z/OS

# MVS JCL Reference

*Version 2 Release 2*

SA23-1385-03

**Note**
Before using this information and the product it supports, read the information in "Notices" on page 697.

This edition applies to Version 2 Release 2 of z/OS (5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

**© Copyright IBM Corporation 1988, 2015.**
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents


**Figures** .. xix

**Tables** .. xxi

**About this document** .. xxiii
Who should use this document .. xxiii
Where to find more information .. xxiii
Related information .. xxiii
Programs .. xxiv
Hardware .. xxiv

**How to send your comments to IBM** xxv
If you have a technical problem .. xxv

**Summary of changes for z/OS Version 2 Release 2 (V2R2) as updated December, 2015** .. xxvii
Summary of changes for z/OS Version 2 Release 2 (V2R2) .. xxvii
Summary of changes for z/OS Version 2 Release 1 (V2R1) as updated December 2013 .. xxviii
z/OS Version 2 Release 1 summary of changes xxix

**Chapter 1. Job control statements** .. 1
JCL statements .. 1
JECL statements .. 2

**Chapter 2. Job Control Tasks** .. 5
Entering Jobs .. 5
Processing Jobs .. 6
Requesting Resources .. 6
Task Charts .. 6

**Chapter 3. Format of statements.** .. 13
JCL statement fields .. 13
Parameter field .. 15
JES2 control statement fields .. 16
JES3 control statement fields .. 16
Continuing statements .. 16
Continuing JCL statements .. 16
Examples of continued statements .. 17
Continuing JES2 control statements .. 18
Continuing JES3 control statements .. 18

**Chapter 4. Syntax of parameters.** .. 19
Notation used to show syntax .. 19
Character sets .. 21
Syntax notes .. 23
Backward references .. 23
Examples of backward references .. 24

**Chapter 5. Procedures and symbols** .. 27
Cataloged and in-stream procedures .. 27
In-stream procedures .. 27
Cataloged procedures .. 27
Using a procedure .. 28
Testing a procedure .. 28
Modifying procedures .. 29
Modifying EXEC statement parameters .. 29
Modifying OUTPUT JCL and DD statements .. 30
Examples of procedures .. 33
Nested procedures .. 35
Nesting procedures .. 35
Modifying nested procedures .. 36
Examples of modifying nested procedures .. 36
Using system symbols and JCL symbols .. 38
What are system symbols? .. 38
What are JCL symbols? .. 39
Coding symbols in JCL .. 43
Determining equivalent JCL .. 45
Examples of defining and coding symbols in JCL 50
Using symbols in nested procedures .. 51
Examples of coding symbols in nested procedures .. 52
Using symbols in JES in-stream data .. 54
JCL symbol service (IEFSJSYM) .. 55
JES symbol service (IAZSYMBL) .. 56
Using symbols in batch JCL .. 56

**Chapter 6. Job control statements on the output listing** .. 57

**Chapter 7. Started tasks** .. 59
Determining whether to use a started task .. 59
Determining the source JCL for the started task .. 59
START command processing when the member is a procedure .. 60
START command processing when the member is a job .. 60
Review current started tasks .. 61
Convert procedures to jobs (optional) .. 61
Determining system services for a started task. .. 63
Deciding under which subsystem a started task should run .. 63
Running a started task under a job entry subsystem .. 63
Running a started task under the master subsystem .. 64
Running a started task that uses catalogs .. 65
Set Up the master JCL .. 65
Coding the JCL .. 65
Naming the JCL member .. 65
Coding the JOB statement for the started task .. 66
Using symbols in started task JCL .. 66
Naming a started task (source JCL is a job) .. 70
Setting up operator education for your started task 71


© Copyright IBM Corp. 1988, 2015

**Chapter 8. JCL command statement .. 73**
Description .. 73
Syntax .. 73
Operation field .. 73
Parameter field .. 73
Comments field .. 74
Location in the JCL .. 74
Defaults .. 74
Examples of the command statement .. 74

**Chapter 9. COMMAND statement .. 75**
Description .. 75
Syntax .. 75
Name field .. 76
Operation field .. 76
Parameter field .. 76
Comments field .. 76
Location in the JCL .. 76
Defaults .. 76
Examples of the COMMAND statement .. 77

**Chapter 10. Comment statement. .. 79**
Description .. 79
Syntax .. 79
Location in the JCL .. 79
Listing of comments statements .. 79
Examples of the comment statement .. 79

**Chapter 11. CNTL statement .. 81**
Description .. 81
Syntax .. 81
Label field .. 81
Operation field .. 81
Parameter field .. 81
Comments field .. 81
Location in the JCL .. 82
Program control statements .. 82
Program control statements in procedures .. 82
Example of the CNTL statement .. 82

**Chapter 12. DD statement .. 83**
Description .. 83
Syntax .. 83
Name field .. 83
Operation field .. 85
Parameter field .. 85
Comments field .. 97
Location in the JCL .. 97
Examples of DD statements and ddnames .. 100
* Parameter .. 101
Syntax .. 101
Defaults .. 101
Relationship to other parameters .. 101
Relationship to other control statements .. 102
Location in the JCL .. 102
Unread records .. 103
Examples of the * parameter .. 103
ACCODE parameter .. 104
Syntax .. 104
Subparameter definition .. 104
Defaults .. 105
Overrides .. 105
Example of the ACCODE parameter .. 105
AMP parameter .. 105
Syntax .. 105
Subparameter definition .. 107
Relationship to other parameters .. 111
Buffer requirements .. 112
Examples of the AMP parameter .. 112
AVGREC parameter .. 113
Syntax .. 113
Subparameter definition .. 113
Overrides .. 114
Relationship to other parameters .. 114
Examples of the AVGREC parameter .. 114
BLKSIZE parameter .. 114
Syntax .. 115
Subparameter definition .. 115
Defaults .. 115
Overrides .. 115
Relationship to other control statements .. 116
Coexistence considerations .. 116
Examples of the BLKSIZE parameter .. 116
BLKSZLIM parameter .. 116
Syntax .. 116
Subparameter definition .. 117
Defaults .. 117
Relationship to other parameters .. 117
Example of the BLKSZLIM parameter .. 117
BURST parameter .. 117
Syntax .. 118
Subparameter definition .. 118
Defaults .. 118
Overrides .. 118
Relationship to other parameters .. 118
Relationship to other control statements .. 118
Example of the BURST parameter .. 119
CCSID parameter .. 119
Syntax .. 119
Subparameter definition .. 119
Default .. 119
Relationship to other parameters .. 119
Examples of the CCSID parameter .. 120
CHARS parameter .. 121
Syntax .. 122
Subparameter definition .. 122
Defaults .. 122
Overrides .. 122
Relationship to other parameters .. 123
Relationship to other control statements .. 123
Requesting a high-density dump .. 123
Examples of the CHARS parameter .. 123
CHKPT parameter .. 123
Syntax .. 124
Subparameter definition .. 124
Overrides .. 124
Relationship to other parameters .. 124
Relationship to the SYSCKEOV DD statement 124
Checkpointing concatenated data sets .. 124
Examples of the CHKPT parameter .. 124
CNTL parameter .. 125

Syntax . . . . . 125
Subparameter definition . . . . . 125
Examples of the CNTL parameter . . . . 125
COPIES parameter . . . . . 126
Syntax . . . . . 126
Subparameter definition . . . . . 126
Defaults . . . . . 127
Overrides . . . . . 127
Relationship to Other Parameters . . . . 127
Relationship to other control statements . . 128
Examples of the COPIES parameter . . . . 128
DATA parameter . . . . . 129
Syntax . . . . . 129
Defaults . . . . . 130
Relationship to other parameters . . . . 130
Relationship to other control statements . . 131
Location in the JCL . . . . . 131
Unread records . . . . . 131
Examples of the DATA parameter . . . . 131
DATACLAS parameter . . . . . 132
Syntax . . . . . 133
Subparameter definition . . . . . 134
Defaults . . . . . 134
Overrides . . . . . 134
Relationship to other parameters . . . . 135
Examples of the DATACLAS parameter . . 135
DCB parameter . . . . . 135
Syntax . . . . . 136
Subparameter definition . . . . . 137
Completing the data control block . . . . 138
Relationship to other parameters . . . . 138
Examples of the DCB parameter . . . . 139
DCB subparameters . . . . . 140
DDNAME parameter . . . . . 150
Syntax . . . . . 151
Subparameter definition . . . . . 151
Overrides . . . . . 151
Relationship to other parameters . . . . 151
Location in the JCL . . . . . 151
Referenced DD statement . . . . . 152
Backward references . . . . . 153
Examples of the DDNAME parameter . . . 154
DEST parameter . . . . . 155
Syntax . . . . . 156
Subparameter definition for JES2 systems . . 156
Subparameter definition for JES3 systems . . 157
Defaults . . . . . 158
Overrides . . . . . 159
Relationship to other parameters . . . . 159
Relationship to other control statements . . 159
Example of the DEST parameter . . . . 159
DISP parameter . . . . . 159
Syntax . . . . . 161
Subparameter definition . . . . . 161
Defaults . . . . . 166
Relationship to other parameters . . . . 167
Disposition of QSAM data sets . . . . . 167
Disposition of generation data sets . . . . 167
Disposition of temporary data sets . . . . 167
Disposition of partitioned data sets (PDSs and PDSEs) . . . . . 167
Adding a volume to a cataloged data set . . 168
DISP=MOD for a multivolume data set. . . 168
Summary of disposition processing . . . . 170
Examples of the DISP parameter . . . . . 171
DLM parameter . . . . . 172
Syntax . . . . . 173
Subparameter definition . . . . . 173
Default . . . . . 173
Relationship to other parameters . . . . . 173
Invalid delimiters . . . . . 174
Example of the DLM parameter . . . . . 174
DSID Parameter . . . . . 174
Syntax . . . . . 175
Subparameter definition . . . . . 175
Relationship to other parameters . . . . . 175
Example of the DSID parameter . . . . . 175
DSNAME parameter . . . . . 176
Syntax . . . . . 176
Subparameter definition . . . . . 177
Relationship to other parameters . . . . . 182
Examples of the DSNAME parameter . . . 183
DSNTYPE parameter . . . . . 184
Syntax . . . . . 185
Subparameter definition . . . . . 185
Defaults . . . . . 186
Overrides . . . . . 186
Relationship to other parameters . . . . . 187
Examples of the DSNTYPE parameter . . . 187
DUMMY parameter . . . . . 188
Syntax . . . . . 189
Parameters on DD DUMMY statements . . 189
Relationship to other parameters . . . . . 189
Relationship to other control statements . . 189
Relationship to access methods . . . . . 190
Examples of the DUMMY parameter . . . 190
DYNAM parameter . . . . . 191
Syntax . . . . . 191
Relationship to other parameters . . . . . 191
Relationship to other control statements . . 191
Example of the DYNAM parameter . . . . 191
EATTR parameter . . . . . 192
Syntax . . . . . 192
Subparameter definition . . . . . 192
Examples of the EATTR parameter . . . . 192
EXPDT parameter . . . . . 193
Syntax . . . . . 193
Subparameter definition . . . . . 193
Overrides . . . . . 194
Relationship to other parameters . . . . . 194
Deleting a data set before its expiration date 194
Examples of the EXPDT parameter . . . . 194
FCB parameter . . . . . 195
Syntax . . . . . 195
Subparameter definition . . . . . 196
Defaults . . . . . 196
Overrides . . . . . 196
Relationship to other parameters . . . . . 196
Relationship to other control statements . . 197
Defining an FCB image for a work station . . 197
Requesting a high-density dump . . . . . 197
Examples of the FCB parameter . . . . . 197

FILEDATA parameter . . . . . 198
Syntax . . . . . 198
Subparameter definition . . . . . 198
Defaults . . . . . 199
Overrides . . . . . 199
Relationship to other parameters . . . . . 199
Example of the FILEDATA parameter . . . 199
FLASH parameter . . . . . 199
Syntax . . . . . 200
Subparameter definition . . . . . 200
Defaults . . . . . 200
Overrides . . . . . 200
Relationship to other parameters . . . . . 200
Relationship to other control statements . . 201
Verification of forms overlay frame . . . . . 201
Printing without flashing . . . . . 201
Example of the FLASH parameter . . . . . 201
FREE parameter . . . . . 201
Syntax . . . . . 202
Subparameter definition . . . . . 202
Defaults . . . . . 202
Overrides . . . . . 202
Relationship to other parameters . . . . . 202
Relationship to other control statements . . 203
Relationship to the CLOSE macro instruction 203
Examples of the FREE parameter . . . . . 203
FREEVOL parameter . . . . . 204
Syntax . . . . . 204
Subparameter definition . . . . . 204
Defaults . . . . . 204
Overrides . . . . . 205
Relationship to other parameters . . . . . 205
Relationship to other control statements . . 205
GDGORDER parameter . . . . . 205
Syntax . . . . . 205
Subparameter definition . . . . . 205
Defaults . . . . . 205
Example of the GDGORDER parameter . . 205
HOLD parameter . . . . . 206
Syntax . . . . . 206
Subparameter definition . . . . . 206
Defaults . . . . . 207
Overrides . . . . . 207
Relationship to other parameters . . . . . 208
Relationship to other control statements . . 208
Examples of the HOLD parameter . . . . . 208
KEYLABL1 parameter . . . . . 209
Syntax . . . . . 209
Subparameter definition . . . . . 209
Defaults . . . . . 209
Overrides . . . . . 209
Relationship to other parameters . . . . . 210
Examples of the KEYLABL1 parameter . . . 210
KEYLABL2 parameter . . . . . 210
Syntax . . . . . 210
Subparameter definition . . . . . 210
Defaults . . . . . 211
Overrides . . . . . 211
Relationship to other parameters . . . . . 211
Examples of the KEYLABL2 parameter . . . 211
KEYENCD1 parameter . . . . . 211
Syntax . . . . . 212
Subparameter definition . . . . . 212
Overrides . . . . . 212
Relationship to other parameters . . . . . 212
Example of the KEYENCD1 parameter . . . 212
KEYENCD2 parameter . . . . . 212
Syntax . . . . . 213
Subparameter definition . . . . . 213
Overrides . . . . . 213
Relationship to other parameters . . . . . 213
Example of the KEYENCD2 parameter . . . 213
KEYLEN parameter . . . . . 213
Syntax . . . . . 214
Subparameter definition . . . . . 214
Overrides . . . . . 214
Relationship to other parameters . . . . . 214
Examples of the KEYLEN parameter . . . . 214
KEYOFF parameter . . . . . 215
Syntax . . . . . 215
Subparameter definition . . . . . 215
Overrides . . . . . 215
Relationship to other parameters . . . . . 215
Example of the KEYOFF parameter . . . . . 216
LABEL parameter . . . . . 216
Syntax . . . . . 216
Subparameter definition . . . . . 217
Defaults . . . . . 220
Relationship to other parameters . . . . . 220
Relationship to other control statements . . 221
Data conversion . . . . . 221
Examples of the LABEL parameter . . . . . 221
LGSTREAM parameter . . . . . 222
Syntax . . . . . 223
Subparameter definition . . . . . 223
Defaults . . . . . 223
Overrides . . . . . 223
Relationship to other parameters . . . . . 223
Example of the LGSTREAM parameter . . . 224
LIKE parameter . . . . . 224
Syntax . . . . . 225
Subparameter definition . . . . . 225
Overrides . . . . . 225
Relationship to other parameters . . . . . 226
Examples of the LIKE parameter . . . . . 226
LRECL parameter . . . . . 226
Syntax . . . . . 226
Subparameter definition . . . . . 226
Overrides . . . . . 227
Relationship to other parameters . . . . . 227
Examples of the LRECL parameter . . . . . 227
MAXGENS parameter . . . . . 228
Syntax . . . . . 228
Subparameter definition . . . . . 228
Relationship to other parameters . . . . . 228
Examples of the MAXGENS parameter . . . 228
MGMTCLAS parameter . . . . . 228
Syntax . . . . . 229
Subparameter definition . . . . . 229
Defaults . . . . . 229
Overrides . . . . . 229
Relationship to other parameters . . . . . 230

Example of the MGMTCLAS parameter .. 230
MODIFY parameter .. 230
Syntax .. 231
Subparameter definition .. 231
Defaults .. 231
Overrides .. 231
Relationship to other parameters .. 231
Relationship to other control statements .. 232
Example of the MODIFY parameter .. 232
OUTLIM parameter .. 232
Syntax .. 232
Subparameter definition .. 232
Default .. 233
Relationship to other parameters .. 233
Relationship to other control statements .. 233
Example of the OUTLIM parameter .. 233
OUTPUT parameter .. 233
Syntax .. 234
Subparameter definition .. 234
Defaults .. 234
Overrides .. 235
Relationship to other parameters .. 235
Location in the JCL .. 235
No match for OUTPUT name .. 236
Processing options in multiple references .. 236
Examples of the OUTPUT parameter .. 236
PATH parameter .. 237
Syntax .. 237
Subparameter definition .. 238
Defaults .. 238
Relationship to other parameters .. 238
Relationship to other statements .. 239
Dummy z/OS UNIX files .. 240
Example of the PATH parameter .. 240
PATHDISP parameter .. 240
Syntax .. 240
Subparameter definition .. 241
Defaults .. 241
Relationship to other parameters .. 241
Example of the PATHDISP parameter .. 241
PATHMODE parameter .. 242
Syntax .. 242
Subparameter definition .. 242
Defaults .. 244
Relationship to other parameters .. 244
Example of the PATHMODE parameter .. 245
PATHOPTS parameter .. 245
Syntax .. 245
Subparameter definition .. 246
Defaults .. 247
Relationship to other parameters .. 247
File status .. 248
Example of the PATHOPTS parameter .. 249
PROTECT parameter .. 249
Syntax .. 249
Subparameter definition .. 250
Overrides .. 250
Relationship to other parameters .. 250
Requirements for protecting a tape data set .. 250
Requirements for protecting a tape volume .. 250
Requirements for protecting a direct access data set .. 251
Examples of the PROTECT parameter .. 251
RECFM parameter .. 251
Coding RECFM for BDAM access method .. 252
Coding RECFM for BPAM access method .. 252
Coding RECFM for BSAM, EXCP, and QSAM access methods .. 253
Overrides .. 254
Relationship to other parameters .. 254
Examples of the RECFM parameter .. 254
RECORG parameter .. 254
Syntax .. 255
Subparameter definition .. 255
Defaults .. 255
Overrides .. 255
Relationship to other parameters .. 255
Example of the RECORG parameter .. 255
REFDD parameter .. 256
Syntax .. 257
Subparameter definition .. 257
Overrides .. 257
Relationship to other parameters .. 257
Examples of the REFDD parameter .. 258
RETPD parameter .. 258
Syntax .. 258
Subparameter definition .. 259
Overrides .. 259
Relationship to other parameters .. 259
Deleting a data set before its retention period passes .. 259
Examples of the RETPD parameter .. 259
RLS parameter .. 260
Syntax .. 260
Subparameter definition .. 260
Overrides .. 261
Relationship to other parameters .. 261
Examples of the RLS parameter .. 261
SECMODEL parameter .. 262
Syntax .. 262
Subparameter definition .. 262
Overrides .. 263
Relationship to other parameters .. 263
Examples of the SECMODEL parameter .. 263
SEGMENT parameter .. 263
Syntax .. 263
Subparameter definition .. 264
Overrides .. 264
Relationship to other parameters .. 264
Example of the segment parameter .. 264
SPACE parameter .. 264
Syntax .. 265
Subparameter definition .. 266
Overrides .. 271
Relationship to other parameters .. 271
SPACE for new data sets with SMS .. 271
Examples of the SPACE parameter .. 271
SPIN parameter .. 273
Syntax .. 273
Subparameter definition .. 273
Defaults .. 274

Overrides .. 274
Relationship to other parameters .. 274
Examples of the SPIN parameter .. 274
STORCLAS parameter .. 275
Syntax .. 276
Subparameter definition .. 276
Defaults .. 276
Overrides .. 276
Relationship to other parameters .. 276
Examples of the STORCLAS parameter .. 277
SUBSYS parameter .. 277
Syntax .. 278
Subparameter definition .. 278
Relationship to other parameters .. 278
Subsystem support for JCL parameters .. 279
Examples of the SUBSYS parameter .. 279
SYMBOLS parameter .. 280
Syntax .. 280
Relationship to other parameters .. 281
Example of the SYMBOLS parameter .. 281
SYMLIST parameter .. 281
Syntax .. 281
Relationship to other parameters .. 282
Example of the SYMLIST parameter .. 282
SYSOUT parameter .. 283
Syntax .. 284
Subparameter definition .. 284
Defaults .. 285
Overrides .. 285
Relationship to other parameters .. 286
Relationship to other control statements .. 286
Starting an external writer when requested .. 287
Held classes in a JES2 system .. 287
Held classes in a JES3 system .. 287
Significance of output classes .. 287
Examples of the SYSOUT parameter .. 287
TERM parameter .. 288
Syntax .. 289
Subparameter definition .. 289
Relationship to other parameters .. 289
Location in the JCL .. 289
Examples of the TERM parameter .. 289
UCS parameter .. 290
Syntax .. 290
Subparameter definition .. 290
Defaults .. 291
Overrides .. 292
Relationship to other parameters .. 292
Using special character sets .. 292
Examples of the UCS parameter .. 292
UNIT parameter .. 293
Syntax .. 293
Subparameter definition .. 293
Overrides .. 297
Relationship to other parameters .. 298
Relationship to other control statements .. 298
Location in the JCL .. 298
Examples of the UNIT parameter .. 298
VOLUME parameter .. 300
Syntax .. 301
Subparameter definition .. 302
Overrides .. 309
Relationship to other parameters .. 309
VOLUME parameter in a JES3 system .. 309
VOLUME parameter for optical readers .. 309
VOLUME parameter for nonspecific volume requests .. 309
VOLUME parameter for specific multi-volume tape requests .. 310
Examples of the VOLUME parameter .. 310

**Chapter 13. Special DD statements 313**

Description .. 313
Syntax .. 313
Special ddnames .. 313
JOBLIB DD statement .. 313
Syntax .. 313
Parameters on JOBLIB DD statements .. 313
Relationship to other control statements .. 315
Location in the JCL .. 315
Relationship of a JOBLIB to a STEPLIB .. 315
Examples of the JOBLIB DD statement .. 315
STEPLIB DD statement .. 316
Syntax .. 317
Parameters on STEPLIB DD statements .. 317
Relationship to other control statements .. 318
Location in the JCL .. 318
Relationship of a STEPLIB to a JOBLIB .. 318
Examples of the STEPLIB DD statement .. 318
SYSABEND, SYSMDUMP, and SYSUDUMP DD statements .. 319
Syntax .. 320
Location in the JCL .. 320
Storing a dump .. 320
Printing a dump .. 321
Overriding dump DD statements .. 321
Duplicate dump requests .. 322
Examples of the SYSABEND, SYSMDUMP, and SYSUDUMP DD statements .. 322
SYSCHK DD statement .. 323
Syntax .. 323
Parameters on SYSCHK DD statements .. 323
Relationship to other control statements .. 324
Location in the JCL .. 325
Examples of the SYSCHK DD statement .. 325
SYSCKEOV DD statement .. 325
Syntax .. 325
Parameters on SYSCKEOV DD statements .. 325
Location in the JCL .. 326
Example of the SYSCKEOV DD statement .. 326
SYSIN DD statement .. 326
Syntax .. 326
Parameters on SYSIN DD statements .. 326
Location in the JCL .. 326
Examples of SYSIN DD statements .. 326

**Chapter 14. Delimiter statement .. 329**

Description .. 329
Syntax .. 329
Comments field .. 329
Relationship to the DLM parameter .. 329

Location in the JCL .. 330
Examples of the delimiter statement .. 330

**Chapter 15. ENDCNTL statement .. 331**
Description .. 331
Syntax .. 331
Label field .. 331
Operation field .. 331
Comments field .. 331
Location in the JCL .. 331
Example of the ENDCNTL statement .. 331

**Chapter 16. EXEC statement .. 333**
Description .. 333
Syntax .. 333
Name field .. 333
Operation field .. 334
Parameter field .. 334
Comments field .. 337
Location in the JCL .. 337
Examples of EXEC statements .. 337
ACCT parameter .. 338
Syntax .. 338
Subparameter definition .. 339
On an EXEC statement that calls a procedure 339
Examples of the ACCT parameter .. 339
ADDRSPC parameter .. 340
Syntax .. 340
Subparameter definition .. 340
Defaults .. 340
Overrides .. 340
Relationship to the EXEC REGION parameter 341
On an EXEC statement that calls a procedure 341
Examples of the ADDRSPC parameter .. 341
CCSID parameter .. 341
Syntax .. 342
Subparameter definition .. 342
Default .. 342
Relationship to other parameters .. 342
Examples of the CCSID parameter .. 342
COND parameter .. 342
Syntax .. 343
Subparameter definition .. 344
Overrides .. 345
Location in the JCL .. 345
On an EXEC statement that calls a procedure 345
Considerations when using the COND parameter .. 346
Summary of COND parameters .. 347
Examples of the COND parameter .. 348
DYNAMNBR parameter .. 350
Syntax .. 350
Subparameter definition .. 350
Defaults .. 351
On an EXEC statement that calls a procedure 351
Example of the DYNAMNBR parameter .. 351
MEMLIMIT parameter .. 351
Syntax .. 351
Subparameter definition .. 351
Defaults .. 352
Overrides .. 352
Relationship to the REGION parameter .. 352
Considerations When Using the MEMLIMIT parameter .. 352
Example of the MEMLIMIT parameter .. 352
PARM parameter .. 352
Syntax .. 353
Subparameter definition .. 353
On an EXEC statement that calls a procedure 353
Examples of the PARM parameter .. 353
PARMDD parameter .. 354
Syntax .. 355
Relationship to other control statements .. 355
Data set requirements .. 355
Record length requirements .. 355
Parameter string requirements .. 355
Examples of the PARMDD parameter .. 356
PERFORM parameter .. 356
Syntax .. 357
Subparameter definition .. 357
Defaults .. 357
Overrides .. 357
On an EXEC statement that calls a procedure 358
Example of the PERFORM parameter .. 358
PGM parameter .. 358
Syntax .. 358
Subparameter definition .. 358
Examples of the PGM parameter .. 359
PROC and procedure name parameters .. 360
Syntax .. 360
Subparameter definition .. 360
Effect of PROC parameter on other parameters and following statements .. 360
Examples of the PROC parameter .. 360
RD parameter .. 361
Syntax .. 362
Subparameter definition .. 362
Defaults .. 363
Overrides .. 363
Relationship to other control statements .. 363
On an EXEC statement that calls a procedure 363
Examples of the RD parameter .. 363
REGION parameter .. 364
Syntax .. 364
Subparameter definition .. 364
Defaults .. 365
Overrides .. 365
Relationship to the EXEC ADDRSPC parameter 366
On an EXEC statement that calls a procedure 366
Relationship to the MEMLIMIT parameter .. 366
| Relationship to the REGIONX parameter .. 366
Considerations when using the REGION parameter .. 366
Examples of the REGION parameter .. 366
| REGIONX parameter .. 366
| Syntax .. 367
| Subparameter definition .. 367
| Defaults .. 367
| Overrides .. 368
| Relationship to the EXEC ADDRSPC parameter 368
| On an EXEC statement that calls a procedure 368

| Relationship to the MEMLIMIT parameter. .. 368
| Relationship to the REGION parameter. . .. 369
| Examples of the REGIONX parameter . . .. 369
RLSTMOUT parameter . . . . . .. 369
Syntax. . . . . .. 369
Defaults . . . . . .. 370
Examples of the RLSTMOUT parameter . .. 370
TIME parameter . . . . . .. 370
Syntax. . . . . .. 370
Subparameter definition . . . . . .. 370
Defaults . . . . . .. 371
Overrides . . . . . .. 371
On an EXEC statement that calls a procedure 371
Examples of the TIME parameter. . . . . .. 372

**Chapter 17. EXPORT statement. . .. 375**
Description . . . . . .. 375
Syntax. . . . . .. 375
Label field . . . . . .. 375
Operation field . . . . . .. 375
Parameter field . . . . . .. 375
Comments field . . . . . .. 376
Location in the JCL . . . . . .. 376
SYMLIST parameter . . . . . .. 376
Syntax. . . . . .. 376
Subparameter definition . . . . . .. 377
Examples. . . . . .. 377

**Chapter 18. IF/THEN/ELSE/ENDIF statement construct . . . . . .. 379**
Description . . . . . .. 379
Syntax. . . . . .. 379
Name field . . . . . .. 379
Operation field . . . . . .. 380
Relational-expression field . . . . . .. 380
Comments field . . . . . .. 385
Location in the JCL . . . . . .. 385
Relationship to other parameters . . . . .. 385
THEN and ELSE clauses . . . . . .. 386
Considerations when using the IF/THEN/ELSE/ENDIF construct . . . .. 386
Examples of IF/THEN/ELSE/ENDIF statement constructs . . . . . .. 388

**Chapter 19. INCLUDE statement . .. 393**
Description . . . . . .. 393
Syntax. . . . . .. 393
Name field . . . . . .. 393
Operation field . . . . . .. 393
Parameter field . . . . . .. 393
Comments field . . . . . .. 394
Location in the JCL . . . . . .. 394
Considerations for using INCLUDE groups .. 394
Examples of the INCLUDE statement: . . .. 395

**Chapter 20. JCLLIB statement . . .. 397**
Description . . . . . .. 397
Syntax. . . . . .. 397
Name field . . . . . .. 397
Operation field . . . . . .. 398
Parameter field . . . . . .. 398
Comments field . . . . . .. 399
Location in the JCL . . . . . .. 399
Considerations for using the JCLLIB statement 400
Examples of the JCLLIB statement . . . .. 400

**Chapter 21. JOB statement . . . .. 403**
Description . . . . . .. 403
Syntax. . . . . .. 403
Name field . . . . . .. 403
Operation field . . . . . .. 404
Parameter field . . . . . .. 404
Comments field . . . . . .. 408
Location in the JCL . . . . . .. 408
Examples of JOB statements . . . . . .. 408
Accounting information parameter . . . . .. 409
Syntax. . . . . .. 409
Subparameter definition . . . . . .. 410
Relationship to other control statements . .. 410
JES2 accounting information format . . . .. 410
Syntax. . . . . .. 410
Examples of the accounting information parameter . . . . . .. 411
ADDRSPC parameter. . . . . .. 412
Syntax. . . . . .. 412
Subparameter definition . . . . . .. 412
Defaults . . . . . .. 412
Overrides . . . . . .. 412
Relationship to the JOB REGION parameter .. 412
Examples of the ADDRSPC parameter . . .. 413
BYTES parameter . . . . . .. 413
Syntax. . . . . .. 413
Subparameter definition . . . . . .. 413
Defaults . . . . . .. 414
Overrides . . . . . .. 414
Relationship to other parameters . . . . .. 414
Relationship to other control statements . .. 414
Examples of the BYTES parameter . . . .. 414
CARDS parameter. . . . . .. 415
Syntax. . . . . .. 415
Subparameter definition . . . . . .. 415
Defaults . . . . . .. 416
Overrides . . . . . .. 416
Relationship to other parameters . . . . .. 416
Relationship to other control statements . .. 416
Examples of the CARDS parameter . . . .. 416
CCSID parameter . . . . . .. 417
Syntax. . . . . .. 417
Subparameter definition . . . . . .. 417
Default . . . . . .. 417
Overrides . . . . . .. 417
Relationship to other parameters . . . . .. 417
Examples of the CCSID parameter . . . .. 418
CLASS parameter . . . . . .. 418
Syntax. . . . . .. 418
Subparameter definition . . . . . .. 419
Defaults . . . . . .. 419
Overrides . . . . . .. 419
Relationship to other control statements . .. 419
Example of the CLASS parameter . . . .. 419
COND parameter . . . . . .. 419

Syntax . . . . . 420
Subparameter definition . . . . . 420
Overrides . . . . . 420
Summary of COND parameters . . . . . 421
Examples of the COND parameter . . . . 421
DSENQSHR parameter . . . . . 421
Syntax . . . . . 422
Subparameter definition . . . . . 422
Defaults . . . . . 422
Overrides . . . . . 422
Relationship to other control statements . . 422
Examples of the DSENQSHR parameter . . 422
GROUP parameter . . . . . 423
Syntax . . . . . 424
Subparameter definition . . . . . 424
Defaults . . . . . 424
Example of the GROUP parameter . . . . 424
JESLOG parameter . . . . . 424
Syntax . . . . . 424
Subparameter definition . . . . . 424
Defaults . . . . . 425
Examples of the JESLOG parameter . . . . 425
JOBRC parameter . . . . . 426
Syntax . . . . . 426
Subparameter definition . . . . . 426
Defaults . . . . . 426
Overrides . . . . . 426
Relationship to other control statements . . 426
Examples of the JOBRC parameter . . . . 427
LINES parameter . . . . . 427
Syntax . . . . . 427
Subparameter definition . . . . . 427
Defaults . . . . . 428
Overrides . . . . . 428
Relationship to other parameters . . . . . 428
Relationship to other control statements . . 428
Examples of the LINES parameter . . . . 428
MEMLIMIT parameter . . . . . 429
Syntax . . . . . 429
Subparameter definition . . . . . 429
Defaults . . . . . 429
Overrides . . . . . 429
Relationship to the REGION parameter. . . 429
Considerations when using the MEMLIMIT parameter . . . . . 430
Examples of the MEMLIMIT parameter . . 430
MSGCLASS parameter . . . . . 430
Syntax . . . . . 430
Subparameter definition . . . . . 430
Defaults . . . . . 431
Significance of output classes . . . . . 431
Examples of the MSGCLASS parameter . . 431
MSGLEVEL parameter . . . . . 432
Syntax . . . . . 432
Subparameter definition . . . . . 432
Defaults . . . . . 433
Examples of the MSGLEVEL parameter . . 433
NOTIFY parameter . . . . . 433
Syntax . . . . . 434
Subparameter definition for JES2 systems . . 434
Subparameter definition for JES3 systems . . 434
Receiving notification of job completion . . 434
Examples of the NOTIFY parameter . . . . 435
PAGES parameter . . . . . 435
Syntax . . . . . 435
Subparameter definition . . . . . 435
Defaults . . . . . 436
Overrides . . . . . 436
Relationship to other parameters . . . . . 436
Relationship to other control statements . . 436
Examples of the PAGES parameter . . . . 436
PASSWORD parameter . . . . . 437
Syntax . . . . . 438
Subparameter definition . . . . . 438
Relationship to other parameters . . . . . 438
Examples of the PASSWORD parameter . . 438
PERFORM parameter. . . . . 438
Syntax . . . . . 439
Subparameter definition . . . . . 439
Defaults . . . . . 439
Overrides . . . . . 439
Examples of the PERFORM parameter . . . 440
Programmer's name parameter . . . . . 440
Syntax . . . . . 440
Parameter definition . . . . . 441
Examples of the programmer's name parameter 441
PRTY parameter . . . . . 442
Syntax . . . . . 442
Subparameter definition . . . . . 442
Defaults . . . . . 442
Example of the PRTY parameter . . . . . 443
RD parameter . . . . . 443
Syntax . . . . . 444
Subparameter definition . . . . . 444
Defaults . . . . . 445
Overrides . . . . . 445
Relationship to other control statements . . 445
Examples of the RD parameter . . . . . 445
REGION parameter . . . . . 446
Syntax . . . . . 446
Subparameter definition . . . . . 446
Defaults . . . . . 447
Overrides . . . . . 447
Relationship to the JOB ADDRSPC parameter 447
Relationship to the MEMLIMIT parameter. . 448
| Relationship to the REGIONX parameter . . 448
Considerations when using the REGION parameter . . . . . 448
Examples of the REGION parameter . . . 448
| REGIONX parameter . . . . . 448
| Syntax . . . . . 448
| Subparameter definition . . . . . 449
| Defaults . . . . . 449
| Overrides . . . . . 449
| Relationship to the JOB ADDRSPC parameter 450
| On a JOB statement that calls a procedure . . 450
| Relationship to the MEMLIMIT parameter. . 450
| Examples of the REGIONX parameter . . . 450
RESTART parameter . . . . . 450
Syntax . . . . . 451
Subparameter definition . . . . . 451
Relationship to other control statements . . 452

Cautions when coding the RESTART parameter 452
Generation data sets in restarted jobs .. 452
Examples of the RESTART parameter .. 452
SECLABEL parameter .. 453
Syntax .. 454
Subparameter definition .. 454
Defaults .. 454
Relationship to other parameters .. 454
Example of the SECLABEL parameter .. 454
SCHENV parameter .. 454
Syntax .. 455
Subparameter definition .. 455
Defaults .. 455
Relationship to other control statements .. 455
Example of the SCHENV parameter .. 455
SYSAFF parameter .. 455
Syntax .. 456
Subparameter definition .. 456
Defaults .. 457
Relationship to other control statements .. 457
Examples of the SYSAFF parameter .. 457
SYSTEM parameter .. 457
Syntax .. 458
Subparameter definition .. 458
Relationship to other control statements .. 458
Examples of the SYSTEM parameter .. 458
TIME parameter .. 459
Syntax .. 459
Subparameter definition .. 459
Defaults .. 460
Overrides .. 460
Examples of the TIME parameter .. 460
Examples of the TIME parameter on JOB and EXEC statements .. 461
TYPRUN parameter .. 462
Syntax .. 462
Subparameter definition .. 462
Relationship to other control statements .. 463
Example of the TYPRUN parameter .. 463
UJOBCORR parameter .. 464
Syntax .. 464
Subparameter definition .. 465
Examples of the UJOBCORR parameter .. 465
USER parameter .. 465
Syntax .. 466
Subparameter definition .. 466
Defaults .. 466
Relationship to other parameters .. 466
Example of the USER parameter .. 466

**Chapter 22. Null Statement .. 467**

Description .. 467
Syntax .. 467
Location in the JCL .. 467
Example of the null statement .. 467

**Chapter 23. OUTPUT JCL statement 469**

Description .. 469
Syntax .. 469
Name field .. 469
Operation field .. 469
Parameter field .. 470
Comments field .. 478
Location in the JCL .. 478
Overrides .. 479
Relationship to sysout DD statement .. 480
Relationship to the JES2 /*OUTPUT statement 480
Relationship to the JES3 //*FORMAT statement 480
ADDRESS parameter .. 480
Syntax .. 481
Subparameter definition .. 481
Defaults .. 481
Overrides .. 481
Examples of the ADDRESS parameter .. 481
AFPPARMS parameter .. 482
Syntax .. 482
Parameter definition .. 482
Defaults .. 482
Overrides .. 482
Relationship to other control statements .. 482
Example of the AFPPARM keyword .. 482
AFPSTATS parameter .. 483
Syntax .. 483
Parameter definition .. 483
Defaults .. 483
Overrides .. 483
Relationship to other control statements .. 484
Example of the AFPSTATS keyword .. 484
BUILDING parameter .. 484
Syntax .. 484
Subparameter definition .. 484
Defaults .. 484
Overrides .. 485
Example of the BUILDING parameter .. 485
BURST parameter .. 485
Syntax .. 485
Subparameter definition .. 485
Defaults .. 485
Overrides .. 486
Example of the BURST parameter .. 486
CHARS parameter .. 486
Syntax .. 486
Subparameter definition .. 486
Defaults .. 487
Overrides .. 487
Requesting a high-density dump .. 487
Example of the CHARS parameter .. 487
CKPTLINE parameter .. 488
Syntax .. 488
Subparameter definition .. 488
Defaults .. 488
Example of the CKPTLINE parameter .. 488
CKPTPAGE parameter .. 488
Syntax .. 489
Subparameter definition .. 489
Defaults .. 489
Relationship to other parameters .. 489
Example of the CKPTPAGE parameter .. 489
CKPTSEC parameter .. 489
Syntax .. 489
Subparameter definition .. 489

Defaults .. 490
Relationship to other parameters .. 490
Example of the CKPTSEC parameter .. 490
CLASS parameter .. 490
Syntax .. 490
Subparameter definition .. 490
Overrides .. 490
Held Classes in a JES2 system .. 491
Held Classes in a JES3 system .. 491
Significance of output classes .. 491
Examples of the CLASS parameter .. 492
COLORMAP parameter .. 492
Syntax .. 492
Subparameter definition .. 492
Example of the COLORMAP parameter .. 492
COMPACT parameter .. 492
Syntax .. 493
Subparameter definition .. 493
Defaults .. 493
Overrides .. 493
Example of the COMPACT parameter .. 493
COMSETUP parameter .. 493
Syntax .. 493
Subparameter definition .. 494
Example of the COMSETUP parameter .. 494
CONTROL parameter .. 494
Syntax .. 494
Subparameter definition .. 494
Defaults .. 494
Example of the CONTROL parameter .. 495
COPIES parameter .. 495
Syntax .. 495
Subparameter definition .. 495
Defaults .. 496
Overrides .. 496
Relationship to other parameters .. 496
Relationship to other control statements .. 496
Examples of the COPIES parameter .. 496
COPYCNT parameter .. 497
Syntax .. 497
Subparameter definition .. 497
Defaults .. 497
Overrides .. 497
Relationship to other parameters .. 497
Relationship to other control statements .. 497
Examples of the COPYCNT parameter .. 498
DATACK parameter .. 498
Syntax .. 498
Subparameter definition .. 498
Defaults .. 499
Relationship to other parameters .. 499
Example of the DATACK parameter .. 499
DDNAME parameter .. 499
Syntax .. 499
Subparameter definition .. 499
Example of the DDNAME parameter .. 500
DEFAULT parameter .. 500
Syntax .. 500
Subparameter definition .. 500
Defaults .. 500
Location in the JCL .. 501
References to default OUTPUT JCL statements 501
Example of the DEFAULT parameter .. 501
DEPT parameter .. 502
Syntax .. 502
Subparameter definition .. 502
Defaults .. 503
Overrides .. 503
Example of the DEPT parameter .. 503
DEST parameter .. 503
Syntax .. 504
Subparameter definition for JES2 systems .. 504
Subparameter definition for JES3 systems .. 505
Defaults .. 506
Overrides .. 506
Relationship to other parameters .. 506
Examples of the DEST parameter .. 506
DPAGELBL parameter .. 507
Syntax .. 507
Subparameter definition .. 508
Defaults .. 508
Relationship to other parameters .. 508
Example of the DPAGELBL parameter .. 508
DUPLEX parameter .. 508
Syntax .. 508
Subparameter definition .. 508
Relationship to other keywords on this statement .. 509
Example of the DUPLEX parameter .. 509
FCB parameter .. 509
Syntax .. 510
Subparameter definition .. 510
Defaults .. 510
Overrides .. 510
Relationship to other parameters .. 510
Requesting a high-density dump .. 510
Example of the FCB parameter .. 511
FLASH parameter .. 511
Syntax .. 511
Subparameter definition .. 511
Defaults .. 512
Overrides .. 512
Relationship to other parameters .. 512
Verification of forms overlay frame .. 512
Printing without flashing .. 512
Example of the FLASH parameter .. 512
FORMDEF parameter .. 513
Syntax .. 513
Subparameter definition .. 513
Overrides .. 513
Example of the FORMDEF parameter .. 514
FORMLEN parameter .. 514
Syntax .. 514
Subparameter definition .. 514
Relationship to other control statements .. 514
Examples of the FORMLEN parameter .. 514
FORMS parameter .. 515
Syntax .. 515
Subparameter definition .. 515
Defaults .. 515
Overrides .. 515
Example of the FORMS parameter .. 515

FSSDATA parameter . . . . . 515
Syntax . . . . . 516
Subparameter definition . . . . . 516
Defaults . . . . . 516
Overrides . . . . . 517
Relationship to other keywords on this statement . . . . . 517
Relationship to other system functions . . . 517
Examples of the FSSDATA parameter . . . 517
GROUPID parameter . . . . . 518
Syntax . . . . . 518
Subparameter definition . . . . . 518
Relationship to other control statements . . 518
Examples of the GROUPID parameter . . . 518
INDEX parameter . . . . . 519
Syntax . . . . . 520
Subparameter definition . . . . . 520
Defaults . . . . . 520
Relationship to other parameters . . . . . 520
Example of the INDEX parameter . . . . 520
INTRAY parameter . . . . . 520
Syntax . . . . . 520
Subparameter definition . . . . . 520
Relationship to other keywords on this statement . . . . . 520
Example of the INTRAY parameter . . . . 521
JESDS parameter . . . . . 521
Syntax . . . . . 521
Subparameter definition . . . . . 521
Overrides . . . . . 522
Location in the JCL . . . . . 522
Destination for the system data sets . . . . 522
JES2 processing with JESDS . . . . . 522
JES3 processing with JESDS . . . . . 522
Example of the JESDS parameter . . . . . 522
LINDEX parameter . . . . . 523
Syntax . . . . . 523
Subparameter definition . . . . . 523
Defaults . . . . . 523
Relationship to other parameters . . . . . 523
Example of the LINDEX parameter . . . . 523
LINECT parameter . . . . . 524
Syntax . . . . . 524
Subparameter definition . . . . . 524
Defaults . . . . . 524
Example of the LINECT parameter . . . . 524
MAILBCC parameter . . . . . 524
Syntax . . . . . 524
Subparameter definition . . . . . 525
Defaults . . . . . 525
Overrides . . . . . 525
Relationship to other system functions . . . 525
Examples of the MAILBCC parameter . . . 525
MAILCC parameter . . . . . 525
Syntax . . . . . 525
Subparameter definition . . . . . 526
Defaults . . . . . 526
Overrides . . . . . 526
Relationship to other system functions . . . 526
Examples of the MAILCC parameter . . . . 526
MAILFILE parameter . . . . . 526
Syntax . . . . . 526
Subparameter definition . . . . . 527
Defaults . . . . . 527
Overrides . . . . . 527
Relationship to other system functions . . . 527
Example of the MAILFILE parameter . . . 527
MAILFROM parameter . . . . . 527
Syntax . . . . . 527
Subparameter definition . . . . . 527
Defaults . . . . . 527
Overrides . . . . . 528
Relationship to other system functions . . . 528
Example of the MAILFROM parameter . . . 528
MAILTO parameter . . . . . 528
Syntax . . . . . 528
Subparameter definition . . . . . 528
Defaults . . . . . 528
Overrides . . . . . 528
Relationship to other system functions . . . 528
Example of the MAILTO parameter . . . . 528
MERGE parameter . . . . . 529
Syntax . . . . . 529
Subparameter definition . . . . . 529
Defaults . . . . . 529
Example of the MERGE parameter . . . . 529
MODIFY parameter . . . . . 529
Syntax . . . . . 530
Subparameter definition . . . . . 530
Defaults . . . . . 530
Overrides . . . . . 531
Relationship to other parameters . . . . . 531
Example of the MODIFY parameter . . . . 531
NAME parameter . . . . . 531
Syntax . . . . . 531
Subparameter definition . . . . . 531
Defaults . . . . . 531
Overrides . . . . . 532
Example of the NAME parameter . . . . 532
NOTIFY parameter . . . . . 532
Syntax . . . . . 532
Subparameter definitions . . . . . 533
Defaults . . . . . 533
Examples of the NOTIFY parameter . . . . 533
OFFSETXB parameter . . . . . 533
Syntax . . . . . 533
Subparameter definition . . . . . 533
Relationship to other keywords on this statement . . . . . 534
Example of the OFFSETXB parameter . . . 534
OFFSETXF parameter . . . . . 534
OFFSETYB parameter . . . . . 534
OFFSETYF parameter . . . . . 534
OUTBIN parameter . . . . . 534
Syntax . . . . . 534
Subparameter definition . . . . . 534
Defaults . . . . . 535
Overrides . . . . . 535
Relationship to other system functions . . . 535
Example of the OUTBIN parameter . . . . 535
OUTDISP parameter . . . . . 535
Syntax . . . . . 535

Subparameter definitions .. 536
Defaults .. 536
Overrides .. 536
Relationship to other control statements .. 536
Examples of the OUTDISP parameter .. 537
OVERLAYB parameter .. 537
Syntax .. 537
Subparameter definition .. 538
Relationship to other keywords on this statement .. 538
Example of the OVERLAYB parameter .. 538
OVERLAYF parameter .. 538
OVFL parameter .. 538
Syntax .. 538
Subparameter definition .. 538
Defaults .. 538
Example of the OVFL parameter .. 539
PAGEDEF parameter .. 539
Syntax .. 539
Subparameter definition .. 540
Overrides .. 540
Example of the PAGEDEF parameter .. 540
PIMSG parameter .. 540
Syntax .. 541
Subparameter definition .. 541
Defaults .. 541
Examples of the PIMSG parameter .. 541
PORTNO parameter .. 542
Syntax .. 542
Subparameter definition .. 542
Relationship to other system functions .. 542
Example of the PORTNO parameter .. 542
PRMODE parameter .. 542
Syntax .. 542
Subparameter definition .. 542
Defaults .. 543
Printing a line-mode data set using PSF .. 543
Example of the PRMODE parameter .. 543
PRTATTRS parameter .. 543
Syntax .. 543
Parameter definition .. 544
Defaults .. 544
Overrides .. 544
Relationship to other keywords on this statement .. 544
Relationship to other control statements .. 544
Example of the PRTATTRS parameter .. 544
PRTERROR parameter .. 544
Syntax .. 544
Subparameter definition .. 544
Relationship to other control statements .. 545
Examples of the PRTERROR parameter .. 545
PRTOPTNS parameter .. 545
Syntax .. 545
Subparameter definition .. 546
Relationship to other system functions .. 546
Example of the PRTOPTNS parameter .. 546
PRTQUEUE parameter .. 546
Syntax .. 546
Subparameter definition .. 546
Relationship to other system functions .. 546
Example of the PRTQUEUE parameter .. 546
PRTY parameter .. 546
Syntax .. 547
Subparameter definition .. 547
Defaults .. 547
Overrides .. 547
Example of the PRTY parameter .. 547
REPLYTO parameter .. 547
Syntax .. 547
Subparameter definition .. 547
Defaults .. 548
Overrides .. 548
Relationship to other system functions .. 548
Example of the REPLYTO parameter .. 548
RESFMT parameter .. 548
Syntax .. 548
Subparameter definition .. 548
Relationship to other control statements .. 548
Example of the RESFMT parameter .. 548
RETAINS *and* RETAINF parameters .. 548
Syntax .. 549
Subparameter definition .. 549
Relationship to other control statements .. 549
Relationship to other system functions .. 549
Examples of the RETAIN keywords .. 549
RETRYL *and* RETRYT parameters .. 550
Syntax .. 550
Subparameter definition .. 550
Relationship to other control statements .. 550
Relationship to other system functions .. 550
Examples of the RETRY keywords .. 551
ROOM parameter .. 551
Syntax .. 551
Subparameter definition .. 551
Defaults .. 551
Overrides .. 552
Example of the ROOM parameter .. 552
SYSAREA parameter .. 552
Syntax .. 552
Subparameter definition .. 552
Defaults .. 553
Relationship to other parameters .. 553
Example of the SYSAREA parameter .. 553
THRESHLD parameter .. 553
Syntax .. 553
Subparameter definition .. 553
Defaults .. 553
Example of the THRESHLD parameter .. 554
TITLE parameter .. 554
Syntax .. 554
Subparameter definition .. 554
Example of the TITLE parameter .. 554
TRC parameter .. 555
Syntax .. 555
Subparameter definition .. 555
Defaults .. 555
Relationship to other parameters .. 555
Example of the TRC parameter .. 555
UCS parameter .. 556
Syntax .. 556
Subparameter definition .. 556

Defaults . . . . . 556
Overrides . . . . . 557
Using special characters sets . . . . . 557
Example of the UCS parameter . . . . . 558
USERDATA parameter . . . . . 558
Syntax. . . . . 558
Subparameter definition . . . . . 558
Defaults . . . . . 558
Overrides . . . . . 558
Relationship to other keywords on this statement. . . . . 559
Relationship to other control statements . . 559
Relationship to other system functions . . . 559
Examples of the USERDATA parameter . . 559
USERLIB parameter . . . . . 561
Syntax. . . . . 561
Subparameter definitions . . . . . 561
Defaults . . . . . 562
Overrides . . . . . 562
Requirements for USERLIB libraries . . . . 562
Examples of the USERLIB parameter . . . 562
USERPATH parameter . . . . . 562
Syntax. . . . . 563
Subparameter definitions . . . . . 563
Defaults . . . . . 563
Overrides . . . . . 563
Relationship to other system functions . . . 563
Examples of the USERPATH parameter. . . 563
WRITER parameter . . . . . 564
Syntax. . . . . 564
Subparameter definition . . . . . 564
Defaults . . . . . 564
Overrides . . . . . 564
Relationship to other parameters . . . . . 564
Starting an external writer . . . . . 564
Examples of the WRITER parameter. . . . . 564

**Chapter 24. PEND statement . . . . . 567**
Description . . . . . 567
Syntax. . . . . 567
Name field . . . . . 567
Operation field . . . . . 567
Comments field . . . . . 567
Location in the JCL . . . . . 567
Examples of the PEND statement. . . . . 567

**Chapter 25. PROC statement . . . . . 569**
Description . . . . . 569
Syntax. . . . . 569
Name field . . . . . 569
Operation field . . . . . 570
Parameter field . . . . . 570
Comments field . . . . . 570
Overrides . . . . . 570
Location in the JCL . . . . . 570
Examples of the PROC statement. . . . . 570

**Chapter 26. SCHEDULE statement 571**
Description . . . . . 571
Syntax. . . . . 571
Name field . . . . . 571
Operation field . . . . . 571
Parameter field . . . . . 571
Comments field . . . . . 572
Location in the JCL . . . . . 572
Examples of SCHEDULE statement . . . . 572
HOLDUNTL Parameter . . . . . 572
Syntax. . . . . 572
Subparameter definition . . . . . 572
Relationship to other parameters . . . . . 573
JOBGROUP Parameter . . . . . 573
Syntax. . . . . 573
Subparameter definition . . . . . 573
Defaults . . . . . 573
Relationship to other parameters . . . . . 573
Example of the JOBGROUP parameter . . . 573
STARTBY Parameter . . . . . 574
Syntax. . . . . 574
Subparameter definition . . . . . 574
Relationship to other parameters . . . . . 574
WITH Parameter . . . . . 575
Syntax. . . . . 575
Subparameter definition . . . . . 575

**Chapter 27. SET statement . . . . . 577**
Description . . . . . 577
Syntax. . . . . 577
Name field . . . . . 578
Operation field . . . . . 578
Parameter field . . . . . 578
Comments field . . . . . 578
Overrides . . . . . 579
Location in the JCL . . . . . 579
Relationship to other control statements . . 579
Considerations for using the SET statement . . 579
Examples of the SET statement . . . . . 580

**Chapter 28. XMIT JCL statement . . . . . 583**
Description . . . . . 584
Syntax. . . . . 584
Name field . . . . . 584
Operation field . . . . . 584
Parameter field . . . . . 584
Comments field . . . . . 584
Location in the JCL . . . . . 584
Error on XMIT JCL statement . . . . . 585
Examples of the XMIT JCL statement . . . 585
DEST parameter . . . . . 586
Syntax. . . . . 586
Subparameter definition . . . . . 586
Examples of the DEST parameter. . . . . 586
DLM parameter . . . . . 587
Syntax. . . . . 587
Subparameter definition . . . . . 587
Default . . . . . 587
Invalid delimiters . . . . . 587
Examples of the DLM parameter . . . . . 587
SUBCHARS parameter . . . . . 588
Syntax. . . . . 588
Subparameter definition . . . . . 588

Default . . . . . 589
Invalid substitute . . . . . 589
Examples of the SUBCHARS parameter . . 589

## Chapter 29. JES2 Execution Control Statements . . . . . 591

Examples of a JOBGROUP and associated jobs . . 591
JOBGROUP logging job . . . . . 592
Other attributes of a JOBGROUP logging job . . 592
JES2 exit processing for JOBGROUP JCL . . . 593
Associating jobs with a job group . . . . . 593
Processing for jobs in a job group . . . . . 593
Configuring and activating job groups . . . . 593
JOBGROUP statement . . . . . 594
Description . . . . . 594
GJOB statement . . . . . 598
Description . . . . . 598
JOBSET statement . . . . . 599
Description . . . . . 599
SJOB statement . . . . . 600
Description . . . . . 600
ENDSET statement . . . . . 601
Description . . . . . 601
BEFORE statement . . . . . 602
Description . . . . . 602
AFTER statement . . . . . 605
Description . . . . . 605
CONCURRENT statement . . . . . 607
Description . . . . . 608
ENDGROUP statement . . . . . 609
Description . . . . . 609

## Chapter 30. JES2 control statements 611

Description . . . . . 611
Considerations for started tasks . . . . . 611
Considerations for an APPC scheduling environment . . . . . 611
Location in the JCL . . . . . 611
Internal reader . . . . . 611
JES2 command statement . . . . . 611
Syntax . . . . . 612
Parameter definition . . . . . 612
Location in the JCL . . . . . 613
Examples of the command statement . . . 613
/*JOBPARM statement . . . . . 613
Syntax . . . . . 614
Parameter definition . . . . . 614
Overrides . . . . . 618
Location in the JCL . . . . . 618
Execution node . . . . . 618
Examples of the /*JOBPARM statement . . 619
/*MESSAGE statement . . . . . 619
Syntax . . . . . 619
Relationship to the /*ROUTE XEQ statement 619
Location in the JCL . . . . . 619
Example of the /*MESSAGE statement . . . 620
/*NETACCT statement . . . . . 620
Syntax . . . . . 620
Parameter definition . . . . . 620
Defaults . . . . . 620
Overrides . . . . . 620
Location in the JCL . . . . . 620
Example of the /*NETACCT statement . . . 620
/*NOTIFY statement . . . . . 621
Syntax . . . . . 621
Parameter definition . . . . . 621
Overrides . . . . . 621
Location in the JCL . . . . . 622
Examples of the NOTIFY statement . . . . 622
/*OUTPUT statement . . . . . 622
Syntax . . . . . 623
Parameter definition . . . . . 624
Overrides . . . . . 630
Relationship to other control statements . . 630
Location in the JCL . . . . . 630
Example of the /*OUTPUT statement . . . 630
/*PRIORITY statement . . . . . 631
Syntax . . . . . 631
Parameter definition . . . . . 631
Overrides . . . . . 631
Relationship to other control statements . . 631
Location in the JCL . . . . . 631
Example of the PRIORITY statement . . . 632
/*ROUTE statement . . . . . 632
Syntax . . . . . 632
Parameter definition . . . . . 632
Location in the JCL . . . . . 634
Processing of /*ROUTE statements . . . . 634
Multiple /*ROUTE statements . . . . . 634
Examples of the ROUTE statement . . . . 634
/*SETUP statement . . . . . 635
Syntax . . . . . 635
Parameter definition . . . . . 635
Location in the JCL . . . . . 636
Example of the /*SETUP statement . . . . 636
/*SIGNOFF statement . . . . . 636
Syntax . . . . . 636
Location in the JCL . . . . . 636
Example of the /*SIGNOFF statement . . . 636
/*SIGNON statement . . . . . 637
Syntax . . . . . 637
Parameter definition . . . . . 637
Location in the JCL . . . . . 638
Examples of the /*SIGNON statement . . . 638
/*XEQ statement . . . . . 639
Syntax . . . . . 639
Parameter definition . . . . . 639
Location in the JCL . . . . . 640
Multiple /*XEQ statements . . . . . 640
Example of the XEQ statement . . . . . 640
/*XMIT statement . . . . . 640
Syntax . . . . . 641
Parameter definition . . . . . 641
Defaults . . . . . 642
Location in the JCL . . . . . 642
Examples of the XMIT statement . . . . . 642

## Chapter 31. JES3 control statements 643

Description . . . . . 643
Considerations for an APPC scheduling environment . . . . . 643

Considerations for started tasks .. 643
Location in the JCL .. 643
Internal reader .. 643
Examples of JES3 control statements .. 643
JES3 control statement tracking .. 644
JES3 command statement .. 644
Syntax .. 645
Parameter definition .. 645
Location in the JCL .. 646
Examples of the command statement .. 646
//*DATASET statement .. 646
Syntax .. 647
Parameter definition .. 647
Location in the JCL .. 648
Example of the //*DATASET statement .. 648
//*ENDDATASET statement .. 648
Syntax .. 648
Location in the JCL .. 648
Example of the //*ENDDATASET statement 648
//*ENDPROCESS statement .. 649
Syntax .. 649
Location in the JCL .. 649
Example of the //*ENDPROCESS statement 649
//*FORMAT PR statement .. 649
Syntax .. 650
Parameter definition .. 651
Relationship to sysout DD and OUTPUT JCL statements .. 658
Relationship to //*PROCESS statement .. 658
Location in the JCL .. 658
Examples of the //*FORMAT PR statement .. 658
//*FORMAT PU statement .. 659
Syntax .. 660
Parameter definition .. 660
Relationship to sysout DD and OUTPUT JCL statements .. 663
Relationship to //*PROCESS statement .. 663
Location in the JCL .. 663
Examples of the //*FORMAT PU statement .. 664
//*MAIN statement .. 664
Syntax .. 665
Parameter definition .. 666
Location in the JCL .. 678
Examples of the //*MAIN statement .. 678
//*NET statement .. 679
Syntax .. 679
Parameter definition .. 680
Location in the JCL .. 683
Examples of the //*NET statement .. 683
//*NETACCT statement .. 683
Syntax .. 683
Parameter definition .. 683
Defaults .. 684
Location in the JCL .. 684
Example of the //*NETACCT statement .. 684
//*OPERATOR statement .. 684
Syntax .. 684
Location in the JCL .. 684
Example of the //*OPERATOR statement .. 685
//**PAUSE statement .. 685
Syntax .. 685
Location in the JCL .. 685
Example of the //**PAUSE statement .. 685
//*PROCESS statement .. 685
Syntax .. 686
Parameter definition .. 686
Location in the JCL .. 687
Examples of the //*PROCESS statement .. 687
//*ROUTE XEQ statement .. 688
Syntax .. 688
Parameter definition .. 688
Location in the JCL .. 689
JOB Statement after //*ROUTE XEQ .. 689
Example of the //*ROUTE XEQ statement .. 689
/*SIGNOFF statement .. 690
Syntax .. 690
Location in the JCL .. 690
Example of the /*SIGNOFF statement .. 690
/*SIGNON statement .. 690
Syntax .. 691
Parameter definition .. 691
Location in the JCL .. 692
Example of the /*SIGNON statement .. 692
JES2 processing of JES3 control statements .. 692

**Appendix. Accessibility .. 693**
Accessibility features .. 693
Consult assistive technologies .. 693
Keyboard navigation of the user interface .. 693
Dotted decimal syntax diagrams .. 693

**Notices .. 697**
Policy for unsupported hardware .. 698
Minimum supported hardware .. 699
Trademarks .. 699

**Index .. 701**

# Figures

1. Operators on IF/THEN/ELSE/ENDIF Statement Construct . . . . . .. 381
2. Example //*ROUTE XEQ statement . . .. 690

© Copyright IBM Corp. 1988, 2015

# Tables


1. MVS Job Control Language (JCL) Statements 1
2. JES2 Job Entry Control Language (JECL) Statements .. 2
3. JES3 Job Entry Control Language (JECL) Statements .. 3
4. Tasks for Entering Jobs .. 6
5. Tasks for Processing Jobs .. 8
6. Tasks for Requesting Data Set Resources .. 8
7. Tasks for Requesting Sysout Data Set Resources .. 10
8. JCL Statement Fields .. 14
9. Notation used to show syntax .. 19
10. Character Sets .. 21
11. Special Characters Used in Syntax .. 21
12. Special Characters that Do Not Require Enclosing Apostrophes .. 22
13. Summary of Rules 2 through 6 for Symbols in Nested Procedures .. 52
14. Identification of Statements in Job Log .. 57
15. Positional parameters .. 85
16. Keyword parameters .. 86
17. Summary of Disposition Processing .. 170
18. Special Character Sets for the 1403, 3203 Model 5, and 3211 Printers .. 291
19. Positional parameters .. 334
20. Keyword parameters .. 335
21. Bypassing or Execution of Current Step Based on COND Parameter .. 347
22. Effect of EVEN and ONLY Subparameters on Step Execution .. 348
23. SYMLIST keyword parameter on the EXPORT statement .. 376
24. JOB statement keyword parameters .. 405
25. Continuation or Termination of the Job Based on the COND Parameter .. 421
26. JOBCLASS attribute for DSENQSHR 422
27. Keyword parameters .. 470
28. Job- and Step-Level OUTPUT JCL Statements in the JCL .. 479
29. Special Character Sets for the 1403, 3203 Model 5, and 3211 Printers .. 556
30. DSPs for JES3 //*PROCESS Statements 686


© Copyright IBM Corp. 1988, 2015

# About this document

This document describes the job control tasks needed to enter jobs into the z/OS® operating system, control the system's processing of jobs, and request the resources needed to run jobs. The document also contains a chapter that describes "started tasks" and how to set them up. To perform job control or started tasks, programmers code "job control statements." This document describes how to code these statements, which include:

- Job control language (JCL) statements
- Job entry control language (JECL) statements, which encompass:
  - Job entry subsystem 2 (JES2) control statements
  - Job entry subsystem 3 (JES3) control statements

This document is designed as a reference document, to be used while coding the statements. It contains some introductory material. Full explanations of the job control tasks are presented in a companion document, *z/OS MVS JCL User's Guide*, SA23-1386.

## Who should use this document

This document is needed by system and application programmers who enter programs into the operating system. Those using this document should understand the concepts of job management and data management.

## Where to find more information

Where necessary, this document references information in other documents, using shortened versions of the document title. For complete titles and order numbers of the documents for all products that are part of z/OS, see *z/OS V2R2 Information Roadmap*.

You might also need the following information:

| Short Title Used in This Document | Title | Order Number |
|---|---|---|
| *SNA Sync Point Services Architecture* | *Systems Network Architecture Sync Point Services Architecture Reference* | SC31-8134 |

## Related information

To have complete JCL information, you need the following document:

- *z/OS MVS JCL User's Guide*, SA23-1386

Where necessary, this document references information in other documents, using shortened versions of the document title. For complete titles and order numbers of the documents for all products that are part of z/OS, see *z/OS V2R2 Information Roadmap*. The following tables list the short titles, titles, and order numbers for documents related to non-OS/390 products that this document references.

© Copyright IBM Corp. 1988, 2015

## Programs

| Short Title Used in This Document | Title | Order Number |
|---|---|---|
| *PSF for z/OS: Customization* | *PSF for z/OS: Customization* | S550-0427 |
| *PSF for z/OS: User's Guide* | *PSF for z/OS: User's Guide* | S550-0435 |

## Hardware

| Short Title Used in This document | Title | Order Number |
|---|---|---|
| *2821 Component Description* | *IBM 2821 Control Unit Component Description* | GA24-3312 |
| *3540 Programmer's Reference* | *OS/VS2 IBM 3540 Programmer's Reference* | GC24-5111 |
| *Forms Design Reference Guide for the 3800* | *Forms Design Reference Guide for the IBM 3800 Printing Subsystem* | GA26-1633 |

# How to send your comments to IBM

We appreciate your input on this publication. Feel free to comment on the clarity, accuracy, and completeness of the information or provide any other feedback that you have.

Use one of the following methods to send your comments:

1. Send an email to mhvrcfs@us.ibm.com.
2. Send an email from the "Contact us" web page for z/OS (http://www.ibm.com/systems/z/os/zos/webqs.html).

Include the following information:

- Your name and address.
- Your email address.
- Your telephone or fax number.
- The publication title and order number:
  z/OS V2R2 MVS JCL Reference
  SA23-1385-03
- The topic and page number that is related to your comment.
- The text of your comment.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute the comments in any way appropriate without incurring any obligation to you.

IBM or any other organizations use the personal information that you supply to contact you only about the issues that you submit.

## If you have a technical problem

Do not use the feedback methods that are listed for sending comments. Instead, take one of the following actions:

- Contact your IBM service representative.
- Call IBM technical support.
- Visit the IBM Support Portal at z/OS Support Portal (http://www-947.ibm.com/systems/support/z/zos/).

© Copyright IBM Corp. 1988, 2015

# Summary of changes for z/OS Version 2 Release 2 (V2R2) as updated December, 2015

The following changes are made for z/OS V2R2 as updated December, 2015.

### Changed

- "REGIONX parameter" on page 366 is added for the REGIONX parameter.
- The Keyword parameters table in "Parameter field" on page 334 and "Relationship to the REGIONX parameter" on page 366 of the EXEC statement are updated to add the REGIONX parameter.
- The JOB statement keyword parameters table in "Parameter field" on page 404and "Relationship to the REGIONX parameter" on page 448of the JOB statement are updated to add the REGIONX parameter.

# Summary of changes for z/OS Version 2 Release 2 (V2R2)

The following changes are made for z/OS Version 2 Release 2 (V2R2).

### New

- JES3 control statement tracking. For details, see the topic about "JES3 control statement tracking" on page 644.
- New statement, SCHEDULE, is added. See Chapter 26, "SCHEDULE statement," on page 571 for more information.
- See Chapter 29, "JES2 Execution Control Statements," on page 591 for new JCL statements for JES2. The following new JCL statements were added for JES2:
  - AFTER - See "AFTER statement" on page 605 for more information.
  - BEFORE - See "BEFORE statement" on page 602 for more information.
  - CONCURRENT - See "CONCURRENT statement" on page 607 for more information.
  - ENDGROUP - See "ENDGROUP statement" on page 609 for more information.
  - ENDSET - See "ENDSET statement" on page 601 for more information.
  - GJOB - See "GJOB statement" on page 598 for more information.
  - JOBGROUP - See "JOBGROUP statement" on page 594 for more information.
  - JOBSET - See "JOBSET statement" on page 599 for more information.
  - SJOB - See "SJOB statement" on page 600 for more information.
- JES2 processing of JES3 control statements. See "JES2 processing of JES3 control statements" on page 692 for more information.

### Changed

- Modifications are made for SYSUID symbol substitution for the in-stream data sets. For details, see the topic about "Using symbols in JES in-stream data" on page 54.
- Modified the HOLD parameter of the DD statement in "Keyword parameters" table. For details, see the topic about "Parameter field" on page 85.
- A note was added to the FREEVOL parameter. See "FREEVOL parameter" on page 204.

© Copyright IBM Corp. 1988, 2015

- JCL parsing for the MAXGENS keyword is changed to accept a value of 0. For details, see the topic about “Subparameter definition” on page 228.
- Modified *logging-DDname* in syntax of SYMBOLS parameter. For details, see “Syntax” on page 280.
- Job scheduler processing is modified to resolve exported symbols in jobs where a SET statement follows an execute of a procedure. See “SYMLIST parameter” on page 376 for more information.
- Modifications are made to AFF=ddname. See “Subparameter definition” on page 293 for more information.
- Modified Overrides of the OUTPUT JCL statement. For details, see the topic about “Overrides” on page 479.
- Added JES3 information to the PROCLIB=*ddname* parameter of the JCLLIB statement. For details, see the topic about “Parameter field” on page 398.
- Modified the DDNAME parameter of the OUTPUT JCL statement in "Keyword parameters" table. For details, see the topic about “Parameter field” on page 470.
- Modified Relationship to sysout DD statement of the OUTPUT JCL statement. For details, see the topic about “Relationship to sysout DD statement” on page 480.
- Modified DDNAME= keyword syntax of the OUTPUT JCL statement. For details, see the topic about “Syntax” on page 499.
- Modified subparameter definition of the DDNAME parameter of the OUTPUT JCL statement. For details, see the topic about “Subparameter definition” on page 499.
- Modified Defaults and Overrides of the OUTDISP parameter OUTPUT JCL statement. For details, see the topic about “Defaults” on page 536 and “Overrides” on page 536.

### Deleted

No content was removed from this information.

## Summary of changes for z/OS Version 2 Release 1 (V2R1) as updated December 2013

Changes made to z/OS V2R1 as updated December 2013

### New

The MAXGENS parameter on the DD statement specifies the maximum number of generations for members in a Version 2 PDSE. For details, refer to “MAXGENS parameter” on page 228.

### Changed

The REFDD parameter on the DD statement now copies the maximum number of generations for members in a Version 2 PDSE (MAXGENS). For details, refer to “REFDD parameter” on page 256.

## z/OS Version 2 Release 1 summary of changes

See the following publications for all enhancements to z/OS Version 2 Release 1 (V2R1):

- *z/OS V2R2 Migration*
- *z/OS Planning for Installation*
- *z/OS V2R2 Summary of Message and Interface Changes*
- *z/OS V2R2 Introduction and Release Guide*

# Chapter 1. Job control statements

This chapter lists, in Table 1, all but one of the statements in the MVS™ Job Control Language (JCL), and in Table 2 on page 2, all of the Job Entry Control Language (JECL) statements for the JES2 and JES3 subsystems, together with the purpose of each statement. Later chapters describe each statement in detail. (The PRINTDEV JCL statement, for use by the person starting the Print Services Facility™, is documented in the manual *PSF for z/OS: Customization*.)

## JCL statements

*Table 1. MVS Job Control Language (JCL) Statements*

| Statement | Name | Purpose |
|---|---|---|
| // command | JCL command | Enters an MVS system operator command through the input stream. The command statement is used primarily by the operator. Use the COMMAND statement instead of the JCL command statement. |
| // COMMAND | command | Specifies an MVS or JES command that the system issues when the JCL is converted. Use the COMMAND statement instead of the JCL command statement. |
| //* comment | comment | Contains comments. The comment statement is used primarily to document a program and its resource requirements. |
| // CNTL | control | Marks the beginning of one or more program control statements. |
| // DD | data definition | Identifies and describes a data set. |
| /* | delimiter | Indicates the end of data placed in the input stream.<br><br>**Note:** A user can designate any two characters to be the delimiter. |
| // ENDCNTL | end control | Marks the end of one or more program control statements. |
| // EXEC | execute | Marks the beginning of a job step; assigns a name to the step; identifies the program or the cataloged or in-stream procedure to be executed in this step. |
| // EXPORT | export | Makes specific JCL symbols available to the job step program. |
| // IF/THEN/ELSE/ ENDIF | IF/THEN/ELSE/ ENDIF statement construct | Specifies conditional execution of job steps within a job. |
| // INCLUDE | include | Identifies a member of a partitioned data set (PDS) or partitioned data set extended (PDSE) that contains JCL statements to include in the job stream. |

© Copyright IBM Corp. 1988, 2015

*Table 1. MVS Job Control Language (JCL) Statements (continued)*

| Statement | Name | Purpose |
|---|---|---|
| // JCLLIB | JCL library | Identifies the libraries that the system will search for:<br>• INCLUDE groups<br>• Procedures named in EXEC statements. |
| // JOB | job | Marks the beginning of a job; assigns a name to the job. |
| // | null | Marks the end of a job. |
| // OUTPUT | output JCL | Specifies the processing options that the job entry subsystem is to use for printing a sysout data set. |
| // PEND | procedure end | Marks the end of an in-stream or cataloged procedure. |
| // PROC | procedure | Marks the beginning of an in-stream procedure and may mark the beginning of a cataloged procedure; assigns default values to parameters defined in the procedure. |
| // SCHEDULE | schedule | Specifies scheduling attributes for a job such as the job group it is associated with and whether the job should be held for a time before execution. |
| // SET | set | Defines and assigns initial values to symbolic parameters used when processing JCL statements. Changes or nullifies the values assigned to symbolic parameters. |
| // XMIT | transmit | Transmits input stream records from one node to another. |

# JECL statements

*Table 2. JES2 Job Entry Control Language (JECL) Statements*

| JES2 JECL Control Statement | Purpose |
|---|---|
| /*$command | Enters JES2 operator commands through the input stream. |
| /*JOBPARM | Specifies certain job-related parameters at input time. |
| /*MESSAGE | Sends messages to the operator via the operator console. |
| /*NETACCT | Specifies an account number for a network job. |
| /*NOTIFY | Specifies the destination of notification messages. |
| /*OUTPUT | Specifies processing options for sysout data set(s). |
| /*PRIORITY | Assigns a job queue selection priority. |
| /*ROUTE | Specifies the output destination or the execution node for the job. |

*Table 2. JES2 Job Entry Control Language (JECL) Statements (continued)*

| JES2 JECL Control Statement | Purpose |
|---|---|
| /*SETUP | Requests mounting of volumes needed for the job. |
| /*SIGNOFF | Ends a remote job stream processing session. |
| /*SIGNON | Begins a remote job stream processing session. |
| /*XEQ | Specifies the execution node for a job. |
| /*XMIT | Indicates a job or data stream to be transmitted to another JES2 node or eligible non-JES2 node. |

*Table 3. JES3 Job Entry Control Language (JECL) Statements*

| JES3 JECL Control Statement | Purpose |
|---|---|
| //**command | Enters JES3 operator commands, except *DUMP and *RETURN, through the input stream. |
| //*DATASET | Begins an input data set in the input stream. |
| //*ENDDATASET | Ends the input data set that began with a //*dataset statement. |
| //*ENDPROCESS | Ends a series of //*PROCESS statements. |
| //*FORMAT | Specifies the processing options for a sysout or JES3-managed print or punch data set. |
| //*MAIN | Defines selected processing parameters for a job. |
| //*NET | Identifies relationships between predecessor and successor jobs in a dependent job control net. |
| //*NETACCT | Specifies an account number for a network job. |
| //*OPERATOR | Sends messages to the operator. |
| //*PAUSE | Halts the input reader. |
| //*PROCESS | Identifies a nonstandard job. |
| //*ROUTE | Specifies the execution node for the job. |
| /*SIGNOFF | Ends a remote job stream processing session. |
| /*SIGNON | Begins a remote job stream processing session. |

# Chapter 2. Job Control Tasks

For your program to execute on the computer and perform the work you designed it to do, your program must be processed by your operating system.

Your operating system consists of an MVS base control program (BCP) with a job entry subsystem (JES2 or JES3) and DFSMSdfp installed with it.

For the operating system to process a program, programmers must perform certain job control tasks. These tasks are performed through the job control statements, which consist of:

- JCL statements
- JES2 control statements
- JES3 control statements

## Entering Jobs

***Job Steps:*** You enter a program into the operating system as a **job step**. A job step consists of the job control statements that request and control execution of a program and request the resources needed to run the program. A job step is identified by an EXEC statement. The job step can also contain data needed by the program. The operating system distinguishes job control statements from data by the contents of the records.

***Jobs:*** A **job** is a collection of related job steps. A job is identified by a JOB statement.

***Input Streams:*** Jobs placed in a series and entered through one input device form an **input stream**. The operating system reads an input stream into the computer from an input/output (I/O) device or an internal reader. The input device can be a card reader, a magnetic tape device, a terminal, or a direct access device. An internal reader is a buffer that is read from a program into the system as though it were an input stream.

***Cataloged and In-Stream Procedures:*** You often use the same set of job control statements repeatedly with little or no change, for example, to compile, assemble, link-edit, and execute a program. To save time and prevent errors, you can prepare sets of job control statements and place, or catalog, them in a partitioned data set (PDS) or partitioned data set extended (PDSE) known as a procedure library. The data set attributes of a procedure library should match SYS1.PROCLIB (record length of 80 and record format of FB). Such a set of job control statements in the system procedure library, SYS1.PROCLIB (or an installation-defined procedure library), is called a **cataloged procedure**.

To test a procedure before placing it in the catalog, place it in an input stream and execute it; a procedure in an input stream is called an **in-stream procedure**. The maximum number of in-stream procedures you can code in any job is 15.

***Steps in a Job:*** A job can be simple or complex; it can consist of one step or of many steps that call many in-stream and cataloged procedures. A job can consist of up to 255 job steps, including all steps in any procedures that the job calls. Specification of a greater number of steps produces a JCL error.

© Copyright IBM Corp. 1988, 2015

## Processing Jobs

The operating system performs many job control tasks automatically. You can influence the way your job is processed by the JCL and JES2 or JES3 parameters you code. For example, the job entry subsystem selects jobs for execution, but you can speed up or delay selection of your job by the parameters you code.

## Requesting Resources

***Data Set Resources:*** To execute a program, you must request the data sets needed to supply data to the program and to receive output records from the program.

***Sysout Data Set Resources:*** A sysout data set is a system-handled output data set. This data set is placed temporarily on direct access storage. Later, at the convenience of the system, the system prints it, punches it, or sends it to a specified location. Because sysout data sets are processed by the system, the programmer can specify many parameters to control that processing.

## Task Charts

The following charts list the job control tasks, which are described in the *z/OS MVS JCL User's Guide*, in four groups:

- Entering jobs in Table 4
- Processing jobs in Table 5 on page 8
- Requesting data set resources in Table 6 on page 8
- Requesting sysout data set resources in Table 7 on page 10

For each task, the charts list the parameters and statements that can be used to perform it. In many cases, the same task can be performed using different parameters on different statements. Where a parameter can appear on both a JOB and EXEC statement, it applies to the entire job when coded on the JOB statement but only to a step when coded on an EXEC statement.

The system is designed to enable users to perform many types of job control in many ways. To allow this flexibility, only two job entry tasks are required:

- **Identification:** The job must be identified in the *jobname field* of a JOB statement.
- **Execution:** The program or procedure to be executed must be named in a PGM or PROC parameter on an EXEC statement.

Therefore, the following statements are the minimum needed to perform a job control task:

```
//jobname  JOB
//         EXEC  {PGM=program-name   }
                 {PROC=procedure-name}
                 {procedure-name}
```

*Table 4. Tasks for Entering Jobs*

| TASKS FOR ENTERING JOBS | STATEMENTS AND PARAMETERS | | | | |
|---|---|---|---|---|---|
| | JCL Statements | | | JES2 Statements | JES3 Statements |
| | JOB | EXEC | Other JCL | | |
| **Identification** | | | | | |
| of job | jobname field | | null statement (JES3 only) | | |
| of step | | stepname field | | | |
| of procedure | | | PROC PEND | | |

*Table 4. Tasks for Entering Jobs (continued)*

| TASKS FOR ENTERING JOBS | STATEMENTS AND PARAMETERS | | | | |
|---|---|---|---|---|---|
| | JCL Statements | | | JES2 Statements | JES3 Statements |
| | JOB | EXEC | Other JCL | | |
| of INCLUDE group | | | INCLUDE | | |
| of account | accounting information or pano in JOB JES2 accounting information | ACCT | | /*NETACCT | //*NETACCT |
| of programmer | programmer's name and room in JOB JES2 accounting information USER | | | ROOM on /*JOBPARM | PNAME, BLDG, DEPT, ROOM, and USERID on //*NETACCT |
| **Execution** | | | | | |
| of program | | PGM | | | |
| of procedure | | PROC | | | |
| when restarting and with checkpointing | RESTART RD | RD | SYSCHK DD | RESTART on /*JOBPARM | FAILURE and JOURNAL on //*MAIN |
| deadline or periodic | | | | | DEADLINE on //*MAIN |
| when dependent on other jobs | | | | | //*NET |
| at remote node | | | XMIT JCL | /*ROUTE XEQ /*XEQ /*XMIT | //*ROUTE XEQ |
| **Job Input Control** | | | | | |
| by holding job entrance | TYPRUN CLASS | | | | HOLD, UPDATE, or CLASS on //*MAIN //*NET |
| by holding local input reader | | | | | //*PAUSE |
| by copying input stream | TYPRUN CLASS | | | | |
| from remote work station | | | | /*SIGNON /*SIGNOFF | /*SIGNON /*SIGNOFF |
| **Communication** | | | | | |
| from JCL to system | | | COMMAND Command | /*$command | //**command |
| from JCL to operator | | | | /*MESSAGE | //*OPERATOR |
| from JCL to programmer | Comment field unless no parameter field | Comment field | //*comment, also comment field on all statements but null | | Comment field on //*ENDPROCESS and //*PAUSE |
| from JCL to program | | PARM | | | |
| from system to operator | WARNING on BYTES, CARDS, LINES, and PAGES | | | | FETCH on //*MAIN WARNING on BYTES, CARDS, LINES, and PAGES on //*MAIN |
| from system to userid -of job completion -of print completion | NOTIFY | | NOTIFY on OUTPUT JCL statement | /*NOTIFY | ACMAIN on //*MAIN with JOB NOTIFY |
| from TSO/E userid to system | | | | | USER on //*MAIN |
| from functional subsystem to programmer | | | PIMSG on OUTPUT JCL | | |
| through job log | MSGCLASS MSGLEVEL log in JOB JES2 accounting information | | JESDS on OUTPUT JCL | NOLOG on /*JOBPARM | |
| **Protection** | | | | | |
| through RACF | GROUP PASSWORD SECLABEL USER | | | | |

*Table 4. Tasks for Entering Jobs (continued)*

| TASKS FOR ENTERING JOBS | STATEMENTS AND PARAMETERS | | | | |
|---|---|---|---|---|---|
| | JCL Statements | | | JES2 Statements | JES3 Statements |
| | JOB | EXEC | Other JCL | | |
| **Resource Control** | | | | | |
| of program library | | | JOBLIB DD, STEPLIB DD, DD defining PDS or PDSE member | | |
| of procedure library | | | JCLLIB | PROCLIB on /*JOBPARM | PROC and UPDATE on //*MAIN |
| of INCLUDE group | | | JCLLIB | PROCLIB on /*JOBPARM | PROC and UPDATE on //*MAIN |
| of address space | REGION ADDRSPC | REGION ADDRSPC | | | LREGION on //*MAIN |
| of processor | | | | SYSAFF on /*JOBPARM | SYSTEM on //*MAIN |
| of spool partition | | | | | SPART and TRKGRPS on //*MAIN |

*Table 5. Tasks for Processing Jobs*

| TASKS FOR PROCESSING JOBS | STATEMENTS AND PARAMETERS FOR TASK | | | | |
|---|---|---|---|---|---|
| | JCL Statements | | | JES2 Statements | JES3 Statements |
| | JOB | EXEC | Other JCL | | |
| **Processing Control** | | | | | |
| by conditional execution | COND<br><br>CANCEL on BYTES, CARDS, LINES, and PAGES | COND | IF/THEN/ELSE/ ENDIF statement construct | CANCEL on BYTES, CARDS, LINES, and PAGES on /*JOBPARM | CANCEL on BYTES, CARDS, LINES, and PAGES on //*MAIN |
| by timing execution | TIME or time in JOB JES2 accounting information | TIME | | TIME on /*JOBPARM | |
| for testing:<br>1. by altering usual processing<br>2. by dumping after error | TYPRUN CLASS DUMP on BYTES, CARDS, LINES, and PAGES | PGM=IEFBR14<br><br>PGM=JCLTEST<br>PGM=JSTTEST (JES3 only) | SYSMDUMP DD<br>SYSUDUMP DD<br>SYSABEND DD<br><br>To format dump on 3800 Printing Subsystem, FCB=STD3 and CHARS=DUMP on dump DD | | //*PROCESS<br>//*ENDPROCESS<br>DUMP in BYTES, CARDS, LINES, and PAGES on //*MAIN |
| **Performance Control** | | | | | |
| by job class assignment | CLASS | | | | CLASS on //*MAIN |
| by selection priority | PRTY | | | /*PRIORITY | |
| by performance group assignment | PERFORM | PERFORM | | | |
| by I/O-to-processing ratio | | | | | IORATE on //*MAIN |

*Table 6. Tasks for Requesting Data Set Resources*

| TASKS FOR REQUESTING DATA SET RESOURCES | STATEMENTS AND PARAMETERS FOR TASK | | | | |
|---|---|---|---|---|---|
| | JCL Statements | | | JES2 Statements | JES3 Statements |
| | DD | OUTPUT JCL | Other JCL | | |
| **Identification** | | | | | |
| of data set | DSNAME | | | | UPDATE on //*MAIN |

*Table 6. Tasks for Requesting Data Set Resources (continued)*

| TASKS FOR REQUESTING DATA SET RESOURCES | STATEMENTS AND PARAMETERS FOR TASK | | | | |
|---|---|---|---|---|---|
| | JCL Statements | | | JES2 Statements | JES3 Statements |
| | DD | OUTPUT JCL | Other JCL | | |
| of in-stream data set | * or DATA SYSIN DD DLM | | /* or xx delimiter | | //*DATASET<br>//*ENDDATASET |
| of data set on 3540 Diskette Input/Output Unit | DSID | | | | |
| through label | label-type on LABEL | | | | |
| by location on tape | data-set- sequence-number on LABEL | | | | |
| from or to terminal | TERM | | | | |
| **Description** | | | | | |
| of status | DISP | | | | |
| of data attributes - by modeling | DCB<br>AMP<br>DATACLAS<br>KEYLEN<br>DSNTYPE<br>KEYOFF<br>LRECL<br>RECFM<br>RECORG<br><br>LIKE<br>REFDD | | | | |
| of data for ISO/ANSI Version 4 tapes | CCSID | | | | |
| of migration and backup | MGMTCLAS | | | | |
| **Protection** | | | | | |
| through RACF | PROTECT<br>SECMODEL | | | | |
| for ISO/ANSI/FIPS Version 3 tapes and ISO/ANSI Version 4 tapes | ACCODE | | | | |
| by passwords | PASSWORD and NOPWREAD on LABEL | | | | |
| of access to BSAM and BDAM data sets | IN and OUT on LABEL | | | | |
| **Allocation** | | | | | |
| of device | UNIT STORCLAS | | CLASS on JOB (JES3 only) | | SETUP and CLASS on //*MAIN |
| of tape or direct access volume | VOLUME STORCLAS | | | | EXPDTCHK and RINGCHK on //*MAIN |
| of direct access space | SPACE AVGREC<br>DATACLAS | | | | |
| of virtual I/O | UNIT<br>DSNAME=<br>temporary<br>data set | | | | |
| with deferred volume mounting | DEFER on UNIT | | | | |
| with volume pre-mounting | | | | /*SETUP | |
| dynamic | | | DYNAMNBR on EXEC | | |
| **Processing Control** | | | | | |
| by suppressing processing | DUMMY NULLFILE on DSNAME | | | | |

*Table 6. Tasks for Requesting Data Set Resources (continued)*

| TASKS FOR REQUESTING DATA SET RESOURCES | STATEMENTS AND PARAMETERS FOR TASK | | | | |
|---|---|---|---|---|---|
| | JCL Statements | | | JES2 Statements | JES3 Statements |
| | DD | OUTPUT JCL | Other JCL | | |
| by postponing specification | DDNAME | | | | |
| with checkpointing | CHKPT SYSCKEOV DD SYSCHK DD | | RESTART on JOB RD on EXEC | | |
| by subsystem | SUBSYS CNTL | | CNTL ENDCNTL | | |
| **End Processing** | | | | | |
| unallocation | FREE | | | | |
| disposition of data set | DISP<br><br>RETPD<br>EXPDT | | | OUTDISP on /*OUTPUT | |
| release of unused direct access space | RLSE on SPACE | | | | |
| disposition of volume | RETAIN and PRIVATE on VOLUME | | | | |

*Table 7. Tasks for Requesting Sysout Data Set Resources*

| TASKS FOR REQUESTING DATA SET RESOURCES | STATEMENTS AND PARAMETERS FOR TASK | | | | |
|---|---|---|---|---|---|
| | JCL Statements | | | JES2 Statements | JES3 Statements |
| | DD | OUTPUT JCL | Other JCL | | |
| **Identification** | | | | | |
| as a sysout data set | SYSOUT | | | | |
| name (last qualifier) | DSNAME | | | | |
| of output class | class on SYSOUT | CLASS | MSGCLASS on JOB with SYSOUT=* or CLASS=* and SYSOUT=(,) | | |
| of data set on 3540 Diskette Input/Output Unit | DSID | | | | |
| **Description** | | | | | |
| of data attributes | DCB | | | | |
| **Protection** | | | | | |
| of printed output | | DPAGELBL SYSAREA | | | |
| **Performance Control** | | | | | |
| by queue selection | | PRTY | | | |
| **Processing Control** | | | | | |
| with additional parameters | OUTPUT code-name on SYSOUT | DEFAULT | | | |
| by segmenting | SEGMENT | | | | |
| with other data sets | class on SYSOUT | THRESHLD (JES3 only)<br>GROUPID (JES2 only) | | | |
| by external writer | writer-name on SYSOUT | WRITER | | | |
| by mode | | PRMODE | | | |
| by holding | HOLD class on SYSOUT | CLASS OUTDISP | | | |
| by suppressing output | DUMMY class on SYSOUT | OUTDISP=PURGE on OUTPUT | | | |
| with checkpointing | | CKPTLINE CKPTPAGE CKPTSEC | | CKPLNS and CKPPGS on /*OUTPUT | |

*Table 7. Tasks for Requesting Sysout Data Set Resources (continued)*

| TASKS FOR REQUESTING DATA SET RESOURCES | STATEMENTS AND PARAMETERS FOR TASK | | | | |
|---|---|---|---|---|---|
| | JCL Statements | | | JES2 Statements | JES3 Statements |
| | DD | OUTPUT JCL | Other JCL | | |
| by Print Services Facility (PSF) | | AFPPARMS<br>AFPSTATS<br>COLORMAP<br>COMSETUP<br>DUPLEX<br>FORMDEF<br>FORMLEN<br>INTRAY<br>OFFSETXB<br>OFFSETXF<br>OFFSETYB<br>OFFSETYF<br>OVERLAYB<br>OVERLAYF<br>PAGEDEF<br>PRTERROR<br>RESFMT<br>USERLIB<br>USERPATH | | | |
| by Infoprint Server | | FSSDATA<br>MAILBCC<br>MAILCC<br>MAILFILE<br>MAILFROM<br>MAILTO<br>PORTNO<br>PRTATTRSPRTOPTNS<br>PRTQUEUE<br>REPLYTO<br>RETAINF<br>RETAINS<br>RETRYL<br>RETRYT | | | |
| **End Processing** | | | | | |
| unallocation | FREE<br>SPIN | | | | |
| **Destination Control** | | | | | |
| to local or remote device or to another node | DEST class on SYSOUT | DEST<br>COMPACT | | /*ROUTE PRINT<br>/*ROUTE PUNCH | ORG on //*MAIN |
| to another processor | | | | | ACMAIN on //*MAIN |
| to internal reader | INTRDR as writer-name on SYSOUT | | /*EOF<br>/*DEL<br>/*PURGE<br>/*SCAN | | |
| to terminal | TERM | | | | |
| to assist in sysout distribution | | ADDRESS<br>BUILDING<br>DEPT<br>NAME<br>ROOM<br>TITLE | | ROOM on /*OUTPUT | |
| **Output Formatting** | | | | | |
| to any printer | COPIES FCB form-name on SYSOUT UCS | COPIES<br>FCB<br>FORMS<br>LINECT<br>(JES2 only)<br>UCS<br>CONTROL | forms, copies, and linect on JOB JES2 accounting information | COPIES, FORMS, and LINECT on /*JOBPARM<br>COPIES, FCB, and FORMS on /*OUTPUT | COPIES and FORMS on //*FORMAT PR |

*Table 7. Tasks for Requesting Sysout Data Set Resources (continued)*

| TASKS FOR REQUESTING DATA SET RESOURCES | STATEMENTS AND PARAMETERS FOR TASK | | | | |
|---|---|---|---|---|---|
| | JCL Statements | | | JES2 Statements | JES3 Statements |
| | DD | OUTPUT JCL | Other JCL | | |
| to an AFP printer in addition to most of printer parameters | BURST<br>CHARS<br>FLASH<br>MODIFY<br>DCB=<br>OPTCD=J | BURST CHARS FLASH<br>MODIFY TRC | | BURST on /*JOBPARM<br><br>CHARS, FLASH, and BURST on /*OUTPUT | CHARS and FLASH on //*FORMAT PR |
| to 3211 Printer with indexing feature | | INDEX (JES2 LINDEX only) | | | |
| to punch | COPIES FCB form-name on SYSOUT<br>DCB=FUNC=I | COPIES<br>FCB<br>FORMS | | | |
| of dumps on 3800 Printing Subsystem | CHARS=DUMP<br>FCB=STD3 | CHARS=DUMP<br>FCB=STD3 | | | |
| **Output Limiting** | | | | | |
| | OUTLIM | | lines and cards on JOB JES2 accounting information<br>BYTES, CARDS, LINES, and PAGES on JOB | BYTES, CARDS, LINES, and PAGES on /*JOBPARM | BYTES, CARDS, LINES, and PAGES on //*MAIN |
| **USERDATA Specifications** | | | | | |
| Installation specifications | | USERDATA | | | |

# Chapter 3. Format of statements

This topic describes the fields in JCL, JES2, and JES3 statements. It ends with the conventions for continuing statements.

## JCL statement fields

A JCL statement consists of one or more 80-byte records. Each record is comparable to an 80-column punched-card image. Each JCL statement is logically divided into the following five fields. All five fields do not appear on every statement; see Table 8 on page 14 for the fields that can appear on each statement.

**Identifier field**
The identifier field indicates to the system that a statement is a JCL statement rather than data. The identifier field consists of the following:

- Columns 1 and 2 of all JCL statements, except the delimiter statement, contain //
- Columns 1 and 2 of the delimiter statement contain either /* or two other characters designated in a DLM parameter to be the delimiter
- Columns 1, 2, and 3 of a JCL comment statement contain //*

**Name field**
The name field identifies a particular statement so that other statements and the system can refer to it. For JCL statements, code the name as follows:

- The name must begin in column 3.
- The name is 1 through 8 alphanumeric or national ($, #, @) characters. See Table 10 on page 21 for the character sets.
- The first character must be an alphabetic or national ($, #, @).
- The name must be followed by at least one blank.

**Operation field**
The operation field specifies the type of statement, or, for the command statement, the command. Code the operation field as follows:

- The operation field consists of the characters in the syntax box for the statement.
- The operation follows the name field.
- The operation must be preceded and followed by at least one blank.

**Parameter, or operand field**
The parameter field, also sometimes referred to as the operand field, contains parameters separated by commas. Code the parameter field as follows:

- The parameter field follows the operation field.
- The parameter field must be preceded by at least one blank.

See “Parameter field” on page 15 for details on coding the parameter field.

**Comments field**
The comments field contains any information you deem helpful when you code the control statement. Code the comments field as follows:

- The comments field follows the parameter field.
- The comments field must be preceded by at least one blank.

© Copyright IBM Corp. 1988, 2015

You can code comments after the parameter field even though you continue the parameter field on a subsequent statement; see "Continuing JCL statements" on page 16.

For most statements, if you do not code any parameters, do not code any comments.

*Table 8. JCL Statement Fields*

| Statement | Fields |
|---|---|
| JCL Command | // command [parameter] [comments] |
| COMMAND | //[name] COMMAND 'command command-operand' [comments] |
| Comment | //* comments |
| CNTL | //label CNTL [* comments] |
| DD | //[ddname] DD [parameter [comments]]<br>//[ddname] DD |
| Delimiter | /* [comments]<br>xx [comments] |
| ENDCNTL | //[label] ENDCNTL [comments] |
| EXEC | //[stepname] EXEC parameter [comments] |
| EXPORT | //[label] EXPORT [comments] |
| IF/THEN/ELSE/ENDIF | //name IF [relational expression] THEN [comments]<br>//name ELSE [comments]<br>//name ENDIF [comments] |
| INCLUDE | //[name] INCLUDE parameter [comments] |
| JCLLIB | //[name] JCLLIB parameter [comments] |
| JOB | //jobname JOB [parameter [comments]]<br>//jobname JOB |
| Null | // |
| OUTPUT JCL | //name OUTPUT parameter [comments] |
| PEND | //[name] PEND [comments] |
| PROC (cataloged) | //[name] PROC [parameter [comments]]<br>//[name] PROC |
| PROC (in-stream) | //name PROC [parameter [comments]]<br>//name PROC |
| SCHEDULE | //[name] SCHEDULE parameter [comments] |
| SET | //[name] SET parameter [comments] |
| XMIT | //[name] XMIT parameter[,parameter] [comments] |

***Location of fields on statements:*** Code the identifier field beginning in column 1 and the name field immediately after the identifier, with no intervening blanks. Code the operation, parameter, and comments fields in free form. Free form means that the fields need not begin in a particular column. Between fields leave at least one blank; the blank serves as the delimiter between fields.

Do not code fields, except on the comment statement, past column 71. Columns 73-80 are ignored by z/OS and are typically used for sequence numbers. If the total length of the fields would exceed 71 columns, continue the fields onto one or more following statements. Continuing fields is described under "Continuing JCL statements" on page 16. The comment statement can be coded through column 80.

***Use keywords only for parameters or subparameters:*** Do not use parameter or subparameter keywords from any JCL, JES2, or JES3 statements as symbolic parameters, names, or labels.

## Parameter field

The parameter field consists of two types of parameters: **positional parameters** and **keyword parameters**. All positional parameters must precede all keyword parameters. Keyword parameters follow the positional parameters.

***Commas:*** Use commas to separate positional parameters, keyword parameters, and subparameters in the parameter field.

***Positional Parameters:*** A positional parameter consists of:

- Characters that appear in uppercase in the syntax and must be coded as shown
- Variable information, or
- A combination.

For example, DATA on a DD statement, programmer's-name on a JOB statement, and PGM=program-name on an EXEC statement.

Code positional parameters first in the parameter field in the order shown in the syntax. If you omit a positional parameter and code a following positional parameter, code a comma to indicate the omitted parameter. Do not code the replacing comma if:

- The omitted positional parameter is the last positional parameter.
- All following positional parameters are also omitted.
- Only keyword parameters follow.
- All positional parameters are omitted.

***Keyword parameters:*** A keyword consists of characters that appear in uppercase in the syntax and must be coded as shown followed by an equals sign followed by either characters that must be coded as shown or variable information. For example, RD=R and MSGCLASS=class-name on the JOB statement.

Code any of the keyword parameters for a statement in any order in the parameter field after the positional parameters. Because of this positional independence, never code a comma to indicate the absence of a keyword parameter.

***Multiple subparameters:*** A positional parameter or the variable information in a keyword parameter sometimes consists of more than one item, called a subparameter list. A subparameter list can consist of both positional and keyword subparameters. These subparameters follow the same rules as positional and keyword parameters.

When a parameter contains more than one subparameter, separate the subparameters by commas and enclose the subparameter list in parentheses or, if indicated in the syntax, by apostrophes. If the list is a single keyword subparameter or a single positional subparameter with no omitted preceding subparameters, omit the parentheses or apostrophes.

***Null positional subparameters:*** You are allowed to specify null (that is, omitted) positional subparameters except where the Syntax section of a particular parameter states otherwise. (For example, null positional subparameters are **not** allowed on a COND parameter of an EXEC statement or on an AMP parameter of a DD statement.) You specify a null positional subparameter by following the coding rules listed previously for an omitted positional parameter.

## JES2 control statement fields

The rules for coding JES2 control statements are the same as the rules for JCL statements, with the following additions:

- Columns 1 and 2 always contain the characters /*. Columns 73-80 are ignored by z/OS and are typically used for sequence numbers.
- Do not code comments on any JES2 statements. Where comments are needed, code a JCL comment statement.
- If you code the same parameter on the same statement more than once, JES2 uses the value in the last parameter.

When coding a JES2 control statement more than once, be aware of the following JES2 actions:

- If the same parameter appears on more than one statement, JES2 uses the value coded on the last statement.
- If the statements contain different parameters, JES2 uses all parameters combined.

## JES3 control statement fields

The rules for coding JES3 control statements are the same as the rules for JCL statements, with the following additions:

- Columns 1, 2, and 3 generally contain the characters //* (slash-slash-asterisk). Some JES3 control statements may contain, and certain other JES3 control statements *must* contain only a single slash-asterisk (/*) in columns 1 and 2.
- Columns 3 and 4 must not be blank.
- To code a comment on a JES3 control statement, code a blank after the control statement, and end the comment before column 72. Columns 73-80 are ignored by z/OS and are typically used for sequence numbers.

## Continuing statements

You can continue some JCL statements, as described in the following sections.

### Continuing JCL statements

When the total length of the fields on a control statement exceeds 71 columns, continue the fields onto one or more records. Each line is a record.

The following are JCL statements that you **cannot** continue. While you cannot continue these statements, you can code as many separate statements as you need.

- JCL Command statement
- Comment statement
- Delimiter statement
- Null statement

For all other JCL statements, you can continue the parameter field or the comments field on the JCL statement. If you continue both the parameter field and the comments field on the same record, the system ignores the indication to continue the comment. How you continue a parameter field depends on whether the parameter is enclosed in apostrophes.

### Continuing the parameter field

1. Interrupt the field after a complete parameter or subparameter, including the comma that follows it, at or before column 71.
2. Code // in columns 1 and 2 of the following statement.
3. Code a blank character in column 3 of the following statement. If column 3 contains anything but a blank or an asterisk, the system assumes the following statement is a new statement. The system issues an error message indicating that no continuation is found and fails the job.
4. Continue the interrupted parameter or field beginning in any column from 4 through 16.

### Continuing parameter fields enclosed in apostrophes

To continue a parameter that is enclosed in apostrophes:

1. Extend the parameter to column 71. Do not code an apostrophe in column 71 of a JCL statement that is continued. The system interprets the apostrophe in column 71 as the final character in the statement and ignores the continuation.
2. Code // in columns 1 and 2 of the following statement.
3. Continue the parameter in column 16 of the following statement even if this splits the parameter. Trailing blanks or commas within the apostrophes do not indicate a continued statement; the system treats them as part of the parameter.

The following example shows the specification of a long file name in the PARM field:

```
//STEP1 EXEC                          PGM=IEFBR14,PARM=(PARM1,'/DIR1/DIR2
//             /DIR3/DIR4/DIR5/DIR6/DIR7/DIR8/DIR9/DIR10/DIR11/DIR12/DI
//             R13/FILENM')
```

### Continuing the comments field

Include comments by following an interrupted parameter field with at least one blank. To continue a comment:

1. Interrupt the comment at a convenient place before column 72, up to and including column 71.
2. Code a nonblank character in column 72.
3. Code // in columns 1 and 2 of the following statement.
4. Code a blank character in column 3 of the following statement.
5. Continue the comments field beginning in any column after column 3.

You can use JCL comment statements as an alternative way to imbed comments in the JCL stream.

## Examples of continued statements

***Example 1***

```
//DD1  DD   DSNAME=SWITCH.LEVEL18.GROUP12,UNIT=3390,
//     VOLUME=335023,SPACE=(TRK,(80,15)),DISP=(,PASS)
```

Example 1 shows continuation of a DD statement. The DD statement is continued from the first card image to the second card image. The comma on the first line indicates that continuation is expected.

***Example 2***

```
//DS1 DD  DSNAME=INDS,DISP=OLD,CHKPT=EOV,   MY INPUT DATA SET
//     UNIT=SYSSQ,VOLUME=SER=(TAPE01,TAPE02,TAPE03)
```

Example 2 shows continuation of a DD statement with a comment on the first statement. The DD statement is continued from the first card image to the second card image. The comma followed by a blank on the first line indicates that continuation is expected. The comment on the first card image is not continued to the next card image.

*Example 3*

```
//STP4  EXEC PROC=BILLING,COND.PAID=((20,LT),EVEN),
//     COND.LATE=(60,GT,FIND),
//     COND.BILL=((20,GE),(30,LT,CHGE))  THIS STATEMENT CALLS          X
//     THE BILLING PROCEDURE AND SPECIFIES RETURN CODE TESTS           X
//     FOR THREE PROCEDURE STEPS.
```

Example 3 shows continuation of an EXEC statement with a comment at the end that also is continued on multiple lines. The EXEC statement is continued from the first card image to the second card image. The comma followed by a blank on the first line indicates that continuation is expected. Continuation from the second card to the third uses the same logic. The comment on the third card image is continued to the next card image via an X in column 72. The comment on fourth card image is continued to the fifth card image via an X in column 72.

*Example 4*

```
//S1       EXEC PGM=IEFBR14,PARM='THIS IS A LONG PARAMETER WITHIN APOST
//             ROPHES, CONTINUED IN COLUMN 16 OF THE NEXT RECORD'
```

Example 4 shows continuation of a parameter field when a parameter is enclosed in apostrophes. The parameter field is continued from column 71 of the first card image to column 16 of the second.

## Continuing JES2 control statements

The only JES2 control statement that you can continue is the /*OUTPUT statement. For all other JES2 control statements, code the statement as many times as needed.

## Continuing JES3 control statements

Continue JES3 statements, except the command statement or //*NETACCT statement, by:

1. Coding a comma as the last character of the first statement.
2. Coding //* in columns 1 through 3 of the continuation statement.
3. Resuming the code in column 4 of the continuation statement.

On the JES3 //*NET statement, each parameter must appear entirely on one statement; a subparameter cannot be continued after a comma, except for the RELEASE parameter. To continue the RELEASE parameter, end the statement with the comma following a jobname and continue the next statement with the next jobname. The left parenthesis appears at the beginning of the jobname list and the right parenthesis appears at the end of the list. For example:

```
//*NET NETID=EXP1,RELEASE=(JOB35,JOB27Z,MYJOB,
//*WRITJB,JOBABC)
```

If the parameters on a //*NETACCT statement cannot fit on one statement, code more than one //*NETACCT statement.

# Chapter 4. Syntax of parameters

Syntax rules define how to code the fields and parameters on job control statements. The syntax indicates:

- What the system requires.
- What is optional for the specific purpose or process you are requesting.
- How the parameters are to appear.

The syntax rules apply to all job control statements: JCL statements, JES2 control statements, and JES3 control statements.

**Note:** You must follow the syntax rules in coding job control statements to achieve specific results. If you do not follow the rules, you may get error messages or unpredictable results. IBM does not support the use of statements or parameters to achieve results other than those stated in this publication.

## Notation used to show syntax

The syntax for job control statements and their parameters appear in the description for each statement. The notation used in this publication for the syntax is shown in Table 9.

*Table 9. Notation used to show syntax*

| Notation | Meaning | Examples |
|---|---|---|
| Uppercase letters, words, and characters | Code uppercase letters, words, and the following characters exactly as they appear in the syntax.<br>**&** ampersand<br>* asterisk<br>**,** comma<br>**=** equal sign<br>**( )** parentheses<br>**.** period<br>**/** slash | |
| Lowercase letters, words, and symbols | Lowercase letters, words, and symbols in the syntax represent variables. Substitute specific information for them. | Syntax: on JOB statement<br>CLASS=jobclass<br>Coded:<br>CLASS=A |
| \| (vertical bar) | A vertical bar indicates an exclusive OR. Never code \| on a control statement. It is used between choices within braces or brackets; it indicates that you code only one of the items within the braces or brackets. | Syntax: on DD DCB parameter BFALN={F\|D}<br>Coded:<br>BFALN=F<br>or<br>BFALN=D |

© Copyright IBM Corp. 1988, 2015

*Table 9. Notation used to show syntax (continued)*

| Notation | Meaning | Examples |
|---|---|---|
| { } (braces) | Braces surround *required*, related items and indicate that you must code one of the enclosed items. Never code { or } on a control statement. | Syntax: on DD SPACE parameter<br>`{TRK    }`<br>`{CYL    }`<br>`{blklgth}`<br>`{reclgth}`<br><br>Coded:<br>TRK or<br>CYL or<br>960 |
| [ ] (brackets) | Brackets surround an *optional* item or items and indicate that you can code one or none of the enclosed items. Never code [ or ] on a control statement. | Syntax: on DD UNIT parameter<br><br>[,DEFER]<br><br>Coded:<br>,DEFER<br>or<br>omitted<br><br>Syntax: on DD LABEL parameter<br>`[,RETPD=nnnn        ]`<br>`[,EXPDT=  {yyddd   }]`<br>`[         {yyyy/ddd}]`<br><br>Coded:<br>,RETPD=nnnn<br>or<br>,EXPDT=yyddd<br>or<br>,EXPDT=yyyy/ddd<br>or<br>omitted |
| { , } or [ , ] | One of the items in braces or brackets can be a comma. Code the comma when you do not code any of the other items in the braces or brackets but you are coding a following part of the parameter. | Syntax: on DD UCS parameter UCS=(character-set-code[,FOLD\|,] [,VERIFY])<br>• Coded:UCS=(character-set-code)<br>• UCS=(character-set-code,FOLD)<br>• UCS=(character-set-code,FOLD, VERIFY)<br>• UCS=(character-set-code,,VERIFY)<br><br>Note that the comma is not coded if both FOLD and VERIFY are omitted, but must appear if FOLD is omitted and VERIFY follows. |
| ‗ (underline) | An underline indicates the default that the system uses when you do not code a subparameter. | Syntax: on JOB or EXEC statement<br>ADDRSPC={VIRT\|REAL}<br><br>Coded:<br>ADDRSPC omitted means<br>ADDRSPC=VIRT |
| ... (ellipsis) | An ellipsis follows an item that you can code more than once. Never code ... on a control statement. | Syntax: on DD statement<br>COND=((code,operator)[,(code,operator)]....)<br><br>Coded:<br>Can repeat ,(code,operator)<br>Thus:<br>COND=((12,GE),(8,EQ),(4,EQ)) |

*Table 9. Notation used to show syntax (continued)*

| Notation | Meaning | Examples |
|---|---|---|
| .. (two consecutive periods) | Two consecutive periods indicate that a parameter consists of a symbolic parameter followed by a period and then by other code, so that only part of the parameter is variable. | Coded: &DEPT..NYC<br><br>Meaning:<br>If &DEPT is D27:<br>D27.NYC is the value |

## Character sets

To code job control statements, use characters from the character sets in Table 10. Table 11 lists the special characters that have syntactical functions in job control statements.

*Table 10. Character Sets*

| Character set | Contents | Details |
|---|---|---|
| Alphanumeric | Alphabetic<br>Numeric | Capital A through Z<br>0 through 9 |
| National<br>(See note) | "At" sign<br>Dollar sign<br>Pound sign | @ (Characters that can be<br>$ represented by hexadecimal<br># values X'7C', X'5B', and<br>X'7B') |
| Special | Comma<br>Period<br>Slash<br>Apostrophe<br>Left parenthesis<br>Right parenthesis<br>Asterisk<br>Ampersand<br>Plus sign<br>Hyphen<br>Equal sign<br>Blank | ,<br>.<br>/<br>'<br>(<br>)<br>*<br>&<br>+<br>-<br>= |
| EBCDIC text | EBCDIC printable character set | Characters that can be represented by hexadecimal X'40' through X'FE' |
| **Note:** The system recognizes the following hexadecimal representations of the U.S. National characters; @ as X'7C'; $ as X'5B'; and # as X'7B'. In countries other than the U.S., the U.S. National characters represented on terminal keyboards might generate a different hexadecimal representation and cause an error. For example, in some countries the $ character may generate a X'4A'. | | |

*Table 11. Special Characters Used in Syntax*

| Character | Syntactical function |
|---|---|
| , | To separate parameters and subparameters |
| = | To separate a keyword from its value, for example, BURST=YES |
| ( b ) | To enclose subparameter list or the member name of a PDS or PDSE |
| & | To identify a symbolic parameter, for example, &LIB |
| && | To identify a temporary data set name, for example, &&TEMPDS, and, to identify an in-stream or sysout data set name, for example, &&PAYOUT |

*Table 11. Special Characters Used in Syntax (continued)*

| Character | Syntactical function |
|---|---|
| . | To separate parts of a qualified data set name, for example, A.B.C., or parts of certain parameters or subparameters, for example, nodename.userid |
| * | To refer to an earlier statement, for example, OUTPUT=*.name, or, in certain statements, to indicate special functions: //label CNTL * //ddname DD * RESTART=* on the JOB statement |
| ' | To enclose specified parameter values which contain special characters |
| (blank) | To delimit fields |

***Special characters in parameters:*** The syntax or parameter description indicates if the variable that you code can contain special characters or not. Parameters and subparameters that can contain special characters not used for syntactical functions usually must be enclosed in apostrophes, for example, ACCT='123+456'. Code each apostrophe that is part of the parameter or subparameter as two consecutive apostrophes, for example, code O'NEIL as 'O''NEIL'.

Table 12 lists the parameters that can contain certain special characters without requiring enclosing apostrophes.

Ampersands are used in JCL to indicate the beginning of a symbolic parameter (see "Using system symbols and JCL symbols" on page 38). If a parameter contains an ampersand and you do not want the system to interpret the ampersand as a symbolic parameter, code the ampersand as two consecutive ampersands. For example, code

```
//S1   EXEC  PGM=IEFBR14,ACCT='&&ABC'
//DD1  DD    DSN=&&TEST,UNIT=SYSDA,SPACE=(TRK,(1,1))
```

The system treats double ampersands as a single character. IBM recommends that you use apostrophes to enclose parameters that contain ampersands (other than a DSNAME parameter representing a temporary data set) to further reduce the possibility of error.

*Table 12. Special Characters that Do Not Require Enclosing Apostrophes*

| Statement and parameter or subparameter | Special characters not needing enclosing apostrophes | Examples |
|---|---|---|
| JOB accounting information | Hyphens (-) | //JOBA JOB D58-D04 |
| JOB programmer's-name | Hyphens (-), leading periods, or embedded periods. Note that a trailing period requires enclosing apostrophes. | //JOBB JOB ,S-M-TU<br>//JOBC JOB ,.ABC<br>//JOBD JOB ,P.F.M<br>//JOBE JOB ,'A.B.C.' |
| EXEC ACCT | Hyphens (-) or plus zero (+0, an overpunch) | //S1 EXEC PGM=A,ACCT=D58-LOC<br>//S2 EXEC PGM=B,ACCT=D04+0 |
| DD DSNAME | Hyphens (-) | DSNAME=A-B-C |
| | Periods to indicate a qualified data set name | DSNAME=A.B.C |
| | Double ampersands to identify a temporary data set name, and to identify an in-stream or sysout data set name | DSNAME=&&TEMPDS<br><br>DSNAME=&&PAYOUT |

*Table 12. Special Characters that Do Not Require Enclosing Apostrophes (continued)*

| Statement and parameter or subparameter | Special characters not needing enclosing apostrophes | Examples |
|---|---|---|
| | Parentheses to enclose the member name of a partitioned data set (PDS) or partitioned data set extended (PDSE) or the generation number of a generation data set | DSNAME=PDS1(MEMA)<br>DSNAME=ISDS(PRIME)<br>DSNAME=GDS(+1) |
| | Plus (+) or minus (-) sign to identify a generation of a generation data group | DSNAME=GDS(-2) |
| DD VOLUME=SER | Hyphens (-) | VOLUME=SER=PUB-RD |
| DD UNIT device-type | Hyphens (-) | UNIT=SYSDA |

## Syntax notes

JCL positional parameters and keywords can have at most two levels of subparameters. Therefore, when parentheses are used to delimit a list of subparameters, a maximum of two levels of parenthesis nesting is permitted. This restriction applies even if the parentheses are empty.

## Backward references

Many parameters in job control statements can use a backward reference to fill in information. A backward reference is a reference to an earlier statement in the job or in a cataloged or in-stream procedure called by a job step. A backward reference is in the form:

- ***.name** or ***.ddname** where name or ddname is the name field of the referenced statement.
- ***.stepname.name** or ***.stepname.ddname** where the referenced statement, name or ddname, is in an earlier step, stepname, in the same job.
- ***.stepname.procstepname.name** or ***.stepname.procstepname.ddname** where this job step or an earlier job step, stepname, calls a procedure; the procedure contains procedure step, procstepname, which contains the referenced statement, name or ddname.

If stepname is specified without a procstepname, it identifies an EXEC statement that contains a PGM parameter, not one that invokes a procedure. Similarly, if stepname.procstepname is coded, procstepname identifies an EXEC statement containing the PGM parameter in the procedure invoked by stepname.

The backward reference lets you copy previously coded information or refer to an earlier statement. The following parameters can make backward references:

- DD CNTL refers to earlier CNTL statement
- DD DCB refers to earlier DD statement to copy its DCB parameter
- DD DSNAME refers to earlier DD statement to copy its DSNAME parameter, whether or not the data set is a partitioned data set, and whether or not the data set is a temporary data set
- DD OUTPUT refers to earlier OUTPUT JCL statement
- DD REFDD refers to earlier DD statement to copy its data set attributes

- DD VOLUME=REF refers to earlier DD statement to use the same volume(s). The LABEL label type subparameter is also copied from the referenced DD statement.
- EXEC PGM refers to an earlier DD statement that defines the program to be executed as a member of a partitioned data set

The following statements cannot be referenced:

- DD * statement in DCB, DSNAME, or VOLUME parameter
- DD DATA statement in DCB, DSNAME, or VOLUME parameter
- DD DUMMY statement in VOLUME or UNIT parameter. The referring DD statement acquires a dummy status.
- DD DYNAM statement
- DD statement containing FREE=CLOSE in VOLUME or UNIT parameters
- Nested procedure statements
- Sysout DD statement
- DD statement that is the target of a DDNAME= reference.
- A DD statement containing a PATH parameter

## Examples of backward references

***Example 1:***

```
//JOB1    JOB   ...
//STEPA   EXEC  ...
//DD1     DD    DSNAME=REPORT
              .
              .
//DD4     DD    DSNAME=*.DD1
```

The referring and referenced DD statements are in the same step.

***Example 2:***

```
//JOB2    JOB   ...
//STEP1   EXEC  ...
//DDA     DD    DSNAME=D58.POK.PUBS01
              .
              .
//STEP2   EXEC  ...
//DDB     DD    DSNAME=*.STEP1.DDA
```

The referring and referenced DD statements are in different steps in the same job.

***Example 3:*** Cataloged procedure PROC1 contains:

```
//PS1     EXEC  ...
              .
              .
//PSTEP1  EXEC  ...
//DS1     DD    DSNAME=DATA1
//PSTEP2  EXEC  ...
//DS2     DD    DSNAME=DATA2
              .
```

The job contains:

```
//JOB5    JOB   ...
//CALLER  EXEC  PROC=PROC1
              .
```

```
//REF1    DD    DSNAME=*.CALLER.PSTEP2.DS2
//NEXT    EXEC  ...
//REF2    DD    DSNAME=*.CALLER.PSTEP1.DS1
            .
```

DD statement REF1 in the calling step refers to DD statement DS2 in procedure step PSTEP2. DD statement REF2 in a step after the calling step refers to DD statement DS1 in procedure step PSTEP1. Note that the entire procedure is processed when the calling EXEC statement is processed; therefore, all DD statements in the procedure are earlier than all DD statements in the calling step.

# Chapter 5. Procedures and symbols

This section describes how to use procedures, including nested procedures. It also explains how to use JCL symbols and system symbols.

## Cataloged and in-stream procedures

For jobs that you run frequently or types of jobs that use the same job control, prepare sets of job control statements, called procedures.

### In-stream procedures

When you place a procedure in the job input stream, it is called an **in-stream procedure**.

An in-stream procedure must begin with a PROC statement, end with a PEND statement, and include only the following other JCL statements: CNTL, comment, DD, ENDCNTL, EXEC, IF/THEN/ELSE/ENDIF, INCLUDE, OUTPUT JCL, and SET. You must observe the following restrictions regarding in-stream procedures:

- Do not place any JCL statements (other than the ones listed here) or any JES2 or JES3 control statements in the procedure.
- Do not define one in-stream procedure within another, that is, do not use *nested* in-stream procedures. Refer to "Nested procedures" on page 35 for information on methods for nesting procedures.
- Do not use an in-stream procedure if the procedure will be run as a started job under the MASTER subsystem, that is, includes a JOB statement and is started via a **START** command such as **S membername,SUB=MSTR**.

### Cataloged procedures

A procedure that you catalog in a library is called a **cataloged procedure**.

A cataloged procedure may consist of these JCL statements: CNTL, command, DD, ENDCNTL, EXEC, IF/THEN/ELSE/ENDIF, INCLUDE, OUTPUT JCL, and SET. Optionally, a cataloged procedure can begin with a PROC statement and end with a PEND statement. If coded, PROC must be the first statement in the procedure.

#### Cataloging a procedure

The library containing cataloged procedures is a partitioned data set (PDS) or a partitioned data set extended (PDSE). The system procedure library is SYS1.PROCLIB. The installation can have many more procedure libraries with different names. You can also have procedures in a private library. The name of a cataloged procedure is its member name or alias in the library.

When a cataloged procedure is called, the calling step receives a copy of the procedure; therefore, a cataloged procedure can be used simultaneously by more than one job.

If you are modifying a cataloged procedure, do not run any jobs that use the procedure during modification.

In a JES3 system, you can specify UPDATE on the JES3 //*MAIN statement to update a procedure library.

© Copyright IBM Corp. 1988, 2015

## Using a procedure

To execute a procedure, call it on an EXEC statement in an in-stream job. Specify the name of the procedure in the PROC parameter of the EXEC statement. The step uses the JCL statements in the procedure as if the JCL statements appeared in the input stream immediately following the EXEC statement. If necessary, you can modify the procedure for the current execution of the job step.

When you call a procedure, the system retrieves it using the following search order:

1. **From the input stream**

   If the called procedure is an in-stream procedure, the system retrieves it from the job input stream. You must place the in-stream procedure before the EXEC statement that calls it.

2. **From a private library**

   If the called procedure is cataloged in a private library, the system retrieves it from the private library that you specify on the JCLLIB statement that appears earlier in the job stream.

3. **From the system library (in a non-APPC scheduling environment)**

   If the called procedure is cataloged in a system library, the subsystem retrieves it as follows:

   - In JES2, from the library name on the PROCLIB= parameter on a JES2 /*JOBPARM statement. See "/*JOBPARM statement" on page 613 for more information.
   - In JES3, from the library name on the PROC= parameter of the JES3 //*MAIN statement. See "//*MAIN statement" on page 664 for more information.
   - In MSTR, the data set specified by the IEFPDSI DD statement in the currently active master JCL is searched for procedures. The default master JCL specifies SYS1.PROCLIB.

## Testing a procedure

Before putting a procedure into a system procedure library, you should test it. There are two ways to test a procedure:

- Place a PROC statement before the procedure and a PEND statement after it and place it in a job input stream. For the test, call this in-stream procedure with an EXEC statement that appears later in the same job.
- Put a procedure to be tested in a private library, identify the library on a JCLLIB statement, and call the procedure with an EXEC statement.

After testing a procedure, the type of environment in which you are running the job determines where you can catalog it.

- **In an APPC scheduling environment:** Catalog the procedure in a private library, and define the library with a JCLLIB statement.
- **In a non-APPC scheduling environment:** Catalog the procedure in the system procedure library SYS1.PROCLIB, an installation-defined procedure library, or a private library. Cataloging the procedure in the system procedure library allows anyone to use the procedure by calling it with an EXEC statement.

Cataloged and in-stream procedures are not checked for correct syntax until an EXEC statement that calls the procedure is checked for syntax. Therefore, a procedure can be tested only if an EXEC statement calls it.

# Modifying procedures

There are two ways you can modify a procedure:

- Using system symbols and JCL symbols
- Using overrides.

Using system symbols and JCL symbols, you can design your procedures to be easily modified. If the procedure does not contain required system symbols and JCL symbols, you can override the statement.

For its current execution, you can override an in-stream or cataloged procedure by:

- Overriding, nullifying, or adding EXEC statement parameters
- Overriding, nullifying, or adding parameters to DD or OUTPUT JCL statements
- Adding DD or OUTPUT JCL statements

Overriding a parameter modifies only that parameter; the system uses all other parameters on the original statement. For example, if you override the data set name on a DD statement that includes a UNIT and VOL=SER parameter, the system will still use the UNIT and VOL=SER parameters.

Invalid parameters in a procedure cannot be corrected through overrides. Before processing overrides, the system scans the original procedure statements for errors and issues error messages.

## Modifying EXEC statement parameters

All keyword parameters on the calling EXEC statement affect the execution of the procedure, as follows:

### All procedure statements

If a keyword parameter is to override the parameter or be added to every EXEC statement in the procedure, code the parameter in the usual way. For example, the ACCT parameter applies to all steps:

```
//STEP1  EXEC  PROC=RPT,ACCT=5670
```

**Note:** A PARM parameter without a procstepname qualifier applies only to the first procedure step. A TIME parameter without a procstepname qualifier applies to the entire procedure and nullifies any TIME parameters on procedure step EXEC statements.

If the keyword parameter is to nullify the parameter on every EXEC statement in the procedure, code it without a value following the equal sign. For example, the ACCT parameter is nullified in all steps:

```
//STEP2  EXEC  PROC=RPT,ACCT=
```

### A single procedure statement

If the keyword parameter is to override the parameter or be added to only one EXEC statement in the procedure, code **.procstepname** immediately following the keyword. The procstepname is the name field of the procedure EXEC statement containing the keyword parameter to be overridden. For example, the ACCT parameter applies to only step PSTEPWED:

```
//STEP1  EXEC  PROC=RPT,ACCT.PSTEPWED=5670
```

If the keyword parameter is to nullify the parameter on only one EXEC statement in the procedure, code it with the procstepname. For example:

```
//STEP2  EXEC  PROC=RPT,ACCT.PSTEPTUE=
```

The override, nullification, or addition applies only to the current execution of the job step; the procedure itself is not changed.

### Rules for modifying EXEC parameters

The following rules apply for modifying EXEC parameters:

- You cannot modify a PGM parameter.
- The calling EXEC statement can contain changes for more than one parameter and for the same parameter in more than one step in a called procedure. (If you code multiple overrides for any parameter in the same step, only the last specification will be effective.)
- Modifying parameters should appear in the following order:
  - Parameters without a procstepname qualifier.
  - All parameters modifying the first step, then the second step, then the third step, and so forth.
- You do not need to code the parameters for each step in the same order as they appear on the procedure EXEC statement.
- You must code an entire overriding parameter even if you are changing only part of it.
- You can use a different parameter to override the parameter in a procedure statement, if the two parameters are mutually exclusive. The override operation automatically nullifies the procedure parameter. This is an exception to the general rule that the only way to override a parameter is to specify it explicitly. For example, if the EXEC statement in a procedure contains a PARM= specification and you override it with a PARMDD= specification, the value specified by PARM= is nullified and the value specified by PARMDD= is substituted.

## Modifying OUTPUT JCL and DD statements

OUTPUT JCL and DD statements that follow the calling EXEC statement

- Override, nullify, or add parameters to OUTPUT JCL and DD statements in the procedure, or
- Are added to the procedure.

These changes affect only the current execution of the job step; the procedure itself is not changed. When an OUTPUT JCL statement is modified, the sysout data set is processed according to the parameters as modified by the overriding statement.

In a procedure, to ensure that OUTPUT JCL and DD statements are overridden correctly by modifying statements, place the OUTPUT JCL statements before the DD statements in each step of the procedure.

### Location in the JCL

Place modifying OUTPUT JCL and DD statements in the following order, after the EXEC statement that calls the procedure:

- For each procedure step in the invoked procedure:
  1. Overriding statements can appear in any order when they explicitly specify the step that is being overridden. Added statements can appear in any order when they specify the step explicitly.
  2. Overriding and added statements that do not explicitly specify the step are applied to the step named in the previous overriding or added OUTPUT JCL

or DD statement. If no previous override statement named a step, then they are applied to the first step in the procedure.

- For all procedure steps in the invoked procedure, place the modifying statements for each procedure step in the same order in which the procedure steps are specified.

## Coding an overriding OUTPUT JCL or DD statement

To override, nullify, or add parameters to a procedure OUTPUT JCL or DD statement, code in the name field of the overriding OUTPUT JCL or DD statement the name of the procedure step containing the overridden statement, followed by a period, followed by the name of the procedure OUTPUT JCL statement or the ddname of the procedure DD statement.

```
//pstepname.name   OUTPUT parameters
  //pstepname.ddname DD     parameters
```

## Rules for modifying OUTPUT JCL or DD parameters

The override operation merges the parameters from an overriding statement with those in the overridden statement. Follow these rules in coding overriding statements:

- You can code more than one change on an overriding statement.
- You can code modifying parameters in any order on an overriding statement.
- Code an entire overriding parameter even when changing only part of that parameter.
- If you code a parameter on an overriding statement that is not on the procedure statement, the override operation adds it to the procedure statement.
- Nullify a parameter by not coding a value after the equal sign. Omitting the value causes the system to treat the keyword as if it had been removed from the procedure statement. This is the only way to nullify keywords that do not permit a null parameter value.
- If you add a parameter that is mutually exclusive with a parameter on a procedure statement, the override operation automatically nullifies the procedure parameter. This is the only exception to the rule that the only way to override a parameter is to specify it explicitly.

```
Example:  If a DD statement within a procedure reads:

             //ddname DD DSN=FRED,DISP=SHARE,UNIT=TAPE,VOL=SER=111111

          and you wish to modify that DD statement to read in
          data set GEORGE, which is cataloged to a DASD volume,
          it is NOT sufficient to specify:

             //stepname.ddname DD DSN=GEORGE

          Instead you must specify:

             //stepname.ddname DD DSN=GEORGE,UNIT=,VOL=

          This nullifies the UNIT and VOLUME information, allowing
          the system to retrieve that information from the catalog.
          (An overriding DD statement without those parameters would
          cause the system to find data set GEORGE on tape volume
          serial 111111.)
```

## Additional rules for modifying DD parameters

The following additional rules apply for modifying DD parameters:

- To nullify all parameters but the DCB parameter, code DUMMY on the overriding DD statement.

- Special rules apply when overriding a DCB parameter:
  - Code only the keyword subparameters to be changed; the other DCB subparameters remain unchanged.
  - If a positional subparameter is needed, code it, regardless of whether it appears in the overridden DCB parameter. If a positional subparameter is not needed or is to be nullified, omit it from the overriding DCB parameter.
  - To nullify the entire DCB parameter, nullify each subparameter appearing in the overridden DCB parameter.
- To nullify a DUMMY parameter on the procedure statement, code one of the following on the overriding statement:
  - A DSNAME parameter with a name other than NULLFILE
  - A SYSOUT parameter
  - A * or DATA parameter
  - A SUBSYS parameter.

## Adding an OUTPUT JCL or DD statement

To add OUTPUT JCL or DD statements to a procedure step, code in the name field of the added OUTPUT JCL or DD statement the name of the procedure step, followed by a period, followed by a name or ddname. The name must not appear on any procedure statement.

```
//pstepname.name   OUTPUT parameters
  //pstepname.ddname DD     parameters
```

If you omit the procedure step name, the statement is added to the step named in the previous OUTPUT JCL or DD statement that named a step. If no previous statements named steps, the statement is added to the first step in the procedure.

Added OUTPUT JCL and DD statements can contain symbols. If the statement is being added to the last procedure step, any symbols it contains must appear elsewhere in the procedure.

## Supplying in-stream data for a procedure

To supply a procedure step with data from the input stream, code a DD * or DD DATA statement in the calling step after the last overriding and added DD statement. The name field of this statement must contain the name of the procedure step, followed by a period, followed by a ddname. The ddname can be of your choosing or predefined in the procedure. If it is predefined, it appears in a DDNAME parameter on a procedure DD statement. For example:

```
//PROCSTP1.ANYNAME DD *
//PROCSTP2.PREDEFN DD DATA
```

## Embedding in-stream data in a procedure

In JES2 and JES3, you can embed in-stream data directly within in-stream or cataloged procedure code. For example, in JES2:

```
//HELLO     PROC
//STEPA     EXEC  PGM=IEBGENER
//SYSIN     DD    DUMMY
//SYSPRINT  DD    SYSOUT=A
//SYSUT2    DD    SYSOUT=A
//SYSUT1    DD    DATA
HELLO WORLD
/*
//          PEND
```

### Rules for modifying DD statements in concatenated data sets

- To override the first DD statement in a concatenation, code only one overriding DD statement.
- To override any following DD statements in the concatenation, code an overriding DD statement for each concatenated DD statement.
- The overriding DD statements must be in the same order as the concatenated DD statements.
- Code a ddname on only the first overriding DD statement. Leave the ddname field blank on the following statements.
- To leave a concatenated statement unchanged, code its corresponding overriding DD statement with a blank operand (or parameter) field.

## Examples of procedures

***Example 1***

```
In the input stream:

//JOBA        JOB  ACCT23,'G. HILL'
//STEPA       EXEC PROC=REP
//PSTEP1.INDS DD   *
               .
               (data)
               .
/*

In SYS1.PROCLIB member REP:

//       PROC
//PSTEP1 EXEC PGM=WRIT22
//OUTDS  DD   SYSOUT=A
```

In this example, the EXEC statement STEPA calls the cataloged procedure named REP and supplies in-stream data. The procedure executes a program named WRIT22. The output from the program will appear in the sysout class A data set.

***Example 2***

```
In the input stream:

//JOB1         JOB     ,'H.H. MORRILL'
//ADD1         OUTPUT  COPIES=2
//STEPA        EXEC    PROC=P
//PS1.OUTA     OUTPUT  CONTROL=DOUBLE,COPIES=5
//PS1.DSB      DD      OUTPUT=*.ADD1
//PS1.DSE      DD      *
                .
                (data)
                .
/*
//PS2.OUTB     OUTPUT  DEFAULT=YES,DEST=STL

In SYS1.PROCLIB member P:

//PS1         EXEC    PGM=R15
//OUTA        OUTPUT  CONTROL=PROGRAM
//DSA         DD      SYSOUT=C,OUTPUT=*.OUTA
//DSB         DD      SYSOUT=D,OUTPUT=*.OUTA
//PS2         EXEC    PGM=T48
//DSC         DD      SYSOUT=A
```

In this example, added statements are:

- ADD1, which is an OUTPUT JCL statement added at the job level.
- PS1.DSE, which is an in-stream data set added to procedure step PS1.
- PS2.OUTB, which is a default OUTPUT JCL statement added to procedure step PS2.

Overriding statements are:

- PS1.OUTA, which changes the CONTROL parameter and adds a COPIES parameter to OUTPUT statement OUTA in procedure step PS1.
- PS1.DSB, which changes the OUTPUT parameter on DD statement DSB in procedure step PS1.

***Example 3***

```
//JOBB        JOB  ACCT23,'G. HILL'
//STEPB       EXEC PROC=WRIT35,COND.PSTEP3=(4,GT,PSTEP1),RD=R
//PSTEP1.DD1  DD   VOLUME=SER=,UNIT=SYSDA,DISP=(NEW,CATLG)
//PSTEP1.INDS DD   *
              .
              .
              (data)
              .
/*
//PSTEP2.DD3  DD   DISP=(OLD,KEEP)
//PSTEP3.DD5  DD   DUMMY
//PSTEP3.DD6  DD   DSNAME=A.B.C
//PSTEP3.DD8  DD   EXPDT=


In SYS1.PROCLIB member WRIT35:

//       PROC
//PSTEP1 EXEC PGM=WT1,TIME=(,50)
//DD1    DD   DSNAME=DATA1,DISP=(NEW,DELETE),SPACE=(TRK,(10,2)),
//        UNIT=3390,VOL=SER=1095
//DD2    DD   DSNAME=&&WORK,UNIT=SYSDA,SPACE=(CYL,(10,1)),
//        DISP=(,PASS)
//PSTEP2 EXEC PGM=WT2,TIME=(,30)
//DD3    DD   DSNAME=*.PSTEP1.DD2,DISP=(OLD,DELETE)
//PSTEP3 EXEC PGM=UPDT,TIME=(,45),RD=RNC
//DD4    DD   SYSOUT=*
//DD5    DD   DSNAME=DATA3,UNIT=3390,DISP=OLD,
//        VOLUME=SER=335006
//DD6    DD   DSNAME=QOUT,UNIT=3390
//DD7    DD   SYSOUT=H
//DD8    DD   DSNAME=A.B,DISP=(NEW,CATLG,DELETE),
//            SPACE=(TRK,(1)),EXPDT=92365,UNIT=SYSDA
```

In this example, EXEC statement STEPB calls the cataloged procedure WRIT35. The COND parameter is added to the EXEC statement for PSTEP3. The RD parameter is added to the EXEC statements for PSTEP1 and PSTEP2, and overrides the RD parameter on the EXEC statement for PSTEP3.

In-stream DD statement PSTEP1.DD1 modifies DD statement DD1 in PSTEP1; it nullifies the VOLUME=SER parameter and overrides the UNIT and DISP parameters. Note that the parameters are not in the same order in the overriding and overridden statements.

In-stream DD statement PSTEP1.INDS is added to PSTEP1, supplying in-stream data to be read by program WT1.

In-stream DD statement PSTEP2.DD3 modifies DD statement DD3 in PSTEP2; it overrides the DISP parameter. Note that the entire parameter is coded, even though only the second subparameter is being changed.

In-stream DD statement PSTEP3.DD5 nullifies DD statement DD5 in PSTEP3. However, DD statement DD5 will be checked for correct syntax.

In-stream DD statement PSTEP3.DD6 modifies DD statement DD6 in PSTEP3; it overrides the DSNAME parameter.

In-stream DD statement PSTEP3.DD8 modifies DD statement DD8 in PSTEP3; it nullifies the EXPDT parameter. Note that the EXPDT keyword cannot have a null value. Therefore, you cannot nullify EXPDT by setting it to a substitution text in the procedure DD and then nullifying the symbol on the invoking EXEC statement. EXPDT can only be nullified by not coding a value for it on the overriding DD statement.

Note that procedure DD statements DD2, DD4, and DD7 were not modified.

## Nested procedures

Cataloged and in-stream procedures can invoke other procedures (up to 15 levels of nesting). In a procedure, an EXEC statement can invoke another procedure, which can contain an EXEC statement to invoke another procedure, and so on.

Note that an in-stream procedure cannot be defined within another procedure. The sequence PROC, PROC, PEND, PEND is not valid.

### Nesting procedures

The following example shows how procedures can be nested:

```
Procedure C:
   //C       PROC
   //CS1     EXEC  PGM=GHI
              .
   //        PEND

Procedure B:
   //B       PROC
   //BS1     EXEC  PROC=C
              .
   //BS2     EXEC  PGM=DEF
              .
   //        PEND

Procedure A:
   //A       PROC
   //AS1     EXEC  PROC=B
              .
   //AS2     EXEC  PGM=ABC
              .
   //        PEND

Job Stream:
   //JOB1    JOB
   //STEP1   EXEC  PROC=A
              .
   //STEP2   EXEC  PGM=JKL
              .
              .
```

The following statements are equivalent to the nested procedures shown above and show the levels of nesting (scoping) for the procedures.

```
//JOB1    JOB                  Level 0
//CS1     EXEC  PGM=GHI        Level 3
          .
//BS2     EXEC  PGM=DEF        Level 2
          .
//AS2     EXEC  PGM=ABC        Level 1
          .
//STEP2   EXEC  PGM=JKL        Level 0
           .
           .
```

## Modifying nested procedures

The rules for modifying OUTPUT JCL and DD statements described in "Modifying OUTPUT JCL and DD statements" on page 30 apply to nested procedures.

**In addition,** the following rules apply to modifying statements in nested procedures.

1. Procedure and step names referenced by other statements in the job should be unique within the job.
2. Modifying or additional JCL statements must appear in the job stream following the EXEC statement for the procedure they are to modify and prior to the next EXEC statement.
3. Modifying or additional JCL statements apply to one level of nesting only. You can use statements to modify statements in a procedure only for the level of nesting at which the EXEC statement for that procedure appears.
4. Modifying or additional JCL statements cannot themselves be modified. Do not modify statements that are overrides or additions to a procedure.
5. Modifying or additional JCL statements can only have procstepname.name or procstepname.ddname in their name field. Do not specify backward references to nested procedures, such as procstepname.procstepname.ddname DD parameters.

These rules are illustrated in the examples in this topic.

## Examples of modifying nested procedures

***Example 1:*** The following example shows overrides and additions to DD statements.

```
Procedure C:
   //C       PROC
   //CS1     EXEC  PGM=CCC
   //CS1DD1  DD    DSNAME=A.B.C,DISP=SHR
   //CS1DD2  DD    SYSOUT=A
   //        PEND

Procedure B:
   //B       PROC
   //BS1     EXEC  PROC=C
   //CS1.CS1DD1  DD  DSNAME=X.Y.Z       This statement is a valid
   //*                                  override of procedure C, stepCS1
   //*                                  for DD CS1DD1
   //*
   //CS1.CS1DD3  DD  SYSOUT=A           This statement is a valid
   //*                                  addition to procedure C, step CS1
   //BS2     EXEC  PGM=BBB
   //BS2DD1  DD    DSNAME=E,DISP=SHR
   //        PEND
```

```
Procedure A:
   //A       PROC
   //AS1     EXEC  PROC=B
   //BS2.BS2DD2  DD  DSNAME=G,DISP=SHR  This statement is a valid
   //*                                  addition to procedure B, step BS2
   //AS2     EXEC  PGM=AAA
   //AS2DD1  DD    DSNAME=E,DISP=SHR
   //        PEND

Job Stream:
   //JOB1    JOB
   //STEP1   EXEC  PROC=A
   //AS2.AS2DD2  DD  DSNAME=G,DISP=SHR  This statement is a valid
   //*                                  addition to procedure A, step AS2
   //STEP2   EXEC  PGM=IEFBR14
              .
```

The following statements are equivalent to the nested procedures shown above.

```
//JOB1    JOB
//CS1     EXEC  PGM=CCC
//CS1DD1  DD    DSNAME=X.Y.Z,DISP=SHR
//*
//CS1DD2  DD    SYSOUT=A
//CS1DD3  DD    SYSOUT=A
//*
//BS2     EXEC  PGM=BBB
//BS2DD1  DD    DSNAME=E,DISP=SHR
//BS2DD2  DD    DSNAME=G,DISP=SHR
//*
//AS2     EXEC  PGM=AAA
//AS2DD1  DD    DSNAME=E,DISP=SHR
//AS2DD2  DD    DSNAME=G,DISP=SHR
//STEP2   EXEC  PGM=IEFBR14
```

***Example 2:*** The following example shows nested procedures and **invalid** overrides of DD statement parameters that result in JCL errors. The example refers to the rules that appear in "Modifying nested procedures" on page 36.

```
Procedure C:
      //C        PROC
      //CS1      EXEC  PGM=CCC
      //CS1DD1   DD    DSN=A.B.C,DISP=SHR
      //CS1DD2   DD    SYSOUT=A
      //         PEND

   Procedure B:
      //B        PROC
      //BS1      EXEC  PROC=C
      //CS1.CS1DD1  DD  DSNAME=X.Y.Z
      //CS1.CS1DD3  DD  SYSOUT=A
      //BS2      EXEC  PGM=BBB
      //BS2DD1   DD    DSN=E,DISP=SHR
      //         PEND

   Procedure A:
      //A        PROC
      //AS1      EXEC  PROC=B
      //BS1.CS1.CS1DD1 DD DSN=X.Y.Z    This statement is an invalid
      //*                              override of procedure B, step BS1,
      //*                              DD CS1.CS1DD1 (rules 4 and 5)
      //*
      //BS1.CS1.CS1DD3 DD SYSOUT=A     This statement is an invalid
      //*                              override of procedure B, step BS1,
      //*                              DD CS1.CS1DD3 (rules 4 and 5)
      //*
```

```
   //BS1.BS1DD1 DD  DSN=G,DISP=SHR  This statement is an invalid
   //*                              addition to procedure B, step BS1
   //*                              (rule 3)
   //AS2      EXEC  PGM=AAA
   //AS2DD1   DD    DSN=E,DISP=SHR
   //         PEND

Job Stream:
   //JOB1     JOB
   //STEP1    EXEC  PROC=A
   //AS1.BS1.CS1.CS1DD1 DD DSN=X    This statement is an invalid
   //*                              override of procedure A, step AS1,
   //*                              DD BS1.CS1.CS1DD1 (rules 3 and 5)
   //STEP2    EXEC  PGM=IEFBR14
```

# Using system symbols and JCL symbols

System symbols and JCL symbols are character strings that represent variable information in JCL. They allow you to modify JCL statements in a job easily. A symbol-defining string is limited to eight characters, not including the identifying ampersand (&) character.

This section:

- Describes system symbols and JCL symbols and the differences between them
- Explains how to define JCL symbols
- Shows how to code system symbols and JCL symbols.

## What are system symbols?

System symbols represent values that are unique to each system. The system replaces system symbols with its own values when it processes started task JCL and batch job JCL (jobs and procedures read from a procedure library), and TSO logons. A started task is a task resulting from JCL that is processed immediately as a result of a START command. Batch job JCL is scheduled by JES2 or JES3 and is run based on system resources and other controls. For additional information about started tasks, see “Using symbols in started task JCL” on page 66. For additional information about batch job JCL, see “Using symbols in batch JCL” on page 56.

The following rules govern the use of system symbols:

- To use system symbols in batch JCL, you must first specify SYSSYM=ALLOW on the class definition, and assign the batch job to that class.
- You can use system symbols in started task JCL and batch job JCL (for both jobs and procedures), and in TSO logon procedures.
- Within started task JCL and batch job JCL, you can use system symbols wherever you use JCL symbols (described under “What are JCL symbols?” on page 39).
-

Note the following differences between system symbols and JCL symbols:

- Substitution texts for system symbols are either fixed for the life of an IPL (static system symbols) or determined by the system (dynamic system symbols).
- Substitution texts for JCL symbols can be controlled through input job stream modifications to their definitions.

Before you use system symbols in JCL, see *z/OS MVS Initialization and Tuning Reference* for a complete list of symbols and for details about how they work. Then read the rest of this section for specific information about using system symbols in started task JCL.

### Displaying static system symbols

If you are authorized to do so, you can enter the DISPLAY SYMBOLS command to display the static system symbols and associated substitution texts that are in effect for a member. The output from DISPLAY SYMBOLS shows you the system symbols that you can specify. See the description of DISPLAY SYMBOLS in *z/OS MVS System Commands* for the command syntax.

### Using system symbols in started task JCL

The general rules and recommendations for using system symbols (which are described in *z/OS MVS Initialization and Tuning Reference*) apply to started task JCL. The following are exceptions to those general rules and recommendations:

- Normally, you can specify an optional period at the end of system symbols. In started task JCL, you must follow the rules for JCL symbols when placing a period at the end of system symbols. See "Using symbols before fixed code" on page 48 for details.
- Although dynamic system symbols are supported in started task JCL, IBM does not recommend that you code them in started task JCL. The system substitutes text for dynamic system symbols at conversion time, which means that the system could assign different substitution texts to the same dynamic system symbol within the same job.

  For example, the resolved substitution text for the &JOBNAME dynamic system symbol is the name of the job assigned to the address space in which the JCL is converted, *not* the name of the JCL job being processed.

For further information about specifying system symbols in started task JCL, including examples, see "Using symbols in started task JCL" on page 66.

## What are JCL symbols?

JCL symbols differ from system symbols in that you must define them in started task JCL before you can use them in that JCL. The JCL symbols that you define are valid only for the current job. Conversely, there is no need to define system symbols; they are either defined to MVS or defined by your installation, and you can use them in any set of started task JCL.

The rules for coding JCL symbols are the same as for coding system symbols. You can code JCL symbols anywhere in started task JCL that you code system symbols.

This section explains how to define, nullify, and use JCL symbols in JCL.

### Defining and nullifying JCL symbols

When you code JCL symbols, you must define or nullify them in your JCL each time a job runs; otherwise, the system does not substitute text for JCL symbols.

The maximum length of any substitution text that you can assign to a JCL symbol is 255 characters.

To define or nullify a JCL symbol, code the substitution text on one or more of the following:

1. **The EXEC statement that calls procedures:** Use the EXEC statement to define substitution texts on statements in the called procedures. The substitution texts you assign override the default substitution texts assigned on the PROC statement. For example:

   ```
   //STEP1   EXEC   PROC=SEARCH,PARM1='MYDS1.PGM'
   ```

   The system uses a JCL symbol defined on the EXEC statement for any procedures that it invokes. A JCL symbol defined on an EXEC statement is not in effect for subsequent job steps in the same level of procedure nesting. See "Using symbols in nested procedures" on page 51 for more information.

   If you specify duplicate JCL symbols on an EXEC statement, the system uses the first substitution text as the default.
2. **The PROC statement that begins a procedure:** The PROC statement must begin in-stream procedures and can begin cataloged procedures. Use the PROC statement to define default substitution texts for JCL symbols in the procedure (you can override the defaults on the EXEC statement). If you do not define or nullify the substitution text for a JCL symbol on the EXEC statement, the system uses the default substitution text. For example:

   ```
   //PROC1   PROC   PARM2=OLD,PARM3=111222
   ```

   If you specify duplicate JCL symbols on a PROC statement, the system uses the first substitution text as the default.

   Assign only one substitution text to each JCL symbol used in a procedure.
3. **The SET statement that defines and nullifies: JCL symbols** Code the SET statement in the JCL before the first use of the JCL symbol. Use the SET statement to define JCL symbols that are used on:
   - JCL statements in the JCL stream
   - Statements in a procedure (when the EXEC statement that calls the procedure and the PROC statement for the procedure do not also define JCL symbols).

   For example:

   ```
   //LEVEL1   SET   PARM2=NEW,PARM3=DELETE
   ```

   If you define duplicate JCL symbols on a SET statement, the system assigns the last substitution text to the JCL symbol.

   **Note:** The substitution text specified on the SET statement is assigned to the JCL symbol regardless of the logic of the construct. This is because the SET statement is not executed conditionally (such as in the THEN and ELSE clauses of an IF/THEN/ELSE/ENDIF statement construct).

If the SET statement defines a value for a JCL symbol but that symbol is not coded in the JCL, there is no JCL error. Otherwise:

- All JCL symbols for which values are defined must be coded in the JCL.
- All JCL symbols coded in the JCL must have defined values.

***Syntax of JCL symbol definitions:*** To define a substitution text to a JCL symbol, code:

```
JCL_symbol_name=substitution_text
```

***Rules for defining JCL symbols:***

- Define a substitution text that is 1-255 characters long.
- Enclose within apostrophes substitution texts that do not fit on a single line. Continue values that do not fit on a single line as described in "Continuing JCL statements that contain symbols" on page 46.
- Do not specify the ampersand that identifies the JCL symbol in the procedure.

- Define JCL symbols on EXEC, PROC, or SET statements, as described in "Defining and nullifying JCL symbols" on page 39. For example, if the JCL symbol &NUMBER appears on one or more DD statements in a procedure, and you want to substitute the text **3390** for &NUMBER, code one or more of the following:

```
//SET1   SET   NUMBER=3390

//STEP1  EXEC   PROC=PROC1,NUMBER=3390

//PROC1  PROC   NUMBER=3390
```

- Do not specify JCL symbols within other JCL symbols. The results can be unpredictable, especially if the imbedded JCL symbol is not previously defined.

***Defining names for JCL symbols:*** IBM recommends that your installation define standard names for frequently used JCL symbols and enforce the use of those names. For example, if your installation frequently assigns department numbers in procedures, define the &DEPT JCL symbol and use it consistently. If your installation plans to provide a standard set of JCL symbols, ensure that all system and application programmers know about those JCL symbols.

You can define names for JCL symbols that are the same as system symbol names. When a JCL symbol has the same name as a system symbol, **the substitution text for the JCL symbol overrides the substitution text for the system symbol.** For example, if JCL defines a symbol with the name &SYSNAME, which is also the name of a system symbol, the system uses the substitution text that is defined in the JCL.

***Defining default substitution texts to JCL symbols:*** The substitution texts that you define to JCL symbols on the PROC statement serve as defaults. You should assign default values to all JCL symbols in a procedure. The system uses the default values on the PROC statement when no calling EXEC statement or SET statement overrides them.

***Using special characters in substitution texts:*** If a substitution text contains certain special characters, enclose the substitution text in apostrophes (for example, LOC='O''HARE'). The enclosing apostrophes are not considered to be part of the substitution text. See Table 11 on page 21 for a list of special characters.

If the substitution text contains multiple ampersands and is not enclosed in apostrophes, the system treats each pair of ampersands as a single character.

If the special characters include apostrophes, code each apostrophe as two consecutive apostrophes. You must code four consecutive apostrophes in substitution texts that are to be substituted into a parameter that is enclosed in apostrophes. For example:

```
//    SET LOC='O''''HARE'
//S1  EXEC PGM=IEFBR14,PARM='&LOC'
```

produces the following equivalent JCL, which is processed correctly:

```
//S1  EXEC PGM=IEFBR14,PARM='O''HARE'
```

However, if you code the following:

```
//    SET LOC='O''HARE'
//S1  EXEC PGM=IEFBR14,PARM='&LOC'
```

The equivalent JCL is:

```
//S1  EXEC PGM=IEFBR14,PARM='O'HARE'
```

The system fails this statement because the apostrophes resulting from the substitution are unbalanced.

When you want to code a JCL symbolic that consists of two parameters separated by a comma, you may have to enclose the JCL symbolic in triple apostrophes. For example:

```
//JOB1   EXEC   PROC1
//PROC1  PROC   WORK='''1000,500'''
//STEP1  EXEC   PROC2,WORK=&WORK
```

The substitution JCL would be:

```
//STEP1  EXEC   PROC2,WORK='1000,500'
```

If the substitution text begins and ends with matched parentheses, do not enclose the value in apostrophes. The parentheses are considered part of the substitution text. For example:

```
//TPROC PROC DISP=(NEW,PASS)
```

If the substitution text within the parentheses contains apostrophes, the apostrophes are considered part of the substitution text. The system does not remove them.

***Syntax for nullifying JCL symbols:*** To nullify a JCL symbol, code:

```
JCL_symbol_name=
```

- Do not code the ampersand that identifies the JCL symbol in the procedure.
- Do not code a substitution text after the equal sign.
- Do not code literal blanks (for example, VALUE=' ').

For example, if the JCL symbol &NUMBER appears in one or more DD statements in a procedure, code one or more of the following to nullify UNIT=&NUMBER:

```
//SET2     SET   NUMBER=

//CALLER   EXEC   PROC=ABC,NUMBER=,ACCT=DID58

//ABC      PROC  NUMBER=,LOC=POK
```

When nullifying JCL symbols, keep the following in mind:

- When you nullify a JCL symbol, delimiters, such as leading or trailing commas, are not nullified. In some cases, the remaining comma is required; in others it causes a syntax error.
- Do not nullify JCL symbols that appear on JCL keywords that do not accept NULL values. The syntax descriptions of the individual keywords specify whether the keywords allow NULL values.
- If you use an EXEC statement to nullify a JCL symbol, and a PROC statement specifies a default substitution text for the JCL symbol, the JCL symbol is nullified.

The following sections explain special considerations to make when JCL symbols are positional and not positional.

*When a JCL Symbol is Positional:* When a JCL symbol is a positional parameter, and another parameter follows it, code a comma to omit the positional parameter. Code

commas both before and after the JCL symbol; the required commas remain after the JCL symbol is nullified. For example, &NUMBER for the unit count:

```
UNIT=(3390,&NUMBER,DEFER)
```

When &NUMBER is nullified, the parameter correctly becomes:

```
UNIT=(3390,,DEFER)
```

*When a JCL Symbol is Not Positional:* When a JCL symbol is *not* a positional parameter, *do not* code a comma to omit the parameter. Do not code a comma before the JCL symbol; no commas remain after the JCL symbol is nullified. For example, serial numbers in the VOLUME=SER parameter:

```
VOLUME=SER=(&FIRST&SECOND)
```

If either of the JCL symbols is nullified, a leading or trailing comma does not remain. If you nullify &FIRST and assign 222222 for &SECOND, the parameter correctly becomes:

```
VOLUME=SER=(222222)
```

If you nullify &SECOND and define 111111 to &FIRST, the parameter correctly becomes:

```
VOLUME=SER=(111111)
```

Code a comma when it is required in a substitution text. Enclose the comma in apostrophes (because it is a special character). For example:

```
//CALLER  EXEC  PROC=ABC,FIRST=111111,SECOND=',222222'
```

## Coding symbols in JCL

JCL symbols and system symbols can represent parameters, subparameters, or values in procedures or in the parameter field of statements; those that vary each time a job runs are good candidates to be coded as symbols.

You can code *JCL symbols* in:

- JCL statements in the input job stream, submitted either in batch mode or from a TSO session (but not in the job stream read in response to a START command)
- Statements in cataloged or in-stream procedures (which do not include started task JCL)
- DD statements that are added to a procedure (something that is possible, but not practical for a started task procedure).

You can code *system symbols* in started task JCL and batch job JCL (jobs and procedures), which can be read only from a procedure library. Therefore, you can code system symbols only in statements in cataloged procedures.

Symbolic parameters are not permitted in place of the "*" or "DATA" positional parameters on SYSIN type DD statements. SYSIN DD statements do not necessarily have SYSIN as the ddname. See “SYSIN DD statement” on page 326 for a description of the SYSIN DD statement.

For example, if the data set name on a DD statement in an INCLUDE group can vary each time the INCLUDE group is imbedded in the JCL, you can code the DSNAME parameter as a system symbol on the DD statement:

```
DSNAME=&DAY
```

If a job step is charged to different account numbers each time the procedure is executed, code the ACCT parameter on the EXEC statement as one or more system symbols or JCL symbols:

```
ACCT=&ALLNOS
ACCT=&FIRST&SECOND&THIRD
```

- For information about using symbols in nested procedures, see "Using symbols in nested procedures" on page 51.
- For information about using symbols in started task JCL, see "Using symbols in started task JCL" on page 66.
- For information about using symbols in batch job JCL, see "Using symbols in batch JCL" on page 56.

## Rules for coding symbols in JCL

Follow these rules when coding symbols in JCL:

1. Do not code EXEC statement parameter and subparameter keywords as names for JCL symbols.

   **Example:** Do not code &REGION=200K or REGION=&REGION; correctly code REGION=&SIZE.
2. Do not code DD or JOB statement keywords as JCL symbols in procedures or jobs that are started by a START command from the operator console. This rule includes the following obsolete keywords:
   - AFF
   - SEP
   - SPLIT
   - SUBALLOC
   - MODE

   This rule also includes DCB subparameters. For example, do not use the following DCB subparameters as symbol values:
   - BFALN
   - LRECL

   For a complete list of DCB subparameters, see "DCB subparameters" on page 140.
3. When coding a JCL symbol that has the same name as a system symbol, keep in mind that the substitution text for the JCL symbol overrides the substitution text for the system symbol with the same name.
4. Do not use symbols to change the identifier field, name field, or operation field of a JCL statement.

In addition to the preceding rules for coding symbols in JCL, you also need the general rules for coding system symbols. See the coding system symbols information in *z/OS MVS Initialization and Tuning Reference*.

**Note:**

1. JCL supports *substringing* of system symbols but not JCL symbols. You can use substringing to specify a subset of characters in substitution text. For an explanation of substringing symbols, see the substringing symbols information in the general rules for coding symbols in *z/OS MVS Initialization and Tuning Reference*
2. You can also use double ampersand notation in your JCL code. See *z/OS MVS Initialization and Tuning Reference* for further information.

For instance, suppose you want to enter a substringed symbol as a parameter of an EXEC statement of a started task. By using a double ampersand you can force MVS to defer processing the statement until after the JCL is executed and the program is running. For example, given a value of '05' for SYSCLONE, the statement could read:

```
//Step1 EXEC PGM=MVSCMD,PARM='F RMF,S III,MEMBER(3&&SYSCLONE(2:1))'
```

The MVS converter will change that to:

```
//Step1 EXEC PGM=MVSCMD,PARM='F RMF,S III,MEMBER(3&SYSCLONE(2:1))'
```

which is the JCL that gets executed. Your MVSCMD program would then take what is in the PARM on its EXEC statement and issue it as an MVS command:

```
F RMF,S III,MEMBER(3&SYSCLONE(2:1))
```

which the command symbolic substitution routine then processes and changes to:

```
F RMF,S III,MEMBER(35)
```

## Determining equivalent JCL

When you submit JCL that specifies symbols, the system responds as if you had coded the equivalent JCL (without symbols) produced by the following sequence of operations:

1. Determine the substitution texts. The system:
   - Does not consider apostrophes that enclose symbols as part of their substitution texts.
   - Considers parentheses that enclose symbols as part of their substitution texts.
   - Compresses two-to-one the double apostrophes within symbols.
   - Compresses two-to-one the double ampersands in symbols that are not enclosed in apostrophes.
   - Does not compress double ampersands within symbols that are enclosed in apostrophes.
2. Substitute all symbols.
   - Resolution of all symbols might determine the processing of subsequent statements. For example, a JCLLIB or INCLUDE statement might contain symbols that determine which statements are used in the job.
   - Symbols on JCL records are treated as if they were resolved simultaneously.

The following example shows a procedure that defines JCL symbols:

```
//EXAMPLE   PROC SYM1='What''''s up, Doc?',SYM2=(DEF),SYM3=&&&&TEMP1,
//       SYM4='&&TEMP2',SYM5=&&TEMP3,TEMP3=TEMPNAME,
//       SYM6=&TEMP3
//S1        EXEC PGM=WTO,PARM='&SYM1',ACCT=&SYM2
//DD1       DD   DSN=&SYM3,UNIT=SYSDA,SPACE=(TRK,(1,1))
//DD2       DD   DSN=&SYM4,UNIT=SYSDA,SPACE=(TRK,(1,1))
//DD3       DD   DSN=&SYM5,UNIT=SYSDA,SPACE=(TRK,(1,1))
//DD4       DD   DSN=&SYM6,UNIT=SYSDA,SPACE=(TRK,(1,1))
//          PEND
```

The PROC statement assigns the following substitution texts to the JCL symbols:

```
SYM1      What''s up, Doc?
SYM2      (DEF)
SYM3      &&TEMP1
SYM4      &&TEMP2
SYM5      &TEMP3
TEMP3     TEMPNAME
SYM6      &TEMP3
```

The equivalent JCL produced by the substitution, when the procedure is expanded, is:

```
//S1        EXEC PGM=WTO,PARM='What''s up, Doc?',ACCT=(DEF)
//DD1       DD   DSN=&&TEMP1,UNIT=SYSDA,SPACE=(TRK,(1,1))
//DD2       DD   DSN=&&TEMP2,UNIT=SYSDA,SPACE=(TRK,(1,1))
//DD3       DD   DSN=&TEMP3,UNIT=SYSDA,SPACE=(TRK,(1,1))
//DD4       DD   DSN=&TEMP3,UNIT=SYSDA,SPACE=(TRK,(1,1))
```

Note the following in the example:

- SYM1 requires four apostrophes in its original definition because it is substituted into a parameter enclosed in apostrophes. The system compresses the apostrophes in the symbol definition when the value of the symbol is determined, and again when the `EXEC PARM` parameter is processed. The parameter passed to the `WTO` program is:

  ```
  What's up, Doc?
  ```

- The single ampersand produced by `SYM5` in the `DSN` parameter of DD3 cannot be interpreted as the start of a new JCL symbol, since substitution is performed only once for a given statement. All symbols are treated as if they were resolved simultaneously. If the symbol TEMP3 defined on the PROC statement is not used elsewhere in the procedure, a JCL error results.
- The symbol TEMP3 cannot be used to assign a value for the symbol SYM6 on the same statement. Because all symbolic parameters are resolved simultaneously, the value assigned to SYM6 cannot depend on another symbol defined at the same time. The system assigns the value &TEMP3, not &&TEMP2, to SYM6. Again, if the symbol TEMP3 is not used elsewhere in the procedure, a JCL error will result.

## Continuing JCL statements that contain symbols

The system evaluates continuations of JCL statements that contain symbols as follows:

1. The system substitutes all symbols on an 80-character record.
2. The system determines if the record continues to another record. If symbolic substitution produces a null record (a line that is blank except for slashes in columns 1 and 2) as the continuation record, the substitution is not valid.

For example, consider the following JCL:

```
//SET1   SET  VAL1='ABC,',VAL2=DEF,NULLSYM=''
//S1     EXEC PGM=IEFBR14,PARM=&VAL1
//         TIME=30
//S2     EXEC PGM=IEFBR14,PARM=&VAL2
//         TIME=30
//S3     EXEC PGM=IEFBR14,PARM=&VAL1
//         &NULLSYM
```

The JCL records that define step S1 form a valid continuation; the JCL symbol `VAL1` introduces a comma, and the continuation is correctly coded.

Steps S2 and S3 are not valid. In step S2, the first record does not end in a comma after substitution of `VAL2`. In step S3, the record containing `NULLSYM` evaluates to a null record after symbolic substitution.

It may be that the number and length of symbols form a parameter that does not fit within the limits imposed by an 80-character record. (In reality the limit is 68 characters, because columns 1, 2, and 3 must contain respectively a slash, slash, and blank, and column 72 must be blank.) Two techniques for handling this situation are: (1) defining shorter symbols to substitute for the longer ones, or (2)

dividing the series of symbols so as to form two parameters, which would allow you to place a comma after the first and move the second to a continuation record.

## Coding symbols in comments

The system does not process symbols in comment statements or in comment fields of JCL statements. Comments on JCL statements that contain symbols are evaluated as follows:

- In the original submitted JCL, the system recognizes the beginning of the comment field when it encounters the blank character at the end of the parameter field. For purposes of symbolic substitution, the system disregards text occurring after this blank.
- After performing symbolic substitution, the system re-evaluates the resulting equivalent JCL to determine where the parameter field ends. The system recognizes the beginning of the comment field in the substituted JCL when it encounters the blank character at the end of the (potentially modified) parameter field. The system disregards text occurring after this blank in subsequent processing.

Example:

```
//     SET  QUOTE=''''
//S1   EXEC PGM=IEFBR14,PARM=&QUOTE.ABC   DEF&QUOTE
//DD1  DD   DUMMY
```

The equivalent JCL produced by substitution is

```
//S1   EXEC PGM=IEFBR14,PARM='ABC   DEF&QUOTE
//DD1  DD   DUMMY
```

DEF&QUOTE is considered a comment because it follows the blank that ends the parameter field, so the second instance of &QUOTE will not be replaced during symbolic substitution. Because the first &QUOTE symbol resolves to a single quotation mark, the system expects to either find another single quotation at the end of a subparameter list, or find a continuation to the next line. The EXEC statement receives an error message indicating that the system did not receive an expected continuation.

Example:

```
//    SET  CONT='  ',T='(30,0)'
//S1  EXEC PGM=IEFBR14&CONT,PARM='ABC  DEF',TIME=&T
```

The equivalent JCL is:

```
//S1  EXEC PGM=IEFBR14  PARM='ABC  DEF',TIME=(30,0)
```

The text (30,0) is substituted for the symbol &T. However, because substitution introduced a blank character after the program name parameter, all text following the blank is considered to be a comment. Thus the system does not process the PARM and TIME parameters.

## Coding symbols in apostrophes

You can code symbols in apostrophes on the following keywords:

- The DD statement AMP parameter
- The DD statement PATH parameter
- The DD statement SUBSYS parameter
- The EXEC statement ACCT parameter
- The EXEC statement PARM parameter.

When you specify these parameters, the system regards a string beginning with an ampersand (&) inside the apostrophes as a symbol when the following conditions are true:

- The character following the ampersand is not another ampersand.
- The characters following the ampersand are ended by a character that is not alphabetic, numeric, or national. The ending character must be not more than 9 characters after the ampersand. The symbol cannot be more than 8 characters long.
- The string of characters delimited by the & (ampersand) character and the ending character is:
  - Defined as a symbol on a PROC, EXEC, or SET statement
  - A system symbol.

The system treats a string beginning with an ampersand but not meeting these criteria as a literal sequence of characters. Thus the system does not substitute text for symbols and does not issue error messages.

In the following example, &XXX is a JCL symbol that is defined in the STEP2 EXEC statement. &INPUT is not a symbol because it is not defined.

```
//TPROC PROC
//STEP1 EXEC PGM=IEFBR14,PARM='&INPUT&XXX'
//      PEND
//STEP2 EXEC TPROC,XXX=VALUE
```

The ending character for &XXX is the apostrophe.

The result of the example is:

```
//EXEC PGM=IEFBR14,PARM='&INPUTVALUE'
```

On parameters that are not in the list, the system correctly resolves a symbol that is enclosed in apostrophes when the symbol is immediately preceded by a symbol that is *not* enclosed in apostrophes. For example, both A and B are substituted correctly in:

```
//DD1  DD  &A'&B',DISP=OLD
```

A symbol within apostrophes cannot be broken at column 71 and continued to the next line. For example, the following JCL statement is incorrect:

```
//    SET  SYMBOL=VALUE
//S1          EXEC PGM=IEFBR14,TIME=(30,0),REGION=4K,PARM='Print &SYMB
//             OL'
```

The JCL symbol SYMBOL is not substituted because it must be coded on a single JCL record. A JCL error may result.

## Using symbols before fixed code

A period is required after a symbol when the code that follows the symbol is fixed and begins with:

- An alphanumeric or national character ($, #, @)
- A period.
- A left parenthesis, when it is at the start of the designation of a relative generation of a generation data group (GDG) that does not contain a plus or minus sign.

The system recognizes the period as a delimiter. The period does not appear after you assign a substitution text to a symbol or nullify a symbol.

For example, if the first part of a data set name varies and the last does not, as in MONDATA, TUESDATA, and so forth, code:

```
DSNAME=&DAY.DATA
```

When coding a system symbol in a data set name with a relative generation number, you must place a period between the system symbol and the generation number if the following conditions are met:

- The system symbol immediately precedes the generation number.
- The generation number is not preceded by a plus or minus sigh.

For example, if &SYSNAME resolves to SY1, type DSNAME=PROD.&SYSNAME.(0) if the desired data set name is PROD.SY1(0). If a plus or minus sign is included, the period is optional: DSNAME=PROD.&SYSNAME.(+0) and DSNAME=PROD.&SYSNAME(+0) are both acceptable.

Code two consecutive periods (..) if a period follows a symbol. For example, code &DEPT..POK when the desired value is D58.POK and DEPT=D58 is the value assignment.

### Using symbols as positional parameters

When a symbol is a positional parameter followed by other parameters in the statement, follow the symbol with a period instead of a comma. For example:

```
//DS1   DD  &POSPARM.DSNAME=ATLAS,DISP=OLD
```

If &POSPARM is nullified, the statement appears as:

```
//DS1   DD  DSNAME=ATLAS,DISP=OLD
```

When assigning a substitution text to &POSPARM, include the comma:

```
POSPARM='DUMMY,'
```

### Using two or more symbols in succession

Code two or more symbols in succession without including a comma. For example:

```
PARM=&DECK&CODE
```

If the substitution text is to contain a comma, include the comma in the substitution text.

### Using multiple symbols

The same symbol can appear more than once in a job. You can assign different substitution texts to the same symbol on different statements.

The same symbol can appear more than once in a procedure, as long as its substitution text is the same throughout the procedure. For example, &DEPT can appear several times in a procedure, if the department number is always to be the same.

### Using the SYSUID system symbol

As long as you observe the rules listed in "Rules for Coding Symbols in JCL," you can code the SYSUID system symbol anywhere in your JCL where you would code a user ID *except* on the keywords and statements listed in the topic “Restrictions on coding SYSUID” on page 50. The system replaces &SYSUID with the user ID under whose authority the job will run, which is normally one of the following:

- The USER parameter from the JOB statement, if specified, or
- The user ID from which the job was submitted.

**Note:** If userid propagation does not occur, (for example a security product is not active or the submitting userid is not allowed to propagate), SYSUID will be null. A security product is considered "not active" in OS/390® if it has been disabled. If RACF® is running in a fail soft mode, the security product is considered "active."

**Note:** If RACF is active and the job is running with a user ID not defined to RACF, the system provides substitute characters for &SYSUID and may fail the job because of this JCL error. The same results may occur if &SYSUID is not resolved to a valid user when RACF is *not* active.

You can, for example, use &SYSUID as a generic qualifier in a data set name specified in a transaction program profile that will be invoked by a transaction program. Code SYSUID on the DSNAME parameter as the high-level qualifier of the data set name:

```
//DD1   DD  DSNAME=&SYSUID..PROFILE,DISP=(NEW,KEEP)
```

The system replaces the symbol with the userid of the transaction program invoker. If userid ROGERS invokes the transaction program, the system will create the data set name ROGERS.PROFILE.

### Restrictions on coding SYSUID

Do not code &SYSUID on the following keywords and statements:

- Job statement USER, GROUP, PASSWORD, and SECLABEL parameters when a security product like RACF is active.
- The XMIT JCL statement; coding &SYSUID on XMIT causes a JCL error and the job is flushed.
- JES2 or JES3 control statements.
- Job statement accounting information and programmer name fields.

In an APPC scheduling environment:

- Avoid coding &SYSUID on the DD statement SUBSYS parameter; symbol substitution is unpredictable on SUBSYS.
- Avoid coding &SYSUID on the JOB statement NOTIFY parameter; if the user ID specified through the Allocate service is longer than 7 characters, the Allocate request will fail.

Avoid using &SYSUID as an unqualified data set name. Depending on the other statements in the transaction program profile, the system might interpret the data set name as a temporary data set name.

## Examples of defining and coding symbols in JCL

***Example 1:***

```
//JOBA     JOB  ...
//INSTREAM PROC LOC=POK
//PSTEP    EXEC PGM=WRITER
//DSA      DD   SYSOUT=A,DEST=&LOC
//         PEND
//CALLER   EXEC PROC=INSTREAM,LOC=NYC
//
```

In this example of an in-stream procedure, the &LOC symbol has a default value of POK on the PROC statement; then it is assigned an execution value of NYC on the calling EXEC statement.

***Example 2:***

```
//JOBB      JOB  ...
//INSTREAM  PROC LOC=POK,NUMBER=3390
//PSTEP     EXEC ...
//PIN       DD   DSNAME=REPORT,DISP=(OLD,KEEP),UNIT=&NUMBER
//POUT      DD   SYSOUT=A,DEST=&LOC
//          PEND
//CALLER    EXEC PROC=INSTREAM,NUMBER=,LOC=STL
//PSTEP.INDATA  DD   *
                .
                (data)
                .
/*
```

This code nullifies the &NUMBER JCL symbol. The calling EXEC statement assignment of STL for the &LOC symbol overrides the PROC statement assignment of POK.

***Example 3:*** This example illustrates execution of an in-stream procedure to test symbols before placing the procedure in a procedure library. The in-stream procedure named TESTPROC is:

```
//TESTPROC PROC A=IMB406,B=ABLE,C=3390,D=WXYZ1,
//           E=OLD,F=TRK,G='10,10,1'
//STEP     EXEC PGM=&A
//DD1      DD   DSNAME=&B,UNIT=&C,VOLUME=SER=&D,DISP=&E,
//           SPACE=(&F,(&G))
//         PEND
```

To execute this in-stream procedure and override &A with IEFBR14, &B with BAKER, and &E with (NEW, KEEP) but leave the other parameters the same, call the in-stream procedure with:

```
//CALLER1 EXEC PROC=TESTPROC,A=IEFBR14,B=BAKER,E=(NEW,KEEP)
```

Note that the value (NEW,KEEP) does not require apostrophes because it contains a matched pair of parentheses. See Table 12 on page 22 for more information.

After symbolic substitution, the statements are:

```
//STEP  EXEC PGM=IEFBR14
//DD1   DD   DSNAME=BAKER,UNIT=3390,VOLUME=SER=WXYZ1,
//           DISP=(NEW,KEEP),SPACE=(TRK,(10,10,1))
```

***Example 4:*** To execute the in-stream procedure in the previous example and change DD1 to resemble a temporary scratch space, code the following statement:

```
//CALLER2 EXEC PROC=TESTPROC,A=IEFBR14,B=,C=3390,D=,E=
```

After symbolic substitution, the statements are:

```
//STEP  EXEC PGM=IEFBR14
//DD1   DD DSNAME=,UNIT=3390,VOLUME=SER=,DISP=,SPACE=(TRK,(10,10,1))
```

## Using symbols in nested procedures

The general rules described in "Using system symbols and JCL symbols" on page 38 also apply to symbols in nested procedures, along with the following rules:

1. Within a nested procedure, assign only one substitution text per symbol. You can use the same symbol in other nested procedures and assign it different values.
2. If you assign or nullify the value for a symbol on an EXEC statement that calls a nested procedure, the substitution text that you specify on the EXEC statement is used in the procedure. The EXEC statement overrides any default value you specify on the PROC statement of the nested procedure.
3. When the EXEC statement that calls the nested procedure does not assign a substitution text to the symbol, the system uses the default substitution text specified on a PROC statement.

   One way to provide an override value for a symbolic in a nested procedure is to design the procedure so that it requires no assignment of default symbolic parameter values. If the PROC statement of the inner procedure contains no default value, the system uses the value specified on the EXEC statement of the outer procedure. For example:

```
//TESTJCL   PROC
//STEP1     EXEC  TESTJCL1
//          PEND
//TESTJCL1  PROC
//STEP2     EXEC  PGM=IEFBR14,PARM=&PVAL
//SYSUDUMP  DD    SYSOUT=A
//          PEND
//RUNIT     EXEC  TESTJCL,PVAL=EXEC0
```

4. If you assign or nullify a substitution text for a symbol on a SET statement, the substitution text that you specify on the SET statement is used in all subsequent statements, procedures, and nested procedures. However, if the calling EXEC statement or the PROC statement of the procedure assigns or nullifies the symbol, it only applies to subsequent statements within that PROC and subsequent nested procedures within that procedure.
5. If you do not assign or nullify a value for a JCL symbol in a nested procedure, the value used for the JCL symbol in this procedure is obtained from the procedure in which this procedure is nested.
6. If a JCL symbol is not assigned a substitution text or is not nullified, it is an undefined JCL symbol which might cause errors in the JCL.

Table 13 shows rules 2 through 6 in a summary table, which is the order in which the value for a symbol is resolved.

*Table 13. Summary of Rules 2 through 6 for Symbols in Nested Procedures*

| | **Where the symbol is defined** | | | | |
|---|---|---|---|---|---|
| **Value Used** | **EXEC**<br><br>**(Rule 2)** | **PROC**<br>**Not EXEC**<br><br>**(Rule 3)** | **SET**<br>**Not PROC**<br>**Not EXEC**<br><br>**(Rule 4)** | **Nested Value**<br>**Not SET**<br>**Not PROC**<br>**Not EXEC**<br>**(Rule 5)** | **None**<br><br>**(Rule 6)** |
| EXEC Value | X | | | | |
| PROC Value | | X | | | |
| SET Value | | | X | | |
| Nested Value | | | | X | |
| Undefined | | | | | X |

## Examples of coding symbols in nested procedures

***Example 1:*** The following example defines symbols A, B, and C with multiple assignments in nested procedures:

```
                                                Current value of symbol:
//MYJOB    JOB    ...                          Level 0:
//SET1     SET    A=123,B=456                  A=123,B=456,C=undefined
            .
//PROC1    PROC   A=234,C=GHI
//PSTEP1   EXEC   PROC=PROC2,A=ABC,B=DEF
//PSTEP2   EXEC   PGM=IEFBR14
//         PEND
            .
//PROC2    PROC
//P2STEP1  EXEC   PGM=IEBGENER
//SYSPRINT DD     SYSOUT=A
//SYSUT1   DD     DSN=&A..&B,DISP=SHR
//SYSUT2   DD     SYSOUT=A,DCB=LRECL=&C
//         PEND
            .
//STEP1    EXEC   PROC=PROC1,A=,C=789          Level 1:
            .                                   A=,B=456,C=789
++PROC1    PROC   A=234,C=GHI
++PSTEP1   EXEC   PROC=PROC2,A=ABC,B=DEF       Level 2:
            .                                   A=ABC,B=DEF,C=789
++PROC2    PROC
++P2STEP1  EXEC   PGM=IEBGENER
++SYSPRINT DD     SYSOUT=A
++SYSUT1   DD     DSN=ABC.DEF,DISP=SHR
++SYSUT2   DD     SYSOUT=A,DCB=LRECL=789
++         PEND
            .
++PSTEP2   EXEC   PGM=IEFBR14                  Level 1:
            .                                   A=,B=456,C=789
++         PEND
//BARNEY   EXEC   PGM=IEFBR14                  Level 0:
//...         .                                 A=123,B=456,C=undefined
```

The processing of symbols in MYJOB is:

- When the SET statement SET1 is processed, symbols A and B are defined and initialized to the values 123 and 456, respectively. (The C symbol C is not yet defined.) The level of nesting (scoping) is 0.
- EXEC statement STEP1 references in-stream procedure PROC1. The symbols are changed as follows: A is nullified, B remains 456 from SET statement SET1, and C is defined and assigned the substitution text 789. The level of nesting (scoping) is now 1.

  PROC statement PROC1 defines the default values for the symbols A and C as A=234 and C=GHI. However, these values are overridden by the values on the EXEC statement STEP1 as: A=, and C=789.B remains 456 from SET statement SET1. The level of nesting is still 1.
- EXEC statement PSTEP1 is processed. The substitution texts for the symbols are updated again as: A=ABC and B=DEF. (C remains 789 from EXEC statement STEP1.) The substitution texts are passed to procedure PROC2 referenced by EXEC statement PSTEP1. The level of nesting is now 2.
- The statements in procedure PROC2 are processed. The values used to resolve the symbols on DD statements SYSUT1 and SYSUT2 are those from level 2, namely A=ABC, B=DEF, C=789. The level of nesting returns to level 1.
- EXEC statement PSTEP2 in PROC1 is processed. This statement does not change the values of the symbols. However, because the expansion of PROC2 is complete, the values of the symbols return to the level 1 values held prior to procedure PROC2, which are A=, B=456, and C=789. The level of nesting returns to level 0.

- EXEC statement BARNEY is at level 0 and the substitution texts for symbols are restored to their original values: A=123,B=456, and C=undefined. The substitution texts, defined by SET statement SET1, are retained throughout this level of nesting (level 0).

***Example 2:*** To illustrate the scope of symbolics in the case of nested procedures, consider the following example, where PROC1 calls PROC2:

```
//JOB2   JOB    ...
//PROC1  PROC   WORK='''1000,500'''
//S1     EXEC   PROC2,WORK=&WORK
//S2     EXEC   PROC2,WORK=&WORK
//       PEND
//PROC2  PROC   WORK='''500,250''',LABEL=DUMMY
//P1     EXEC   PGM=IEFBR14
//DD1    DD     UNIT=SYSDA,SPACE=(TRK,(&WORK)),DSN=&LABEL
//       PEND
//J1     EXEC   PROC1,WORK='''500,250''',LABEL=UNUSED
//J2     EXEC   PROC1
```

In the prior example, the symbolic LABEL is defined as UNUSED in EXEC statement J1, which calls PROC1. The symbolic LABEL is not used in PROC1 but *is* used in PROC2, which is called by PROC1 and therefore *is* in the scope of the original definition of the symbolic.

## Using symbols in JES in-stream data

For programming flexibility and efficiency, symbolic substitution is supported for data that is contained within JES in-stream data sets. Unlike symbolic substitution in the JCL stream of a job, which is performed by the JCL converter during processing of JCL statements, in-stream symbolic substitution is performed by JES when an in-stream data set is read.

The three types of symbols that can be used for JES in-stream substitution are JCL Symbols, JES Symbols, and System Symbols:

**JCL symbols**

By default, JCL Symbols are only available to the job at the converter phase and are lost by the time the job runs. However, by using the **EXPORT** and **SET** JCL statements, JCL symbols can be made available to the job execution phase.

Any JCL symbols that are inherited from a submitting job through the internal reader SYMLIST facility are implicitly exported. Exported JCL Symbols can be accessed during the job execution phase using the JCL Symbol Service (IEFSJSYM) or the JES Symbol Service (IAZSYMBL).

SYSUID is a special type of JCL symbol, which is set and maintained by JES. Unlike other JCL symbols, the value for the SYSUID symbol is substituted in the in-stream data even if it was not explicitly EXPORTed in the JCL stream of the job. The value that is used for substitution in the in-stream data is the same as the one used during conversion, except when the SYSUID symbol was modified by the SET JCL statement but not EXPORTed. In this case, the conversion uses the modified value of the symbol, whereas the original value is used for the in-stream data. If this is not the result you want, use the **EXPORT JCL** statement to ensure that the value used for the in-stream data is the same as that used by conversion.

The JCL Symbol Service (**IEFSJSYM**) is documented in *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG*.

**JES symbols**
JES Symbols are dynamic symbols that can be managed and manipulated using the JES Symbol Service (**IAZSYMBL**). The JES Symbol Service is documented in *z/OS JES Application Programming*.

**System symbols**
System Symbols are specific to the MVS system. Refer to "Using system symbols and JCL symbols" on page 38 and the **ASASYMBM** service in *z/OS MVS Programming: Assembler Services Reference ABE-HSP*). System symbols are defined in the **IEASYM***xx* member of **SYS1.PARMLIB**, and are described in *z/OS MVS Initialization and Tuning Reference*.

The type of symbol substitution used for the in-stream data is controlled by the SYMBOLS keyword coded on the DD statement that defines the in-stream data set. Without the SYMBOLS keyword, JES does not perform symbol substitution for the in-stream data set and the application interprets the data exactly as it is entered in the data set.

The SYMBOLS keyword can be defined as follows:

**SYMBOLS=JCLONLY**
Names of JCL symbols and JES symbols found in the in-stream data set are replaced with their values.

**SYMBOLS=EXECSYS**
Substitution follows the SYMBOLS=JCLONLY rule. In addition, system symbols defined on the system during job execution can be used in the in-stream data.

**SYMBOLS=CNVTSYS**
Substitution follows the SYMBOLS=EXECSYS rule, with the exception that system symbols used for substitution are taken not from the system where the job is executing, but from the system where the job has undergone JCL conversion.

The symbols will have values that they had at the time of JCL conversion.

The syntax rules for using symbols in in-stream data include those described previously for using symbols in JCL. One important difference is the handling of blanks in the input data. When symbols are substituted in JCL statements, there is no special treatment of blanks—as symbols are substituted, the resulting string expands or contracts depending on whether the symbol value is longer or shorter than the symbol expression (symbol name with a leading ampersand character and optional period at the end of the symbol name). When symbols are substituted in in-stream data, the system attempts to maintain the position of non-blank characters. This is achieved by adding or removing blanks between non-blank character sequences. At least one blank is always preserved to maintain syntactical validity of the data. The resulting string never contracts and only expands if there are not enough blanks to remove to maintain data positioning. Refer to "Defining and nullifying JCL symbols" on page 39 for additional information.

## JCL symbol service (IEFSJSYM)

The JCL Symbol Service (**IEFSJSYM**) gives applications read-only access to JCL symbols that are made available to the job execution phase. The specific JCL symbols that are made available at the job execution phase are defined by the EXPORT SYMLIST statement. The JCL Symbol Service is documented in *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG*.

### JES symbol service (IAZSYMBL)

The JES Symbol Service (**IAZSYMBL**) manages JES symbols, which can be used to pass data between applications that are running in the same job step, to create JCL symbols for submitted jobs, and to pass information between applications and JES. The JES Symbol Service is documented in *z/OS JES Application Programming*.

## Using symbols in batch JCL

You can code both system symbols and JCL symbols in batch JCL for both jobs and procedures. This information provides examples of how to code system symbols and JCL symbols in batch JCL. For details on how to code system symbols in JCL, and how to define and code JCL symbols in JCL, see "Using system symbols and JCL symbols" on page 38.

Because a batch job can be routed to another system for execution, the symbol values that are resolved when the job is initially converted must also resolve correctly when the job is executed. Therefore, use one of the following methods to use system symbols in batch JCL:

- Use the SYSTEM= keyword (or your JES2 or JES3 JECL equivalent) to ensure that the batch job executes on a system where the resolved symbol values are valid.
- Only use symbols that have the same value on every system in your JES complex. For example, you can define a system symbol for the location of your JES complex: thereafter, any batch job that uses that symbol can run on a system in your JES complex.

# Chapter 6. Job control statements on the output listing

Use the JOB statement MSGLEVEL parameter to request that job control statements be printed in the job log output listing. Code MSGLEVEL=(1,1) to receive the maximum amount of information, in the following order:

- JES messages and job statistics.
- All job control statements in the input stream and procedures.
- Messages about job control statements.
- JES and operator messages about the job's processing: allocation of devices and volumes, execution and termination of job steps and the job, and disposition of data sets.

***Statements in listing:*** To identify the source and type of each statement, the system prints certain characters in columns 1 and 2 or 1, 2, and 3 of the listing. These identifying characters are explained in Table 14. The listing shows all procedure statements as they appear in the cataloged procedure; the listing does not show parameter substitutions and overrides on the statement itself.

***Symbolic parameters:*** The job log listing shows the symbolic parameters in procedure statements. The values assigned to the parameters are given in IEF653I messages. These messages appear immediately after each statement that contains symbolic parameters.

***EXEC overriding parameters:*** A procedure EXEC statement appears in the job log listing exactly as it appears in the procedure. Overridden parameters must be shown by the program being executed:

- For the EXEC statement that executes the assembler program, the **Diagnostic Cross Reference and Assembler Summary** produced by the assembler program shows the overriding parameters.
- For the EXEC statement that executes the linkage editor, the linkage editor listing shows the overriding parameters.

*Table 14. Identification of Statements in Job Log*

| **Columns 1, 2, and 3** | **Source and type of statement** |
|---|---|
| **Job control statements in the input stream** | |
| // | JCL statement |
| //* | Job control statement that is not a JCL comment statement but one that the system considers to contain only comments |
| /* | JES2 statement |
| //* | JES3 statement |
| /* | Certain JES3 control statements |
| //* | JCL comment statement |
| **Cataloged procedure statements** | |
| XX | DD statement that was not overridden and all other JCL statements, except the JCL comment statement. Each statement appears in the listing exactly as it appears in the procedure. |

© Copyright IBM Corp. 1988, 2015

*Table 14. Identification of Statements in Job Log (continued)*

| Columns 1, 2, and 3 | Source and type of statement |
|---|---|
| X/ | DD statement that was overridden (preceded by the overriding DD statement) |
| XX* | Job control statement that is not a JCL comment statement but one that the system considers to contain only comments |
| XX* | JCL comment statement |
| **In-stream procedure statements** | |
| ++ | DD statement that was not overridden and all other JCL statements, except the JCL comment statement. Each statement appears in the listing exactly as it appears in the procedure. |
| +/ | DD statement that was overridden (preceded by the overriding DD statement) |
| ++* | Job control statement that is not a JCL comment statement but one that the system considers to contain only comments |
| ++* | JCL comment statement |

# Chapter 7. Started tasks

This topic describes the decisions your system programmer needs to make in order for your installation to use started tasks, and the steps that users with operator authority will perform to use started tasks.

## Determining whether to use a started task

When you determine where and when you want specific JCL to run, you will consider using batch jobs or started tasks. Batch jobs are scheduled by a job entry subsystem (JES2 or JES3) and are scheduled to run based on the resources they require and their availability, or based on controls that you place on the batch system. Controlling where and when a batch job runs is more complex than using a started task.

A started task is a set of JCL that is run immediately as the result of a START command. Started tasks are generally used for critical applications. An advantage offered by started tasks are control over where and when the JCL is run. For example, you could have the JCL started at each IPL of the system.

For more information about system symbols and JCL symbols, see "Using symbols in started task JCL" on page 66.

## Determining the source JCL for the started task

If you decide to use a started task, you must then determine what the source JCL will be and where the JCL will be located. The source JCL can be a JOB (located in a member of a data set defined in the IEFJOBS or IEFPDSI concatenation of master JCL) or a procedure (located in a subsystem procedure library, for example, SYS1.PROCLIB). In the latter case, the system will process only the JCL associated with the first JOB statement in the procedure; it will bypass the second and subsequent jobs.

For information about master JCL considerations to support started tasks, see *z/OS MVS Initialization and Tuning Reference*.

Before determining whether you will use a job or a procedure as source JCL for a given started task, you need to understand the advantages of each. After you have identified whether the source JCL will be a job or a procedure, then determine the system services that the started task will require. (See "Determining system services for a started task" on page 63.)

In most cases, you will use a procedure unless you need greater control of your started task. For example, EREP formats the logrec data set information; you may not need to change the way this currently works. The best candidates for procedures are started tasks that require minimal maintenance.

The major advantage of using a job as the source JCL for a started task is the control provided over certain aspects of the started task, such as:

- Ability to specify accounting data

  For example, to determine which resources are being used by individual users.
- Ability to pass parameters to the started task

© Copyright IBM Corp. 1988, 2015

For example, using SYSIN data, you can pass data to programs in the started task.

- Control of output

  For example, many installations purge all output from started tasks because of the volume of output. With the output control allowed within a job, you can specify to receive output only if something abnormal occurs with the started task.

Started tasks are initiated by the START command which identifies the member that contains the source JCL for the task. (See *z/OS MVS System Commands* for information on the START command.) "START command processing when the member is a procedure" and "START command processing when the member is a job" describe how the system processes the START command (depending on whether the source JCL is a job or a procedure) and the JCL that results.

Note the following restriction: If you are running a started task you cannot override the PARM= parameter on the START command. However, you can circumvent this restriction as follows:

- Make the PARM= a symbolic in the EXEC statement where it is pertinent. Example:

  ```
  //JOB1   JOB   parameters
  //STEP1  EXEC  PGM=programname,PARM=&PARM1
  ```

- Then, on the START command, change the value of the symbolic. Example:

  ```
  START  JOB1,,PARM1=parameters
  ```

## START command processing when the member is a procedure

During START command processing, if the member specified does not start with a JOB statement, the system creates a JOB statement and EXEC statement that will invoke the procedure of the same name as the member.

For example, the member INIT exists in SYS1.PROCLIB as follows:

```
//IEFPROC   EXEC   PGM=IEFIIC
```

JES2 automatically issues the command `S INIT.INIT,,,JES2,SUB=JES2` and the following JCL is created:

```
//INIT     JOB MSGLEVEL=1
//INIT     EXEC INIT
```

## START command processing when the member is a job

If a JOB statement is the first statement in the member, the system uses the JCL provided in the member. For example, given the following JOB statement and JCL in the INIT member:

```
//INIT     JOB    'accounting_info',MSGLEVEL=1
//JESDS    OUTPUT JESDS=ALL,OUTDISP=(PURGE,WRITE)
//INIT     EXEC   INIT
//DD1      DD     DSN=SYSTEM.ACCOUNT.DATA,DISP=SHR
//*
```

JES2 automatically issues the command `S INIT.INIT,,,JES2,SUB=JES2` and the preceding JCL is invoked, starting the MVS initiator by calling the INIT procedure. The `S INIT.INIT,,,JES2,SUB=JES2` command now uses the source JCL and invokes the same procedure.

## Review current started tasks

Some of your existing started tasks may offer you greater benefits if the source JCL were a job. Review existing started tasks and identify the ones that should be a job by comparing their needs with the support provided (for example, output or accounting).

When you have identified that the source JCL will be a job, determine which method you will use to convert existing procedures, and determine whether the system services that the started task will require have changed. (See "Determining system services for a started task" on page 63.)

## Convert procedures to jobs (optional)

You may decide to convert some of your existing started task procedures to jobs. Before doing so, you should understand how the started task JCL and processing will change.

If the following command is issued for a started task procedure:

```
S DUMPCHK,SG=ALL,JDATE=93119,DAY=THURSDAY
```

and the procedure being started is:

```
//DUMPCHK  PROC SG=ALL,JDATE=,DAY=
//DUMPCHK  EXEC PGM=DMPCHKO,REGION=5M,PARM='/&SG,&JDATE,&DAY'
//STEPLIB  DD   DSN=JCR.PGM.LOAD,DISP=SHR
//CDS      DD   DSN=DATAMGT.CDS,DISP=SHR
//         DD   DSN=DATAMGT.CDS.CLEAR,DISP=SHR
//         DD   DSN=DATAMGT.CDS.Y43DUMPS,DISP=SHR
//LOG      DD   DSN=SYS1.TSODUMP.LOG,DISP=SHR
//SYSPRINT DD   SYSOUT=*
```

MVS creates the following JCL to invoke this procedure:

```
//DUMPCHK  JOB  MSGLEVEL=1
//STARTING EXEC DUMPCHK,SG=ALL,JDATE=93119,DAY=THURSDAY
```

To convert an existing procedure to a job, remove the PROC and PEND statements and add a JOB statement and any other JCL you plan to use.

To invoke as existing procedure, you can choose one of the following alternatives.

- "Alternative 1 - Add the member and JCL to the IEFJOBS-Defined data set"
- "Alternative 2 - Add the job JCL to the existing procedure" on page 62
- "Alternative 3 - Add the member and invoke a procedure in another DD concatenation" on page 62

**Note:** It is important to note that if system symbols are used on the PROC statement, they cannot be overridden by the START command system symbols.

### Alternative 1 - Add the member and JCL to the IEFJOBS-Defined data set

If you plan to define an IEFJOBS concatenation in MSTJCLxx with a data set of SYS1.STCJOBS, create a DUMPCHK member in SYS1.STCJOBS. Place the job in this member and add an EXEC statement that will run the existing procedure. For example:

```
//DUMPCHK  JOB  'accounting_info',MSGLEVEL=(1,1)
//         EXEC DUMPCHK
```

When the START command is issued, MVS inserts a JCL SET statement after the JOB statement, resulting in the following JCL:

```
//DUMPCHK  JOB  'accounting_info',MSGLEVEL=(1,1)
//         SET  SG=ALL,JDATE=93119,DAY=THURSDAY
//         EXEC DUMPCHK
```

### Alternative 2 - Add the job JCL to the existing procedure

If you do not plan to define an IEFJOBS concatenation in MSTJCLxx and the procedure DUMPCHK is already defined in SYS1.PROCLIB or one of the other data sets in the IEFPDSI concatenation of MSTJCLxx, use a JOB statement in the DUMPCHK member that formerly contained only the procedure, along with an EXEC statement that will run the existing procedure, and convert the existing procedure to an in-stream procedure by adding PROC and PEND statements, if they are not already present. For example:

```
//DUMPCHK  JOB  'accounting_info',MSGLEVEL=(1,1)
//DUMPCHK  PROC
//DUMPCHK  EXEC PGM=DMPCHKO,REGION=5M,PARM='/&SG,&JDATE,&DAY'
//STEPLIB  DD   DSN=JCR.PGM.LOAD,DISP=SHR
//CDS      DD   DSN=DATAMGT.CDS,DISP=SHR
//         DD   DSN=DATAMGT.CDS.CLEAR,DISP=SHR
//         DD   DSN=DATAMGT.CDS.Y43DUMPS,DISP=SHR
//LOG      DD   DSN=SYS1.TSODUMP.LOG,DISP=SHR
//SYSPRINT DD   SYSOUT=*
//         PEND
//         EXEC DUMPCHK
```

When the START command is issued, MVS inserts a JCL SET statement after the JOB statement, resulting in the following JCL:

```
//DUMPCHK  JOB  'accounting_info',MSGLEVEL=(1,1)
//         SET  SG=ALL,JDATE=93119,DAY=THURSDAY
//DUMPCHK  PROC
//DUMPCHK  EXEC PGM=DMPCHKO,REGION=5M,PARM='/&SG,&JDATE,&DAY'
//STEPLIB  DD   DSN=JCR.PGM.LOAD,DISP=SHR
//CDS      DD   DSN=DATAMGT.CDS,DISP=SHR
//         DD   DSN=DATAMGT.CDS.CLEAR,DISP=SHR
//         DD   DSN=DATAMGT.CDS.Y43DUMPS,DISP=SHR
//LOG      DD   DSN=SYS1.TSODUMP.LOG,DISP=SHR
//SYSPRINT DD   SYSOUT=*
//         PEND
//         EXEC DUMPCHK
```

**Note:** This alternative will not work for converting procedures to jobs when the job will run under the MASTER subsystem (SUB=MSTR either explicitly specified on the START command or defaulted to).

### Alternative 3 - Add the member and invoke a procedure in another DD concatenation

If you do not plan to define an IEFJOBS concatenation in MSTJCLxx and the procedure DUMPCHK is not in any of the data sets in the IEFPDSI concatenation of MSTJCLxx, create a member to contain the job and add that member to one of the data sets in the IEFPDSI concatenation and place the EXEC statement that will run the existing procedure (contained in the JES PROCLIB) in the JCL. The existing procedure will be invoked just as it was in the past. For example:

```
//DUMPCHK  JOB  'accounting_info',MSGLEVEL=(1,1)
//         EXEC DUMPCHK
```

When the START command is issued, MVS inserts a JCL SET statement after the JOB statement, resulting in the following JCL:

```
//DUMPCHK  JOB  'accounting_info',MSGLEVEL=(1,1)
//         SET  SG=ALL,JDATE=93119,DAY=THURSDAY
//         EXEC DUMPCHK
```

**Note:** This alternative will not work for converting procedures to jobs when the job will run under the MASTER subsystem (SUB=MSTR either explicitly specified on the START command or defaulted to).

# Determining system services for a started task

Before you begin to code the JCL for a started task, you should determine under which subsystem the JCL will run, and the changes the master JCL will require.

## Deciding under which subsystem a started task should run

To decide under which subsystem your started task should run, determine what services the task requires and what support the primary (job entry) subsystem, the master subsystem, and other subsystems provide.

Inform the system programmer responsible for the master JCL of your decision. Then code the name of the subsystem on the START command's SUB= keyword.

Without a SUB= keyword on the START command, the operating system will create the started task under the primary job entry subsystem (JES2 or JES3) unless the task itself is a subsystem, that is, it is either defined

- in the member IEFSSNxx of SYS1.PARMLIB, or
- dynamically by the SETSSI command or IEFSSI macro.

(A subsystem, unless requested to start under the primary JES subsystem by setting flag SSCTUPSS in the SSCVT, starts under the master subsystem, MSTR.)

A started task, regardless of the subsystem under which it runs, is demand-selected and runs in its own address space. Several considerations apply:

- The task can be a multi-step procedure or a job.
- It may not use operating system restart facilities. (The system does not support step restarts or checkpoint restarts for started tasks.)
- The JCL for the started task may contain the following statements:
  - COMMAND
  - ELSE
  - ENDIF
  - IF/THEN
  - INCLUDE
  - SET
- The system defines the system symbolic parameter &SYSUID. If the member name that is the target of the START command matches an entry in the started procedures table, &SYSUID contains the corresponding userid from that table. Otherwise, &SYSUID contains a null string. For information on the started procedures table, see *z/OS Security Server RACF System Programmer's Guide*.

## Running a started task under a job entry subsystem

These additional considerations apply to a started task running under a job entry subsystem (JES):

- The JCL for the started task may contain commands and JES2 JECL statements. It may not use JES3 JECL.
- In JES3, the JCL may contain a JCLLIB statement, and the started task may have a SYSIN data set, but these are permitted only if the JCL being started is a complete job. For example, the following will work in JES3:

```
//STC    JOB
//       JCLLIB ORDER=...
//STEP1  EXEC   PGM=...
//MYDATA DD     *
These are the times that try men's souls.
/*
//
```

- In JES2, the JCL may contain a JCLLIB statement, and the started task may have a SYSIN data set. For example, the following will work in JES2:

```
//HELLO     PROC
//STEPA     EXEC  PGM=IEBGENER
//SYSIN     DD    DUMMY
//SYSPRINT  DD    SYSOUT=A
//SYSUT2    DD    SYSOUT=A
//SYSUT1    DD    DATA
HELLO WORLD
/*
//          PEND
```

- The started task may have SYSOUT data sets.
- JES exits get control (to validate and modify the task).

## Running a started task under the master subsystem

These additional considerations apply to a started task that runs under the master subsystem:

- Any started task that can operate under the master subsystem can also run under the primary JES subsystem.
- A started task running under the master subsystem (SUB=MSTR) may choose to use JES services. To do so, the task must issue a Request Job ID call to the JES. (See *z/OS MVS Using the Subsystem Interface* for additional information about the Request Job ID call.)
- The started task may include JES statements and commands with the // COMMAND statement. Note, however, that if JES is not running, the system may queue or purge these statements.
- The JCL may *not* include a JCLLIB statement.
- The JCL may include PROC and PEND statements if the JCL is a procedure, but not if it is a job.
- The started task may not have SYSIN data sets.
- The system will initially allocate only data sets that are cataloged in the master catalog or a user catalog. Catalogs must reside on online volumes.
- You may dynamically allocate data sets that are not cataloged in the master catalog to a task running under the master subsystem during execution.
- You may dynamically allocate SYSOUT data sets after successfully completing a Request Job ID SSI call.
- JES exits do not get control during startup processing of a started task. If, however, the started task issues a Request Job ID SSI call, JES exits will get control for the minimal JCL used to construct the JES job structure.
- SMF exits, such as IEFUJV, get control with the subsystem shown as SYS.
- SRM determines performance characteristics based on the master subsystem.

- You may not specify JES3-managed devices in the procedure; JES3 cannot manage devices for tasks that run under the master subsystem.
- You must code a TIME= value on the EXEC statement of the procedure (such as TIME=NOLIMIT), or else specify the program as a system task in the program properties table (PPT). Otherwise, the task will end abnormally with a time-out condition.
- You must specify the region size that will be used if REGION is not specified in the JCL.

### Running a started task that uses catalogs

A catalog describes data set attributes and indicates the volumes on which a data set is located. Catalogs are allocated by the catalog address space (CAS), a system address space for the DFSMSdfp catalog function.

For a started task to use data sets cataloged in a catalog, either of the following must occur:

- You start the started task after the CAS is fully active, or
- The started task is one of the following:
  - Not a subsystem
  - A subsystem that is used to start another task
  - A subsystem that is started under the primary JES subsystem

If neither of those conditions is met and the task attempts to obtain catalog information, the system ends the started task abnormally. To avoid this potential abend, either specify unit and volume information in your JCL for each data set cataloged in a catalog, or catalog the data sets in the master catalog.

### Set Up the master JCL

Before adding or changing a started task, contact the system programmer who controls the master JCL. With that person, identify and define the data sets to which you will need access, and what you intend to change. For information on setting up the master JCL, the system programmer can see *z/OS MVS Initialization and Tuning Reference*.

## Coding the JCL

When you have determined what the started task source JCL will be, where it will run, and have set up the necessary support for it, you are ready to code the JCL for the started task.

This section explains how to:

- Name the PDS member that contains the JCL
- Code a JOB statement for a started task
- Use symbols in started task JCL.

### Naming the JCL member

The name specified on the START command is used to search for the JCL for the started task.

The system first searches the data sets specified in the IEFJOBS DD of the Master JCL, looking for a member with the specified name and which begins with a JOB statement. If one is found, that JCL is submitted. Any procedures within the JCL

are expanded using the data sets appropriate for the subsystem under which the job will be run (for example, SUB=MSTR, SUB=JES2 or SUB=JES3).

If a member with a JOB statement is not found in the IEFJOBS DD statement of the Master JCL, the system searches the data sets specified in the IEFPDSI DD of the Master JCL, looking for a member with the specified name and which begins with a JOB statement. If one is found, that JCL is submitted. Any procedures within the JCL are expanded using the data sets appropriate for the subsystem under which the job will be run (for example, SUB=MSTR, SUB=JES2 or SUB=JES3).

If a member with a JOB statement is not found in the IEFPDSI DD statement of the Master JCL, the system builds a JOB statement and searches the data sets specified in the procedure libraries appropriate for the subsystem under which the job will be run (for example, SUB=MSTR, SUB=JES2 or SUB=JES3), looking for a member with the specified name. If one is found, that JCL is appended to the JOB statement which was constructed and the JCL is submitted. Any procedures within the JCL are expanded using the data sets appropriate for the subsystem under which the job will be run (for example, SUB=MSTR, SUB=JES2 or SUB=JES3).

The following should be taken into consideration when naming members to be placed in the IEFJOBS data sets.

- Do not use the member name IEESYSAS. This name is reserved by the system for use in starting system address spaces. The IEESYSAS procedure is shipped in SYS1.PROCLIB.
- Be careful when using member names that are already in use in SYS1.PROCLIB and any other data sets specified in the IEFPDSI DD of the Master JCL. Doing so will cause the IEFJOBS data set member to override the existing JCL.

## Coding the JOB statement for the started task

If you choose to code a started task with a JOB statement, the rules are slightly different than the rules for other jobs:

- The statement must start with //
- The jobname is 1 through 8 non-blank characters
- If a name is not valid, a JCL error results.
- The jobname must be followed by at least 1 blank.
- JOB must follow the blank(s) after the jobname.
- JOB must be followed by at least 1 blank.

## Using symbols in started task JCL

You can code both system symbols and JCL symbols in started task JCL for both jobs and procedures. This section provides examples of how to code system symbols and JCL symbols in started task JCL. For details on how to code system symbols in JCL, and how to define and code JCL symbols in JCL, see “Using system symbols and JCL symbols” on page 38.

**Note:** You can also use system symbols in batch JCL.

### Example: using system symbols

Suppose you want to start a task whose source JCL is in the DUMPCHK member of a partitioned data set. You can specify system symbols for the task in one of the following two ways:

**On the START command:**

Suppose you enter the following command to start the DUMPCHK task:

```
START DUMPCHK,SG=ALL,JDATE=93119,DAY=THURSDAY,SUB=CICS&SYSNAME
```

If the substitution text for the &SYSNAME system symbol is SYS1 on the system that processes the START command, the system substitutes the text **SYS1** for the &SYSNAME system symbol. The equivalent source JCL is:

```
//DUMPCHK  JOB  MSGLEVEL=1
//STARTING EXEC DUMPCHK,SG=ALL,JDATE=93119,DAY=THURSDAY,SUB=CICSSYS1
```

**In the source JCL:**

You can also specify system symbols in the source JCL for started tasks. Keep in mind that system symbols in the source JCL are resolved during JCL processing, rather than command processing.

For example, suppose you code the following JCL in the DUMPCHK procedure:

```
//DUMPCHK  PROC
//S1       EXEC PGM=DUMPPROG,PARM=CICS&SYSNAME
```

As in the previous example for the START command, if the substitution text for the &SYSNAME system symbol is SYS1 on the system that processes the JCL, the system substitutes the text **SYS1** for the &SYSNAME system symbol. The equivalent JCL is:

```
//DUMPCHK  PROC
//S1       EXEC PGM=DUMPPROG,PARM=CICSSYS1
```

The DUMPCHK procedure can also include system symbols on other statements. For example, you might specify system symbols in DD statements that must specify data sets with unique names on different systems.

Suppose that two systems, named SYS1 and SYS2, are to process a DUMPCHK procedure that contains the following statement:

```
//LOG   DD  DSN=&SYSNAME..LOG,DISP=......
```

When each system processes the statement, the following data set names result:

```
SYS1.LOG on system SYS1
SYS2.LOG on system SYS2
```

You can include a substring of a system symbol on a JCL statement. For example, you might specify system symbols in DD statements that must specify data sets with unique names on different systems, but only have two characters to use. Suppose that two systems, named SYS1 and SYS2, are to process a procedure that contains the following statement:

```
//DD1 DD DSN=SYS1.PARMLIB.SYSTEM&SYSNAME(-2:2).,DISP=....
```

When each system processes the statement, the following data set names result:

```
SYS1.PARMLIB.SYSTEMS1 on system SYS1
SYS1.PARMLIB.SYSTEMS2 on system SYS2
```

## Example: using JCL symbols

Suppose that processing for some JCL is charged to multiple departments, all with different accounting numbers, and the JCL is to reflect the number of the department to be charged for the processing.

Code a symbol in the source JCL to represent the different account numbers:

```
ACCT=&ACCTNO
```

Assume that the source JCL is a started task named TEST. There are three departments (A, B, and C) with three accounting codes (ACODE, BCODE, and

CCODE) respectively. You can have each department indicate its accounting code on the START command. For example, when department A enters the following command:

```
START TEST,ACCTNO=ACODE
```

The system places the ACODE value in the ACCTNO field.

You can also use symbols to set default values that can later be overridden (as needed).

For example, if the procedure TEST has the following JCL coded:

```
ACCT=&ACCTNO
```

you can set the value of ACCT to ACODE by including the following JCL on the PROC statement of procedure TEST:

```
ACCTNO=ACODE
```

ACODE is provided as the default value.

If another value is provided on the START command (for example, START TEST, ACCT=BCODE), the new value (BCODE) overrides the default (ACODE) provided in the JCL, but only for this instance of the started task. If the START command is entered again without a value, the default will again be provided.

**Note:** This example modifies the step-level accounting data defined by the EXEC statement ACCT parameter. The START command JOBACCT parameter can be used to specify job-level accounting data.

## Using symbols on certain JCL statements

You might need to specify symbols within JCL for each invocation of a started task. Consider the following statements for possible use of symbols:

- DD statements
- EXEC statements.

If DD statement keywords (or the positional parameters for UNIT and VOL=SER) are specified on a START command, the following DD statement is added to the JCL processed by the system:

```
//IEFPROC.IEFRDER DD keyword=value...
```

The added JCL either adds a DD statement (if an IEFRDER statement is not specified in the source JCL) or modifies an existing IEFRDER DD statement in the source JCL. The DD statement override allows you to determine the characteristics for one DD statement when you issue the START command.

The DD statement keyword parameters can be any keyword that is valid on the MVS JCL DD statement. The IEFRDER DD statement contains all of the DD keywords specified on the START command. For example:

```
START  ABLE.LOAD,DSNAME=MY.LOADLIB,DISP=SHR
```

creates the following DD statement:

```
//IEFPROC.IEFRDER  DD  DSNAME=MY.LOADLIB,DISP=SHR
```

**Note:** If you are overriding a dataset name in the cataloged procedure and the name of the data set is 44 characters long, use DSN=*name.* If you specify DSNAME=*name*, the START procedure stops and returns a JCL error.

Also, DD statement keywords can be specified on the START command for positional parameters on the DD statement in the procedure. For example:

```
START  CICS.CICS,333,U30PAK
```

is the same as:

```
START  CICS.CICS,UNIT=333,VOL=SER=U30PAK
```

## Using JCL statement keywords and symbols to override JCL

You can use JCL statement keywords and symbols to override existing JCL.

With the exception of the keywords listed in Note 2 below, JOB statement keyword parameters include those keywords that are defined for the MVS JCL JOB statement. Such keywords add to or override the specification of the JOB statement keywords. EXEC statement keyword parameters include those keywords that are defined for the MVS JCL EXEC statement. The treatment of these keywords depends on whether the target of the START command is a job or a procedure; see the following table for more information. EXEC keywords that are also JOB keywords, such as TIME and REGION, are treated as JOB keywords.

JOB statement keyword parameters are those keywords defined for the MVS JCL JOB statement. These keywords will add to or override the specification of the JOB statement keywords. The EXEC statement keyword parameters are those keywords defined for the MVS JCL EXEC statement. The treatment of these keywords depends on whether the target of the START command is a job or a procedure. See the following table. EXEC keywords that are also JOB keywords, such as TIME and REGION, are treated as JOB keywords.

In this next example, assume ABC is a procedure, not a job. The following START command creates a REGION=200K parameter on the JOB statement and a DYNAMNBR=2 parameter on the EXEC statement:

```
  START  ABC.DEF,REGION=200K,DYNAMNBR=2
```

The result of the command is the following JCL:

```
//ABC     JOB REGION=200K,MSGLEVEL=1
//DEF     EXEC ABC,DYNAMNBR=2
```

You can use symbols to override other symbols that are specified in the procedure to be started. For example, the following command starts customer information control system (CICS®) with a 20K region:

```
START  CICS,A=20K
```

A=20K overrides A=10K on the following PROC statement:

```
//CICS  PROC  A=10K
//      EXEC  PGM=XYZ,REGION=&A
```

The command results in the following JCL:

```
//CICS     JOB MSGLEVEL=1
//STARTING EXEC  CICS,A=20K
```

**Note:** Select names for symbols carefully; see “Coding symbols in JCL” on page 43 for rules to use when coding and naming symbols.

The following table describes the actions that result from specifying various keywords and symbols on the START command:

| Source JCL | Keyword | Result |
|---|---|---|
| JOB | JOB (see Note 2) | Overrides or added to source JOB statement |
| JOB | EXEC | Placed on SET statement as a symbol |
| JOB | DD | Overrides, or added to, source IEFRDER DD statement |
| Procedure | Other (see Note 1) | Placed on SET statement as a symbol |
| Procedure | JOB | Overrides, or added to, source JOB statement |
| Procedure | EXEC | Placed on EXEC memname statement overriding keyword |
| Procedure | DD | Overrides, or added to, source IEFRDER DD statement |
| JOB | Other (see Note 1) | Placed on EXEC memname statement as symbol |

**Notes:**

- **Note 1:** *Other* does not include the START command reserved words SUB, JOBNAME, and JOBACCT.
- **Note 2:** The following keywords are not merged into the JOB statement, which allows their use as symbol names on the START command:
  - SYSAFF
  - SYSTEM

## Naming a started task (source JCL is a job)

If you plan to run the started task more than once on the same system or on different systems within a sysplex, consider using unique job names for each instance of the started task. For example, you may want to name started tasks according to the system tasks they support; you can name one set of jobs for CICS terminal-owning regions (CICSTOR1, CICSTOR2) and another set for CICS application-owning regions (CICSAOR1, CICSAOR2).

**Note:** You are not required to change the name of your started task; you probably will not want to change the name of a started task that typically has only one instance (OAM or LLA, for example).

There are four ways that you can name or identify a started task:

- JOBNAME parameter

  Use the JOBNAME parameter on the START command to rename the started task dynamically (see the description of START in *z/OS MVS System Commands* for details).
- Membername

  If you do not use the JOBNAME parameter on the START command and the source JCL is a procedure, the system automatically assigns the membername as the jobname.
- Source JCL

  If you do not use the JOBNAME parameter on the START command and the source JCL for the started task is a job, the jobname provided on the JOB statement is assigned as the jobname.
- Identifier

If specified on the START command, and the started task runs in a system address space that is created using common system address space procedure IEESYSAS, the identifier is assigned to the started task.

**Note:** Given the capability to assign the jobname dynamically, it is recommended that you use the JOBNAME parameter instead of the identifier. Only operators can view the identifier for a started task, limiting automation and identification by other users.

If you decide to change the names of started tasks, be sure to update other applications to recognize the new names.

## Setting up operator education for your started task

When you have set up the system support necessary and have coded the JCL, educate the system operators about any overrides you want them to use on the START command for your started task, and inform them of when they should use the overrides. Also, educate them on how to display information about your started task (using the DISPLAY command) as well as how to manage your started task (using the MODIFY, STOP, CANCEL, RESET, and FORCE commands).

# Chapter 8. JCL command statement

***Purpose:*** Use the JCL command statement to enter an MVS operator command through the input stream on a JES2 or JES3 system.

However, the COMMAND statement is the preferred way within the job control language to specify MVS and JES commands.

**Note:** To enter a JES2 command, use the JES2 command statement. To enter a JES3 command, use the JES3 command statement.Note also that the JCL Converter does not identify every input command that is not valid, but relies also on MVS command processing: see the following Example 3.

If an in-stream command is to be executed (see the explanation in the following section "Defaults" ), the system usually executes it as soon as it is read. Therefore, the command will **not** be synchronized with the execution of any job or step in the input stream. To synchronize a command with the job processing, tell the operator the commands you want entered and when they should be issued, and let the operator enter them from the console.

The system processes each command according to installation options for both the input device from which the job was read, and the job class.

***Considerations for an APPC Scheduling Environment:*** The command statement has no function in an APPC scheduling environment. If you code the command statement, the system will check it for syntax and ignore it.

***References:*** For more information on MVS commands and for descriptions of their parameters, see *z/OS MVS System Commands*.

## Description

### Syntax

```
//  command  [parameter] [comments]
```

The command statement consists of the characters // in columns 1 and 2 and three fields: operation (command), parameter, and comments.

Do not continue a command statement.

### Operation field

The operation field contains the MVS operator command and is coded as follows:

- Precede and follow the command with one or more blanks. It can begin in any column.
- Code the command or a valid abbreviation for the command.

### Parameter field

Code any required parameters. When more than one parameter is coded, separate them with commas.

© Copyright IBM Corp. 1988, 2015

## Comments field

The comments field follows the parameter field after at least one intervening blank. The system removes the comments field from the command before processing the command.

## Location in the JCL

A command statement can appear anywhere after a JOB statement and before the end of the job. If a command statement appears between jobs, it is ignored. A command statement should not be placed before the first JOB statement in an input stream.

If a command statement contains errors, it is not executed. If the erroneous statement is between two jobs in the input stream, the system does not issue a message to indicate that the command is not executed.

## Defaults

Two ways to control command authority are through JES initialization parameters and RACF. For information about controlling command authority through initialization parameters see, *Initialization and Tuning* for the appropriate subsystem at your installation. For information about controlling command authority using RACF see, *z/OS MVS Planning: Operations*.

## Examples of the command statement

***Example 1***

```
//   DISPLAY TS,LIST
```

In response to this command statement, the system displays the number and userid of all active time-sharing users of the system.

***Example 2***

```
// F NETVIEW,CLOSE IMMED
```

In response to this command statement, the system shuts down NETVIEW. The system considers IMMED to be a comment due to the delimiting blank.

***Example 3***

```
//  SETDANNO  ABCDEFG
```

MVS will fail this command because no such command exists: IEE305I SETDANNO COMMAND INVALID.

# Chapter 9. COMMAND statement

***Purpose:*** Use the COMMAND statement to specify a system or JES command that the system issues when the submitted JCL is converted.

The *COMMAND* statement is the preferred way within the job control language to specify commands, rather than using the *JCL command* statement, which is described in Chapter 8, "JCL command statement," on page 73. That is because the *COMMAND* statement is in standard JCL statement format, is parsed and processed using code common to the other JCL statements, and if necessary may be continued across multiple records, that is, is not limited to 80 characters. Note that some z/OS subsystems, including TSO, JES2, and JES3, offer additional ways to enter system commands outside JCL, which may be preferable under certain circumstances.

When the system encounters an in-stream command it issues message IEFC165I to inform the operator. If the operator is requested to authorize running of commands entered through the input stream, the system then issues message IEFC166D asking for the operator to respond. The operator should respond REPLY id,'Y' if the command displayed in message IEFC165I is to be run, and REPLY id,'N' otherwise.

Because the system usually executes an in-stream command as soon as it is converted, execution of the command will **not** be synchronized with the execution of any job or job step in the input stream. To synchronize a command with job processing, tell the operator the commands you want entered and when they should be issued, and let the operator enter them from the console.

The system processes each command according to installation options for both the input device from which the job was read, and the job class.

On a JES3 system, the system does not record in a job's JESMSGLG data set any commands you enter with the COMMAND statement.

***References:*** For more information on MVS and JES commands and for descriptions of their parameters, see *z/OS MVS System Commands*, *z/OS JES2 Commands*, and *z/OS JES3 Commands*.

***Considerations for an APPC Scheduling Environment:*** The COMMAND statement has no function in an APPC scheduling environment. If you code a COMMAND statement, the system will check it for syntax and then ignore it.

## Description

### Syntax

```
//[name]  COMMAND  'command command-operand'   [comments]
```

© Copyright IBM Corp. 1988, 2015

The COMMAND statement consists of the characters // in columns 1 and 2 and four fields: name, operation (COMMAND), 'command command-operand', and comments.

**Continuation onto Another Statement**: To continue a COMMAND statement, end the statement in column 71 and continue the statement in column 16 of the next statement. For example:

```
                                                                  (column 71)
                                                                      |
//      COMMAND  'START  XYZ,PARM=''ABC,DEF,GHI,JK'',TIME=1440,REGION=4
//             096K'
                |
          (column 16)
```

Do not code an apostrophe in column 71; see "Continuing parameter fields enclosed in apostrophes" on page 17 if you need more information.

## Name field

A name is optional on a COMMAND statement. If used, code it as follows:

- The name should be unique within the job.
- The name must begin in column 3.
- The name is 1 through 8 alphanumeric or national ($, #, @) characters.
- The first character must be alphabetic or national ($, #, @).
- The name must be followed by at least one blank.
- The name may be preceded by up to 8 alphanumeric or national characters, and then separated by a period. Coding the name in this way should not be confused with specifying an override, as can be done when coding DD statements.

If a name is not coded, column 3 must be blank.

## Operation field

The operation field consists of the characters COMMAND and must be preceded and followed by at least one blank. It can begin in any column.

## Parameter field

The parameter field specifies the name of the command, at least one blank, and then operands for the command. The command and its operands must be preceded by at least one blank, enclosed in apostrophes, and followed by at least one blank. The maximum length of the command is 123 characters. If the command operand contains an apostrophe, code it as two apostrophes. You can specify any MVS command that can be issued from the operator's console.

## Comments field

The comments field follows the parameter field after at least one intervening blank.

## Location in the JCL

A COMMAND statement can appear anywhere in the job after the JOB statement.

## Defaults

Two ways to control command authority are through RACF and through JES initialization parameters. For information about controlling command authority using RACF, see *z/OS MVS Planning: Operations*. For information about controlling

command authority through initialization parameters, see either *z/OS JES2 Initialization and Tuning Reference* or *z/OS JES3 Initialization and Tuning Reference*, as appropriate for the subsystem at your installation.

## Examples of the COMMAND statement

***Example 1:*** The following shows an example COMMAND statement with the START command.

```
//     COMMAND  'S VTAM'    start VTAM
```

***Example 2:*** The following is an example of a command that is continued with the command operand in apostrophes.

```
//       COMMAND  'SEND  ''This message will be sent to user SCOTTC
//             when this job is converted'',USER=(SCOTTC)'
```

The command statement must end in column 71 and be continued in column 16.

# Chapter 10. Comment statement

***Purpose:*** Use the comment statement to enter a comment on the output listing. The comment statement is used primarily to document a job and its resource requirements.

## Description

### Syntax

```
//*comments
```

The comment statement consists of the characters //* in columns 1, 2, and 3 and one field: comments.

Code the comments in columns 4 through 80. The comments field does not need to be preceded or followed by blanks. (In a JES3 system, do not use a JES3 keyword as the first word in column 4 of the comment field, or the comment might be taken for a JES3 statement.)

Do not continue a comment statement using continuation conventions. Instead, code additional comment statements.

### Location in the JCL

Place a comment statement anywhere after the JOB statement. You can place a comment statement between continuations of JCL statements.

### Listing of comments statements

Use the MSGLEVEL parameter on the JOB statement to request that the job log output listing contain all the JCL statements for your job.

See Table 14 on page 57 for the comment statement characters used in columns 1, 2, and 3.

### Examples of the comment statement

```
//* THE COMMENT STATEMENT CANNOT BE CONTINUED,
//* BUT IF YOU HAVE A LOT TO SAY, YOU CAN FOLLOW A
//* COMMENT STATEMENT WITH MORE COMMENT
//* STATEMENTS.
```

© Copyright IBM Corp. 1988, 2015

# Chapter 11. CNTL statement

***Purpose:*** Use the CNTL statement to mark the beginning of program control statements in the input stream. Program control statements specify control information for a subsystem. The program control statements are ended by an ENDCNTL statement and are called a CNTL/ENDCNTL group.

The DD statement that defines a data set to be processed by a subsystem must refer to the CNTL statement in order for the subsystem to use the program control statements in processing the data set.

***References:*** The program control statements are documented in the publications for the subsystems. For example, for information on program control statements for the Print Services Facility (PSF) see *PSF for z/OS: Customization*.

## Description

### Syntax

```
//label CNTL  [ *  comments]
```

The CNTL statement consists of the characters // in columns 1 and 2 and four fields: label, operation (CNTL), parameter (*), and comments. The * parameter is required only when comments follow.

### Label field

Code a label on every CNTL statement, as follows:

- The label must begin in column 3.
- The label is 1 through 8 alphanumeric or national ($, #, @)characters.
- The first character must be alphabetic or national ($, #, @)
- The label must be followed by at least one blank.
- The label may be preceded by up to 8 alphanumeric or national characters, and then separated by a period. Coding the label in this way should not be confused with specifying an override, as can be done when coding DD statements.

### Operation field

The operation field consists of the characters CNTL and must be preceded and followed by at least one blank. It can begin in any column.

### Parameter field

The parameter field contains only an asterisk. When present, the asterisk must be preceded and followed by at least one blank. The asterisk is required only when the statement contains comments.

### Comments field

The comments field follows the asterisk after at least one intervening blank.

© Copyright IBM Corp. 1988, 2015

## Location in the JCL

A CNTL statement must appear before the DD statement that refers to it. The CNTL and its referencing DD statement must be in the same job step or in the same cataloged or in-stream procedure step. A CNTL statement can be in a procedure and the referencing DD statement can be in the calling job step, but not vice versa.

You can define CNTL/ENDCNTL groups at the job level and the step level. A job-level CNTL/ENDCNTL group appears before the first EXEC statement of the job. A step-level CNTL/ENDCNTL group appears within the same job step or procedure step. If you code multiple step-level CNTL/ENDCNTL groups, the label on each CNTL statement must be unique within that step. Likewise, multiple job-level CNTL statements must also have unique labels. You can, however, use the same name on a step-level CNTL label and a job-level CNTL label. In this case, the step-level CNTL group overrides the job-level CNTL group.

## Program control statements

Program control statements supply control information for a subsystem. A subsystem can require one or more program control statements. The one or more statements must be immediately preceded by a CNTL statement and immediately followed by an ENDCNTL statement.

Do not code JCL statements within a program control group.

## Program control statements in procedures

You can code symbolic parameters on program control statements in a cataloged or in-stream procedure.

You can override parameters on program control statements in a procedure. Follow the rules used for overriding DD statement parameters in a procedure. For more information, see "Modifying OUTPUT JCL and DD statements" on page 30.

## Example of the CNTL statement

```
//STEP1     EXEC      PGM=PRINT
//ALPHA     CNTL      *  PROGRAM CONTROL STATEMENT FOLLOWS
//PRGCNTL   PRINTDEV  BUFNO=20,PIMSG=YES,DATACK=BLOCK
//OMEGA     ENDCNTL
//AGAR      DD        UNIT=AFP1,CNTL=*.ALPHA
```

The PSF subsystem uses the BUFNO, PIMSG, and DATACK options of the PRINTDEV control statement to print the data set for DD statement AGAR on an AFP printer. For information about the PRINTDEV statement, see *PSF for z/OS: Customization*.

# Chapter 12. DD statement

***Purpose:*** Use the DD (data definition) statement to describe a data set and to specify the input and output resources needed for the data set.

The parameters you can specify for data set definition are arranged alphabetically in the following pages.

***References:*** For information about the JES initialization parameters that provide installation defaults, see *z/OS JES2 Initialization and Tuning Reference* and *z/OS JES3 Initialization and Tuning Reference*.

## Description

### Syntax

```
// [ddname             ] DD [positional-parameter][,keyword-parameter]...[comments]
   [procstepname.ddname]

// [ddname             ] DD
   [procstepname.ddname]
```

- The DD statement consists of the characters // in columns 1 and 2 and four fields: name, operation (DD), parameter, and comments. Do not code comments if the parameter field is blank.
- A DD statement is required for each data set.
- The maximum number of DD statements per job step is 3273, based on the number of single DD statements allowed for a TIOT (task input output table) control block size of 64K. This limit can be different depending on the installation-defined TIOT size. The IBM-supplied default TIOT size is 32K. For information about changing the size of the TIOT, see the ALLOCxx parmlib member information in *z/OS MVS Initialization and Tuning Reference*. For information about how dynamic allocation might cause changes to the task input/output table (TIOT), see *z/OS MVS Programming: Authorized Assembler Services Guide*.

  In a JES3 system, the installation might further reduce the maximum number of DD statements per job.

### Name field

When specified, code a ddname as follows:

- Each ddname should be unique within the job step. If duplicate ddnames appear in a job step, processing is as follows:
  - **In a JES2 system**: The system performs device and space allocation and disposition processing for both DD statements; however, it directs all references to the first DD statement in the step.
  - **In a JES3 system**: If both DD statements request JES3 or jointly-managed devices, the system cancels the job during JES3 interpretation. If only one or neither DD statement requests a JES3 or jointly-managed device, the system performs device and space allocation processing for both DD statements; however, it directs all references to the first DD statement in the step.
- The ddname must begin in column 3.
- The ddname is 1 through 8 alphanumeric or national ($, #, @) characters.
- The first character must be alphabetic or national ($, #, @).
- The ddname must be followed by at least one blank.

© Copyright IBM Corp. 1988, 2015

***Omitting the ddname:*** Do not code a ddname when the DD statement defines a data set that is concatenated to the data set of the preceding DD statement. You can code PATH or DSNAME in either or both of the DD statements if your program uses BSAM, QSAM or BPAM.

**Note:** Prior to z/OS V1R7 you could define indexed sequential data sets using DSORG=IS and a DD statement without a name.

***Name field when overriding a procedure DD statement:*** Code the following in the name field of a DD statement that is to override a procedure DD statement:

1. The name of the procedure step that contains the DD statement to be overridden
2. Followed by a period
3. Followed by the ddname of the procedure DD statement that is to be overridden.

***Name field when adding a DD statement to a procedure:*** Code the following in the name field of a DD statement that is to be added to a procedure:

1. The name of the procedure step to which the DD statement is to be added
2. Followed by a period
3. Followed by a ddname of your choosing.

For example:

```
//PROCSTP1.DDA DD parameters
```

***Name field when adding a DD statement to a program:*** When you code a DD statement with a ddname of procstepname.ddname within a program step, the system:

1. Checks the syntax of both the procstepname qualifier and the ddname qualifier
2. Uses only the ddname qualifier as the statement ddname
3. Adds the DD statement to the program step that contains the statement
4. Issues an informational message because procstepname is coded outside of a procedure.

***Special ddnames:*** Use the following special ddnames only when you want to use the facilities these names represent to the system. These facilities are explained in Chapter 13, “Special DD statements,” on page 313.

```
JOBLIB
STEPLIB
SYSABEND
SYSCKEOV
SYSMDUMP
SYSUDUMP
```

Do not use the following ddnames. They are reserved for compatibility with the prior releases of MVS.

```
JOBAT
STEPCAT
```

Do not use the following ddnames on a DD statement in a JES2 system. They have special meaning to JES2.

```
JESJCLIN
JESJCL
JESMSGLG
JESYSMSG
```

The following ddnames have special meaning to JES3; do not use them on a DD statement in a JES3 system.

```
JCBIN
JCBLOCK
JCBTAB
JESJCLIN
JESInnnn
JESJCL
JESMSGLG
JOURNAL
JST
JESYSMSG
JES3CATLG
J3JBINFO
J3SCINFO
J3STINFO
STCINRDR
TSOINRDR
```

## Operation field

The operation field consists of the characters DD and must be preceded and followed by at least one blank. It can begin in any column.

## Parameter field

A DD statement has two kinds of parameters: positional and keyword. All parameters are optional.

Leave the parameter field blank only in the following cases:

- When SMS will provide the necessary DD description.
- When leaving a DD statement within a concatenation unchanged and overriding parameters on subsequent DD statements within that concatenation.

***Positional parameters:*** A DD statement can contain **one** positional parameter. If coded, this positional parameter must precede all keyword parameters.

*Table 15. Positional parameters*

| POSITIONAL PARAMETERS | VALUES | PURPOSE |
|---|---|---|
| `[ * ]`<br>`[DATA]`<br><br>See section "* Parameter" on page 101 or "DATA parameter" on page 129 | *: for data sets containing no JCL DATA: for data sets containing JCL | In a non-APPC scheduling environment, begins an in-stream data set. |
| `DUMMY`<br><br>See section "DUMMY parameter" on page 188 | | Specifies no space allocation, no disposition processing, and, for BSAM and QSAM, no I/O. |
| `DYNAM`<br><br>See section "DYNAM parameter" on page 191 | | (Parameter is supported to provide compatibility with previous systems.) |

***Keyword parameters:*** A DD statement can contain the following keyword parameters. You can code any of the keyword parameters in any order in the parameter field after a positional parameter, if coded.

Do not use DD statement keywords as symbolic parameters in procedures to be started by a START command from the operator console.

*Table 16. Keyword parameters*

| KEYWORD PARAMETERS | VALUES | PURPOSE |
|---|---|---|
| ACCODE=access-code<br><br>See section "ACCODE parameter" on page 104 | For ISO/ANSI/FIPS Version 3 tapes, access-code: 1 - 8 characters, first must be upper case A - Z.<br><br>For ISO/ANSI Version 4 tapes, access-code: 1 - 8 characters, first must be upper case A - Z, number 0 - 9, or one of these special characters: ! * " % & ' ( ) + , - . / : ; < = > ? _ | Specifies or changes an accessibility code for an ISO/ANSI/FIPS Version 3 or ISO/ANSI Version 4 tape output data set. |
| AMP=(subparameter)<br>AMP=('subparameter[,subparameter]...')<br><br>subparameters:<br><br>AMORG<br>BUFND=number<br>BUFNI=number<br>BUFSP=bytes<br>CROPS= {RCK}<br>{NCK}<br>{NRE}<br>{NRC}<br><br>FRLOG= {NONE}<br>{REDO}<br><br>OPTCD= {I }<br>{L }<br>{IL}<br><br>RECFM= {F }<br>{FB}<br>{V }<br>{VB}<br><br>STRNO=number<br>SYNAD=modulename<br>TRACE<br>ACCBIAS=[USER ]<br>[SYSTEM]<br>[DO ]<br>[DW ]<br>[SO ]<br>[SW ]<br><br>SMBDFR= {Y \| N}<br><br>SMBHWT= nn<br><br>SMBVSP= {nnK \| nnM}<br><br>MSG=SMBBIAS<br>RMODE31=[ALL ]<br>[BUFF]<br>[CB ]<br>[None]<br><br>See section "AMP parameter" on page 105 | see *z/OS DFSMS Using Data Sets* | Completes information in an access method control block (ACB) for a VSAM data set. |

*Table 16. Keyword parameters (continued)*

| KEYWORD PARAMETERS | VALUES | PURPOSE |
|---|---|---|
| With SMS only:<br>`AVGREC= {U}`<br>`{K}`<br>`{M}`<br><br>See section "AVGREC parameter" on page 113 | U: space specified in records<br>K: space specified in thousands of records<br>M: space specified in millions of records | Specifies a record request and the quantity of primary and secondary space specified on the SPACE parameter. |
| `BLKSIZE= {value}`<br>`{valueK}`<br>`{valueM}`<br>`{valueG}`<br><br>See section "BLKSIZE parameter" on page 114 | value: the specified maximum length, in bytes, of a block (The maximum is depending on the device type.)<br><br>valueK: the specified maximum length, in kilobytes, of a block (The maximum is 2097152K.)<br><br>valueM: the specified maximum length, in megabytes, of a block (The maximum is 2048M.)<br><br>valueG: the specified maximum length, in gigabytes, of a block (The maximum is 2G.) | Specifies the maximum length of a block. |
| `BLKSZLIM= {value}`<br>`{valueK}`<br>`{valueM}`<br>`{valueG}`<br><br>See section "BLKSZLIM parameter" on page 116 | value: 32,760 bytes - 2,147,483,648 bytes (two gigabytes)<br><br>valueK: 32K - 2,097,152K (two gigabytes)<br><br>valueM: 1M - 2048M (two gigabytes)<br><br>valueG: 1G - 2G (two gigabytes) | Specifies an upper limit on a data set's block size if BLKSIZE is omitted from all sources and the system determines the block size for the data set. |
| `BURST= {YES}`<br>`{Y }`<br>`{NO }`<br>`{N }`<br><br>See section "BURST parameter" on page 117 | YES or Y: burster-trimmer-stacker<br>NO or N: continuous forms stacker | Directs output to a stacker on a continuous-forms AFP printer. |
| CCSID=nnnnn<br><br>See section "CCSID parameter" on page 119 | nnnnn: 1 - 65535 | Specifies the coded character set identifier indicating the character code conversion performed on reads from and writes to tapes accessed in ISO/ANSI Version 4 format. |
| `CHARS= {table-name }`<br>`{(table-name[,table-name]...)}`<br>`{DUMP }`<br>`{(DUMP[,table-name]...) }`<br><br>See section "CHARS parameter" on page 121 | 1 - 4 table-name subparameters:<br>1 - 4 alphanumeric or $, #, @ characters<br><br>DUMP: 204-character print lines on 3800 model 1 | Names coded fonts for printing on an AFP printer. Requests a high-density dump on a SYSABEND or SYSUDUMP DD statement. |
| `CHKPT=EOV`<br><br>See section "CHKPT parameter" on page 123 | | Requests a checkpoint at each end-of-volume except the last. |
| `CNTL= {*.label }`<br>`{*.stepname.label }`<br>`{*.stepname.procstepname.label}`<br><br>See section "CNTL parameter" on page 125 | label: names CNTL statement<br>stepname: CNTL in named step<br>procstepname: step in named procedure | Causes the system to execute statements following an earlier CNTL statement. |
| `COPIES= {nnn }`<br>`{(nnn,(group-value[,group-value]...))}`<br>`{(,(group-value[,group-value]...)) }`<br><br>See section "COPIES parameter" on page 126 | nnn (JES2): 1 - 255<br>nnn (JES3): 0 - 255<br>1 - 8 group-values (JES2): 1 - 255<br>1 - 8 group values (JES3): 1 - 254 | Specifies number of copies printed. For an AFP printer, can instead specify number of copies of each page printed before the next page is printed. |

*Table 16. Keyword parameters (continued)*

| KEYWORD PARAMETERS | VALUES | PURPOSE |
|---|---|---|
| With SMS only:<br>`DATACLAS=data-class-name`<br><br>See section "DATACLAS parameter" on page 132 | data-class-name: installation-defined name of a data class | Specifies the data class for a new data set. |
| `DCB=(subparameter[,subparameter]...)`<br><br>`DCB= ( {dsname                          } )`<br>`     ( {*.ddname                        } )`<br>`     ( {*.stepname.ddname               } )`<br>`     ( {*.stepname.procstepname.ddname} )`<br>`     (  [,subparameter]...               )`<br><br>See section "DCB parameter" on page 135 | subparameter: see tables in DCB parameter description<br><br>*.ddname: copy DCB parameter from named cataloged data set<br><br>dsname: copy DCB information from named earlier DD statement<br><br>stepname: DD in named step<br><br>procstepname: step in named procedure | Completes information in data control block (DCB). |
| `DDNAME=ddname`<br><br>See section "DDNAME parameter" on page 150 | ddname: names later DD statement | Postpones defining the data set until later in same step: on a DD statement in the calling step or in a procedure called by the step. |
| `DEST=destination`<br><br>`destination (JES2):`<br>`LOCAL`<br>`name`<br>`Nnnnnn`<br>`NnRmmmmm to NnnnnnRm`<br>`(node,remote)`<br>`Rnnnnn or RMnnnnn or RMTnnnnn`<br>`Unnnnn`<br>`(node,userid)`<br><br>`destination (JES3):`<br>`ANYLOCAL`<br>`device-name`<br>`device-number`<br>`group-name`<br>`nodename`<br>`(node,userid)`<br><br>See section "DEST parameter" on page 155 | LOCAL or ANYLOCAL: local device<br><br>name: named local or remote device<br><br>Nnnnnn: node (1 - 32,767)<br><br>NnRmm: node (1 - 32,767) and remote work station (1 - 32,767); 6 digits maximum for n and m combined<br><br>Rnnnnn or RMnnnnn or RMTnnnnn: remote terminal (1 - 32,767)<br><br>Unnnnn: local terminal (1 - 32,767)<br><br>(node,userid): node (1 - 8 alphanumeric or $, #, @ characters) and TSO/E userid (1 - 7 alphanumeric or $, #, @ characters) or VM userid (1 - 8 alphanumeric or $, #, @ characters)<br><br>device-number: 3-digit or 4-digit hexadecimal number (/ required before 4-digit number)<br><br>device-name: local device (1 - 8 alphanumeric or $, #, @ characters)<br><br>group-name: 1 or more local devices or remote stations (1 - 8 alphanumeric or $, #, @ characters)<br><br>nodename: node (1 - 8 alpha- numeric or $, #, @ characters) | Sends a sysout data set to the specified destination. |

*Table 16. Keyword parameters (continued)*

| KEYWORD PARAMETERS | VALUES | PURPOSE |
|---|---|---|
| DISP=[status]<br>DISP=([status][,normal-termination-disp]<br>[,abnormal-termination-disp])<br><br>See section "DISP parameter" on page 159 | status: NEW, OLD, SHR (for shared), MOD (for data set to be modified)<br><br>normal-termination-disp: DELETE, KEEP, PASS, CATLG, or UNCATLG<br><br>abnormal-termination-disp: DELETE, KEEP, CATLG, or UNCATLG | Describes the status of the data set and tells the system to do the following with the data set after normal or abnormal termination of the step or job: delete or keep it on its volume(s), pass it to a later step, or add it to or remove it from the catalog. |
| DLM=delimiter<br><br>See section "DLM parameter" on page 172 | delimiter: 2 characters | In a non-APPC scheduling environment, terminates an in-stream data set. |
| DSID= {id }<br>{(id,[V])}<br><br>See section "DSID Parameter" on page 174 | id: 1 - 8 characters<br>V: label was verified (only on a SYSIN DD statement) | Identifies a data set on a diskette of a 3540 Diskette Input/Output Unit. |
| {DSNAME} = {dsname }<br>{DSN } {dsname(member-name) }<br>{dsname(generation-number) }<br>{&&dsname(member-name) }<br>{*.ddname }<br>{*.stepname.ddname }<br>{*.stepname.procstepname.ddname}<br>{NULLFILE }<br><br>See section "DSNAME parameter" on page 176 | unqualified dsname: 1 - 8 alphanumeric or $, #, @ characters, -, +0<br><br>qualified dsname: multiple names joined by periods<br><br>member-name: member in PDS or PDSE<br><br>generation-number: 0 or signed integer<br><br>stepname: DD in named step<br><br>procstepname: step in named procedure<br><br>NULLFILE: dummy data set | Names the data set. |
| DSNTYPE= {BASIC}<br>{LARGE}<br>{EXTREQ}<br>{EXTPREF}<br>{LIBRARY}<br>{(LIBRARY,1)}<br>{(LIBRARY,2)}<br>{HFS }<br>{PDS }<br>{PIPE }<br><br>See section "DSNTYPE parameter" on page 184 | BASIC: basic format data set<br>LARGE: large format data set<br>EXTREQ: extended format data set<br>EXTPREF: extended format data set (preferred)<br>LIBRARY: partitioned data set extended (PDSE), with optional version number<br>HFS: hierarchical file system (HFS) data set<br>PDS: partitioned data set<br>PIPE: FIFO special file | Specifies the type of data set. |
| EATTR=[OPT\|NO]<br><br>See section "EATTR parameter" on page 192 | OPT: Extended attributes are optional<br><br>NO: No extended attributes | Indicate whether the data set can support extended attributes (format 8 and 9 DSCBs) or not. |
| EXPDT= {yyddd }<br>{yyyy/ddd}<br><br>See section "EXPDT parameter" on page 193 | yyddd: expiration date (yy: 2-digit year, ddd: day 001-366)<br><br>yyyy/ddd: expiration date (yyyy: 4-digit year, ddd: day 001-366) | Specifies an expiration date for the data set. |
| FCB= {fcb-name }<br>{(fcb-name [,ALIGN ] ) }<br>[,VERIFY ]<br><br>See section "FCB parameter" on page 195 | fcb-name: 1 - 4 alphanumeric or $, #, @ characters<br>ALIGN: operator check forms alignment<br>VERIFY: operator verify FCB image | Specifies FCB image, carriage control tape for 1403 Printer, or data-protection image for 3525 Card Punch. |

*Table 16. Keyword parameters (continued)*

| KEYWORD PARAMETERS | VALUES | PURPOSE |
|---|---|---|
| FILEDATA= {BINARY}<br>{TEXT }<br>{RECORD}<br><br>See section "FILEDATA parameter" on page 198 | BINARY: byte-stream file<br>TEXT: delimited by the EBCDIC newline character<br>RECORD: the data consist of records with prefixes. | Specifies the content type of a z/OS UNIX file. |
| FLASH= {overlay-name }<br>{(overlay-name[,count])}<br>{NONE }<br><br>See section "FLASH parameter" on page 199 | overlay-name: forms overlay frame (1 - 4 alphanumeric or $, #, @ characters)<br>count: copies with overlay (0 - 255)<br>NONE: suppresses flashing | For printing on a 3800 Printing Subsystem, indicates that the data set is to be printed with the named forms overlay and can specify how many copies are to be flashed. |
| FREE= {END }<br>{CLOSE}<br><br>See section "FREE parameter" on page 201 | END: unallocate at end of last step<br>CLOSE: unallocate when data set is closed | Specifies when to unallocate the resources for this data set. |
| HOLD= {YES}<br>{Y }<br>{NO }<br>{N }<br><br>See section "HOLD parameter" on page 206 | YES or Y: holds this sysout data set<br><br>NO or N: allows normal processing for this sysout data set's output class | Tells the system to hold this sysout data set until released by the operator. |
| KEYLEN=bytes<br><br>See section "KEYLEN parameter" on page 213 | bytes: number of bytes (1-255 for key-sequenced (KS), 0-255 for sequential (PS) or partitioned (PO)) | Specifies the length of the keys in the data set. |
| With SMS only:<br>KEYOFF=offset-to-key<br><br>See section "KEYOFF parameter" on page 215 | offset-to-key: position of key (0 to difference of LRECL and KEYLEN minus 1) | Specifies the offset of the first byte of the record key. |
| LABEL= ([data-set-seq-no][,label-type] [,PASSWORD]) [,IN ] [,RETPD=nnnn ])<br>[,NOPWREAD]) [,OUT] [,EXPDT= {yyddd }]<br>[, ]) [, ] [ {yyyy/ddd}]<br><br>See section "LABEL parameter" on page 216 | | |

*Table 16. Keyword parameters (continued)*

| KEYWORD PARAMETERS | VALUES | PURPOSE |
|---|---|---|
| | data-set-seq-no: data set position on tape volume (1 - 4 decimal digits)<br><br>label-type: SL: IBM® standard labels SUL: IBM standard and user labels AL: ISO/ANSI Version 1 and ISO/ANSI/FIPS Version 3 labels AUL: user labels and ISO/ANSI Version 1 and ISO/ANSI/FIPS Version 3 labels NSL: nonstandard labels NL: no labels BLP: bypass label processing LTM: leading tapemark<br><br>PASSWORD: password required to access data set<br><br>NOPWREAD: password required to change or delete data set<br><br>IN: only read BSAM data set opened for INOUT or BDAM data set opened for UPDAT<br><br>OUT: only write to BSAM data set opened for OUTIN or OUTINX<br><br>RETPD=nnnn: retention period (nnnn: 1 - 4 decimal digits)<br><br>EXPDT=yyddd: expiration date (yy: 2-digit year, ddd: day 001 - 366)<br><br>EXPDT=yyyy/ddd: expiration date (yyyy: 4-digit year, ddd: day 001 - 366) | Specifies information about a data set's label, password, opening, expiration date, and, for a tape data set, relative position on the volume. |
| `LGSTREAM=name`<br><br>See section "LGSTREAM parameter" on page 222 | name: the name of the prefix that the system logger uses for the forward recovery log stream | Specifies the prefix of the name of the log stream for an SMS-managed VSAM data set. |
| With SMS only:<br>`LIKE=data-set-name`<br><br>See section "LIKE parameter" on page 224 | data-set-name: dsname of model data set | Specifies the attributes of a new data set. |
| `LRECL=(bytes)`<br><br>See section "LRECL parameter" on page 226 | bytes: length in bytes (1-32,760 for PS or PO, 1-32,761 for KS, ES, or RR) | Specifies the length of the records in the data set. |
| `MAXGENS=maximum-generations`<br><br>See section "MAXGENS parameter" on page 228 | maximum-generations: 0 - 2000000000 | Specifies the maximum number of generations for members in a Version 2 PDSE. |
| With SMS only:<br>`MGMTCLAS=data-class-name`<br><br>See section "MGMTCLAS parameter" on page 228 | data-class-name: installation- defined name of a data class | Specifies the management class for a new data set. |
| `MODIFY= {module-name }`<br>`{(module-name[,trc])}`<br><br>See section "MODIFY parameter" on page 230 | module-name: 1 - 4 alphanumeric or $, #, @ characters<br><br>trc: table-name in CHARS parameter (0 for first, 1 for second, 2 for third, and 3 for fourth table-name) | Specifies a copy-modification module in SYS1.IMAGELIB to be used by JES to print the data set on a 3800 Printing Subsystem. |
| `OUTLIM=number`<br><br>See section "OUTLIM parameter" on page 232 | number: 1 - 16777215 logical records maximum | Limits the logical records in this sysout data set. |

*Table 16. Keyword parameters (continued)*

| KEYWORD PARAMETERS | VALUES | PURPOSE |
|---|---|---|
| `OUTPUT= {reference }`<br>`{(reference[,reference]...)}`<br><br>reference:<br>• *.name<br>• *.stepname.name<br>• *.stepname.procstepname.name<br><br>See section "OUTPUT parameter" on page 233 | name: names earlier OUTPUT JCL statement<br>stepname: OUTPUT JCL in named step<br>procstepname: step in named procedure | Associates this sysout data set with one or more OUTPUT JCL statements. |
| `PATH=pathname`<br><br>See section "PATH parameter" on page 237 | pathname: pathname for a file | Specifies the name of a UNIX File. |
| `PATHDISP=(normal-termination-disposition,`<br>`abnormal-termination-disposition)`<br><br>See section "PATHDISP parameter" on page 240 | normal-termination-disposition: KEEP, DELETE<br>abnormal-termination-disposition: KEEP, DELETE | Tells the system to keep or delete the file after the job step ends. |
| `PATHMODE=file-access-attribute`<br>`PATHMODE=(file-access-attribute`<br>`[,file-access-attribute]...)`<br><br>See section "PATHMODE parameter" on page 242 | file-access-attribute for file owner class: SIRUSR, SIWUSR, SIXUSR, SIRWXU<br><br>file-access-attribute for file group class: SIRGRP, SIWGRP, SIXGRP, SIRWXG<br><br>file-access-attribute for file other class: SIROTH, SIWOTH, SIXOTH, SIRWXO<br><br>file-access-attribute to set process IDs: SISUID, SISGID | Specifies file access attributes when creating a UNIX File. |
| `PATHOPTS=file-option`<br>`PATHOPTS=(file-option[,file-option]...)`<br><br>See section "PATHOPTS parameter" on page 245 | file-option for access group: ORDONLY, OWRONLY, ORDWR<br><br>file-option for status group: OAPPEND, OCREAT, OEXCL, ONOCTTY, ONONBLOCK, OSYNC, OTRUNC | Specifies access and status for a file. |
| `PROTECT=YES`<br><br>See section "PROTECT parameter" on page 249 | | Requests that RACF create a discrete profile to protect a data set on direct access or a tape volume. |

*Table 16. Keyword parameters (continued)*

| KEYWORD PARAMETERS | VALUES | PURPOSE |
|---|---|---|
| Coding RECFM for BDAM Access Method:<br>`RECFM= {U }`<br>`{V }`<br>`{VS }`<br>`{VBS}`<br>`{F }`<br>`{FT }`<br><br>Coding RECFM for BPAM Access Method:<br>`RECFM= {U } [A]`<br>`{UT } [M]`<br>`{V }`<br>`{VB }`<br>`{VS }`<br>`{VT }`<br>`{VBS }`<br>`{VBT }`<br>`{VBST}`<br>`{F }`<br>`{FB }`<br>`{FT }`<br>`{FBT }`<br><br>Coding RECFM for BSAM, EXCP, and QSAM Access Methods:<br>`RECFM= {U } [A]`<br>`{UT } [M]`<br>`{F }`<br>`{FB }`<br>`{FS }`<br>`{FT }`<br>`{FBS }`<br>`{FBT }`<br>`{V }`<br>`{VB }`<br>`{VS }`<br>`{VT }`<br>`{VBS }`<br>`{VBT }`<br>`{VBST}`<br><br>For BSAM, EXCP, and QSAM using ISO/ANSI/FIPS data sets on tape:<br>`RECFM= {D } [A]`<br>`{DB }`<br>`{DS }`<br>`{DBS}`<br>`{U }`<br>`{F }`<br>`{FB }`<br><br>See section “RECFM parameter” on page 251 | Record format is:<br>F: fixed length<br>B: blocked<br>S: spanned<br>V: variable length<br>U: undefined length<br>T: track-overflow feature<br>D: variable-length ISO/ANSI tape records<br>Control characters are:<br>A: ISO/ANSI code<br>M: machine code | Specifies the format and characteristics of the records in a data set. |
| With SMS only:<br>`RECORG= {KS}`<br>`{ES}`<br>`{RR}`<br>`{LS}`<br><br>See section “RECORG parameter” on page 254 | Organization of records:<br>KS: key-sequenced<br>ES: entry-sequenced<br>RR: relative record<br>LS: linear space | Specifies the organization of the records in a VSAM data set. |
| With SMS only:<br>`REFDD= {*.ddname }`<br>`{*.stepname.ddname }`<br>`{*.stepname.procstepname.ddname}`<br><br>See section “REFDD parameter” on page 256 | Referenced DD statement:<br>ddname: unqualified name<br>stepname: qualified by step name<br>procstepname: step in procedure | Specifies the attributes of a new data set by referring to a previous DD statement. |

*Table 16. Keyword parameters (continued)*

| KEYWORD PARAMETERS | VALUES | PURPOSE |
|---|---|---|
| `RETPD=nnnn`<br><br>See section "RETPD parameter" on page 258 | nnnn: number of days (0-9999) | Specifies the retention period for a new data set. |
| `RLS= {NRI}`<br>`{CR }`<br>`{CRE }`<br><br>See section "RLS parameter" on page 260 | NRI: can read uncommitted changes CR: can read only committed changes CRE: ensures that records read by a unit of recovery are not changed by other units of recovery until the reading unit of recovery issues a syncpoint. | Specifies the record-level sharing protocol to be used with a VSAM data set. |
| With SMS only:<br>`SECMODEL=(profile-name[,GENERIC])`<br><br>See section "SECMODEL parameter" on page 262 | profile-name: name of model profile<br>GENERIC: model is generic profile | Specifies a RACF profile to be used for a new data set. |
| `SEGMENT=page-count`<br><br>See section "SEGMENT parameter" on page 263 | page-count: number of pages of a sysout data set | Specifies the number of pages produced for the current segment of the sysout data set before the data set is spun-off for output processing. (JES2 only) |
| For system assignment of space:<br>`SPACE=({TRK,    } (primary-qty [,second-qty] [,directory]) [,RLSE] [,CONTIG] [,ROUND] )`<br>`{CYL,    }              [,          ]                [,    ] [,MXIG  ]`<br>`{blklgth,}                                                  [,ALX   ]`<br>`{reclgth,}                                                  [,      ]`<br><br>To request specific tracks:<br>`SPACE=`<br>`(ABSTR, (primary-qty,address [,directory]))`<br><br>To request directory blocks (with SMS only):<br>`SPACE=(,(,,directory))`<br><br>See section "SPACE parameter" on page 264 | | |

*Table 16. Keyword parameters (continued)*

| KEYWORD PARAMETERS | VALUES | PURPOSE |
|---|---|---|
| | TRK: allocation in tracks<br><br>CYL: allocation in cylinders<br><br>blklgth: allocation in average blocks, 1 - 65535<br><br>reclgth: allocation in average records (SMS)<br><br>primary-qty: number of tracks, cylinders or blocks to be allocated<br><br>second-qty: additional tracks or cylinders to be allocated, if more are needed<br><br>directory: number of 256-byte records for PDS directory<br><br>RLSE: release unused space when data set is closed<br><br>CONTIG: contiguous primary allocation<br><br>MXIG: allocation in largest available space<br><br>ALX: allocation of up to 5 separate contiguous primary quantities<br><br>ROUND: allocation by block length rounded to integral cylinders<br><br>ABSTR: allocation at the specified address<br><br>address: track number of first track to be allocated | Requests space for a new data set on direct access storage. |
| `SPIN= {UNALLOC} (JES2 only)`<br>`{NO }`<br><br>See section "SPIN parameter" on page 273 | UNALLOC (JES2 only): the data set is available for printing immediately upon unallocation. UNALLOC is supported on JES2 only. NO: the data set is available for printing at the end of the job | Specifies that the output for a sysout data set is available for printing immediately upon unallocation or at the end of the job. |
| With SMS only:<br>`STORCLAS=storage-class-name`<br><br>See section "STORCLAS parameter" on page 275 | storage-class-name: installation- defined name of a storage class | Specifies the storage class for a new data set. |
| `SUBSYS= (subsystem-name )`<br>`([,subsystem-parameter]...)`<br><br>See section "SUBSYS parameter" on page 277 | subsystem-name: identifies the subsystem<br>subsystem-parameter: specifies information for the subsystem | Requests a subsystem to process this data set. |
| `SYSOUT=class`<br>`SYSOUT=([class] [,writer-name] [,form-name])`<br>`[,INTRDR ] [,code-name]`<br>`[, ]`<br>`SYSOUT=*`<br>`SYSOUT=(,)`<br><br>See section "SYSOUT parameter" on page 283 | class: A - Z, 0 - 9<br><br>writer-name: 1 - 8 alphanumeric or $, #, @ characters<br><br>form-name: 1 - 4 alphanumeric or $, #, @ characters<br><br>code-name: 1 - 4 alphanumeric or $, #, @ characters (JES2 only)<br><br>*: same output class as MSGCLASS parameter on JOB statement | Defines this data set as a sysout data set and (1) assigns it to an output class, (2) requests external writer to process it, (3) identifies print or punch forms, and (4) refers to the code-name of a JES2 /*OUTPUT statement. |

*Table 16. Keyword parameters (continued)*

| KEYWORD PARAMETERS | VALUES | PURPOSE |
|---|---|---|
| TERM=TS<br><br>See section "TERM parameter" on page 288 | | The TERM parameter has no function in an APPC scheduling environment. In a foreground job,indicates that this data set is coming from or going to a TSO/E userid. In a batch job, indicates that this DD statement begins an in-stream data set. |
| UCS= {character-set-code }<br>{(character-set-code [,FOLD] [,VERIFY])}<br>{ [, ] }<br><br>See section "UCS parameter" on page 290 | character-set-code: 1 - 4 alpha- numeric or $, #, @ characters<br><br>FOLD: operator load chain or train in fold mode<br><br>VERIFY: operator verify UCS image | Specifies universal character set, print train, or font for an AFP printer. |
| UNIT= ([ddd ] [,unit-count][,DEFER])<br>[/ddd ] [,P ]<br>[/dddd ] [, ]<br>[device-type ]<br>[group-name ]<br><br>UNIT=AFF=ddname<br><br>See section "UNIT parameter" on page 293 | device-number: 3-digit or 4-digit hexadecimal number (/ required before 4-digit number and optional before 3-digit number)<br><br>device-type: machine type and model<br><br>group-name: 1 - 8 alphanumeric or $, #, @ characters<br><br>unit-count: 1 - 59<br><br>P: allocate same number of devices as volumes for parallel mount<br><br>DEFER: defers mounting until open<br><br>SMSHONOR: indicates that the system should honor the device number or group-name for a SMS-managed tape library request<br><br>AFF=ddname: requests allocation of same devices as for DD statement ddname | Requests allocation to a specific device, a type or group of devices, or the same device(s) as another data set. Also can specify how many devices and deferred mounting. |
| {VOLUME}=([PRIVATE] [,RETAIN] [,volume-seq-no][,volume-count][,][SER=(serial-number[,serial-number]...)])<br>{VOL } [, ] [, ] [REF=dsname ]<br>[REF=*.ddname ]<br>[REF=*.stepname.ddname ]<br>[REF=*.stepname.procstepname.ddname ]<br>[REF=*.procstepname.ddname ]<br><br>See section "VOLUME parameter" on page 300 | | |

*Table 16. Keyword parameters (continued)*

| KEYWORD PARAMETERS | VALUES | PURPOSE |
|---|---|---|
| | PRIVATE: requests a private volume<br><br>RETAIN: requests private tape volume remain mounted and unwound or requests public tape volume be retained at device<br><br>volume-seq-no: begins processing with volume 1 - 255 of existing multivolume data set<br><br>volume-count: maximum volumes for output data set (1 - 255)<br><br>serial-number subparameters (1 - 255): volume serial numbers (1 - 6 alphanumeric, $, #, @, or special characters)<br><br>REF: copy volume serial numbers from another data set or earlier DD statement, or copy storage class for SMS-managed data sets<br><br>dsname: from cataloged or passed data set<br><br>ddname: from named earlier DD statement<br><br>stepname: DD in named step<br><br>procstepname: step in named procedure | Identifies the volume(s) on which a data set resides or will reside. |

## Comments field

The comments field follows the parameter field after at least one intervening blank. If you do not code any parameters on a DD statement, do not code any comments.

## Location in the JCL

Most DD statements define data sets to be used in a job step, in a cataloged procedure step, or in an in-stream procedure step; these appear after the EXEC statement for the step. Some DD statements define data sets for the job, for example, the JOBLIB DD statement; these appear after the JOB statement and before the first EXEC statement.

***When overriding or adding to procedures:*** Place DD statements that override, nullify, or add parameters immediately following the EXEC statement that calls the procedure. Place overriding and nullifying DD statements first, followed by all added DD statements. Last in the calling step are any DD * or DD DATA statements with their in-stream data.

To override more than one DD statement in a procedure, place the overriding DD statements in the same order as the overridden DD statements in the procedure.

### Concatenating data sets

When data sets are concatenated, the application program can treat them as if they were one logical data set. In general, most of the logical attributes of the first DD statement apply to all of them. You can **concatenate** input data sets for the duration of a job step. Each of the concatenated data sets can reside on a different

volume. For details on concatenating data sets, see *z/OS DFSMS Using Data Sets*. Note that you cannot concatenate output data sets.

## Two types of concatenation

**Partitioned concatenation:** This is any combination of the following:

- Partitioned data set (PDS).
- Partitioned data set extended (PDSE).
- z/OS UNIX directory. Code the PATH keyword.

In general, do not code a member name in these cases. If you code a member name on the first DD statement, it causes the system to position to that member first unless the application program overrides it. The application program uses BPAM to read.

**Sequential concatenation:** There are two types of sequential concatenation:

- Like. The same logical record length (LRECL) value and record format apply to all the data sets. The block size (BLKSIZE) values and device characteristics might differ.
- Unlike. Any of data set characteristics might differ. The application program must have logic to support this capability.

The DD statements are for any combination of sequential disk or tape data sets, members of PDSs and PDSEs, spooled input stream ("sysin") data sets, z/OS UNIX files, a TSO terminal (TERM=TS) and unit record (such as virtual card reader) devices. The application program uses BSAM, QSAM or EXCP to read.

## Coding a concatenation

To concatenate data sets, omit the ddnames from all the DD statements except the first in the sequence. The data sets are processed in the same sequence as the DD statements defining them.

## Devices for concatenated data sets

Concatenated data sets can reside on different devices and different types of devices. (This may require internal DCB modifications, see *z/OS DFSMS Using Data Sets*.)

## Block sizes for concatenated data sets

Concatenated data sets can have different block sizes. In a few cases, the data set with the largest block size must appear first in the concatenation. (Note that you can state a value equal to the largest block size for BLKSIZE on the first DD statement, regardless of what the actual block size of this data set is.) Certain data sets can be concatenated in any order of block size; these are:

- Partitioned data sets (PDSs), and partitioned data sets extended (PDSEs) without member names coded on the DD statements.
- Sequential data sets that are DASD-resident, tape-resident, or in-stream, and are accessed by QSAM and use system-created buffers.
- Sequential data sets that are DASD-resident or in-stream, and are accessed by BSAM.

For these data sets, the BLKSIZE obtained is the largest in the concatenation. Note that this block size can cause invalid attribute combinations when combined with the attributes obtained from the first data set in the concatenation.

If you do not specify a block size, the system can, under certain conditions, determine an optimum block size. For detailed information about system-determined block size, see *z/OS DFSMS Using Data Sets*.

## Logical record lengths for concatenated data sets

Concatenated data sets with format-V records can have different logical record lengths as long as the data set with the largest logical record length appears first in the concatenation. (Note that you can state a value equal to the largest logical record length for LRECL on the first DD statement, regardless of what the actual logical record length of this data set is.)

## References to concatenated data sets

If you make a **backward reference** to a concatenation (using *.), the system obtains information only from the first data set defined in the sequence of DD statements.

If you make a **forward reference** to a concatenation (using the DDNAME parameter), the forward reference resolves to the first data set in the concatenation. If there are no DD statements between the forward reference and the concatenation, the rest of the data sets in the concatenation are appended to the first data set in the concatenation. The following example illustrates this.

```
//STEP1      EXEC PGM=IEBGENER
//SYSPRINT   DD   SYSOUT=*
//SYSUT1     DD   DDNAME=INPUT
//INPUT      DD   DSN=TSTDATA1,DISP=SHR
//           DD   DSN=TSTDATA2,DISP=SHR
//SYSUT2     DD   SYSOUT=*
//SYSIN      DD   DUMMY
```

In this example, SYSUT1 will resolve to the first data set, TSTDATA1, defined by the DDNAME forward reference INPUT. TSTDATA2, the second data set in the DDNAME forward reference INPUT, will be appended to SYSUT1 as well. IEBGENER will recognize TSTDATA1 and TSTDATA2 as input.

If there are any DD statements between the forward reference and the concatenation, the rest of the data sets in the concatenation are appended to the last DD statement preceding the concatenation. For example:

```
//STEP1      EXEC PGM=IEBGENER
//SYSUT1     DD   DDNAME=INPUT
//SYSPRINT   DD   SYSOUT=*
//SYSUT2     DD   SYSOUT=*
//INPUT      DD   DSN=TSTDATA1,DISP=SHR
//           DD   DSN=TSTDATA2,DISP=SHR
//SYSIN      DD   DUMMY
```

In the preceding example, SYSUT1 will resolve to the first data set, TSTDATA1, defined in the DDNAME forward reference INPUT. TSTDATA2 will be appended to SYSUT2, the last DD statement preceding the concatenation. In this example, IEBGENER will recognize only TSTDATA1 as input.

If a concatenated DD is added to a procedure, the remaining concatenated data sets will be concatenated to the last DD in the step named in an override or addition (or to the first step if no step was named in an override or addition). Note that this may result in these concatenated DDs being added to an unexpected DD. The following example illustrates this.

```
//TPROC    PROC
//S1         EXEC PGM=IEFBR14
//DD1        DD   DDNAME=INPUT
//DD2        DD   DSN=MYDSN2,DISP=SHR
```

```
//DD3         DD   DSN=MYDSN3,DISP=SHR
//S2          EXEC PGM=IEFBR14
//DDA         DD   DDNAME=INPUT
//DDB         DD   DSN=MINE2,DISP=SHR
//DDC         DD   DSN=MINE3,DISP=SHR
//           PEND
//STEP1       EXEC TPROC
//INPUT       DD   DSN=MYDSN1,DISP=SHR
//           DD   DSN=MYDSN4,DISP=SHR
//S2.INPUT    DD   DSN=MINE1,DISP=SHR
//           DD   DSN=MINE4,DISP=SHR
```

In this example, the result of the DDNAME forward reference INPUT is:

- In step S1, DD1 resolves to data set MYDSN1 and data set MYDSN4 is concatenated to data set MYDSN3.
- In step S2, DDA resolves to data set MINE1 and data set MINE4 is concatenated to data set MINE3.

### Do not concatenate data sets after a DUMMY data set

If you define a data set using the **DUMMY parameter**, do not concatenate other data sets after it. When the processing program asks to read a dummy data set, the system takes an end-of-data set exit immediately and ignores any data set that might be concatenated after the dummy.

### Do not code other statements between concatenated DD statements

Do not code other types of statements between two or more concatenated data definition (DD) statements. (Comments are the only exception; you can code them between DD statements.) For example, do not code a SET statement as follows:

```
//DD1    DD  DSN=A
//       DD  DSN=B
//       SET  ...
//*      Wrong!!! SET statement not allowed (this comment IS allowed)
//       DD  DSN=C
```

## Examples of DD statements and ddnames

***Example 1***

```
//MYDS   DD  DSNAME=REPORT
//A      DD  DSNAME=FILE
```

***Example 2***

```
//INPUT  DD  DSNAME=FGLIB,DISP=(OLD,PASS)
//       DD  DSNAME=GROUP2,DISP=SHR
```

In this example, because the ddname is missing from the second DD statement, the system concatenates the data sets defined in these statements.

***Example 3***

```
//PAYROLL.DAY DD  DSNAME=DESK,DISP=SHR
```

In this example, if procedure step PAYROLL contains a DD statement named DAY, this statement overrides parameters on DD statement DAY. If the step does not contain DD statement DAY, the system adds this statement to procedure step PAYROLL for the duration of the job step.

***Example 4***

```
//STEPSIX.DD4 DD  DSNAME=TEXT,DISP=(NEW,PASS)
//            DD  DSNAME=ART,DISP=SHR
```

In this example, the second data set is concatenated to the first, and both are added to procedure step STEPSIX. The ddname is omitted from the second DD statement in order to concatenate data set ART to data set TEXT.

Because the system does not allow you to write to a concatenation of data, you need another data set with DISP=OLD in order to read from TEXT. Write to the new DD name before reading from DD4.

## * Parameter

***Parameter Type***

Positional, optional

***Purpose***

Use the * (asterisk) parameter to begin an in-stream data set. The data records immediately follow the DD * statement; the records may be in any code such as EBCDIC. The data records end when one of the following is found:

- /* in the input stream
- // to indicate another JCL statement
- The two-character delimiter specified by a DLM parameter on this DD statement
- The input stream runs out of records

Use a DATA parameter instead of the * parameter if any of the data records start with //.

***Considerations for an APPC Scheduling Environment***

The * parameter has no function in an APPC scheduling environment. If you code *, the system will check it for syntax and otherwise ignore it.

### Syntax

```
//ddname DD *[,parameter]...  [comments]
```

### Defaults

When you do not code BLKSIZE and LRECL, JES uses installation defaults specified at initialization.

**Note:** If the input stream is from NJE (network job entry), JES uses the size specified at the sending node.

### Relationship to other parameters

You can specify the following DD parameters with the DD * and DD DATA parameters. All other parameters are either ignored or result in a JCL error.

```
DCB=BLKSIZE DCB=BUFNO DCB=DIAGNS DCB=LRECL DLM DSID
LIKE RECL REFDD DCB=MODE=C DSNAME VOLUME=SER
```

***Restrictions when coding LRECL:*** If you code LRECL with the * parameter, you cannot submit a data set to JES3 with a record length greater than 80 bytes.

You cannot use the TSO/E SUBMIT command to submit a data set to JES2 or JES3 with a record length greater than 80 bytes.

**Note:** In-stream data within a cataloged procedure is also limited to 80 bytes.

You can submit a data set to JES2 or JES3 with a record length greater than 80 bytes by submitting the following JCL:

```
//SUBMIT   JOB   ...
//S1       EXEC  PGM=IEBGENER
//SYSPRINT DD    SYSOUT=*
//SYSIN    DD    DUMMY
//SYSUT2   DD    SYSOUT=(,INTRDR)
//SYSUT1   DD    DSN=IBMUSER.LONGDATA.JCL,DISP=SHR
```

In this example, IBMUSER.LONGDATA.JCL contains the data with a record length greater than 80 bytes.

In a JES3 system, the record length limit is the size of the installation-defined spool buffer, minus 46. (For example, if the buffer size is 4084, the record length limit is 4038.) JES3 fails any job that exceeds this limit.

If the records longer than 80 bytes include JCL to be transmitted to a remote system using JES3 `// XMIT` or `//*ROUTE XEQ`, or JES2 `/*ROUTE XEQ` or `/*XMIT` with JES3 in the network, the records are truncated to 80 bytes.

***For JES3 SNA RJP input:***

- The only parameters you can specify for JES3 systems network architecture (SNA) remote job processing (RJP) input devices are BLKSIZE and LRECL.
- Code DCB=LRECL=nnn, where nnn is 1 to 255 when SYSIN data records are greater than 80 bytes. (The default LRECL is 80 bytes.)

***For 3540 diskette input/output units:*** VOLUME=SER, BUFNO, and DSID on a DD * statement are ignored except when they are detected by a diskette reader as a request for an associated data set. See *3540 Programmer's Reference*. On a DD * or DD DATA statement processed by a diskette reader, you can specify DSID and VOLUME=SER parameters to indicate that a diskette data set is to be merged into the input stream following the DD statement.

## Relationship to other control statements

Do not refer to an earlier DD * statement in DCB, DSNAME, or VOLUME parameters on following DD statements.

## Location in the JCL

A DD * statement begins an in-stream data set.

***In-stream Data for Cataloged or In-stream Procedures:*** A cataloged procedure can contain a DD * statement.

An in-stream procedure can contain a DD * statement.

When you call an in-stream procedure, you can add input stream data to an in-stream procedure step by placing one or more DD * or DD DATA statements in the calling step. You can alternatively include in-stream data directly within the in-stream or cataloged procedure.

***Multiple In-Stream Data Sets for a Step:*** You can code more than one DD * or DD DATA statement in a job step in order to include several distinct groups of data for the application program. Precede each group with a DD * or DD DATA statement and follow each group with a delimiter statement.

***Omitted Data Delimiters:*** If you omit a DD statement before the input data, the system provides a DD * statement with the ddname of SYSIN. If you omit a delimiter statement after input data, the system ends the data when it reads a JCL statement or runs out of records.

## Unread records

If the processing program does not read all the data in an in-stream data set, the system skips the remaining data without abnormally terminating the step.

## Examples of the * parameter

***Example 1***

```
//INPUT1   DD  *
           .
           .
           data
           .
//INPUT2   DD  *
           .
           .
           data
           .
   /*
```

This example defines two groups of data in the input stream.

***Example 2***

```
//INPUT3   DD  *,DSNAME=&&INP3
           .
           data
           .
/*
```

This example defines an in-stream data set with INP3 as the last qualifier of the system-generated data set name. A name such as userid.jobname.jobid.Ddsnumber.INP3 is generated.

***Example 3***

```
//STEP2         EXEC PROC=FRESH
//SETUP.WORK    DD   UNIT=3400-6,LABEL=(,NSL)
//SETUP.INPUT1  DD   *
                .
                .
                data
                .
/*
//PRINT.FRM     DD   UNIT=180
//PRINT.INP     DD   *
                .
                .
                data
                .
/*
```

This example defines two groups of data in the input stream. The input data defined by DD statement SETUP.INPUT1 is to be used by the cataloged procedure step named SETUP. The input data defined by DD statement PRINT.INP is to be used by the cataloged procedure step named PRINT.

# ACCODE parameter

***Parameter Type***

Keyword, optional

***Purpose***

Use the ACCODE parameter to specify or change an accessibility code for an ISO/ANSI/FIPS Version 3 or ISO/ANSI Version 4 tape output data set. An installation-written file-access exit routine verifies the code after the code is written to tape. If the code is authorized, the job step's program can use the data set; if not, the system issues messages and may abnormally terminate the job step.

A data set protected by an accessibility code should reside only on a volume protected by RACF or a volume accessibility code. The volume should not contain any unprotected data sets.

**Note:** ACCODE is supported only for ISO/ANSI/FIPS Version 3 and ISO/ANSI Version 4 tape data sets. ACCODE is ignored for all label types except AL and AUL label tapes.

***References***

For more information on ISO/ANSI/FIPS Version 3 and ISO/ANSI Version 4 tape data sets, see *z/OS DFSMS Using Magnetic Tapes*. Also *z/OS DFSMS Access Method Services Commands*.

## Syntax

```
ACCODE=access-code
```

## Subparameter definition

**access-code**

Specifies an accessibility code. The access code is 1 through 8 characters. In ISO/ANSI/FIPS Version 3 the first character must be an upper case letter from A through Z. In ISO/ANSI Version 4 the first character must be an upper case letter from A to Z, number from 0 to 9, or one of the special characters ! * " % ' ( ) + , - . / : ; < = > ? and _ .

Enclose the ACCODE in apostrophes if you specify special characters. For example, ACCODE='AB/CD'. Specify two apostrophes if you include an apostrophe as a special character. For example, to specify DAY'SEND, use ACCODE='DAY''SEND'.

**Note:** ISO/ANSI/FIPS Version 3 and ISO/ANSI Version 4 use only the first character as the accessibility code; the installation can use the other seven characters. If the first character is other than those allowed, the installation does not give control to the file-access exit routine.

## Defaults

If you do not specify an accessibility code on a DD statement that defines an ISO/ANSI/FIPS Version 3 or ISO/ANSI Version 4 tape data set, the system writes an ASCII blank character (X'20') in the tape label. A blank authorizes unlimited access to the tape's data sets unless access is limited by RACF data set protection.

If the installation does not supply a file-access exit routine, the system prevents access to any ISO/ANSI/FIPS Version 3 or ISO/ANSI Version 4 tape volume.

## Overrides

If PASSWORD or NOPWREAD is coded on the DD statement LABEL parameter, password access overrides the ACCODE parameter.

## Example of the ACCODE parameter

```
//TAPE   DD  UNIT=3390,VOLUME=SER=T49850,DSNAME=TAPEDS,
//            LABEL=(,AL),ACCODE=Z
```

In this example, the DD statement ACCODE parameter specifies an accessibility code of Z for tape volume T49850. The volume has ISO/ANSI/FIPS Version 3 or ISO/ANSI Version 4 labels. The data set TAPEDS is first on the tape.

# AMP parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the AMP parameter to complete information for a VSAM data set.

AMP is supported only for VSAM data sets.

**Note:** With SMS, you can create new VSAM data sets with JCL DD statements. See the DATACLAS parameter and the RECORG parameter.

***References***

For more information about VSAM data sets, see *z/OS DFSMS Using Data Sets*, *z/OS DFSMS Macro Instructions for Data Sets*, and *z/OS MVS JCL User's Guide*.

## Syntax

```
AMP=(subparameter)

AMP=('subparameter[,subparameter]...')

AMP='subparameter[,subparameter]...'

The subparameters are:

      AMORG

      BUFND=number

      BUFNI=number
```

```
BUFSP=number

CROPS= [NCK]
       [NRC]
       [NRE]
       [RCK]

FRLOG= {NONE}
       {REDO}

OPTCD= {I }
       {L }
       {IL}

RECFM= [F ]
       [FB]
       [V ]
       [VB]

STRNO=number

SYNAD=module

TRACE=(subparameter[,subparameter]...)

ACCBIAS=[USER  ]
        [SYSTEM]
        [DO    ]
        [DW    ]
        [SO    ]
        [SW    ]

SMBDFR= {Y | N}

SMBHWT= nn

SMBVSP= {nnK | nnM}

MSG=SMBBIAS
RMODE31=[ALL ]
        [BUFF]
        [CB  ]
        [None]
```

**Parentheses**: Parentheses are required only when you are continuing the statement.

**Multiple subparameters**: When a parameter contains more than one subparameter, separate the subparameters by commas and enclose the subparameter list in apostrophes inside the parentheses. For example, AMP=('AMORG,STRNO=4').

**Null Positional subparameters**: Null positions in the AMP parameter are invalid.

**Special characters**: When a parameter contains only one subparameter and that subparameter contains special characters, enclose the subparameter in apostrophes inside the parentheses. For example, AMP=('STRNO=4').

**Note:** Do not enclose a subparameter in a subparameter list in apostrophes.

If you code a symbolic parameter on the AMP parameter, you can code the symbolic parameter in apostrophes.

**Continuation onto another statement**: Enclose the subparameter list in only one set of parentheses. Enclose all the subparameters on each statement in apostrophes. End each statement with a comma after a complete subparameter. For example:

```
//DS1  DD  DSNAME=VSAMDATA,AMP=('BUFSP=200,OPTCD=IL,RECFM=FB',
//      'STRNO=6')
```

## Subparameter definition

**AMORG**

Indicates that the DD statement describes a VSAM data set. Code AMORG when data set access is through an ISAM interface program and the DD statement contains VOLUME and UNIT parameters.

It is unnecessary to code AMP=AMORG for a data set that is SMS-managed. An SMS data set is cataloged at allocation; all information pertaining to the data set creation (such as RECORG) must be fully defined at allocation to ensure the success of the job.

**BUFND=number**

Specifies the number of I/O buffers that VSAM is to use for data records. The minimum is 1 plus the STRNO subparameter number. This value overrides the BUFND value specified in the ACB or GENCB macro, or provides a value if one is not specified. If you omit STRNO, BUFND must be at least 2.

If you omit BUFND from AMP and from the ACB macro instruction, the system uses the STRNO number plus 1.

**BUFNI=number**

Specifies the number of I/O buffers that VSAM is to use for index records. This value overrides the BUFNI value specified in the ACB or GENCB macro, or provides a value if one is not specified. If you omit BUFNI from AMP and from the ACB macro instruction, VSAM uses as many index buffers as the STRNO subparameter number; if you omit both BUFNI and STRNO, VSAM uses 1 index buffer.

If data access is through the ISAM interface program, specify for the BUFNI number 1 more than the STRNO number, or specify 2 if you omit STRNO, to simulate having the highest level of an ISAM index resident. Specify a BUFNI number 2 or more greater than the STRNO number to simulate having intermediate levels of the index resident.

**BUFSP=number**

Specifies the maximum number of bytes for the data and index buffers in the user area. This value overrides the BUFSP value specified in the ACB or GENCB macro, or provides a value if one is not specified.

If BUFSP specifies fewer bytes than the BUFFERSPACE parameter of the access method services DEFINE command, the BUFFERSPACE number overrides the BUFSP number.

**CROPS=NCK**
**CROPS=NRC**
**CROPS=NRE**
**CROPS=RCK**

Requests a checkpoint/restart option. For more information, see *z/OS DFSMSdfp Checkpoint/Restart*.

**NCK**

Requests no data set post-checkpoint modification tests.

**NRC**
Requests neither a data-erase test nor data set post-checkpoint modification tests.

**NRE**
Requests no data-erase test.

**RCK**
Requests a data-erase test and data set post-checkpoint modification tests. If the CROPS subparameter is omitted, RCK is the default.

If you request an inappropriate option, such as the data-erase test for an input data set, the system ignores the option.

**FRLOG=NONE**
**FRLOG=REDO**

Specifies if VSAM batch logging will be performed for your VSAM data set.

**NONE**
Disables the VSAM batch logging function for your VSAM data set. Changes made by applications will not be written to the MVS log stream indicated on the LOGSTREAMID parameter.

**REDO**
Enables the VSAM batch logging function for you VSAM data set. Changes made by applications will be written to the MVS log stream indicated on the LOGSTREAMID parameter.

**Note:**

1. If FRLOG=REDO is specified, the LOGSTREAMID parameter must be specified for the VSAM data set(s). If LOGSTREAMID is not specified, IEC161I is issued.
2. There is no default JCL value for FRLOG. If FRLOG is omitted, the catalog value will be used.

**OPTCD=I**
**OPTCD=L**
**OPTCD=IL**

Indicates how the ISAM interface program is to process records that the step's processing program flags for deletion.

**I** Requests, when the data control block (DCB) contains OPTCD=L, that the ISAM interface program is not to write into the data set records marked for deletion by the processing program.

If AMP=('OPTCD=I') is specified without OPTCD=L in the DCB, the system ignores deletion flags on records.

**L** Requests that the ISAM interface program is to keep in the data set records marked for deletion by the processing program.

If records marked for deletion are to be kept but OPTCD=L is not in the DCB, AMP=('OPTCD=L') is required.

**Note:** This parameter has the same meaning and restrictions for the ISAM interface as it has for ISAM. While it was not required in the ISAM job control language, you should code it in the AMP parameter.

**IL** Requests that the ISAM interface program is not to write into the data set

records marked for deletion by the processing program. If the processing program had read the record for update, the ISAM interface program deletes the record from the data set.

AMP=('OPTCD=IL') has the same effect as AMP=('OPTCD=I') coded with OPTCD=L in the DCB.

**RECFM=F**
**RECFM=FB**
**RECFM=V**
**RECFM=VB**

For data sets with SMS, see the DD RECFM parameter described in .

Identifies the ISAM record format used by the processing program. You must code this RECFM subparameter when the record format is not specified in the DCB.

**Note:** This parameter has the same meaning and restrictions for the ISAM interface as it has for ISAM. While it was not required in the ISAM job control language, you should code it in the AMP parameter.

All VSAM requests are for unblocked records. If the processing program requests blocked records, the ISAM interface program sets the overflow-record indicator for each record to indicate that each is being passed to the program unblocked.

**F** Indicates fixed-length records.

**FB** Indicates blocked fixed-length records.

**V** Indicates variable-length records. If no RECFM is specified in the AMP parameter or in the DCB, V is the default.

**VB** Indicates blocked variable-length records.

**STRNO=number**

Indicates the number of request parameter lists the processing program uses concurrently. The number must at least equal the number of BISAM and QISAM requests that the program can issue concurrently. If the program creates subtasks, add together the number of requests for each subtask plus 1 for each subtask that sequentially processes the data set. This value overrides the STRNO value specified in the ACB or GENCB macro, or provides a value if one is not specified.

**SYNAD=module**

Names a SYNAD exit routine. The ISAM interface program is to load and exit to this routine if a physical or logical error occurs when the processing program is gaining access to the data set.

The SYNAD parameter overrides a SYNAD exit routine specified in the EXLST or GENCB macro instruction that generates the exit list. The address of the intended exit list is specified in the access method control block that links this DD statement to the processing program. If no SYNAD exit is specified, the system ignores the AMP SYNAD parameter.

**TRACE=(subparameter[,subparameter]...)**

Indicates that the generalized trace facility (GTF) executes with your job to gather information about the opening, closing, and end-of-volume processing for the data set defined on this DD statement. You can use the interactive problem control system to print the trace output; see *z/OS MVS IPCS User's Guide*.

The TRACE subparameters are: HOOK, ECODE, KEY, PARM1, and PARM2. See *z/OS DFSMS Using Data Sets* for full information on the TRACE subparameter and the VSAM trace facility, which you use to obtain diagnostic information during VSAM processing.

**ACCBIAS=USER**
**ACCBIAS=SYSTEM**
**ACCBIAS=DO**
**ACCBIAS=DW**
**ACCBIAS=SO**
**ACCBIAS=SW**

Specify one of these six values to override record access bias in the data class in order to use System-Managed Buffering (SMB) without changing the data class. See *z/OS DFSMS Using Data Sets* for details on System-Managed Buffering.

**USER**
Obtain buffers the same way the system would without SMB. This is the default if you code no specification for the ACCBIAS subparameter.

**SYSTEM**
Force SMB and let the system determine the buffering technique based on the ACB MACRF and storage class specification.

**Note:** USER and SYSTEM are the only values you may use to specify record access bias in the data class.

**DO** SMB with direct optimization.

**DW** SMB weighted for direct processing.

This option provides the capability to use hiperspace.

**SO** SMB with sequential optimization.

**SW** SMB weighted for sequential processing.

**SMBDFR=Y** *or* **SMBDFR=N**
With direct optimization, use this subparameter to instruct VSAM whether to defer writing of changed buffers to the medium until either the data set is closed or the buffers are required for some other request. See *z/OS DFSMS Using Data Sets* for further details on using SMBDFR.

**SMBHWT=nn**
Specify a requirement for hiperspace where nn is an integer from 0 to 99. Use this parameter with direct optimization. The default value is 0, which means that the system does not obtain any hiperspace.

**SMBVSP=nnK** *or* **SMBVSP=nnM**
Specify the amount of virtual buffer space to acquire for direct optimized processing when opening the data set, where nn is 1 to 2048000 kilobytes or 1 to 2048 megabytes.

**MSG=SMBBIAS**
When you specify MSG = SMBBIAS, the system issues message IEC161I to indicate which access bias SMB has chosen. If SYSTEM is specified in ACCBIAS or Record Access Bias, then the system will choose an access bias for SMB, otherwise it will use what the user specified in ACCBIAS. The possible values in the message are DO, DW, SO, SW, CO, CR or ??, where '??' means OPEN could not determine which one of the six bias values was used to create the initial control block structure. The default is no message.

**RMODE31=ALL**
**RMODE31=BUFF**
**RMODE31=CB**
**RMODE31=NONE**

Designate the residency for buffers and control blocks.

This subparameter allows you to specify whether or not to allocate the buffers and control blocks in 31-bit addressable storage. You can use this field independently of SMB. With SMB the default location is in 31-bit addressable storage ("above the 16-megabyte line"). Without SMB, the default is in 24-bit addressable storage ("below the line").

The values you may specify for RMODE31 are:

**ALL** —Control blocks and buffers above the line.

**BUFF** —Buffers (only) above the line.

**CB** —Control blocks (only) above the line.

**NONE**
—Control blocks and buffers below the line.

When you do not specify ACCBIAS, or when you specify ACCBIAS=USER, if you specify nothing for RMODE31 in either the JCL or the ACB, the system obtains the buffers and control blocks in virtual storage with a 24-bit address.

When ACCBIAS=SYSTEM, if you specify nothing for RMODE31 in either the JCL or the ACB, the system obtains the buffers in storage with an address greater than 16 million bytes.

When you specify CB or NONE for RMODE31, the system obtains the buffers in 24-bit addressable storage.

When you specify BUFF or NONE for RMODE31, the system obtains the control blocks in 24-bit addressable storage.

If your program runs in 24-bit mode and you use locate mode processing for the VSAM data set, you must obtain the buffers and control blocks in 24-bit addressable storage.

**Note:** If your program runs with local or global shared resources (LSR/GSR) and uses journaling (JRNAD) or user processing (UPAD) exit routines, the exits must run in 31-bit mode if you obtained the control blocks above the line.

This capability to allocate above the line is necessary when either or both of the following conditions exists:

- The number of data sets open to a job is quite large.
- The number of buffers is such as to cause a storage shortage if kept in 24-bit addressable storage.

You may specify RMODE31 only with the JCL DD AMP parameter or in the ACB. The RMODE31 subparameter of AMP overrides any RMODE31 values specified in the ACB.

The RMODE31 subparameter is available for all data set types.

## Relationship to other parameters

Do not code the following parameters with the AMP parameter.

*
BURST

CHARS
COPIES
DATA
DCB

DDNAME
DYNAM
FCB
FLASH
FREE
MODIFY

QNAME
RECFM
SUBSYS
SYSOUT
TERM
UCS

***Invalid ddnames:*** The following ddnames are invalid for VSAM data sets:

JOBLIB

STEPLIB

SYSABEND

SYSCHK

SYSCKEOV

SYSMDUMP

SYSUDUMP

***Invalid DSNAMEs:*** When you code the AMP parameter, the DSNAME must not contain parentheses, a minus (hyphen), or a plus (+) sign. The forms of DSNAME valid for ISAM, partitioned access method (PAM), and generation data groups (GDG) are invalid with VSAM data sets.

## Buffer requirements

For a key-sequenced data set, the total minimum buffer requirement is three: two data buffers and one index buffer. For an entry-sequenced data set, two data buffers are required.

If the number of buffers specified in the BUFND and BUFNI subparameters causes the virtual storage requirements to exceed the BUFSP space, the number of buffers is reduced to fit in the BUFSP space.

If BUFSP specifies more space than required by BUFND and BUFNI, the number of buffers is increased to fill the BUFSP space.

## Examples of the AMP parameter

***Example 1***

```
//VSAMDS1  DD  DSNAME=DSM.CLASS,DISP=SHR,AMP=('BUFSP=200,BUFND=2',
//           'BUFNI=3,STRNO=4,SYNAD=ERROR')
```

In this example, the DD statement defines the size of the user area for data and index buffers, specifies the number of data and index buffers, specifies the number of requests that require concurrent data set positioning, and specifies an error exit routine named ERROR.

***Example 2***

```
//VSAMDS2  DD  DSNAME=DSM.CLASS,DISP=SHR,AMP=('BUFSP=23456,BUFND=5',
//           'BUFNI=10,STRNO=6,SYNAD=ERROR2,CROPS=NCK',
//           'TRACE=(PARM1=F00203000010,KEY=ABCDEF)')
```

In this example, the DD statement defines the values for BUFSP, BUFNI, STRNO, and SYNAD, as in the previous example. It also specifies that a data set post-checkpoint modification test is not to be performed when restarting at a checkpoint and that GTF is to provide a trace of specified data areas.

## AVGREC parameter

***Parameter type***

Keyword, optional — use this parameter only with SMS

***Purpose***

Use the AVGREC parameter when you define a new data set to specify that:

- The units of allocation requested for storage space are records.
- The primary and secondary space quantity specified on the SPACE parameter represents units, thousands, or millions of records.

When you use AVGREC with the SPACE parameter, the first subparameter (reclgth) on the SPACE parameter must specify the average record length of the records.

Code the AVGREC parameter when you want to (1) specify records as the units of allocation or (2) override the space allocation defined in the data class for the data set.

If SMS is not installed or is not active, the system checks the syntax and then otherwise ignores the AVGREC parameter.

### Syntax

```
AVGREC= {U}
        {K}
        {M}
```

### Subparameter definition

**U** Specifies a record request and that the primary and secondary space quantity specified on the SPACE parameter represents the number of records in units (multiplier of 1).

**K** Specifies a record request and that the primary and secondary space quantity specified on the SPACE parameter represents the number of records in thousands (multiplier of 1024).

**M** Specifies a record request and that the primary and secondary space quantity specified on the SPACE parameter represents the number of records in millions (multiplier of 1048576).

## Overrides

AVGREC overrides the space allocation defined in the DATACLAS parameter for the data set. See "Overrides" on page 134 section under the Dataclass keyword.

## Relationship to other parameters

Do not code AVGREC with the TRK, CYL, or ABSTR subparameters of the SPACE parameter.

Do not code the following DD parameters with the AVGREC parameter.

*

DATA

DDNAME

DYNAM

QNAME

## Examples of the AVGREC parameter

***Example 1***

```
//SMSDS3  DD  DSNAME=MYDS3.PGM,DATACLAS=DCLAS03,DISP=(NEW,KEEP),
//          SPACE=(128,(5,2)),AVGREC=K
```

In the example, the space allocation defined in the DCLAS03 data class is overridden by the SPACE and AVGREC parameters, which indicate an average record length of 128 bytes, a primary quantity of 5K (5,120) records, and a secondary quantity of 2K (2,048) records.

***Example 2***

```
//SMSDS3A DD  DSNAME=MYDS3.PGM,DATACLAS=DCLAS03A,DISP=(NEW,KEEP),
//          AVGREC=K
```

In the example, the space allocation defined in the DCLAS03A data class is overridden by the AVGREC parameter, which indicates that the primary and secondary quantity represents thousands of records.

# BLKSIZE parameter

***Parameter type***

Keyword, optional

***Purpose***

Code the BLKSIZE parameter to specify the maximum length of a block.

## Syntax

```
BLKSIZE= {value}
         {valueK}
         {valueM}
         {valueG}
```

## Subparameter definition

**value**
Specifies the maximum length, in bytes, of a block.

The number of bytes that you specify for BLKSIZE depends on the device type, the record format for the data set and other programs that will read or write the data set. The maximum value that you can code is 2,147,483,648 (coded without the commas) or 2G. When your program uses the data set, the OPEN function may impose a smaller limit. The maximum allowed by OPEN is:

- 2,147,483,648 for dummy data sets. Note that you cannot actually get a buffer that large.
- 2,147,483,647 for tape that does not have ISO/ANSI Version 3 or Version 4 labels. When writing with BSAM and QSAM, the system imposes a limit that depends on the tape subsystem model. Currently the maximum for certain models is 256K. See *z/OS DFSMS Using Data Sets* for more details about block size limits for specific models. OPEN allows EXCP programs to have higher limits that depend on the hardware.
- 2048 for tape that has ISO/ANSI Version 3 labels, where the minimum value for BLKSIZE is 18 bytes. To allow a block size greater than 2048, use installation exit routine IFG0193G, described in *z/OS DFSMS Installation Exits*.Version 4 labels do not have this restriction.
- 32,760 for DASD, ISO/ANSI Version 4 tape labels, and other data sets.

**valueK**
Specifies the maximum length, in kilobytes, of a block. (1 kilobyte = 1024 bytes.) The maximum is 2097152. If you code 2097152K, the block size is the maximum: 2,147,483,648 bytes.

**valueM**
Specifies the maximum length, in megabytes, of a block. (1 megabyte = 1024 kilobytes.) The maximum is 2048. If you code 2048M, the block size is the maximum: 2,147,483,648 bytes.

**valueG**
Specifies the maximum length, in gigabytes, of a block. (1 gigabyte = 1024 megabytes.) The maximum is 2G. If you code 2G, the block size assigned is the maximum: 2,147,483,648 bytes.

## Defaults

If you do not code BLKSIZE, the system can, under certain conditions, determine an optimum block size. For detailed information about system-determined block size, see *z/OS DFSMS Using Data Sets*.

## Overrides

If you code a non-zero value for the BLKSIZE subparameter on a DCB or DCBE macro instruction or on a DD statement that defines an existing data set with standard labels, the DCB or DCBE BLKSIZE overrides the block size specified in the label.

## Relationship to other control statements

Do not code the BLKSIZE parameter with the DCB subparameter BUFSIZE.

If you code BLKSIZE it will have no effect on EXCP processing unless the application takes special steps to use it. (For information about EXCP processing see *z/OS DFSMSdfp Advanced Services*.)

## Coexistence considerations

Not all programs and operating systems prior to z/OS can read blocks longer than 32,760 bytes. For example, Version 2 Release 10 is the first release of OS/390 that can read such long blocks using standard access methods.

## Examples of the BLKSIZE parameter

```
//DD1B  DD    DSNAME=EVER,DISP=(NEW,KEEP),UNIT=3380,
//            RECFM=FB,LRECL=326,BLKSIZE=23472,
//            SPACE=(23472,(200,40))
```

DD statement DD1B defines a new data set named EVER on a 3380. The DD keywords RECFM, LRECL, and BLKSIZE contain the information necessary to complete the data control block.

```
//DD2B  DD    DSNAME=NEVER,DISP=(NEW,KEEP),UNIT=3590,
//            RECFM=FB,LRECL=256,BLKSIZE=204K
```

DD statement DD2B defines a new data set named NEVER on a 3590. The DD keywords RECFM, LRECL, and BLKSIZE contain the information necessary to complete the data control block. The block size, which in this example is 204 x 1024 = 208,896 bytes, must be divisible by the logical record length, and each program that reads or writes this data set must be capable of handling block sizes this large.

# BLKSZLIM parameter

Keyword, optional

***Purpose***

Use the BLKSZLIM parameter to specify an upper limit on a data set's block size if BLKSIZE is omitted from all sources and the system determines the block size for the data set. If a BLKSIZE value is available from any source (such as the DD statement, data set label, or the program), then the block size limit has no effect. The BLKSZLIM parameter is useful mainly when writing new magnetic tape data sets with programs that can handle blocks longer than 32,760 bytes. Currently the maximum block size supported on any tape is 256 KB. You can safely code a larger value for BLKSZLIM. The BLKSZLIM value does not have to be a multiple of the LRECL value. For more information, see *z/OS DFSMS Using Data Sets*.

## Syntax

```
BLKSZLIM= {value}
          {valueK}
          {valueM}
          {valueG}
```

## Subparameter definition

**value**
Specifies in bytes an upper limit on a data sets's block size if BLKSIZE is omitted from all sources and the system determines the block size for the data set. The maximum value is 2,147,483,648 bytes (two gigabytes). The minimum value is 32K (32,768 bytes). The minimum value is 32K (32,760 bytes).

**valueK**
Specifies the block size limit in kilobytes (units of 1024). The maximum value is 2,097,152K (two gigabytes). The minimum value is 32K.

**valueM**
Specifies the block size limit in megabytes (units of 1024K). The maximum value is 2048M (two gigabytes). The minimum value is 1M.

**valueG**
Specifies the block size limit in gigabytes (units of 1024M). The maximum allowable value is 2G (two gigabytes). The minimum value is 1G.

## Defaults

If you omit BLKSZLIM, the system determines the block size from one of the following sources, starting with the first:

1. Data class
2. DEVSUPxx value
3. 32,768

## Relationship to other parameters

The system ignores BLKSZLIM when you specify BLKSIZE.

## Example of the BLKSZLIM parameter

```
//DD1BB DD DSNAME=EVER,DISP=(NEW,KEEP),UNIT=3390,
//         RECFM=FB,LRECL=326,BLKSZLIM=32760,
//         SPACE=(23472,(200,40))
```

DD statement DD1B defines a new data set named EVER on a 3390 DASD. The DD keywords RECFM and LRECL contain the information necessary to complete the data control block. BLKSZLIM places an upper limit on the block size to be determined by the system.

```
//DD2B  DD DSNAME=NEVER,DISP=(NEW,KEEP),UNIT=3590,
//         RECFM=FB,LRECL=80,BLKSZLIM=40K
```

DD statement DD2B defines a new data set named NEVER on a 3590 TAPE device. The DD keywords RECFM and LRECL contain the information necessary to complete the data control block. BLKSZLIM places an upper limit on the block size to be determined by the system.

# BURST parameter

Keyword, optional

***Purpose***

Use the BURST parameter to specify that the output for this sysout data set printed on a continuous-forms AFP printer is to go to:

- The burster-trimmer-stacker, to be burst into separate sheets.
- The continuous forms stacker, to be left in continuous fanfold.

If the specified stacker is different from the last stacker used, or if a stacker was not previously requested, JES issues a message to the operator to thread the paper into the required stacker.

**Note:** BURST applies only for an output data set printed on an AFP printer equipped with a burster-trimmer-stacker.

## Syntax

```
BURST= {YES}
       {Y  }
       {NO }
       {N  }
```

## Subparameter definition

**YES**
Requests that the printed output is to be burst into separate sheets. This subparameter can also be coded as Y.

**NO**
Requests that the printed output is to be in a continuous fanfold. This subparameter can also be coded as N.

## Defaults

If you do not code a BURST parameter, but you code a DD SYSOUT parameter and the sysout data set is printed on an AFP printer that has a burster-trimmer-stacker, JES uses an installation default specified at initialization.

If you do not code a BURST parameter or a DD SYSOUT parameter, the default is NO.

## Overrides

A BURST parameter on a sysout DD statement overrides an OUTPUT JCL BURST parameter.

## Relationship to other parameters

Do not code the following parameters with the BURST parameter.

| | | |
|---|---|---|
| * | DISP | PROTECT |
| AMP | DSID | QNAME |
| DATA | DYNAM | VOLUME |
| DDNAME | LABEL | |

## Relationship to other control statements

The burster-trimmer-stacker can also be requested using the following:

- The BURST parameter on the OUTPUT JCL statement.
- The STACKER parameter on the JES3 //*FORMAT PR statement.
- The BURST parameter on the JES2 /*OUTPUT statement.

## Example of the BURST parameter

```
//RECORD DD SYSOUT=A,BURST=Y
```

In this example, the DD statement requests that JES send the output to the burster-trimmer-stacker of the AFP printer. The stacker separates the printed output into separate sheets instead of stacking it in a continuous fanfold.

# CCSID parameter

***Parameter type***

Keyword, optional

***Purpose***

You can request the access method to convert data between the coded character set identifier (CCSID) specified on the JOB or EXEC statement and the CCSID specified on the DD statement. Data conversion is supported on access to ISO/ANSI Version 4 tapes using access methods BSAM or QSAM, but not using EXCP.

ISO/ANSI tapes are identified by the LABEL=(,AL) or LABEL=(,AUL) keyword. The CCSID parameter does not apply to ISO/ANSI Version 1 or ISO/ANSI/FIPS Version 3 tapes or to tapes with labels other than AL or AUL. See *z/OS DFSMS Using Data Sets*for selecting ISO/ANSI Version 4 tapes. It also contains a list of supported CCSIDs.

The CCSID value of 65535 has a special meaning: it suppresses conversion.

When CCSID is not specified at the JOB, EXEC, or DD levels, data passed to BSAM and QSAM is converted to 7-bit ASCII when writing to ISO/ANSI tapes. This might result in data loss on conversion. On READ operations the CCSID (if recorded) on the tape header label is used for conversion.

The CCSID is recorded in the tape header label if conversion is not defaulted.

## Syntax

```
CCSID= nnnnn
```

## Subparameter definition

**nnnnn**
The CCSID as a decimal number from 1 through 65535.

## Default

367.

## Relationship to other parameters

Do not code the following parameters with the CCSID parameter:

| | | |
|---|---|---|
| * | DDNAME | QNAME |
| BURST | DYNAM | SYSOUT |

| | | |
|---|---|---|
| CHARS | FCB | TERM |
| COPIES | FLASH | UCS |
| DATA | MODIFY | |

## Examples of the CCSID parameter

***Example 1***

```
//JOB1    JOB    (123456)
//S1      EXEC   PGM=MYPGM
//DD1     DD     DSN=A,DISP=NEW,UNIT=3590,
//               VOL=SER=T00001,LABEL=AL
```

In this example, the data on the new ISO/ANSI tape is converted from EBCDIC to 7-bit ASCII because CCSID was not specified at the JOB, EXEC, or DD levels. If the data passed to the access methods contain graphic or special characters there could be data loss on conversion to 7-bit ASCII. This is the default operation for ISO/ANSI/FIPS Version 3 and ISO/ANSI Version 4 tapes.

***Example 2***

```
//JOB2    JOB    (123456)
//S1      EXEC   PGM=MYPGM
//DD1     DD     DSN=A,DISP=OLD,UNIT=3590,
//               VOL=SER=T00001,LABEL=AL
```

In this example the data on the ISO/ANSI tape is converted from 7-bit ASCII (default) to EBCDIC. This is the default operation for ISO/ANSI/FIPS Version 3 and ISO/ANSI Version 4 tapes.

***Example 3***

```
//JOB3    JOB    (123456)
//S1      EXEC   PGM=MYPGM
//DD1     DD     DSN=A,DISP=NEW,UNIT=3590,
//               CCSID=65535,VOL=SER=T00003,LABEL=AL
```

In this example the data written to the ISO/ANSI Version 4 tape is not converted (CCSID=65535).

***Example 4***

```
//JOB4    JOB    (123456)
//S1      EXEC   PGM=MYPGM
//DD1     DD     DSN=A,DISP=OLD,UNIT=3590,
//               CCSID=65535,VOL=SER=T00004,LABEL=AL
```

In this example the user did not want any conversion (CCSID=65535) on data read by the access methods.

***Example 5***

```
//JOB5    JOB    (123456),CCSID=37
//S1      EXEC   PGM=MYPGM1
//DD1     DD     DSN=A,DISP=NEW,LABEL=(,AL),
//               VOL=SER=T00005,UNIT=3590,CCSID=437
```

In this example the user wants conversion from a CCSID of 37 (CECP: USA, Canada, Netherlands, Portugal, Brazil, Australia, New Zealand) to 437 (Base PC-data) for data written using BSAM or QSAM for ISO/ANSI Version 4 tape. The CCSID of 437 is recorded on the tape header label.

*Example 6*

```
//JOB6    JOB    (123456),CCSID=37
//S1      EXEC   PGM=MYPGM2
//DD1     DD     DSN=A,DISP=OLD,UNIT=3590,
//               VOL=SER=T00006,CCSID=437
```

In this example the user wants data conversion from a CCSID of 437 to a CCSID of 37 for data read by the access method. Note that the CCSID does not have to be specified if it is recorded in the label.

*Example 7*

```
//JOB7    JOB    (123456),CCSID=37
//S1      EXEC   PGM=MYPGM
//DD1     DD     DSN=A,DISP=OLD,UNIT=3590,
//               VOL=SER=T00007
```

In this example the ISO/ANSI labeled tape had a recorded CCSID of 437 and a CCSID was not specified on the DD statement. Data read from this tape by the access method is converted from a CCSID of 437 to a CCSID of 37.

*Example 8*

```
//JOB8    JOB    (123456),CCSID=37
//S1      EXEC   PGM=MYPGM1
//DD1     DD     DSN=A,DISP=NEW,LABEL=(,AL),UNIT=3590,
//               VOL=SER=T00008,CCSID=437
//S2      EXEC   PGM=MYPGM2,CCSID=65535
//DD1     DD     DSN=B,DISP=NEW,LABEL=(,AL),UNIT=3590,
//               VOL=SER=T00009
```

This example illustrates overriding the CCSID specified on the JOB statement by the specification on the EXEC statement.

In this example, in step S1 the user wants conversion from a CCSID of 37 to 437 for data written using BSAM or QSAM for the ISO/ANSI Version 4 tape.

In step S2 the JOB level CCSID of 37 is overridden by the EXEC level CCSID of 65535. Since a CCSID of 65535 prevents conversion, the data written to tape is not converted. A CCSID of 65535 is recorded in the tape header label because no CCSID was specified on the DD statement.

## CHARS parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the CHARS parameter to specify the name of one or more coded fonts for printing this sysout data set on an AFP printer.

**Note:** CHARS applies only for an output data set that is either printed on an AFP printer or processed by Infoprint Server.

***References***

For more information on coded font names, see *IBM AFP Fonts: Font Summary for AFP Font Collection*.

## Syntax

```
CHARS= {font-name                  }
       {(font-name[,font-name]...)}
       {DUMP                       }
       {(DUMP[,font-name]...)      }
```

- You can omit the parentheses if you code only one font-name or only DUMP.
- Null positions in the CHARS parameter are invalid. For example, you **cannot** code CHARS=(,font-name) or CHARS=(font-name,,font-name).

## Subparameter definition

**font-name**
Names a coded font or character-arrangement table. Each font-name is 1 through 4 alphanumeric or national ($, #, @) characters. Code from one to four names.

**DUMP**
Requests a high-density dump of 204-character print lines from a 3800. If more than one font-name is coded, DUMP must be first.

**Note:** Use DUMP on a SYSABEND or SYSUDUMP DD statement.

## Defaults

If you do not code the DD CHARS parameter, JES uses the following, in order:

1. The CHARS parameter on an OUTPUT JCL statement, if referenced by the DD statement.
2. The DD UCS parameter value, if coded.
3. The UCS parameter on an OUTPUT JCL statement, if referenced.

If no character-arrangement table is specified on the DD or OUTPUT JCL statements, JES uses an installation default specified at initialization.

## Overrides

A CHARS parameters on a sysout DD statement overrides the OUTPUT JCL CHARS parameter.

For a data set scheduled to the Print Services Facility (PSF), PSF uses the following parameters, in override order, to select the font list:

1. Font list in the library member specified by an OUTPUT JCL PAGEDEF parameter.
2. DD CHARS parameter.
3. OUTPUT JCL CHARS parameter.
4. DD UCS parameter.
5. OUTPUT JCL UCS parameter.
6. JES installation default for the device.
7. Font list on the PAGEDEF parameter in the PSF-cataloged procedure.

See "PAGEDEF parameter" on page 539 for more information.

## Relationship to other parameters

Do not code the following parameters with the CHARS parameter.

| | | |
|---|---|---|
| * | DISP | PROTECT |
| AMP | DSID | QNAME |
| DATA | DYNAM | VOLUME |
| DDNAME | LABEL | |

## Relationship to other control statements

CHARS can also be coded on the following:

- The OUTPUT JCL statement.
- The JES3 //*FORMAT PR statement.
- The JES2 /*OUTPUT statement.

## Requesting a high-density dump

You can request a high-density dump on the 3800 through two parameters on the DD statement for the dump data set or on an OUTPUT JCL statement referenced by the dump DD statement:

- FCB=STD3. This parameter produces dump output at 8 lines per inch.
- CHARS=DUMP. This parameter produces 204-character print lines.

You can code one or both of these parameters. You can place both on the same statement or one on each statement.

## Examples of the CHARS parameter

***Example 1***

```
//DD1  DD  SYSOUT=A,CHARS=(GT10,GB12)
```

In this example, the CHARS parameter specifies two fonts to be used when printing the data set: GT10 and GB12.

***Example 2***

```
//SYSABEND DD  UNIT=3800,CHARS=DUMP,FCB=STD3
```

The CHARS parameter on this SYSABEND DD statement specifies a high-density dump with 204 characters per line. The FCB parameter requests the dump output at 8 lines per inch.

**Note:** This example pertains only to 3800 printers.

# CHKPT parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the CHKPT parameter to request that a checkpoint be written when each end-of-volume is reached on the multivolume data set defined by this DD

statement. Checkpoints are written for all volumes except the last. Checkpoints can be requested for input or output data sets.

**Note:** CHKPT is supported only for multivolume QSAM or BSAM data sets. CHKPT is ignored for single-volume QSAM or BSAM data sets or for ISAM, BDAM, BPAM, or VSAM data sets. CHKPT is not supported for partitioned data sets extended (PDSEs).

*References*

For more information, see *z/OS DFSMSdfp Checkpoint/Restart*.

## Syntax

```
CHKPT=EOV
```

## Subparameter definition

**EOV**
Requests a checkpoint at each end-of-volume.

## Overrides

- The RD parameter values of NC and RNC on the JOB or EXEC statements override the CHKPT parameter.
- The CHKPT parameter overrides cataloged procedure values or START command values for checkpoints at end-of-volume.

## Relationship to other parameters

Do not code the following parameters with the CHKPT parameter.

| | |
|---|---|
| * | DYNAM |
| DATA | QNAME |
| DDNAME | SYSOUT |

## Relationship to the SYSCKEOV DD statement

If you specify CHKPT, you must also provide a SYSCKEOV DD statement in the job or step.

## Checkpointing concatenated data sets

For concatenated BSAM or QSAM data sets, CHKPT must be coded on each DD statement in the concatenation, if checkpointing is desired for each data set in the concatenation.

## Examples of the CHKPT parameter

*Example 1*

```
//DS1 DD  DSNAME=INDS,DISP=OLD,CHKPT=EOV,
//     UNIT=SYSSQ,VOLUME=SER=(TAPE01,TAPE02,TAPE03)
```

In this example, the DD statement defines data set INDS, a multivolume QSAM or BSAM data set for which a checkpoint is to be written twice: once when end-of-volume is reached on TAPE01 and once when end-of-volume is reached on TAPE02.

*Example 2*

```
//DS2 DD  DSNAME=OUTDS,DISP=(NEW,KEEP),
//     CHKPT=EOV,UNIT=SYSDA,VOLUME=(,,,8)
```

In this example, OUTDS is a multivolume data set that is being created. The data set requires eight volumes. Seven checkpoints will be written: when the end-of-volume is reached on volumes one through seven.

# CNTL parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the CNTL parameter to reference a CNTL statement that appears earlier in the job. The reference causes the subsystem to execute the program control statements within the referenced CNTL/ENDCNTL group.

The system searches for an earlier CNTL statement with a label that matches the **label** in the CNTL parameter. If the system finds no match, the system issues an error message.

## Syntax

```
CNTL= {*.label                       }
      {*.stepname.label              }
      {*.stepname.procstepname.label}
```

## Subparameter definition

**`*.label`**
Identifies an earlier CNTL statement, named label. The system searches for the CNTL statement first earlier in this step, then before the first EXEC statement of the job.

**`*.stepname.label`**
Identifies an earlier CNTL statement, named label, that appears in an earlier step, stepname, in the same job.

**`*.stepname.procstepname.label`**
Identifies a CNTL statement, named label, in a cataloged or in-stream procedure. Stepname is the name of the job step that calls the procedure; procstepname is the name of the procedure step that contains the CNTL statement named label.

## Examples of the CNTL parameter

*Example 1*

```
//MONDAY   DD  CNTL=*.WKLYPGM
```

In this example, the DD statement requests that the system use the program control statements following the CNTL statement named WKLYPGM and located earlier in this step or preceding the first step.

***Example 2***

```
//TUESDAY  DD  CNTL=*.SECOND.BLOCKS
```

In this example, the DD statement requests that the system use the program control statements following the CNTL statement named BLOCKS and located in a preceding step named SECOND.

***Example 3***

```
//WEDNES   DD  CNTL=*.THIRD.PROCTWO.CANETTI
```

In this example, the DD statement requests that the system use the program control statements following the CNTL statement named CANETTI and located in the procedure step PROCTWO of the procedure called in the preceding job step THIRD.

## COPIES parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the COPIES parameter to specify how many copies of this sysout data set are to be printed. The printed output is in page sequence for each copy.

For printing on an AFP printer, this parameter can instead specify how many copies of each page are to be printed before the next page is printed.

**Note:** For more information about the subparameters supported for AFP printers, see *PSF for z/OS: User's Guide*.

### Syntax

```
COPIES= {nnn                                }
        {(nnn,(group-value[,group-value]...))}
        {(,(group-value[,group-value]...))   }
```

- You can omit the parentheses if you code only COPIES=nnn.
- The following are **not** valid:
  - A null group-value, for example, COPIES=(5,(,)) or COPIES=(5,)
  - A zero group-value, for example, COPIES=(5,(1,0,4))
  - A null within a list of group-values, for example, COPIES=(5,(1,,4))

### Subparameter definition

**nnn**
A number (from 1 through 255 in a JES2 system, from 0 through 255 in a JES3 system) that specifies how many copies of the data set are to be printed.

For a data set printed on an AFP printer, JES ignores nnn if any group-values are specified.

**group-value**
Specifies how many copies of each page are to be printed before the next page is printed. Each group-value is a number from 1 through 255 in a JES2 system and from 1 through 254 in a JES3 system. You can code a maximum of eight group-values. Their sum must not exceed 255 or 254. The total copies of each page equals the sum of the group-values.

## Defaults

On any of the following statements, if you do not code a COPIES parameter, code it incorrectly, or code COPIES=0, the system uses the DD COPIES parameter default of 1.

- DD statement
- OUTPUT JCL statement
- For JES2, the /*OUTPUT statement

**Note:** In JES3 a copy count of zero in the OUTPUT JCL statement will give you a copy count of zero, not one.

## Overrides

A COPIES parameter on a sysout DD statement overrides an OUTPUT JCL COPIES parameter.

If this DD statement references an OUTPUT JCL statement and that OUTPUT JCL statement contains a FORMDEF parameter, which specifies a library member, the COPYGROUP parameter on a FORMDEF statement in that member overrides any group-value subparameters on the OUTPUT JCL COPIES parameter or the sysout DD COPIES parameter. For more information, see “FORMDEF parameter” on page 513.

## Relationship to Other Parameters

Do not code the following parameters with the COPIES parameter.

| | | |
|---|---|---|
| * | DDNAME | LABEL |
| AMP | DISP | QNAME |
| DATA | DYNAM | VOLUME |

***Relationship to FLASH Parameter:*** If this DD statement or a referenced OUTPUT JCL statement also contains a FLASH parameter, JES prints with the forms overlay the number of copies specified in one of the following:

- COPIES=nnn, if the FLASH count is larger than nnn. For example, if COPIES=10 and FLASH=(LTHD,12) JES prints 10 copies, all with the forms overlay.
- The sum of the group-values specified in the COPIES parameter, if the FLASH count is larger than the sum. For example, if COPIES=(,(2,3,4)) and FLASH=(LTHD,12) JES prints nine copies in groups, all with the forms overlay.
- The count subparameter in the FLASH parameter, if the FLASH count is smaller than nnn or the sum from the COPIES parameter. For example, if COPIES=10 and FLASH=(LTHD,7) JES prints seven copies with the forms overlay and three copies without.

***Restriction When Coding UNIT Parameter:*** The COPIES parameter is normally coded with the SYSOUT parameter. If, however, both COPIES and UNIT appear on a DD statement, JES handles the COPIES parameter as follows:

- nnn defaults to 1.
- Only the first group-value is used, if group-values are specified and printing is on a 3800.

## Relationship to other control statements

The number of copies can also be specified on the COPIES parameter of the following:

- The OUTPUT JCL statement.
- The JES2 /*OUTPUT statement.
- The JES3 //*FORMAT PR statement.
- The JES3 //*FORMAT PU statement.

**For JES2**, if you request copies of the entire job on the JES2 /*JOBPARM COPIES parameter and also copies of the data set on the DD COPIES or OUTPUT JCL COPIES parameter, and if this is a sysout data set, JES2 prints the number of copies equal to the **product** of the two requests.

### Using OUTPUT JCL COPIES by nullifying DD copies

If both a DD statement and a referenced OUTPUT JCL statement contain COPIES parameters, the DD COPIES parameter normally overrides the OUTPUT JCL COPIES parameter. For example, four copies are printed of sysout data set DDA:

```
//OTA  OUTPUT  COPIES=3
//DDA  DD      SYSOUT=A,OUTPUT=*.OTA,COPIES=4
```

However, if the DD COPIES is a null parameter, the OUTPUT JCL COPIES parameter is used. For example, three copies are printed of sysout data set DDB:

```
//OTB  OUTPUT  COPIES=3
//DDB  DD      SYSOUT=A,OUTPUT=*.OTB,COPIES=
```

The following example shows a null COPIES parameter on an in-stream DD statement that overrides a procedure DD statement. The null COPIES parameter on DD statement PS.DDA nullifies the COPIES parameter on the procedure DD statement DDA, thereby allowing the COPIES parameter on OUTPUT JCL statement OT to be used. The system prints three copies of the DDA sysout data set.

```
//JEX     JOB    ACCT34,'PAUL BENNETT'
//INSTR   PROC
//PS      EXEC   PGM=ABC
//OT      OUTPUT COPIES=3
//DDA     DD     SYSOUT=A,OUTPUT=*.OT,COPIES=2
//        PEND
//STEP1   EXEC   PROC=INSTR
//PS.DDA  DD     COPIES=
/*
```

**Note:** If a null COPIES parameter appears on a DD statement that either does not reference an OUTPUT JCL statement or references an OUTPUT JCL statement that does not contain a COPIES parameter, the system uses a default of 1.

## Examples of the COPIES parameter

***Example 1***

```
//RECORD1 DD  SYSOUT=A,COPIES=32
```

This example requests 32 copies of the data set defined by DD statement RECORD1 when printing on an impact printer or an AFP printer.

***Example 2***

```
//RECORD2 DD  SYSOUT=A,COPIES=(0,(1,2))
```

In this example, when printing on an AFP printer, three copies of the data set are printed in two groups. The first group contains one copy of each page. The second group contains two copies of each page. When printing on an impact printer, one copy (the default for nnn) is printed.

***Example 3***

```
//RECORD3 DD  SYSOUT=A,COPIES=(8,(1,3,2))
```

In this example, when printing on an AFP printer, six copies of the data set are printed in three groups. The first group contains one copy of each page, the second group contains three copies of each page, and the last group contains two copies of each page. When the output device is not an AFP printer, the system prints eight collated copies.

***Example 4***

```
//RECORD4 DD  UNIT=AFP1,COPIES=(1,(2,3))
```

Because the UNIT parameter is coded and the device is an AFP printer, the system prints only the first group-value: two copies of each page.

## DATA parameter

***Parameter type***

Positional, optional

***Purpose***

Use the DATA parameter to begin an in-stream data set that may contain statements with // in columns 1 and 2. The data records immediately follow the DD DATA statement; the records may be in any code, such as EBCDIC. The data records end when:

- An EBCDIC /* or the two-character delimiter specified by a DLM parameter on this DD statement is found in the input stream, or
- The input stream runs out of records.

Note that, unlike a DD * statement, the data is not ended by the // that indicates another JCL statement.

***Considerations for an APPC scheduling environment***

The DATA parameter has no function in an APPC scheduling environment. If you code DATA, the system will check it for syntax and ignore it.

### Syntax

```
//ddname DD DATA[,parameter]...  [comments]
```

## Defaults

When you do not code BLKSIZE and LRECL, JES uses installation defaults specified at initialization.

## Relationship to other parameters

The following DD parameters may be specified with the DD * and DD DATA parameters. All other parameters are either ignored or result in a JCL error.

| | |
|---|---|
| DCB=BLKSIZE | DSNAME |
| DCB=BUFNO | LIKE |
| DCB=LRECL | LRECL |
| DCB=DIAGNS | REFDD |
| DCB=MODE=C | VOLUME=SER |
| DLM | DSID |

For JES3, when using the DCB=MODE=C subparameter with the DATA parameter, DCB=MODE=C must be the only parameter specified with the DATA parameter.

You cannot use the TSO/E SUBMIT command to submit a data set to JES2 or JES3 with a record length of greater than 80 bytes. The records are truncated to 80 bytes.

**Note:** In-stream data within a cataloged procedure is also limited to 80 bytes per record.

You can submit a data set to JES2 or JES3 with a record length of greater than 80 bytes by submitting JCL like the following. In this example JCL, IBMUSER.LONGDATA.JCL contains the data with a record length of greater than 80 bytes. In a JES3 system, the record length is limited to the installation-defined spool buffer size minus 46. (For example, if the buffer size is defined as 4084, the record length limit is 4038.) JES3 input service fails any job that exceeds this limit.

If the records longer than 80 bytes include JCL to be transmitted to a remote system using JES3 // XMIT or //*ROUTE XEQ, or JES2 /*ROUTE XEQ or /*XMIT with JES3 in the network, the records are truncated to 80 bytes.

```
//SUBMIT   JOB   ...
//S1       EXEC  PGM=IEBGENER
//SYSPRINT DD    SYSOUT=*
//SYSIN    DD    DUMMY
//SYSUT2   DD    SYSOUT=(,INTRDR)
//SYSUT1   DD    DSN=IBMUSER.LONGDATA.JCL,DISP=SHR
```

JES3 will honor the BUFNO specification for SYSIN data sets. Values between 0 and 255 are accepted. When 0 or 1 is specified, a default of 2 is used. When 255 is specified, it is reduced to 254.

***For JES3 SNA RJP input:***

- The only parameters you can specify for JES3 systems network architecture (SNA) remote job processing (RJP) input devices are BLKSIZE and LRECL.
- Code DCB=LRECL=nnn, where nnn is 1 to 255 when SYSIN data records are greater than 80 bytes. (The default LRECL is 80 bytes.)

***For 3540 diskette input/output units:*** VOLUME=SER, BUFNO, and DSID on a DD DATA statement are ignored except when they are detected by a diskette reader as a request for an associated data set. See *3540 Programmer's Reference*. On a DD * or DD DATA statement processed by a diskette reader, you can specify DSID and

VOLUME=SER parameters to indicate that a diskette data set is to be merged into the input stream following the DD statement.

## Relationship to other control statements

Do not refer to an earlier DD DATA statement in DCB, DSNAME, or VOLUME parameters on following DD statements.

## Location in the JCL

A DD DATA statement begins an in-stream data set.

***In-stream Data for Cataloged or In-stream Procedures:*** A cataloged procedure can contain a DD DATA statement.

An in-stream procedure can contain a DD DATA statement.

When you call an in-stream procedure, you can add input stream data to an in-stream procedure step by placing one or more DD * or DD DATA statements in the calling step. You can alternatively include in-stream data directly within the in-stream or cataloged procedure.

***Multiple in-stream data sets for a step:*** You can code more than one DD * or DD DATA statement in a job step in order to include several distinct groups of data for the processing program. Precede each group with a DD * or DD DATA statement and follow each group with a delimiter statement.

***Omitted data delimiters:*** If you omit a DD statement before input data, the system provides a DD * statement with the ddname of SYSIN and ends the data when it reads a JCL statement or runs out of records. If you omit a delimiter after input data, the system ends the data when it reads a JCL statement or runs out of records.

## Unread records

If the processing program does not read all the data in an in-stream data set, the system skips the remaining data without abnormally terminating the step.

## Examples of the DATA parameter

***Example 1:***

```
//GROUP1   DD  DATA
           .
           .
           data
           .
/*
//GROUP2   DD  DATA
           .
           .
           data
           .
/*
```

This example defines two groups of data in the input stream.

***Example 2:***

```
//GROUP3   DD  DATA,DSNAME=&&GRP3
           .
           .
           data
           .
/*
```

This example defines an in-stream data set with GRP3 as the last qualifier of the system-generated data set name. A name such as userid.jobname.jobid.Ddsnumber.GRP3 is generated.

***Example 3:***

```
//STEP2       EXEC PROC=UPDATE
//PREP.DD4    DD   DSNAME=A.B.C,UNIT=3390,VOLUME=SER=D88230,
//              SPACE=(TRK,(10,5)),DISP=(,CATLG,DELETE)
//PREP.IN1    DD   DATA
              .
              .
              data
              .
/*
//ADD.IN2     DD   *
              .
              .
              data
              .
/*
```

This example defines two groups of data in the input stream. The input defined by DD statement PREP.IN1 is to be used by the cataloged procedure step named PREP. This data contains job control statements. The input data defined by DD statement ADD.IN2 is to be used by the cataloged procedure step named ADD. Because this data is defined by a DD * statement, it must not contain job control statements.

## DATACLAS parameter

***Parameter type***

Keyword, optional

This parameter is useful only if SMS is running. Without SMS, use the DCB parameter (described on section “DCB parameter” on page 135) or the AMP parameter (described on section “AMP parameter” on page 105). If you use a data class for your new data set, SMS must be running but your data set does not have to be SMS-managed.

***Purpose***

Use the DATACLAS parameter to specify a data class for a new data set. The storage administrator at your installation defines the names of the data classes you can code on the DATACLAS parameter.

If SMS is not installed or is not active, the system syntax checks and then ignores the DATACLAS parameter.

SMS ignores the DATACLAS parameter if you specify it for (1) an existing data set or (2) a data set that SMS does not support.

You can use the DATACLAS parameter for both VSAM data sets and physical sequential (PS) or partitioned (PO) data sets.

A data class defines the following data set allocation attributes:

- Data set organization
  - Record organization (RECORG) or
  - Record format (RECFM)
- Record length (LRECL)
- Key length (KEYLEN)
- Key offset (KEYOFF)
- Type, PDS, PDSE, basic format, extended format, large format, or HFS (DSNTYPE)
- Space allocation (AVGREC and SPACE)
- Retention period (RETPD) or expiration date (EXPDT)
- Volume-count (VOLUME)
- Compaction
- Media interchange type
- Space constraint relief
- Block size limit
- For VSAM data sets (IMBED or REPLICATE, CISIZE, FREESPACE, SHAREOPTIONS, REUSE, INITIAL LOAD, SPANNED/NONSPANNED, BWO (backup while open), and LOGGING OPTIONS)

***Note***

The volume-count on the VOLUME parameter in the data class specifies the maximum number of SMS-managed volumes that a data set can span. The maximum volume-count allowed by data class is 255. For SMS-managed DASD data set, the maximum volumes that a data set can span is 59, a greater than 59 volume-count in data class will be overridden with 59. The volume-count is ignored for data sets to which no storage class is assigned.

For tape data sets, only the following attributes apply:

- EXPDT
- LRECL
- RECFM
- RETPD
- VOLUME COUNT

***References***

See *z/OS DFSMS Using the Interactive Storage Management Facility* for information on how to use ISMF to view your installation-defined data classes.

## Syntax

```
DATACLAS=[data_class_name]
```

**Note:** If you specify a null DATACLAS, the JCL parser accepts it but ignores it.

## Subparameter definition

**data-class-name**
Specifies the name of a data class to be used for allocating the data set.

The name, one to eight characters, is defined by the storage administrator at your installation.

## Defaults

If you do not specify DATACLAS for a new data set and the storage administrator has provided an installation-written automatic class selection (ACS) routine, the ACS routine may select a data class for the data set. Check with your storage administrator to determine if an ACS routine will select a data class for the new data set, in which case you do not need to specify DATACLAS.

When RECORG is not specified, data sets associated with a data class, either by JCL or assigned by an ACS routine, will have DSORG defaulted to either physical sequential or a partitioned organization.

## Overrides

Normally, JCL specifications override data class specifications. However, the OVERRIDE SPACE attribute in the SMS data class allows you to specify whether the space attributes that are specified in the data class override corresponding attributes in the JCL. This exception applies to SPACE on the DD statements and also to dynamic allocation and IDCAMS DEFINE CLUSTER control statements. The OVERRIDE SPACE attribute value in the SMS data class can be YES or NO; the default is NO.

When OVERRIDE SPACE takes the default value 'NO', explicit specification of SPACE on the DD statement overrides both the SPACE and the AVGREC values specified in the data class.

An ACS routine can override the data class that you specify on the DATACLAS parameter.

Attributes obtained with the LIKE and REFDD parameters override the corresponding attributes in the DATACLAS parameter except when the data class assigned specifies 'YES' for the OVERRIDE SPACE attribute.

When OVERRIDE SPACE data class attribute takes the value 'YES', the JCL SPACE parameters are replaced with the data class SPACE related attributes.

For Non-VSAM data sets, the following data class attributes are used to override the JCL space specifications:

- AVGREC
- AVGVAL
- Primary quantity
- Secondary quantity
- Directory blocks (specify 0 for PS , non-zero for PO)

For VSAM data sets , the following data class attributes are used to override user specifications:

- AVGREC
- AVGVAL

- Primary quantity
- Secondary quantity
- Cisize Data
- %Freespace CI
- %Freespace CA

However, depending on the Recfm, Cisize data, %freespace values can be ignored.

## Relationship to other parameters

Do not code the following DD parameters with the DATACLAS parameter.

```
*
DATA
DDNAME
DYNAM
QNAME
```

## Examples of the DATACLAS parameter

***Example 1***

```
//SMSDS1  DD  DSNAME=MYDS1.PGM,DATACLAS=DCLAS01,DISP=(NEW,KEEP)
```

In the example, the attributes in the data class named DCLAS01 are used by SMS to handle the data set. Note that installation-written ACS routines may select a management class and storage class and can override the specified data class.

***Example 2***

```
//SMSDS2  DD  DSNAME=MYDS2.PGM,DATACLAS=DCLAS02,DISP=(NEW,KEEP),
//          LRECL=256,EXPDT=1996/033
```

In the example, the logical record length of 256 and the expiration date of February 2, 1996, override the corresponding attributes defined in the data class for the data set. Note that installation-written ACS routines may select a management class and storage class and can override the specified data class.

# DCB parameter

***Parameter Type***

Keyword, optional

***Purpose***

Use the DCB parameter to complete during execution the data set information in the data control block (DCB).

The data control block is constructed by the DCB macro instruction in assembler language programs or by file definition statements or language-defined defaults in programs in other languages.

**Note:**

1. With SMS, you do not need to use the DCB parameter to specify data set attributes. See the DATACLAS parameter (described on section "DATACLAS parameter" on page 132), the LIKE parameter (described on section "LIKE parameter" on page 224), and the REFDD parameter (described on section "REFDD parameter" on page 256).

2. For JES3 SNA RJP, code DCB=LRECL=nnn; where nnn is 1 to 255 when SYSIN data records are greater than 80 bytes. (The default LRECL is 80 bytes.)
3. Cross checking of DCB subparameters is done when the data set is opened, not when it is allocated. It is therefore possible to create a data set which may not be usable. This may then result in the issuance of a system completion code (ABEND) or unpredictable results when the data set is opened.

***References***

For more information on constructing the data control block, see *z/OS DFSMS Using Data Sets*.

## Syntax

```
[ DCB=(subparameter[,subparameter]...)                        ]

[ DCB= ( {dsname                          }[,subparameter]...) ]
[      ( {*.ddname                        }                  ) ]
[      ( {*.stepname.ddname               }                  ) ]
[      ( {*.stepname.procstepname.ddname}                    ) ]
```

**Parentheses**: You can omit the parentheses if you code:

- Only one keyword subparameter.
- Only a data set name, dsname, without any subparameters.
- Only a backward reference without any subparameters. A backward reference is a reference to an earlier DD statement in the job or in a cataloged or in-stream procedure called by a job step. A backward reference is in the form *.ddname or *.stepname.ddname or *.stepname.procstepname.ddname.

For example, DCB=RECFM=FB or DCB=WKDATA or DCB=*.STEP3.DD2

**Multiple subparameters**: When the parameter contains more than one subparameter, separate the subparameters by commas and enclose the subparameter list in parentheses. For example, DCB=(RECFM=FB,LRECL=133,BLKSIZE=399) or DCB=(*.DD1,BUFNO=4)

**Continuation onto another statement**: Enclose the subparameter list in only one set of parentheses. End each statement with a comma after a complete subparameter. For example:

```
//INPUT  DD  DSNAME=WKDATA,DCB=(RECFM=FB,LRECL=80,BLKSIZE=800,
//         BUFL=800,BUFNO=4)
```

***Alternate syntax for DCB keyword subparameters:*** All of the DCB keyword subparameters can be specified without the need to code DCB=. For example, the following DD statement:

```
//DDEX  DD  DSNAME=WKDATA,DCB=(RECFM=FB,LRECL=80,BLKSIZE=800),DISP=MOD
```

can also be specified as:

```
//DDEX  DD  DSNAME=WKDATA,RECFM=FB,LRECL=80,BLKSIZE=800,DISP=MOD
```

Note that KEYLEN, LRECL, and RECFM are described as DD parameters.

**Note:**

- If the BUFMAX subparameter is coded with or without the DCB parameter, it can have a null value only when coded on a DD which either:
  - Overrides a DD in a procedure

– Is added to a procedure.

## Subparameter definition

**subparameter**
(With SMS, see the DD DATACLAS parameter.)

Specifies a DCB keyword subparameter needed to complete the data control block.

An alphabetic summary of the DCB keyword subparameters follows this parameter description.

You must supply DCB information through the DCB subparameters if your processing program, the data set label, or your language's defined values do not complete the data control block.

**dsname**
(With SMS, see the DD LIKE parameter.)

Names a cataloged data set. The system is to copy DCB information from the data set's label. The data set must reside on a direct access volume, and the volume must be mounted before the job step is executed.

If dsname represents a VSAM data set, and you are allocating a new data set, you must also supply the RECORG parameter. You can specify RECORG explicitly (through the RECORG parameter), or implicitly, through the DATACLAS or LIKE parameters.

A hyphen is a valid character in a catalogued data set name. A data set name that contains a hyphen must be enclosed in apostrophes if it is used as a DCB subparameter.

The dsname cannot contain special characters, except for periods used in qualifying the name. Do not specify a generation data group (GDG) base name, a GDG relative generation member name, or a member name of a non-GDG data set.

The system copies the following DCB information from the data set label:

DSORG (used in a backward reference)
RECFM
OPTCD
BLKSIZE
LRECL
KEYLEN
RKP

If you do not specify the expiration date of the cataloged data set, the system copies it from the data set label. The system also copies the system code.

If you code any DCB subparameters after the dsname, these subparameters override the corresponding subparameters in the data set label. The system copies from the referenced label only those subparameters not specified on the referencing DD statement.

***.ddname**
***.stepname.ddname**
***.stepname.procstepname.ddname**
(With SMS, see the DD REFDD parameter or the DD LIKE parameter to select a comparable refer back function.)

Specify a backward reference to an earlier DD statement. The system is to copy DCB information from the DCB parameter specified on that DD statement. The DCB parameter of the referenced DD statement must contain subparameters, and it cannot name a cataloged data set or refer to another DD statement.

*.ddname specifies the ddname of an earlier DD statement in the same step. *.stepname.ddname specifies the ddname of a DD statement in an earlier step, stepname, in the same job. *.stepname.procstepname.ddname specifies the ddname of a DD statement in a cataloged or in-stream procedure called by an earlier job step. Stepname is the name of the job step that calls the procedure, and procstepname is the name of the procedure step that contains the DD statement.

If you code any DCB subparameters after the reference, these subparameters override the corresponding subparameters on the referenced DD statement. The system copies from the referenced DD statement only those subparameters not specified on the referencing DD statement.

Do not reference a DD * or a DD DATA statement.

**Note:** The system also copies the UCS and FCB parameters from the referenced DD statement, unless you override them in the referencing DD statement.

## Completing the data control block

The system obtains data control block information from the following sources, in override order:

- The processing program, that is, the DCB macro instruction in assembler language programs or file definition statements or language-defined defaults in programs in other languages.
- The DCB subparameter of the DD statement.
- The data set label.

Therefore, if you supply information for the same DCB field in your processing program and on a DD statement, the system ignores the DD DCB subparameter. If a DD statement and the data set label supply information for the same DCB field, the system ignores the data set label information.

**Note:** When concatenated data sets are involved, the DCB is completed based on the type of data set and how the processing program uses the data set. See *z/OS DFSMS Using Data Sets* for more information.

## Relationship to other parameters

See the descriptions of the individual DCB subparameters for the DD parameters and DCB subparameters that should not be coded with a specific DCB subparameter.

Do not code the following parameters with the DCB parameter.

```
AMP
DYNAM
```

With the DDNAME parameter, code **only** the BLKSIZE, BUFNO, and DIAGNS DCB subparameters.

With the QNAME parameter, code **only** the BLKSIZE, LRECL, OPTCD, and RECFM DCB subparameters.

The DD parameter KEYLEN and DCB subparameters KEYLEN, MODE, PRTSP, STACK, and TRTCH apply to specific device types. If you specify one of these subparameters on a DD statement for a device different from the type to which it applies, the system interprets the value incorrectly.

With the SPACE parameter, the value specified for BLKSIZE directly affects the amount of space obtained for data sets allocated in records, and for data sets allocated in blocks where the block length (blklgth) is zero.

***For 3540 Diskette Input/Output Units:*** The VOLUME=SER, DCB=BUFNO, and DSID parameters on a DD * or DD DATA statement are ignored except when they are detected by a diskette reader as a request for an associated data set. See *3540 Programmer's Reference*.

## Examples of the DCB parameter

***Example 1***

```
//DD1   DD    DSNAME=ALP,DISP=(,KEEP),VOLUME=SER=44321,
//            UNIT=3400-6,DCB=(RECFM=FB,LRECL=240,BLKSIZE=960,
//            DEN=1,TRTCH=C)
```

DD statement DD1 defines a new data set named ALP. The DCB parameter contains the information necessary to complete the data control block.

***Example 2***

```
//DD1A  DD    DSNAME=EVER,DISP=(NEW,KEEP),UNIT=3380,
//            DCB=(RECFM=FB,LRECL=326,BLKSIZE=23472),
//            SPACE=(23472,(200,40))
```

DD statement DD1A defines a new data set named EVER on a 3380. The DCB parameter contains the information necessary to complete the data control block.

```
//DD1B  DD    DSNAME=EVER,DISP=(NEW,KEEP),UNIT=3380,
//            RECFM=FB,LRECL=326,
//            SPACE=(23472,(200,40))
```

DD statement DD1B is the same as the DD1A statement except that it shows the alternate syntax for the DCB keyword subparameters. Also, because BLKSIZE is omitted, the system will select an optimum block size for the data.

***Example 3***

```
//DD2   DD    DSNAME=BAL,DISP=OLD,DCB=(RECFM=F,LRECL=80,
//            BLKSIZE=80)
//DD3   DD    DSNAME=CNANN,DISP=(,CATLG,DELETE),UNIT=3400-6,
//            LABEL=(,NL),VOLUME=SER=663488,DCB=*.DD2
```

DD statement DD3 defines a new data set named CNANN and requests that the system copy the DCB subparameters from DD statement DD2, which is in the same job step.

***Example 4***

```
//DD4   DD    DSNAME=JST,DISP=(NEW,KEEP),UNIT=SYSDA,
//            SPACE=(CYL,(12,2)),DCB=(A.B.C,KEYLEN=8)
```

DD statement DD4 defines a new data set named JST and requests that the system copy the DCB information from the data set label of the cataloged data set named A.B.C. If the data set label contains a key length specification, it is overridden by the KEYLEN coded on this DD statement.

*Example 5*

```
//DD5   DD    DSNAME=SMAE,DISP=OLD,
//            DCB=(*.STEP1.PROCSTP5.DD8,BUFNO=5)
```

DD statement DD5 defines an existing, cataloged data set named SMAE and requests that the system copy DCB subparameters from DD statement DD8, which is contained in the procedure step named PROCSTP5. The cataloged procedure is called by EXEC statement STEP1. Any of the DCB subparameters coded on DD statement DD8 are ignored if they are specified in the program. If the DCB BUFNO subparameter is not specified in the program, five buffers are assigned.

## DCB subparameters

| DCB subparameters | Access method | | | | | | | | Description of subparameters |
|---|---|---|---|---|---|---|---|---|---|
| | **BDAM** | **BPAM** | **BSAM** | **BTAM** | **EXCP** | **GAM** | **QSAM** | **TCAM** | |
| BFALN | X | X | X | | X | | X | | BFALN={F\|D}<br><br>Specifies that each buffer starts either on a word boundary that is not also a doubleword boundary or on a doubleword boundary. If both BFALN and BFTEK are specified, they must be specified from the same source.<br><br>Default: D (doubleword)<br><br>**Note:** Do not code the BFALN subparameter with DCB subparameter GNCP, or with DD parameters DDNAME or QNAME. |

| DCB subparameters | Access method | | | | | | | | Description of subparameters |
|---|---|---|---|---|---|---|---|---|---|
| | BDAM | BPAM | BSAM | BTAM | EXCP | GAM | QSAM | TCAM | |
| BFTEK | X | | X | X | | | X | | **BDAM and BSAM**: BFTEK=R<br><br>**R** Specifies that the data set is being created for or contains variable-length spanned records. Do **not** specify R for a PDSE.<br><br>**BTAM**: BFTEK=D<br><br>**D** Specifies that dynamic buffering is to be used in the processing program; if dynamic buffering is specified, a buffer pool also must be defined.<br><br>**QSAM**: BFTEK={S\|A}<br><br>**S** Specifies simple buffering (default). Simple buffering may be coded at any time for QSAM files.<br><br>**A** Specifies locate mode logical record interface for spanned records. QSAM obtains a logical record area and assembles the physical record segments of a spanned record into that logical record area. This forms a complete logical record before pointing the user to it.<br>• This parameter value may be specified only for RECFM=VS or RECFM=VBS files; if specified without RECFM=VS\|VBS, the specification is ignored.<br>• Locate mode must be used together with this parameter value.<br>**Note:** If you use locate mode on a RECFM=VS\|VBS file and BFTEK=A is **not** specified, the results may include processing the segments of a spanned record as separate records, issuance of system completion (abend) code X'002' with reason code X'04', or other unpredictable results.<br>• If you specify BFTEK=A with move mode, a system completion (abend) code X'013' with reason code X'5C' is issued.<br><br>For information about the locate and move modes in the DCB subparameters BFTEK and VBS, see *z/OS DFSMS Macro Instructions for Data Sets*.<br><br>If you specify both BFALN and BFTEK, you must specify them from the same source.<br><br>**Note:** Do not code the BFTEK subparameter with DCB subparameter GNCP, or with DD parameters DDNAME or QNAME.<br><br>**Note:** For compatibility purposes with previous operating systems, the system accepts BFTEK=E. |

| DCB subparameters | Access method | | | | | | | | Description of subparameters |
|---|---|---|---|---|---|---|---|---|---|
| | BDAM | BPAM | BSAM | BTAM | EXCP | GAM | QSAM | TCAM | |
| BLKSIZE | X | X | X | | X | | X | X | BLKSIZE={value \| valueK \| valueM \| valueG}<br><br>Specifies the maximum length, in bytes, of a block.<br><br>**value** Specifies the maximum length of a block. The number you specify for BLKSIZE depends on the device type and the record format for the data set. The maximum is 32,760 for DASD data sets and 2,147,483,648 for tape, except for data sets on magnetic tape with ISO/ANSI/FIPS labels, where the minimum value for BLKSIZE is 18 bytes and the maximum is 2048 bytes. (To allow a value greater than 2048, use installation exit routine IFG0193G, described in *z/OS DFSMS Installation Exits*.)<br><br>**valueK** Specifies the maximum length, in kilobytes, of a block. (1 kilobyte = 1024 bytes.) The maximum is 2097152. If 2097152K is coded, the block size assigned will be the maximum: 2,147,483,648.<br><br>**valueM** Specifies the maximum length, in megabytes, of a block. (1 megabyte = 1024 kilobytes.) The maximum is 2048. If 2048M is coded, the block size assigned will be the maximum: 2,147,483,648.<br><br>**valueG** Specifies the maximum length, in gigabytes, of a block. (1 gigabyte = 1024 megabytes.) The maximum is 2G. If 2G is coded, the block size assigned will be the maximum: 2,147,483,648.<br><br>If you code the BLKSIZE subparameter in the DCB macro instruction or on a DD statement that defines an existing data set with standard labels, the DCB BLKSIZE overrides the block size specified in the label. BLKSIZE can be coded but will have no effect on EXCP processing.<br><br>The number you specify for BLKSIZE directly affects the amount of space obtained for data sets allocated in records, and for data sets allocated in blocks where the block length (blklgth) is zero.<br><br>Default: If you do not code BLKSIZE, the system can, under certain conditions, determine an optimum block size for the data. For detailed information about system-determined block size, see *z/OS DFSMS Using Data Sets*.<br><br>**Note:** Do not code the BLKSIZE subparameter with the BUFSIZE subparameter. |
| BUFIN | | | | | | | | X | BUFIN=buffers<br><br>Specifies the number of buffers to be assigned initially for receiving operations for each line in the line group. The combined BUFIN and BUFOUT values must not be greater than the number of buffers in the buffer pool for this line group (not including those for disk activity only).<br><br>Default: 1<br><br>**Note:** Do not code the BUFIN subparameter with DCB subparameter BUFNO, or DD parameters DDNAME, QNAME. |
| BUFL | X | X | X | | X | | X | X | BUFL=bytes<br><br>Specifies the length, in bytes, of each buffer in the buffer pool. The maximum is 32,760.<br><br>**Note:** Do not code the BUFL subparameter with DD parameter DDNAME. |
| BUFMAX | | | | | | | | X | BUFMAX=buffers<br><br>Specifies the maximum number of buffers to be allocated to a line at one time. Number must be 2 through 15 and must be equal to or greater than the larger of the numbers specified by the BUFIN and BUFOUT subparameters.<br><br>Default: 2<br><br>**Note:** Do not code the BUFMAX subparameter with DCB subparameter NCP, or DD parameters DDNAME, QNAME. |

| DCB subparameters | Access method | | | | | | | | Description of subparameters |
|---|---|---|---|---|---|---|---|---|---|
| | BDAM | BPAM | BSAM | BTAM | EXCP | GAM | QSAM | TCAM | |
| BUFNO | X | X | X | X | X | | X | | BUFNO=buffers<br><br>Specifies the number of buffers to be assigned to the DCB. The maximum normally is 255, but can be less because of the size of the region.<br><br>**Note:** Do not code the BUFNO subparameter with DCB subparameters BUFIN, BUFOUT, or DD parameter QNAME. |
| BUFOFF | | | X | | | | X | | BUFOFF={n\|L}<br><br>**n** Specifies the length, in bytes, of the block prefix used with an ASCII tape data set. For input, n can be 0 through 99. For output, n must be 0 for writing an output data set with fixed-length or undefined-length records.<br><br>**L** Specifies that the block prefix is 4 bytes and contains the block length. BUFOFF=L is valid only with RECFM=D. For output, only BUFOFF=L is valid.<br><br>**Note:** Do not code the BUFOFF subparameter with DD parameters DDNAME, QNAME. |
| BUFOUT | | | | | | | | X | BUFOUT=buffers<br><br>Specifies the number of buffers to be assigned initially for sending operations for each line in the line group. The combined number of BUFIN and BUFOUT values must not be greater than the number of buffers in the buffer pool for this line group (not including those for disk activity only) and cannot exceed 15.<br><br>Default: 2<br><br>**Note:** Do not code the BUFOUT subparameter with DCB subparameter BUFNO, or DD parameter DDNAME. |
| BUFSIZE | | | | | | | | X | BUFSIZE=bytes<br><br>Specifies the length, in bytes, of each of the buffers to be used for all lines in a particular line group. Length must be 31 through 65535 bytes.<br><br>**Note:** Do not code the BUFSIZE subparameter with DCB subparameter BLKSIZE, or DD parameters DDNAME, QNAME. |
| CPRI | | | | | | | | X | CPRI={R\|E\|S}<br><br>Specifies the relative transmission priority assigned to the lines in this line group.<br><br>**R** Specifies that processor receiving has priority over processor sending.<br><br>**E** Specifies that receiving and sending have equal priority.<br><br>**S** Specifies that processor sending has priority over processor receiving.<br><br>**Note:** Do not code the CPRI subparameter with DCB subparameter THRESH, or DD parameters DDNAME, OUTLIM, QNAME. |
| CYLOFL | | | | | | | | | CYLOFL=tracks<br><br>Specifies the number of tracks on each cylinder to hold the records that overflow from other tracks on that cylinder. The maximum is 99. Specify CYLOFL only when OPTCD=Y.<br><br>**Note:** Do not code the CYLOFL subparameter with DCB subparameter RESERVE, or DD parameters DDNAME, FCB, QNAME, UCS. |

| DCB subparameters | Access method | | | | | | | | Description of subparameters |
|---|---|---|---|---|---|---|---|---|---|
| | BDAM | BPAM | BSAM | BTAM | EXCP | GAM | QSAM | TCAM | |
| DEN | | | X | | X | | X | | DEN={1\|2\|3\|4}<br><br>Specifies the magnetic density, in number of bits-per-inch, used to write a magnetic tape data set.<br><br>**DEN 7-track tape 9-track tape**<br>1 556 -<br>2 800 800 (NRZI)<br>3 - 1600 (PE)<br>4 - 6250 (GCR)<br><br>**NRZI** Non-return-to-zero inverted recording mode.<br><br>**PE** Phase encoded recording mode.<br><br>**GCR** Group coded recording mode.<br><br>**Default:** 800 bpi assumed for 7-track tape and 9-track without dual density.<br><br>1600 bpi assumed for 9-track with dual density or phase-encoded drives.<br><br>6250 bpi assumed for 9-track with 6250/1600 bpi dual density or group coded recording tape.<br><br>**Note:** Do not code the DEN subparameter with DD parameters DDNAME, QNAME. |
| DIAGNS | X | X | X | X | X | X | X | | DIAGNS=TRACE<br><br>Specifies the OPEN/CLOSE/EOV trace option, which gives a module-by-module trace of OPEN/CLOSE/EOV's work area and the DCB. If the generalized trace facility (GTF) is not running and tracing user events, DIAGNS is ignored. See *z/OS DFSMSdfp Diagnosis* for more information. |
| DSORG | X | X | X | X | X | X | X | X | DSORG=organization<br><br>Specifies the organization of the data set and indicates whether the data set contains any location-dependent information that would make the data set unmovable.<br><br>**Note:** Do not code the DSORG subparameter with DD parameters DDNAME, QNAME, RECORG.<br><br>**Organization** \| **Access Method**<br>PS Physical sequential data set \| BSAM,EXCP,QSAM,<br>PSU Physical sequential data set that contains location-dependent information \| BSAM,QSAM,EXCP<br>DA Direct access data set \| BDAM,EXCP<br>DAU Direct access data set that contains location-dependent information \| BDAM,EXCP<br>PO Partitioned data set (PDS or PDSE) \| BPAM,EXCP<br>POU Partitioned data set (PDS) that contains location-dependent information \| BPAM,EXCP<br>CX Communications line group \| BTAM<br>GS Graphic data control block \| GAM |

| DCB subparameters | Access method | | | | | | | | Description of subparameters |
|---|---|---|---|---|---|---|---|---|---|
| | BDAM | BPAM | BSAM | BTAM | EXCP | GAM | QSAM | TCAM | |
| EROPT | | | | X | | | X | | EROPT=x<br><br>**BTAM:** Requests the BTAM on-line terminal test option. x=T<br><br>**QSAM:** Specifies the option to be executed if an error occurs in reading or writing a record.<br><br>**x=ACC** System is to accept the block causing the error.<br><br>**x=SKP** System is to skip the block causing the error.<br><br>**x=ABE** System is to cause abnormal end of task.<br><br>**Default** ABE<br><br>**Note:** Do not code the EROPT subparameter with DD parameters DDNAME, QNAME. |
| FUNC | | | X | | | | X | | FUNC={I\|R\|P\|W\|D\|X\|T}<br><br>Specifies the type of data set to be opened for a 3505 Card Reader or 3525 Card Punch. Unpredictable results will occur if coded for other than a 3505 or 3525.<br><br>**I** Data set is for punching and printing cards.<br><br>**R** Data set is for reading cards.<br><br>**P** Data set is for punching cards.<br><br>**W** Data set is for printing.<br><br>**D** Protected data set is for punching.<br><br>**X** Data set is for both punching and printing.<br><br>**T** Two-line print option.<br><br>The only valid combinations of these values are:<br><br>`I WT RWT RPWXT PWX`<br>`R RP PW RPWD RPWX`<br>`P RPD PWXT RWX RWX`<br>`W RW RPW RWXT`<br><br>Default: P, for output data set. R, for input data set.<br><br>**Note:** Do not code the FUNC subparameter with the data-set-sequence number of the DD LABEL parameter, or DD parameters DDNAME, QNAME. |
| GNCP | | | | | | X | | | GNCP=n<br><br>Specifies the maximum number of I/O macro instructions that the program will issue before a WAIT macro instruction.<br><br>**Note:** Do not code the GNCP subparameter with DCB subparameters BFALN, BFTEK, or DD parameters DDNAME, QNAME. |
| INTVL | | | | | | | | X | INTVL={n\|0}<br><br>Specifies the interval, in seconds, between passes through an invitation list.<br><br>Default: 0<br><br>**Note:** Do not code the INTVL subparameter with DD parameters DDNAME, FCB, QNAME, UCS. |
| IPLTXID | | | | | | | | X | IPLTXID=member<br><br>Specifies the name of the partitioned data set (PDS) member that you want loaded into a 3704/3705 Communications Controller. The DCB IPLTXID subparameter overrides IPLTXID in the TERMINAL macro representing the NCP.<br><br>**Note:** Do not code the IPLTXID subparameter with DD parameters DDNAME, DSNAME, QNAME. |

| DCB subparameters | Access method | | | | | | | | Description of subparameters |
|---|---|---|---|---|---|---|---|---|---|
| | BDAM | BPAM | BSAM | BTAM | EXCP | GAM | QSAM | TCAM | |
| KEYLEN | X | X | X | | X | | | X | KEYLEN=bytes<br><br>The KEYLEN keyword subparameter is described on the DD KEYLEN parameter, section "KEYLEN parameter" on page 213. |
| LIMCT | X | | | | | | | | LIMCT={blocks \| tracks}<br><br>Specifies how many blocks (if relative block addressing is used) or how many tracks (if relative track addressing is used) are to be searched for a free block or available space. This kind of search occurs only when DCB OPTCD=E is also specified; otherwise, LIMCT is ignored. If the LIMCT number equals or exceeds the number of blocks or tracks in the data set, the entire data set is searched.<br><br>**Note:** Do not code the LIMCT subparameter with DD parameters DDNAME, QNAME. |
| LRECL | | X | X | | X | | X | X | LRECL=bytes<br><br>The LRECL keyword subparameter is described on the DD LRECL parameter, section "LRECL parameter" on page 226. |
| MODE | | | X | | X | | X | | `MODE= {C [O]}`<br>`{E [R]}`<br><br>Specifies the mode of operation to be used with a card reader, a card punch, or a card read-punch.<br><br>**C** Card image (column binary) mode<br><br>**E** EBCDIC mode<br><br>**O** Optional mark read mode<br><br>**R** Read column eliminate mode<br><br>If you specify R, you must also specify either C or E. Do not code the MODE subparameter for data entered through the input stream except in a JES3 system.<br><br>Do not code MODE=C for JES2 or JES3 output.<br><br>Default: E<br><br>**Note:** Do not code the MODE subparameter with DCB subparameters KEYLEN, PRTSP, TRTCH, or DD parameters DDNAME, KEYLEN, QNAME. |
| NCP | | X | X | | | | | | NCP=n<br><br>Specifies the maximum number of READ or WRITE macro instructions that may be issued before a CHECK macro instruction is issued to test for completion of the I/O operation. The maximum number is 255 for BSAM and BPAM, but may actually be smaller depending on the size of the address space. If chained scheduling is used, the number should be greater than 1.<br><br>Default: 1<br><br>**Note:** Do not code the NCP subparameter with DCB subparameter BUFMAX, or DD parameters DDNAME, QNAME. |
| NTM | | | | | | | | | NTM=tracks<br><br>Specifies the number of tracks to be used for a cylinder index. When the specified number of tracks has been filled, a master index is created. The DCB NTM is needed only when the DCB OPTCB=M. If you specify OPTCD=M but omit NTM, the master index option is ignored.<br><br>**Note:** Do not code the NTM subparameter with DCB subparameter PCI, or DD parameters DDNAME, QNAME. |

| DCB subparameters | Access method | | | | | | | | Description of subparameters |
|---|---|---|---|---|---|---|---|---|---|
| | BDAM | BPAM | BSAM | BTAM | EXCP | GAM | QSAM | TCAM | |
| OPTCD | X | X | X | | X | | X | X | Specifies the optional services to be performed by the control program. All optional services must be requested in one source, that is, in the data set label of an existing data set, in the DCB macro, or in the DD DCB parameter. However, the processing program can modify the DCB OPTCD field. Code the characters in any order; when coding more than one, do not code commas between the characters.<br><br>**Note:** Do not code the OPTCD subparameter with DD parameter DDNAME.<br><br>`BDAM: OPTCD= {A}[E][F][W]`<br>`{R}`<br><br>**A** indicates that the actual device addresses are to be specified in READ and WRITE macro instructions.<br><br>**R** indicates that relative block addresses are to be specified in READ and WRITE macro instructions.<br><br>**E** indicates that an extended search (more than one track) is to be performed for a block of available space. LIMCT must also be coded. Do not code LIMCT=0 because it will cause an abnormal termination when a READ or WRITE macro instruction is executed.<br><br>**F** indicates that feedback can be requested in READ and WRITE macro instructions and the device is to be identified in the same form as it was presented to the control program.<br><br>**W** requests a validity check for write operations on direct access devices. |
| OPTCD (continued) | | | | | | | | | BPAM: OPTCD= {C\|W\|CW}<br><br>**C** has no effect.<br><br>**W** requests a validity check for write operations.<br><br>`BSAM and QSAM: OPTCD= {B }`<br>`{T }`<br>`{U[C] }`<br>`{C[T][B][U] }`<br>`{H[Z][B] }`<br>`{J[C][U] }`<br>`{W[C][T][B][U]}`<br>`{Z[C][T][B][U]}`<br>`{Q[C][T][B] }`<br>`{Z }`<br><br>**B** requests that the end-of-volume (EOV) routine disregard the end-of-file (EOF) recognition for magnetic tape. For an input data set on a standard-labeled (SL or AL) tape, the EOV routine treats EOF labels as EOV labels until the volume serial list is exhausted. This option allows SL or AL tapes to be read out of volume sequence or to be concatenated to another tape with the same data set name using one DD statement. See "Data Sets that Span Libraries" in *z/OS MVS JCL User's Guide* for a description of allocation processing for multi-volume data sets created in different tape libraries.<br><br>**C** requests chained scheduling.<br><br>**H** requests hopper empty exit for optical readers or bypass of DOS checkpoint records.<br><br>**J** for a data set to be printed on an AFP printer, instructs the system that the logical record for each output data line contains a table reference character (TRC). The TRC identifies which character arrangement table in the CHARS parameter is to be used to print the line. Before specifying OPTCD=J, see *PSF for z/OS: User's Guide*. |

| DCB subparameters | Access method | | | | | | | | Description of subparameters |
|---|---|---|---|---|---|---|---|---|---|
| | BDAM | BPAM | BSAM | BTAM | EXCP | GAM | QSAM | TCAM | |
| OPTCD (continued) | | | | | | | | | BSAM and QSAM (continued):<br>**Q** indicates that all the user data in the data set is in ASCII. BSAM or QSAM converts the records from ASCII to EBCDIC when reading and converts the records from EBCDIC to ASCII when writing. The data set must reside on magnetic tape and must not contain IBM standard labels. The record format (RECFM) must not be V but can be D. If the label type is ISO/ANSI/FIPS, specified as LABEL=(,AL), the system forces OPTCD=Q.<br>**T** requests user totaling facility. T cannot be specified for a SYSIN or sysout data set.<br>**U** for 1403 or 3211 Printers with the Universal Character Set (UCS) feature and for the 3800, permits data checks and allows analysis by an appropriate error analysis routine. If U is omitted, data checks are not recognized as errors.<br>**W** requests a validity check for write operations on direct access devices. Requests "tape write immediate" mode on a cartridge tape device such as the IBM 3490 Magnetic Tape Subsystem.<br>**Z** for magnetic tape reel input, requests that the control program shorten its normal error recovery procedure. When specified, a data check is considered permanent after five unsuccessful attempts to read a record.<br>OPTCD=Z is ignored if chained scheduling or a tape cartridge is used. For a PDSE, all options except OPTCD=J are ignored.<br>EXCP: OPTCD=Z<br>**Z** for magnetic tape reel input, requests that the control program shorten its normal error recovery procedure. When specified, a data check is considered permanent after five unsuccessful attempts to read a record. OPTCD=Z has no effect on a tape cartridge. |
| PCI | | | | | | | | X | `PCI= {([N][,N])}`<br>`{([R][,R])}`<br>`{([A][,A])}`<br>`{([X][,X])}`<br>Specifies (1) whether or not a program-controlled interruption (PCI) is to be used to control the allocation and freeing of buffers and (2) how these operations are to be performed. The first operand applies to receiving operations and the second to sending operations.<br>**N** specifies that no PCIs are taken while filling buffers during receiving operations or emptying buffers during sending operations.<br>**R** specifies that after the first buffer is filled or emptied, a PCI occurs during the filling or emptying of each succeeding buffer. The completed buffer is freed, but no new buffer is allocated to take its place.<br>**A** specifies that after the first buffer is filled or emptied, a PCI occurs during the filling or emptying of the next buffer. The first buffer is freed, and a buffer is allocated to take its place.<br>**X** specifies that after a buffer is filled or emptied, a PCI occurs during the filling or emptying of the next buffer. The first buffer is not freed, but a new buffer is allocated.<br>You can omit the parentheses if you code only the first operand.<br>Default: (A,A)<br>**Note:** Do not code the PCI subparameter with DCB subparameter NTM, or DD parameters DDNAME, QNAME. |

| DCB subparameters | Access method | | | | | | | | Description of subparameters |
|---|---|---|---|---|---|---|---|---|---|
| | BDAM | BPAM | BSAM | BTAM | EXCP | GAM | QSAM | TCAM | |
| PRTSP | | | X | | X | | X | | PRTSP={0\|1\|2\|3}<br><br>Specifies the line spacing for an online printer. PRTSP is valid only for an online printer and only if the RECFM is not A or M. PRTSP=2 is ignored if specified with the DD SYSOUT parameter. 0 - spacing is suppressed, 1 - single, 2 - double, 3 - triple spacing<br><br>JES2 ignores PRTSP for sysout data sets.<br><br>Default: 1<br><br>**Note:** Do not code the PRTSP subparameter with DCB subparameters KEYLEN, MODE, STACK, TRTCH, or DD parameters DDNAME, KEYLEN, QNAME. |
| RECFM | X | X | X | | X | | X | X | RECFM=format<br><br>The RECFM keyword subparameter is described on the DD RECFM parameter, section "RECFM parameter" on page 251. |
| RESERVE | | | | | | | | X | RESERVE=(bytes1,bytes2)<br><br>Specifies the number of bytes (0 through 255) to be reserved in a buffer for insertion of data by the DATETIME and SEQUENCE macros.<br><br>**bytes1** indicates the number of bytes to be reserved in the first buffer that receives an incoming message.<br><br>**bytes2** indicates the number of bytes to be reserved in all the buffers following the first buffer in a multiple-buffer header situation.<br><br>Default: (0,0)<br><br>**Note:** Do not code the RESERVE subparameter with DCB subparameters CYLOFL, RKP, or DD parameters DDNAME, KEYOFF, QNAME, UCS. |
| RKP | | | | | X | | | | RKP=number<br><br>With SMS, use the DD KEYOFF or DATACLAS parameter. Specifies the position of the first byte of the record key in each logical record. The first byte of a logical record is position 0.<br><br>If RKP=0 is specified for blocked fixed-length records, the key begins in the first byte of each record. OPTCD=L must not be specified.<br><br>If RKP=0 is specified for unblocked fixed-length records, the key is not written in the data field. OPTCD=L can be specified.<br><br>For variable-length records, the relative key position must be 4 or greater, if OPTCD=L is not specified; the relative key position must be 5 or greater, if OPTCD=L is specified.<br><br>For EXCP processing, RKP can be coded but is ignored.<br><br>Default: 0<br><br>**Note:** Do not code the RKP subparameter with DCB subparameter RESERVE, or DD parameters DDNAME, FCB, KEYOFF, UCS. |
| STACK | | | X | | X | | X | | STACK={1\|2}<br><br>Specifies which stacker bin is to receive a card.<br><br>Default: 1<br><br>**Note:** Do not code the STACK subparameter with DCB subparameters KEYLEN, PRTSP, TRTCH, or DD parameters DDNAME, KEYLEN, QNAME. |

| DCB subparameters | Access method | | | | | | | | Description of subparameters |
|---|---|---|---|---|---|---|---|---|---|
| | BDAM | BPAM | BSAM | BTAM | EXCP | GAM | QSAM | TCAM | |
| THRESH | | | | | | | | X | THRESH=nn<br><br>Specifies the percentage of the nonreusable disk message queue records that are to be used before a flush closedown occurs.<br><br>Default: Closedown occurs when 95 percent of the records have been used.<br><br>**Note:** Do not code the THRESH subparameter with DCB subparameter CPRI, or DD parameters DDNAME, OUTLIM, QNAME. |
| TRTCH | | | X | | X | | X | | TRTCH={C\|E\|T\|ET} {COMP\|NOCOMP}<br><br>With C, E, T, or ET: specifies the recording technique for 7-track tape.<br><br>**C** specifies data conversion, odd parity, and no translation.<br><br>**E** specifies no data conversion, even parity, and no translation.<br><br>**T** specifies no data conversion, odd parity, and that BCD to EBCDIC translation is required when reading and EBCDIC to BCD translation when writing.<br><br>**ET** specifies no data conversion, even parity, and that BCD to EBCDIC translation is required when reading and EBCDIC to BCD translation when writing.<br><br>Default: no conversion, odd parity, and no translation.<br><br>With COMP or NOCOMP: specifies data compaction or no data compaction on a tape device enabled for compaction. Data compaction is not supported with ISO/ANSI labels.<br><br>**COMP** specifies data compaction.<br><br>**NOCOMP** specifies no data compaction<br><br>Defaults: On an IBM standard label tape, data sets after the first data set have the same compaction value (COMP or NOCOMP) as the first data set. The system ignores any compaction specified on data sets after the first. The system takes the compaction value from the first source that specifies it. The following sources can specify compaction:<br>1. TRTCH subparameter.<br>2. Data class, as set by the storage administrator. The tape, however, does not have to be system-managed.<br>3. DEVSUPxx member of SYS1.PARMLIB.<br>4. The hardware model. For the IBM 3480, the default is NOCOMP. For the IBM 3490, the default is COMP.<br><br>See *z/OS MVS JCL User's Guide* for information about using IEFBR14 and the TRTCH subparameter.<br><br>**Note:** Do not code the TRTCH subparameter with DCB subparameters KEYLEN, MODE, PRTSP, STACK, or DD parameters DDNAME, KEYLEN, QNAME.<br><br>**Note:** TRTCH is not applicable for DASD data sets. If specified, it will be ignored. |

## DDNAME parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Use the DDNAME parameter to postpone defining a data set until later in the same job step. A DDNAME parameter on a DD statement in a cataloged or in-stream procedure allows you to postpone defining the data set until a job step calls the procedure; the data set must be defined in the calling job step.

## Syntax

```
DDNAME=ddname
```

- The DDNAME parameter can have a null value only when coded on a DD which either:
  - Overrides a DD in a procedure
  - Is added to a procedure.

## Subparameter definition

**ddname**
Refers to a later DD statement that defines the data set. **ddname** must match the ddname of the referenced DD statement.

A job step or procedure step can contain up to five outstanding, unresolved DD parameters in a step at one time. Each DDNAME parameter must refer to a different DD statement.

## Overrides

If any DCB subparameter appears on both DD statements, the DCB subparameter on the referenced DD statement overrides the DCB subparameter on the DD statement that contains DDNAME.

## Relationship to other parameters

The **only** DD parameters you can code with the DDNAME parameter are:

```
DCB=BLKSIZE
DCB=BUFNO
DCB=DIAGNS
LIKE
REFDD
```

Do not code the DDNAME parameter on a DD statement with a ddname of JOBLIB.

## Location in the JCL

Place a DD statement containing a DDNAME parameter in a job step or in a cataloged or in-stream procedure. The referenced DD statement must be later in the same job step, must be in the calling job step, or must be in a cataloged or in-stream procedure called by the job step.

Do not use the name of a DDNAME statement more than once within the same step.

***Location of DD statements for concatenated data sets:*** To concatenate data sets to a data set defined with a DDNAME parameter, the unnamed DD statements must follow the DD statement that contains the DDNAME parameter, not the referenced DD statement that defines the data set.

***Errors in location of referenced DD statement:*** The system treats a DDNAME parameter as though it were a DUMMY parameter and issues a warning message in the following cases:

- If the job step or called procedure does not contain the referenced DD statement.
- If the referenced DD statement appears earlier in the job step.

***Location of DD statement requesting unit affinity:*** To use the same device, a DD statement can request unit affinity to an earlier DD statement by specifying UNIT=AFF=ddname. If a DD statement requests unit affinity to a DD statement containing a DDNAME parameter, the DD statement requesting unit affinity must be placed after the referenced DD statement. If the DD statement requesting unit affinity appears before, the system treats the DD statement requesting unit affinity as a DUMMY DD statement.

```
//STEP  EXEC  PGM=TKM
//DD1   DD    DDNAME=DD4
//DD2   DD    DSNAME=A,DISP=OLD
        .
        .
//DD4   DD    DSNAME=B,DISP=OLD
//DD5   DD    UNIT=AFF=DD1
```

DD1 postpones defining the data set until DD4. DD5 requests unit affinity to DD1. Because DD1 has been defined when DD5 is processed, the system assigns DD5 to the same device as DD1.

Instead of specifying UNIT=AFF=ddname, both DD statements can specify the same devices in their UNIT parameters or the same volume serials in their VOLUME parameters.

## Referenced DD statement

If the DDNAME parameter appears in a procedure with multiple steps, the ddname on the referenced DD statement takes the form stepname.ddname. For example, if procedure step STEPCP1 contains:

```
//INDATA  DD  DDNAME=DD1
```

The referenced DD statement in the calling job step is:

```
//STEPCP1.DD1  DD *
```

***Parameters not permitted on the referenced DD statement:*** The referenced DD statement must not contain a DYNAM or PATH parameter.

A DD statement that contains a DDNAME parameter must not override a procedure sysout DD statement that contains an OUTPUT parameter if the referenced DD statement also contains an OUTPUT parameter.

***References to Concatenated Data Sets:*** If you make a **forward reference** to a concatenation, the forward reference resolves to the first data set in the concatenation. If there are no DD statements between the forward reference and the concatenation, the rest of the data sets in the concatenation are appended to the first data set in the concatenation. The following example illustrates this.

```
//STEP1    EXEC PGM=IEBGENER
//SYSPRINT DD   SYSOUT=*
//SYSUT1   DD   DDNAME=INPUT
//INPUT    DD   DSN=TSTDATA1,DISP=SHR
//         DD   DSN=TSTDATA2,DISP=SHR
//SYSUT2   DD   SYSOUT=*
//SYSIN    DD   DUMMY
```

In this example, SYSUT1 will resolve to the first data set TSTDATA1, defined by the DDNAME forward reference INPUT. TSTDATA2, the second data set in the DDNAME forward reference INPUT, will be appended to SYSUT1 as well. IEBGENER will recognize TSTDATA1 and TSTDATA2 as input.

If there are any DD statements between the forward reference and the concatenation, the rest of the data sets in the concatenation are appended to the last DD statement preceding the concatenation. For example:

```
//STEP1    EXEC PGM=IEBGENER
//SYSUT1   DD   DDNAME=INPUT
//SYSPRINT DD   SYSOUT=*
//SYSUT2   DD   SYSOUT=*
//INPUT    DD   DSN=TSTDATA1,DISP=SHR
//         DD   DSN=TSTDATA2,DISP=SHR
//SYSIN    DD   DUMMY
```

In the preceding example, SYSUT1 will resolve to the first data set, TSTDATA1, defined in the DDNAME forward reference INPUT. TSTDATA2 will be appended to SYSUT2, the last DD statement preceding the concatenation. In that example IEBGENER will only recognize TSTDATA1 as input.

If a concatenated DD is added to a procedure, the remaining concatenated data sets will be concatenated to the last DD in the step named in an override or addition (or to the first step if no step was named in an override or addition). Note that this may result in these concatenated DDs being added to an unexpected DD. The following example illustrates this.

```
//TPROC    PROC
//S1       EXEC PGM=IEFBR14
//DD1      DD   DDNAME=INPUT
//DD2      DD   DSN=MYDSN2,DISP=SHR
//DD3      DD   DSN=MYDSN3,DISP=SHR
//S2       EXEC PGM=IEFBR14
//DDA      DD   DDNAME=INPUT
//DDB      DD   DSN=MINE2,DISP=SHR
//DDC      DD   DSN=MINE3,DISP=SHR
//         PEND
//STEP1    EXEC TPROC
//INPUT    DD   DSN=MYDSN1,DISP=SHR
//         DD   DSN=MYDSN4,DISP=SHR
//S2.INPUT DD   DSN=MINE1,DISP=SHR
//         DD   DSN=MINE4,DISP=SHR
```

In the preceding example, the result of the DDNAME forward reference INPUT is:

- In step S1, DD1 resolves to data set MYDSN1 and data set MYDSN4 is concatenated to data set MYDSN3.
- In step S2, DDA resolves to data set MINE1 and data set MINE4 is concatenated to data set MINE3.

**Attention:** The system always issues a warning message IEF694I, even if all data sets in the concatenation are used.

## Backward references

A backward reference is a reference to an earlier DD statement in the job or in a cataloged or in-stream procedure called by a job step. A backward reference is in the form *.ddname or *.stepname.ddname or *.stepname.procstepname.ddname. The ddname in the reference is the ddname of the earlier DD statement. If the earlier DD statement contains a DDNAME parameter, the reference is to the ddname in the name field of the earlier statement, **not** to the ddname in the DDNAME parameter.

The DD statement referenced in a DDNAME parameter cannot refer to a DD statement between the statement containing the DDNAME parameter and itself. For example:

```
//SHOW   EXEC PGM=ABLE
//DD1    DD   DDNAME=INPUT
//DD2    DD   DSNAME=TEMPSPAC,SPACE=(TRK,1),UNIT=SYSDA
//DD3    DD   DSNAME=INCOPY,VOLUME=REF=*.DD1,
//            DISP=(,KEEP),SPACE=(TRK,(5,2))
//DD4    DD   DSNAME=OUTLIST,DISP=OLD
//DD5    DD   DSNAME=MESSAGE,DISP=OLD,UNIT=3390,VOLUME=SER=333333
//INPUT  DD   DSNAME=NEWLIST,DISP=(OLD,KEEP),VOLUME=SER=333333,
//            UNIT=3390
```

The DDNAME parameter on DD1 refers to DD statement INPUT. The VOLUME parameter of DD3 specifies a backward reference to DD1, which is the name field ddname.

DD statement INPUT identifies the volume 333333 in its VOLUME=SER=333333 parameter. DD statement INPUT cannot use a backward reference to the VOLUME parameter on DD5 because DD5 is between the referring DD1 and the referenced INPUT.

## Examples of the DDNAME parameter

***Example 1:*** The following procedure step is the only step in a cataloged procedure named CROWE:

```
//PROCSTEP  EXEC PGM=RECPGM
//DD1       DD   DDNAME=WKREC
//POD       DD   DSNAME=OLDREC,DISP=OLD
```

DD statement DD1 is intended for weekly records in the input stream; these records are processed by this step. Because the * and DATA parameters cannot be used in cataloged procedures, the DDNAME parameter is coded to postpone defining the data set until the procedure is called by a job step. The step that calls the procedure is:

```
//STEPA  EXEC PROC=CROWE
//WKREC  DD   *
         .
         .
         data
         .
/*
```

***Example 2:*** When the procedure contains multiple steps, use the form stepname.ddname for the ddname of the referenced DD statement. For example, the following procedure steps appear in a cataloged procedure named PRICE:

```
//STEP1  EXEC PGM=SUGAR
//DD1    DD   DDNAME=QUOTES
         .
         .
         .
//STEP2  EXEC PGM=MOLASS
//DD2    DD   DSNAME=WEEKB,DISP=OLD
         .
         .
         .
```

The step that calls the procedure is:

```
//STEPA        EXEC PROC=PRICE
//STEP1.QUOTES DD   *
               .
               .
               data
               .
/*
```

***Example 3:*** When the referenced DD statement is to be a concatenation, the procedure must already contain the concatenation. (Such as when the referencing DD statement is to contain in-stream data.) For example, the following procedure step appears in cataloged procedure NEWONE.

```
//NEWONE PROC
//STEP1  EXEC  PGM=TRYIT
//DD1    DD    DDNAME=INSTUFF
//       DD    DSN=OLDSTUFF,DISP=OLD
          .
          .
```

The step that calls the procedure is:

```
//STEPA  EXEC  PROC=NEWONE
//STEP1.INSTUFF DD  *
          .
         data
          .
/*
```

The instream data (DDNAME=INSTUFF) is inserted before OLDSTUFF in the concatenation.

***Example 4:*** In the following example we create a DD concatenation in a procedure using multiple DDNAME forward references, INPUT1—INPUT5. In the example, INPUT1 resolves to data set FIRST, INPUT2 resolves to data set SECOND, and INPUT3 resolves to data set THIRD. INPUT4 and INPUT5 resolve to DUMMY.

```
//ABC    PROC
//SI     EXEC PGM=IEFBR14
//DD1    DD DDNAME=INPUT1
//       DD DDNAME=INPUT2
//       DD DDNAME=INPUT3
//       DD DDNAME=INPUT4
//       DD DDNAME=INPUT5
//STEP1  EXEC ABC
//INPUT1 DD DSN=FIRST,DISP=SHR
//INPUT2 DD DSN=SECOND,DISP=SHR
//INPUT3 DD DSN=THIRD,DISP=SHR
```

## DEST parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the DEST parameter to specify a destination for a sysout data set. The DEST parameter can send a sysout data set to a remote or local terminal, a node, a node and remote workstation, a local device or group of devices, or a node and userid.

**Note:** Code the DEST parameter only on a DD statement with a SYSOUT parameter. Otherwise, the system checks the DEST parameter for syntax, then ignores it.

For more information about USERID and WRITER ID, see *z/OS MVS JCL User's Guide*.

## Syntax

```
DEST=destination

The destination subparameter for JES2 is one of the following:

     LOCAL|ANYLOCAL
     name
     Nnnnnn
     NnRmmmmm
     NnnRmmmm
     NnnnRmmm
     NnnnnRmm
     NnnnnnRm
     (node,remote)
     Rmmmmm
     RMmmmmm
     RMTmmmmm
     Unnnnn
     (node,userid)
     userid

The destination subparameter for JES3 is one of the following:

     ANYLOCAL
     device-name
     device-number
     group-name
     nodename
     (nodename,userid)
     (nodename,devicename)
```

## Subparameter definition for JES2 systems

**LOCAL|ANYLOCAL**
Indicates the local node on a local device.

**name**
Identifies a destination by a symbolic name which is defined by the installation during JES2 initialization. The name can be, for example, a local device, remote device, or a userid. The name is 1 through 8 alphanumeric or national ($, #, @) characters.

**Nnnnnn**
Identifies a node. nnnnn is 1 through 5 decimal numbers from 1 through 32,767.

**NnRmmmmm**
**NnnRmmmm**
**NnnnRmmm**
**NnnnnRmm**
**NnnnnRm**
**(node,remote)**
Identifies a node and a remote work station connected to the node. The node

number, indicated in the format by n, is 1 through 5 decimal numbers from 1 through 32,767. The remote work station number, indicated in the format by m, is 1 through 5 decimal numbers from 1 through 32,767. Do not code leading zeros in n or m. The maximum number of digits for n and m combined cannot exceed six.

**Note:** NnnR0 is equivalent to LOCAL specified at node Nn.

**Rmmmmm**
**RMmmmmm**
**RMTmmmmm**
Identifies a remote workstation. mmmmm is 1 through 5 decimal numbers from 1 through 32,767. Note that with remote pooling, the installation may translate this route code to another route code.

If you send a job to execute at a remote node and the job has a ROUTE PRINT RMTmmmm statement, JES2 returns the output to RMTmmmm at the node of origin. For JES2 to print the output at RMTmmmm at the executing node, code DEST=NnnnRmmm on an OUTPUT JCL statement or sysout DD statement.

**Note:** R0 indicates any local device.

**Unnnnn**
Identifies a local terminal with special routing. nnnnn is 1 through 5 decimal numbers from 1 through 32,767.

If you send a job to execute and the job has a ROUTE PRINT Unnnnn statement, JES2 returns the output to Unnnnn at the node of origin.

**(node,userid)**
Identifies a node and a TSO/E or VM userid at that node. The node is a symbolic name defined by the installation during initialization; node is 1 through 8 alphanumeric or national ($, #, @) characters. The userid must be defined at the node; userid for TSO/E is 1 through 7 alphanumeric or national ($, #, @) characters and for VM is 1 through 8 alphanumeric or national ($, #, @) characters. The userid can also be a destination name defined in a JES2 DESTID initialization statement.

DEST=(node) is valid with a **writer-name** subparameter in the SYSOUT parameter; however, DEST=(node,userid) is not valid. Therefore, you can code SYSOUT=(A,writer-name),DEST=(node), but not SYSOUT=(A,writer-name),DEST=(node,userid).

**Note:** You can code DEST=(nodename,Unnnnn) here; this syntax is a valid subset of DEST=(node,userid).

**userid**
Identifies a userid at the local node. Userid for TSO/E is 1 through 7 alphanumeric or national ($, #, @) characters. The userid can also be a destination name definded in a JES2 DESTID initialization statement.

**Note:** JES2 initialization statements determine whether or not the node name is required when coding a userid. See your system programmer for information regarding how routings will be interpreted by JES2.

## Subparameter definition for JES3 systems

**ANYLOCAL**
Indicates any local device that is attached to the global processor.

**device-name**
Identifies a local device by a symbolic name defined by the installation during JES3 initialization. device-name is 1 through 8 alphanumeric or national ($, #, @) characters.

**device-number**
Identifies a specific device by a 3-digit or 4-digit hexadecimal number. Precede a 4-digit number with a slash (/). A 3-digit number can be specified with or without a slash.

**group-name**
Identifies a group of local devices, an individual remote station, or a group of remote stations by a symbolic name defined by the installation during JES3 initialization. group-name is 1 through 8 alphanumeric or national ($, #, @) characters.

**nodename**
Identifies a node by a symbolic name defined by the installation during JES3 initialization. nodename is 1 through 8 alphanumeric or national ($, #, @) characters. If the nodename you specify is the same as the node you are working on, JES3 treats the output as though you specified ANYLOCAL.

**(nodename,userid)**
Identifies a nodename and a TSO/E or VM userid at that nodename. The nodename is a symbolic name defined by the installation during initialization; node is 1 through 8 alphanumeric or national ($, #, @) characters. The userid must be defined at the nodename; userid for TSO/E is 1 through 7 alphanumeric or national ($, #, @) characters and for VM is 1 through 8 alphanumeric or national ($, #, @) characters.

A userid requires a nodename; therefore, code DEST=(nodename,userid). You **cannot** code a userid without a node.

DEST=(nodename) is valid with a **writer-name** subparameter in the SYSOUT parameter: however, DEST=(nodename,userid) is not valid. Therefore, you can code SYSOUT=(A,writer-name),DEST=(nodename).

**(nodename,devicename)**
Identifies a node by a symbolic name defined by the installation during JES3 initialization. nodename and devicename are each 1 through 8 alphanumeric or national ($, #, @) characters. devicename identifies a device by a symbolic name defined to that node by the installation during JES3 initialization. devicename is 1 through 8 alphanumeric or national ($, #, @) characters.

Use this form of the DEST parameter to override the ORG parameter.

## Defaults

If you do not code a DEST parameter, JES directs the sysout data set to the default destination for the input device from which the job was submitted.

In a JES3 system, if you do not code a DEST parameter, the default destination is the submitting location. For jobs submitted through TSO/E and routed to NJE for execution, the default is the node from which the job was submitted, and the destination ANYLOCAL.

If a specified destination is invalid, the job fails.

If you've coded the ORG parameter but did not explicitly code a primary destination, the default primary destination is the node specified in the ORG parameter, not the submitting node.

## Overrides

The DEST parameter on the sysout DD statement overrides an OUTPUT JCL DEST parameter.

## Relationship to other parameters

Code the DEST parameter only on a DD statement with the SYSOUT parameter.

## Relationship to other control statements

You can also code an output destination using:

- The OUTPUT JCL statement.
- The JES2 /*OUTPUT and /*ROUTE control statements.
- The JES3 //*MAIN, //*FORMAT PR, and //*FORMAT PU control statements.

Because DEST=(node,userid) cannot be coded on JES2 or JES3 control statements, you must code it, if needed, on a DD or OUTPUT JCL statement.

## Example of the DEST parameter

```
//JOB01   JOB  ,'MAE BIRD',MSGCLASS=B
//STEP1   EXEC PGM=INTEREST
//DEBIT   DD   SYSOUT=A
//CALIF   DD   SYSOUT=A,DEST=R555
//FLOR    DD   SYSOUT=A,DEST=(BOCA,'9212U28')
```

In this example, the system sends the sysout data set defined by DD statement DEBIT to the work station that submitted the job, the data set defined by DD statement CALIF to the remote terminal 555, and the data set defined by DD statement FLOR to VM userid 9212U28 at node BOCA.

# DISP parameter

***Parameter Type***

Keyword, optional

***Purpose***

Use the DISP parameter to describe the status of a data set to the system and tell the system what to do with the data set after termination of the step or job. You can specify one disposition for normal termination and another for abnormal termination. Note that if an abend occurs within a step, the step itself may terminate normally or abnormally, depending on whether an ESTAE (or ESPIE routine) intercepts the abend and requests that processing continue. Normal termination of a step is indicated by the presence of message IEF142I; abnormal termination of a step is indicated by the presence of message IEF472I.

**Note:** Disposition of the **data set** is controlled solely by the DISP parameter; disposition of the **volume(s)** on which the data set resides is a function of the volume status when the volume is demounted. If the UNIT parameter specifies a device, such as a printer or telecommunications device, that does not involve a data set, do not code the DISP parameter.

If the system obtains unit and volume information for an OLD, MOD, or SHR status, the data set is treated as if it exists, whether or not it is physically on the device.

When any step of a job requests exclusive control of a data set, with an exception of when the job is allowed to downgrade ENQs via the DSENQSHR specification, the system converts all requests for shared control of that data set within that job (DISP=SHR) to requests for exclusive control. One of two methods can be used to request exclusive control:

- DISP=NEW, DISP=MOD, or DISP=OLD on a JCL request.
- DISP=NEW, DISP=MOD, or DISP=OLD on a dynamic allocation request, including dynamic allocation requests that result from the use of certain utility control statements. For example, utility control statements that delete/scratch a data set will result in exclusive use of that data set.

If a dynamic allocation requests exclusive control of a data set then all subsequent DISP=SHR JCL references to that data set within that job will be upgraded to exclusive control. The job will retain exclusive control of that data set until the end of the last step of that job which references that data set in its JCL. For example:

```
//STEP1  EXEC PGM=anypgm1
//DD1    DD  DSN=A.B.C,DISP=SHR
//STEP2  EXEC PGM=IDCAMS
   DELETE  A.B.C
   DEFINE  A.B.C
//STEP3  EXEC PGM=anypgm3
//DD3    DD  DSN=A.B.C,DISP=SHR
//STEP4  EXEC PGM=anypgm4
```

Before the start of STEP1, the job will request shared control of data set A.B.C. In STEP2, the DELETE/DEFINE of data set A.B.C will cause the shared control from STEP1 to be upgraded to exclusive control. In STEP3, control of data set A.B.C will remain exclusive, since it is not possible to downgrade an ENQ from EXCL to SHR in the case that a dynamic allocation upgrades the ENQ. At the end of STEP3, control of data set A.B.C will be released. The ENQ on data set A.B.C will therefore not be held at all during STEP4. If, however, the job also contained a STEP5 which requested use of data set A.B.C, then exclusive control of that data set would continue to be held by the job all the way through both STEP4 and STEP5.

***DISP and ENQ:*** Before starting the first step of a job, the initiator requests control of all of the data sets in that job by issuing an ENQ for each of them, using the value specified for DISP to determine the kind of ENQ issued. The initiator issues the ENQ for each data set at the highest level required for that data set by any step of the job. For example, if all steps of the job request shared control of a specific data set (DISP=SHR) then the ENQ for that data set is requested as SHR. If, on the other hand, *any* step of the job requests exclusive control of a specific data set (DISP=NEW, DISP=MOD, or DISP=OLD), then the ENQ for that data set is requested EXCL.

If the job is allowed to downgrade ENQs, then ENQs will be downgraded from exclusive control to shared control at the last step for which the data set is referenced EXCL. The downgrade will not occur if the ENQ has been upgraded by a dynamic allocation request in the current or a previous step.

If the job is not allowed to downgrade ENQs, then ENQs cannot be downgraded from EXCL to SHR. If one step needs the ENQ EXCL and a following step only needs it SHR, the ENQ is still held as EXCL.

In both cases, the ENQ is held until the end of the last step which references that data set, at which point the ENQ is released entirely.

***DISP and ENQ for generation data sets:*** The way the initator issues an ENQ to control generation data sets can be different than with other data sets. The initiator only issues the ENQ for the GDG base name for a generation data set that is referenced by either:

- Their relative GDG names (for example, DSN=TEST.GDG.DATASET(0)).
- As GDG ALLs (for example, DSN=TEST.GDG.DATASET)

For example, the initiator issues the ENQ for the GDG base name, TEST.GDG.DATASET for the generation data sets shown in the previous list. Generation data sets referenced by either their relative GDG names or as GDG ALLs are processed this way because the initiator does not know which specific absolute generation names will be required. This is because the conversion from relative generation name to absolute generation name is done during the allocation for the step referencing the data set. The ENQ for the specific, absolute, generation (G0000V00) data set name or names is issued at the start of the step requesting the relative GDG or GDG ALL.

Note that the initiator does not issue an ENQ for the GDG base name for a generation data set that is referenced by its absolute GDG name. Instead it issues an ENQ for that specific G0000V00 data set name.

***References***

For information about tape data set processing, see *z/OS DFSMS Using Magnetic Tapes*.

## Syntax

```
{DISP=[status]                                                    }
{DISP=([status][,normal-termination-disp][,abnormal-termination-disp])}

DISP= ( [NEW] [,DELETE ] [,DELETE ] )
        [OLD] [,KEEP   ] [,KEEP   ]
        [SHR] [,PASS   ] [,CATLG  ]
        [MOD] [,CATLG  ] [,UNCATLG]
        [,  ] [,UNCATLG]
              [,       ]
```

- You can omit the parentheses if you code only the status subparameter.
- If you omit the status subparameter but code subparameters for normal or abnormal termination disposition, you must code a comma to indicate the absence of NEW. For example, DISP=(,KEEP) or DISP=(,CATLG,DELETE).
- If you omit the second subparameter but code the third, you must code a comma to indicate the absence of the second subparameter. For example, DISP=(OLD,,DELETE) or DISP=(,,KEEP).

## Subparameter definition

### Status subparameter

**NEW**
Indicates that a new data set is to be created in this step.

**OLD**
Indicates that the data set exists before this step and that this step requires exclusive (unshared) use of the data set.

If you specify DISP=OLD for an output tape data set and (1) the data set is not protected by RACF or a password or (2) the data set has no expiration date, the system does not verify the data set name in the header label.

**SHR**
Indicates that the data set exists before this step and that other jobs can share it, that is, use it at the same time. This subparameter can also be coded as SHARE.

If you specify DISP=SHR for an output tape data set and (1) the data set is not protected by RACF or a password or (2) the data set has no expiration date, the system does not verify the data set name in the header label.

**MOD**
Indicates one of the following:

- The data set exists and records are to be added to the end of it. The data set must be sequential.
- A new data set is to be created.

In either case, MOD specifies exclusive (unshared) use of the data set.

When the data set is opened, the read/write mechanism is positioned after the last sequential record for an existing data set or at the beginning for a new data set. For subsequent OPENs within the same step, the read/write mechanism is positioned after the last sequential record.

**Note:** You cannot specify DISP=MOD to extend an ISO/ANSI/FIPS Version 3 tape data set unless the ISO/ANSI/FIPS Version 3 label validation installation exit allows the extension. For information on using ISO/ANSI/FIPS Version 3 installation exits, see *z/OS DFSMS Using Magnetic Tapes*.

If the system cannot find volume information for the data set on the DD statement, in the catalog, or passed with the data set from a previous step, the system assumes that the data set is being created in this job step. For a new data set, MOD causes the read/write mechanism to be positioned at the beginning of the data set.

To use DISP=MOD to create a new data set, code one of the following:

- No VOLUME=SER or VOLUME=REF parameter on the DD statement. The data set must not be cataloged or passed from another job step.
- A VOLUME=REF parameter that refers to a DD statement that makes a nonspecific volume request. (A nonspecific volume request is a DD statement for a new data set that can be assigned to any volume or volumes.) If it is tape, the referenced DD must not be opened before the DD with VOL=REF. If it is opened later, it will get a different tape volume. If it is tape and the referenced DD is in a different step it must not be opened before the DD with VOL=REF. If the referenced DD is for a nonspecific volume in the same step then the DD with VOL=REF will always be nonspecific regardless of the order of OPENs in the step. For DASD, one of the following must also be true:
  - The DSNAME parameters in the two DD statements must be different.
  - The two DD statements must request different areas of the same ISAM data set.

- In the case of tape, if you do not specify an explicit volume serial number on the DD statement, then you cannot specify a file sequence number greater than 1 and the system requests the operator to mount a "scratch" tape.

  For a new generation of a generation data group (GDG) data set (where (+n) is greater than 0), you may code VOLUME=REF or VOLUME=SER.

  For an SMS-managed data set the system ignores the volume.

After the system chooses a volume for a new data set, if the system finds another data set with the same name on that volume, the system will try to allocate a different volume. However, SMS-managed data sets require unique data set names. If a new data set is chosen to be SMS-managed and an existing SMS-managed data set has the same name, the request fails.

In a JES3 system, if you code DISP=MOD for a multivolume data set and any of the volumes are JES3-managed, JES3 will not execute the job until all volumes, including scratch volumes being added, are allocated. Such a job will wait on the queue until all volumes are allocated.

To use DISP=MOD to extend an existing data set, code one of the following:

- If the data set is cataloged, do not code a VOLUME=SER or a VOLUME=REF parameter on the DD statement, but code DISP=MOD or DISP=(MOD,KEEP) to make sure that the catalog will be updated with any new volume serial numbers.
- If the data set did not exist at the beginning of the job, but was passed from a prior step and not cataloged, it will be deleted at step termination. However, if you want to keep the data set, you can code DISP=(MOD,KEEP).

## Normal termination disposition subparameter

**DELETE**

Indicates that the data set is no longer needed if this step terminates normally.

For a DASD data set, DELETE means that the space occupied by that data set is available for use by other data sets. The system will physically erase the data set itself *only* if the erase option of a security product, such as RACF, is in effect for this data set. If the erase option is not in effect, the data will remain on the DASD until overwritten by another data set. For information on how to set the erase option, see the documentation for the security product.

For a tape data set, DELETE does not physically erase the data from the tape volume. The data will remain on the tape until overwritten by another data set. If the tape volume is a public volume, specifying DELETE allows the system to reuse the tape volume for other data sets that require a public volume; the system may overwrite the data set.

**Note:** DELETE requests are always treated as requiring exclusive serialization, preventing other jobs using the data set until it is deleted. This occurs even when DISP=(SHR,DELETE) is coded. As such, data sets with DISP=(SHR,DELETE) will not have its control changed, even when DSENQSHR=ALLOW is specified on the JOB card or in the JOBCLASS.

**Existing data sets**:

- If you set a retention period on the DD RETPD parameter, an existing data set is deleted only if its retention period is passed; otherwise the data set is kept.

- If you set an expiration date on the DD EXPDT parameter, an existing data set is deleted if the expiration date has passed.

  If the storage administrator specified OVRD_EXPDT(YES) in the IGDSMSxx member of SYS1.PARMLIB, you can override the expiration date or retention period for SMS-managed data sets by specifying DELETE on the DD DISP parameter. In that case, the data set will be deleted whether or not the expiration date or the retention period has passed. See *z/OS MVS Initialization and Tuning Reference* for information on the IGDSMSxx parmlib member.

**New data sets**:

A new data set is deleted at the end of the step even though a retention period or expiration date is also specified. See the DD EXPDT or RETPD parameters.

If the system retrieves volume information from the catalog because the DD statement does not specify VOLUME=SER or VOLUME=REF, then DELETE implies UNCATLG: the system deletes the data set and removes its catalog entry.

**KEEP**

Indicates that the data set is to be kept on the volume if this step terminates normally.

Without SMS, only KEEP is valid for VSAM data sets. VSAM data sets should not be passed, cataloged, uncataloged, or deleted.

With SMS, all dispositions are valid for VSAM data sets; however, UNCATLG is ignored.

For new SMS-managed data sets, KEEP implies CATLG.

**PASS**

Indicates that the data set is to be passed for use by a subsequent step in the same job.

With SMS, the system replaces PASS with KEEP for existing VSAM data sets. When you refer to the data set later in the job, the system obtains data set information from the catalog.

**Note:**

1. A data set can be passed only within a job.
2. If you specify DISP=(NEW,PASS) but, at the end of the job, one or more data sets were not received by any job step, then the maximum number of DD statements you can specify decreases by one. (The size of the TIOT controls how many DD statements are allowed per job step.) For example, if the current limit is 1635 DD statements, you can specify DISP=(NEW,PASS), and up to 1634 DD statements.
3. Coding PASS does not ensure that the operator will not unload the volume or that the system will not demount it to accommodate another job step allocation. Either can occur when the device on which the volume is mounted is not allocated to the job step that specified PASS or, for unlabeled tapes, when the volume requires verification. If the system does demount a volume for which RETAIN was requested, it will do so by issuing message IEF234E R (retain) for that volume. When the system reaches the next step requiring that volume, it will request the operator to remount the volume on an available device of the appropriate type.

**CATLG**
Indicates that, if the step terminates normally, the system is to place an entry pointing to the data set in the catalog. Note that the data set is kept.

An unopened tape data set is cataloged, unless the volume request is nonspecific or unless the data set is allocated to a dual-density tape drive but no density is specified. A nonspecific volume request is a DD statement for a new data set that can be assigned to any volume or volumes.

For information about the rules for cataloged data set names, see *z/OS DFSMS Access Method Services Commands*.

**UNCATLG**
Indicates that, if the step terminates normally, the system is to delete (1) the entry pointing to the data set in the catalog and (2) unneeded indexes, except for the highest level entry. Note that the data set is kept.

With SMS, UNCATLG is ignored for SMS-managed data sets and VSAM data sets (KEEP is implied).

By default, if the system retrieves volume information from the catalog, UNCATLG is processed. If not, the UNCATLG request is rejected and UNCATLG is treated as KEEP. Volume information is not retrieved from the catalog when VOLUME=REF or VOLUME=SER is specified on the DD statement, or for new data sets. However, if the ALLOCxx Parmlib setting for SYSTEM VERIFY_UNCAT is TRACK, MSGTRACK, or LOGTRACK, the UNCATLG request is honored and the data set is uncataloged regardless of whether or not the volume information in the catalog matches the volume information that is used for the dataset.

## Abnormal termination (conditional) disposition subparameter

**DELETE**
Indicates that the data set's space on the volume is to be released if this step terminates abnormally. The space can be used for other data sets; the data set is not erased from the space.

**Note:** DELETE requests are always treated as requiring exclusive serialization, preventing other jobs using the data set until it is deleted. This occurs even when DISP=(SHR,DELETE) is coded. As such, data sets with DISP=(SHR,DELETE) will not have its control changed, even when DSENQSHR=ALLOW is specified on the JOB card or in the JOBCLASS.

**Existing data sets**:

- If you set a retention period on the DD RETPD parameter, an existing data set is deleted only if its retention period is passed; otherwise the data set is kept.
- If you set an expiration date on the DD EXPDT parameter, an existing data set is deleted if the expiration date has passed.

  You can override the expiration date or retention period for SMS-managed DASD data sets using the OVRD_EXPDT(YES) parameter in the IGDSMSxx SYS1.PARMLIB member. In that case, the data set will be deleted whether or not the data set has expired or the retention period has passed. See *z/OS MVS Initialization and Tuning Reference* for information on the IGDSMSxx parmlib member.

**New data sets**:

A new data set is deleted at the end of the step even though a retention period or expiration date is also specified. See the DD EXPDT or RETPD parameters.

If the system retrieves volume information from the catalog because the DD statement does not specify VOLUME=SER or VOLUME=REF, then DELETE implies UNCATLG: the system deletes the data set and removes its catalog entry.

For a cataloged, passed data set, the catalog is not updated.

**KEEP**
Indicates that the data set is to be kept on the volume if this step terminates abnormally.

Without SMS, only KEEP is valid for VSAM data sets. VSAM data sets should not be passed, cataloged, uncataloged, or deleted.

With SMS, all dispositions are valid for VSAM data sets; however, UNCATLG is ignored.

For new SMS-managed data sets, KEEP implies CATLG.

**CATLG**
Indicates that, if the step terminates abnormally, the system is to place an entry pointing to the data set in the catalog. Note that the data set is kept.

An unopened tape data set is cataloged, unless the volume request is nonspecific or unless the data set is allocated to a dual-density tape drive but no density is specified.

For a cataloged, passed data set, the catalog is not updated. A passed, not received data set is not cataloged if the data set name has a first-level qualifier of a catalog name or alias.

**UNCATLG**
Indicates that, if this step terminates abnormally, the system is to delete (1) the entry pointing to the data set in the catalog and (2) unneeded indexes, except for the highest level entry. Note that the data set is kept.

For a cataloged, passed data set, the catalog is not updated.

With SMS, UNCATLG is ignored for SMS-managed data sets and VSAM data sets (KEEP is implied).

By default, if the system retrieves volume information from the catalog, UNCATLG is processed. If not, the UNCATLG request is rejected and UNCATLG is treated as KEEP. Volume information is not retrieved from the catalog when VOLUME=REF or VOLUME=SER is specified on the DD statement, or for new data sets. However, if the ALLOCxx Parmlib setting for SYSTEM VERIFY_UNCAT is TRACK, MSGTRACK, or LOGTRACK, the UNCATLG request is honored and the data set is uncataloged regardless of whether or not the volume information in the catalog matches the volume information that is used for the dataset.

## Defaults

- If you omit the status subparameter, the default is NEW.
- If you omit the normal termination disposition subparameter, the default is DELETE for a NEW data set or KEEP for an existing data set.
- If you omit the abnormal termination disposition subparameter, the default is the disposition specified or implied by the second subparameter. However, if the second subparameter specified PASS, the default abnormal termination disposition is DELETE for a NEW data set or KEEP for an existing data set.

- If you omit the DISP parameter, the default is a NEW data set with a disposition of DELETE for both normal and abnormal termination disposition. Thus, you can omit the DISP parameter for a data set that is created and deleted during a step.

## Relationship to other parameters

Do not code the following parameters with the DISP parameter.

```
*
BURST
CHARS
COPIES
DATA
DDNAME
DYNAM
FLASH
MODIFY
QNAME
SYSOUT
```

## Disposition of QSAM data sets

Do not code DISP=MOD if the data control block (DCB) specifies RECFM=FBS and the data set is processed by QSAM. If you do and a block is shorter than the specified block size, QSAM assumes that the short block is the last block and starts end-of-file processing. By this action, QSAM can embed short blocks in your data set and so affect the number of records per track.

## Disposition of generation data sets

See Topic "VSAM Data Sets" in *z/OS MVS JCL User's Guide* for additional information about disposition processing for generation data sets.

## Disposition of temporary data sets

Specify a normal termination disposition of PASS or DELETE for a temporary data set or for a data set with a system-generated name, that is, when a DSNAME parameter is omitted from the DD statement.

For a temporary data set name, the system ignores any abnormal termination disposition specified in the third subparameter and always PASSes the data set to subsequent steps.

## Disposition of partitioned data sets (PDSs and PDSEs)

When you specify DISP=MOD or DISP=NEW for a partitioned data set (PDS) or partitioned data set extended (PDSE), and you also specify a member name in the DSNAME parameter, the member name must not already exist. If the member name already exists, the system terminates the job.

When you specify DISP=OLD for a PDS or a PDSE, and you also specify a member name in the DSNAME parameter, the data set must already exist. If the member name already exists and the data set is opened for output, the system replaces the existing member with the new member. If the member name does not already exist and the data set is opened for output, the system adds the member to the data set.

When you specify DISP=MOD for a PDS or a PDSE, and you do not specify a member name, the system positions the read/write mechanism at the end of the data set. The system does not make an automatic entry into the directory.

When you specify DISP=MOD for a PDS or a PDSE, and you do specify a member name, the system positions the read/write mechanism at the end of the data set. If the member name already exists, the system terminates the job.

When you specify DISP=SHR for a partitioned data set extended (PDSE) and also specify a member name, then:

- If the member name exists, the member can have one writer or be shared by multiple readers, or
- If the member name does not exist, the member can be added to the data set. Thus, multiple jobs can access different members of the data set and add new members to the data set concurrently — but concurrent update access to a specific member (or update and read by other jobs) is not valid.

## Adding a volume to a cataloged data set

If you want to add a volume to a cataloged data set and have it properly cataloged after it is kept or passed, code the volume count subparameter of the VOLUME parameter to make the system use the values in the system catalog to process the data set. The following DD statement shows how to keep and extend a cataloged data set using the system catalog. Assume that this data set was created with a volume count of 2.

```
//DDEX2 DD DSNAME=OPER.DATA,DISP=(MOD,KEEP),
//            VOLUME=(,,,3),UNIT=(,P)
```

The VOLUME parameter references the system catalog for volume information about the data set and increases the maximum number of volumes for OPER.DATA. Because the UNIT parameter requests parallel mounting, the system must allocate the same number of units as the number of volumes in the VOLUME parameter; in this case, 3.

The following is an example of the messages in the job log after the job completes.

```
IEF285I    OPER.DATA                                          KEPT
IEF285I    VOL SER NOS= 333001,333002,333003.
IEF285I    OPER.DATA                                          RECATALOGED
IEF285I    VOL SER NOS= 333001,333002,333003.
```

***Non-SMS-managed data sets:*** If you do not reference the catalog when adding a volume to a cataloged data set, the system does not update the catalog with the newly referenced volumes.

## DISP=MOD for a multivolume data set

### Minimizing tape mounts

When you code DISP=MOD and the volume information is for a multivolume data set, normally the first volume(s) will be mounted on the devices(s) allocated. Then, if the data set is opened for output, OPEN starts with the last volume. If the number of tape volumes is more than the number of allocated devices, the system asks the operator to demount the first volume(s) and mount the last. To have the last tape volume mounted without first mounting and then demounting the earlier volume(s), code VOLUME=REF or DEFER in the UNIT parameter, or a volume sequence number in the VOLUME parameter.

### Determining the last volume

If a data set that is not a striped data set resides on multiple volumes, you can code a volume sequence number to specify the volume on which reading or writing is to begin. If you do not code a volume sequence number and the data set

is not striped, the system must identify the volume that contains the logical end of the data. Data might not have been written on all the volumes. After the system identifies the last volume, it positions the read/write mechanism on that volume.

In DASD and tape data set labels there is an indicator on the last volume containing user data. When you do not specify a volume sequence number, the system looks in the data set label for the indicator that identifies the last volume, and then selects the volume on which to begin writing as follows:

In the DD: DISP section "How the System Determines the Last Volume", SMS-managed DASD and Non-SMS-managed DASD processing are described separately. The Non-SMS-managed DASD description can be deleted. The SMS-managed DASD subtitle can be changed to be merely DASD.

***DASD:*** The system tests the data set label on the first volume in the list. If the label indicates it contains the end of the data set, the system selects that volume. Otherwise, it checks each subsequent volume until it finds one that has a last-volume indicator. (To begin writing, the system will not select later volumes that might also have the last-volume indicator by virtue of having previously contained parts of the data set.)

***Tape:*** See the information on Minimizing Tape Mounts.

## Extending on a volume other than the last

When you code DISP=MOD for a multivolume data set, use the volume count and volume sequence number subparameters of the VOLUME parameter if you want to keep the system from positioning the read/write mechanism after the last record on the last volume. For example:

```
//DDEX1  DD  DSNAME=OPER.DATA,DISP=(MOD,KEEP),VOLUME=(,,1,2)
```

The volume sequence number of 1 specifies that you want to use the first volume, and the volume count of 2 specifies that the data set requires two volumes.

## Effect of DCB=dsname parameter

If the DCB parameter refers to a cataloged data set, the system obtains the volume sequence number from the label of the data set, unless the volume sequence number is coded on the DD statement.

Thus, for the following DD statement, even though DISP=MOD is specified, the system positions the read/write mechanism after the last record on the volume specified in the volume sequence number in the label; this volume may or may not be the last volume.

```
//DD1   DD   DSNAME=MULTI1,DISP=MOD,DCB=CATDD
```

To control which volume is processed, code a volume sequence number.

```
//DD2   DD   DSNAME=MULTI2,DISP=MOD,DCB=CATDD,VOLUME=(,,2)
```

## Summary of disposition processing

*Table 17. Summary of Disposition Processing*

| DISP subparameters: | | | Disposition (if data set was allocated): | | | |
|---|---|---|---|---|---|---|
| **Status** | **Normal termination disposition** | **Abnormal termination disposition** | **At normal end of step** | **At abnormal end of step** | | **At End of Job** |
| | | | | **Step abnormally terminated** | **If later allocation failed in step** | |
| NEW permanent data set or MOD treated as new | none | none | deleted | deleted | deleted | |
| | KEEP | | kept | kept | | |
| | DELETE | | deleted | deleted | | |
| | CATLG | | cataloged | cataloged | | |
| | PASS | | passed | passed | passed | deleted |
| | PASS | DELETE<br>KEEP<br>CATLG<br>UNCATLG | passed | passed | passed | If all steps terminated normally: deleted<br>If a step terminated abnormally: third subparameter disposition |
| | DELETE<br>KEEP<br>CATLG<br>UNCATLG | KEEP | second subparameter disposition | kept | deleted | |
| | | DELETE | | deleted | | |
| | | CATLG | | cataloged | | |
| NEW temporary data set | none | DELETE<br>KEEP<br>CATLG<br>UNCATLG | deleted | deleted | deleted | |
| | DELETE | | | | | |
| | PASS | | passed | | | deleted |
| NEW data set in step to be automatically restarted | DELETE<br>KEEP<br>PASS<br>CATLG<br>UNCATLG | DELETE<br>KEEP<br>CATLG<br>UNCATLG | | deleted | | |
| NEW data set in step to be restarted at checkpoint | DELETE<br>KEEP<br>PASS<br>CATLG<br>UNCATLG | DELETE<br>KEEP<br>CATLG<br>UNCATLG | | kept, if being used when checkpoint was taken | | |
| OLD or SHR or MOD treated as old | none | none | kept | kept | kept | |
| | KEEP | | | | | |
| | DELETE | | deleted | deleted | | |
| | CATLG | | cataloged or, if new volumes were added, recataloged | cataloged or, if new volumes were added, recataloged | | |
| | UNCATLG | | uncataloged | uncataloged | | |
| | PASS | | passed | passed | passed | kept |

*Table 17. Summary of Disposition Processing (continued)*

| DISP subparameters: | | | Disposition (if data set was allocated): | | | |
|---|---|---|---|---|---|---|
| **Status** | **Normal termination disposition** | **Abnormal termination disposition** | **At normal end of step** | **At abnormal end of step** | | **At End of Job** |
| | | | | **Step abnormally terminated** | **If later allocation failed in step** | |
| OLD or SHR or MOD treated as old (continued) | PASS | DELETE<br>KEEP<br>CATLG<br>UNCATLG | passed | passed | passed | If all steps terminated normally: kept, if originally old; deleted, if originally new<br>If a step terminated abnormally: third subparameter disposition |
| | DELETE<br>KEEP<br>CATLG<br>UNCATLG | KEEP | second parameter disposition | kept | kept, if step was receiving originally old data set; deleted, if step was receiving originally new data set | |
| | | DELETE | | deleted | | |
| | | CATLG | | cataloged or, if new volumes were added, recataloged | | |
| | | UNCATLG | | uncataloged | | |
| OLD permanent data set passed to this job step | none | none | deleted, if data set was originally new; kept, if originally old | deleted, if data set was originally new; kept, if originally old | | |
| OLD data set in step to be automatically restarted | DELETE<br>KEEP<br>PASS<br>CATLG<br>UNCATLG | DELETE<br>KEEP<br>CATLG<br>UNCATLG | | kept | | |
| OLD data set in step to be restarted at checkpoint | DELETE<br>KEEP<br>PASS<br>CATLG<br>UNCATLG | DELETE<br>KEEP<br>CATLG<br>UNCATLG | | kept, if being used when checkpoint was taken | | |

## Examples of the DISP parameter

***Example 1***

```
//DD2  DD  DSNAME=FIX,UNIT=3420-1,VOLUME=SER=44889,
//         DISP=(OLD,,DELETE)
```

DD statement DD2 defines an existing data set and implies by the omitted second subparameter that the data set is to be kept if the step terminates normally. The statement requests that the system delete the data set if the step terminates abnormally.

***Example 2***

```
//STEPA  EXEC PGM=FILL
//DD1    DD   DSNAME=SWITCH.LEVEL18.GROUP12,UNIT=3390,
//        VOLUME=SER=LOCAT3,SPACE=(TRK,(80,15)),DISP=(,PASS)
//STEPB  EXEC PGM=CHAR
//DD2    DD   DSNAME=XTRA,DISP=OLD
//DD3    DD   DSNAME=*.STEPA.DD1,DISP=(OLD,PASS,DELETE)
//STEPC  EXEC PGM=TERM
//DD4    DD   DSNAME=*.STEPB.DD3,DISP=(OLD,CATLG,DELETE)
```

DD statement DD1 defines a new data set and requests that the data set be passed. If STEPA abnormally terminates, the data set is deleted because it is a new data set, the second subparameter is PASS, and an abnormal termination disposition is not coded.

DD statement DD3 in STEPB receives this passed data set and requests that the data set be passed. If STEPB abnormally terminates, the data set is deleted because of the third subparameter of DELETE.

DD statement DD4 in STEPC receives the passed data set and requests that the data set be cataloged at the end of the step. If STEPC abnormally terminates, the data set is deleted because of the abnormal termination disposition of DELETE.

DD statement DD2 defines an old data set named XTRA. When STEPB terminates, normally or abnormally, this data set is kept.

***Example 3***

```
//SMSDD5 DD  DSNAME=MYDS5.PGM,DATACLAS=DCLAS05,STORCLAS=SCLAS05,
//          DISP=(NEW,KEEP)
```

DD statement SMSDD5 defines a new SMS-managed data set and requests that the data set be kept (which implies that it be cataloged).

***Example 4***

```
//SMSDD7 DD  DSNAME=MYDS7.PGM,DISP=(OLD,UNCATLG)
```

DD statement SMSDD7 defines an existing SMS-managed data set (the data set had been assigned a storage class when it was created) and requests that the data set be uncataloged. However, the data set is kept because UNCATLG is ignored for SMS-managed data sets.

## DLM parameter

***Parameter Type***

Keyword, optional

***Purpose***

Use the DLM parameter to specify a delimiter to terminate this in-stream data set. When the DLM parameter assigns a different delimiter, the in-stream data records can include standard delimiters, such as /* and //, in the data.

**In a JES2 system**, when the DLM delimiter appears on a DD * statement, either the assigned delimiter or // ends the input data set. When the DLM delimiter appears on a DD DATA statement, only the assigned delimiter ends the input data set.

**In a JES3 system**, when the DLM delimiter appears on either a DD * or DD DATA statement, only the assigned delimiter ends the input data set.

**Note:** When the DLM delimiter overrides any implied delimiter, you must terminate the data with the DLM characters. Otherwise, the system keeps reading until the reader is empty.

Except for the JES2 /*SIGNON and /*SIGNOFF statements, the system does not recognize JES2 and JES3 statements in an input stream between the DLM parameter and the delimiter it assigns. The JES2 /*SIGNON and /*SIGNOFF statements are processed by the remote work station regardless of any DLM delimiter.

***Considerations for an APPC Scheduling Environment***

The DLM parameter has no function in an APPC scheduling environment. If you code DLM, the system will check it for syntax and ignore it.

## Syntax

```
DLM=delimiter
```

- If the specified delimiter contains any special characters, enclose it in apostrophes. In this case, a special character is any character that is neither alphanumeric nor national ($, #, @).

  Failing to code enclosing apostrophes produces unpredictable results.
- If the delimiter contains an ampersand or an apostrophe, code each ampersand or apostrophe as two consecutive ampersands or apostrophes. Each pair of consecutive ampersands or apostrophes counts as one character.
- The DLM parameter can have a null value only when coded on a DD which either:
  - Overrides a DD in a procedure
  - Is added to a procedure.

## Subparameter definition

**`delimiter`**
Specifies two characters that indicate the end of this data set in the input stream.

## Default

If you do not specify a DLM parameter, the default is the /* delimiter statement.

If the system finds an error on the DD statement before the DLM parameter, it does not recognize the value assigned as a delimiter. The system reads records until it reads a record beginning with /* or //.

## Relationship to other parameters

Code the DLM parameter only on a DD statement with the * or DATA parameter.

The DLM parameter has meaning only on statements defining data in the input stream, that is, DD * and DD DATA statements. If DLM is specified on any other statement, a JCL error message is issued.

## Invalid delimiters

If the delimiter is not two characters:

- **For JES2**, the delimiter is not recognized. The in-stream data set is terminated when a record starting with // or /* is read. The system fails the job due to the invalid delimiter.
- **For JES3**, if an incorrect number of characters is coded, JES3 terminates the job.

## Example of the DLM parameter

```
//DD1  DD  *,DLM=AA
       .
       .
       data
       .
AA
```

The DLM parameter assigns the characters AA as the delimiter for the data defined in the input stream by DD statement DD1. For JES2, the characters // would also serve as valid delimiters since a DD * statement was used. JES3 accepts only the characters specified for the DLM parameter as a terminator for DD * or DD DATA.

# DSID Parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the DSID parameter to specify the data set identifier of an input or output data set on a diskette of the 3540 Diskette Input/Output Unit.

An input data set is read from a 3540 diskette by a diskette reader program, and an output data set is written on a 3540 diskette by a diskette writer, which is an external writer.

To read a data set from a 3540 diskette, the DD statement must contain:

- A DSID parameter.
- An * or DATA parameter, to begin the input stream data set.

To write a data set on a 3540 diskette, the DD statement must contain:

- A DSID parameter.
- A SYSOUT parameter that specifies the output class that the diskette writer processes and the name of the diskette writer.

Also, a system command, from the operator or in the input stream, must start the diskette writer before this DD statement is processed.

**Note:** The system ignores the DSID parameter on a DD *, DD DATA, or a DD statement with the SYSOUT parameter, except when a diskette reader or writer processes the JCL.

***References***

For more information about associated data sets, see *3540 Programmer's Reference*. For information about external writers, see *z/OS JES2 Initialization and Tuning Guide* or *z/OS JES3 Initialization and Tuning Guide*.

## Syntax

```
DSID=   {id      }
        {(id,[V])}
```

- You can omit the parentheses if you code only an id.
- Null positions in the DSID parameter are invalid.

## Subparameter definition

**id**
Specifies the data set identifier. The id is 1 through 8 characters. The characters must be alphanumeric, national ($, #, @), a hyphen, or a left bracket. The first character must be alphabetic or national ($, #, @).

**V** Indicates that the data set label must have been previously verified on a 3741 Data Station/Workstation. This subparameter is required only on a SYSIN DD statement.

## Relationship to other parameters

Do not code the following parameters with the DSID parameter.

```
BURST
CHARS
DDNAME
DYNAM
FLASH
MODIFY
MVSGP
QNAME
```

***For 3540 diskette input/output units:*** A DSID parameter on a DD *, DD DATA, or sysout DD statement is ignored except when detected by a diskette reader as a request for an associated data set. See *3540 Programmer's Reference*.

On a DD * or DD DATA statement processed by a diskette reader, you can specify DSID, VOLUME=SER, BUFNO, and LRECL to indicate that a diskette data set is to be merged into the input stream following the DD statement.

## Example of the DSID parameter

```
//JOB1     JOB   ,,MSGLEVEL=(1,1)
//STEP     EXEC  PGM=AION
//SYSIN    DD    *,DSID=(ABLE,V),VOLUME=SER=123456,
//             DCB=LRECL=80
//SYSPRINT DD    SYSOUT=E,DCB=LRECL=128,DSID=BAKER
```

In this example, the SYSIN DD statement indicates that the input is on diskette 123456 in data set ABLE and must have been verified. The output will be written on a diskette in data set BAKER.

# DSNAME parameter

***Parameter Type***

Keyword, optional

***Purpose***

Use the DSNAME parameter to specify the name of a data set. For a new data set, the specified name is assigned to the data set; for an existing data set, the system uses the name to locate the data set.

***References***

Data sets are described in *z/OS DFSMS Using Data Sets*.

The names of all data sets that are to be cataloged or SMS-managed must conform to the rules for cataloged data set names. For information about the rules for cataloged data set names, refer to *z/OS DFSMS Using Data Sets*.

## Syntax

```
 {DSNAME}  =name
 {DSN   }

name for permanent data set:

     dsname
     dsname(member)
     dsname(generation)

name for temporary data set:

     &&dsname
     &&dsname(member)

name for in-stream or sysout data set:

     &&dsname

name copied from earlier DD statement:

     *.ddname
     *.stepname.ddname
     *.stepname.procstepname.ddname

name for dummy data set:

     NULLFILE
```

- You can abbreviate DSNAME as DSN.
- Avoid starting a data set name with JES or SYS1. The system uses these characters for system data sets.
- If the data set name begins with a blank character, the system assigns the data set with a unique temporary data set name, and ignores the name specified on the DSNAME parameter
- The system ignores blank characters at the end of a data set name.
- Blanks can be included in a data set name if the name is enclosed in apostrophes, such as DSNAME='AB CD'. However, do not code blanks in the name for an in-stream or sysout data set; for example, SYSOUT=P,DSNAME='&&AB CD' is not valid.
- If the data set is to be managed through SMS, you cannot enclose the data set name in apostrophes. However, the following exception applies: You can enclose the data set name on the DSNAME parameter in apostrophes if the data set is to be assigned to, or already resides on, an SMS-managed mountable tape volume.
- Any data set name enclosed in apostrophes on the DSNAME parameter will be treated as an unqualified name. Data sets with an unqualified name cannot be cataloged.
- The system does not check data set names enclosed in apostrophes for valid characters or valid length. When SMS is not installed or active incorrect characters or length result in data set allocation, but the data set is not cataloged. When SMS is active, it will fail the job for incorrect characters or length.

**Non-significant special characters**: When a data set name contains special characters that are not significant to the system, other than hyphens, enclose it in apostrophes. For example, DSNAME='DS/29'.

Code each apostrophe that is part of the data set name as two consecutive apostrophes. For example, code DAYS'END as DSNAME='DAYS''END'.

The system ignores blank characters at the end of a data set name, even if the data set name is enclosed in apostrophes.

**Significant special characters**: The following special characters are significant to the system. Do not enclose them in apostrophes.

- Periods to indicate a qualified data set name. However, you must enclose in apostrophes a period immediately before a right parenthesis, immediately after a left parenthesis, or immediately before a comma; for example, DSNAME='(.ABC)' and DSNAME='(ABC.)' and DSNAME='A.B.C.'.
- Double ampersands to identify a temporary data set name. Note that if you use apostrophes, DSNAME='&&AB' and DSNAME='&AB' refer to the same data set.
- Double ampersands to identify an in-stream or sysout data set name.
- Parentheses to enclose the member name of a partitioned data set (PDS) or partitioned data set extended (PDSE), or the generation number of a generation data set.
- Plus (+) or minus (-) sign to identify a generation of a generation data group.
- The asterisk to indicate a backward reference.

On a DD statement in a cataloged or in-stream procedure, if the data set name is a symbolic parameter, do not enclose it in apostrophes. If it is enclosed in apostrophes, the system performs correct substitution only if the symbolic parameter enclosed in apostrophes is preceded by a symbolic parameter not enclosed in apostrophes.

The data set name should not contain the 44 special characters (X'04') created by the 12-4-9 multiple punch or any operation that converts the value of characters to X'04'.

## Subparameter definition

The data set names you specify on DSNAME are described in the following topics:

- Data Set Name for Permanent Data Set

- Data Set Name for Temporary Data Set
- Data Set Name for In-Stream or Sysout Data Set
- Data Set Name Copied from Earlier DD Statement
- Data Set Name for Dummy Data Set

## Data set name for permanent data set

Assign a permanent data set either an unqualified or a qualified name:

## Unqualified name

1 through 8 alphanumeric or national ($, #, @) characters, a hyphen, or a character X'C0'. The first character must be alphabetic or national ($, #, @). For example, DSNAME=ALPHA is an unqualified data set name.

For the characters allowed in ISO/ANSI/FIPS tape data set names, see information about label definition and organization in *z/OS DFSMS Using Magnetic Tapes*.

## Qualified name

Multiple unqualified names joined by periods. Each qualifier is coded like an unqualified name; therefore, the name must contain a period after every 8 characters or fewer. For example, DSNAME=ALPHA.PGM is a qualified data set name. The maximum length of a qualified data set name is:

- 44 characters, including periods.
- For a generation data group, 35 characters, including periods.
- For an output tape data set, 17 characters, including periods. If longer than 17 characters, only the rightmost 17 characters, excluding trailing blanks, are written to the tape header label. For more information, see *z/OS DFSMS Using Magnetic Tapes* .

## Name for RACF-protected data set

The OS/390 Security Server, which includes RACF, expects the data set name to have a high-level qualifier that is defined to RACF. See the *z/OS Security Server RACF Security Administrator's Guide* for details. RACF uses the entire data set name, from 1 through 44 characters, when protecting a tape data set.

## Cataloged data set name

For information about the rules for cataloged data set names, see *z/OS DFSMS Access Method Services Commands*.

**dsname**
Specifies a data set name.

**dsname(member)**
Specifies the name of the permanent partitioned data set (PDS) or the partitioned data set extended (PDSE), and the name of a member within that data set. If the member does not exist and DISP=OLD or DISP=SHR is specified, the allocation will succeed, but the job will fail when the data set is opened for input. If the member does not exist and the data set is opened for output, the system will add the member to the data set.

**member**
1 to 8 alphanumeric or national characters, or a character X'C0'. The first character must be alphabetic, national, +, or -. If the first character is + or -, the member is a part of a generation data group.

**dsname(generation)**
Specifies the name of a generation data group (GDG) and the generation number (zero or a signed integer) of a generation data set within the GDG.

**Note:** A VSAM data set cannot be a generation data set.

**generation**

- The first character of a relative generation number is +, -, or 0.
- All characters of a relative generation number that follow the +, -, or 0 must be numeric (0 through 9).
- The numeric portion (not + or -) of a relative generation number must be expressed in 1 to 3 numeric characters. For example, +100, -002, +4, -09, 000.
- A relative generation number cannot exceed 255.

To retrieve all generations of a generation data group, omit the generation number.

## Data set name for temporary data set

A temporary data set is a data set that you create and delete within a job. (For information about coding data set names with the DD *, DATA, and SYSOUT parameters, see "Data Set Name for In-Stream or Sysout Data Set.")

**Note:** SMS manages a temporary data set if you specify a storage class (with the DD STORCLAS parameter) or if an installation-written automatic class selection (ACS) routine selects a storage class for the temporary data set.

When you define a temporary data set, you can code the DSNAME parameter or omit it; in either case, the system generates a qualified name for the temporary data set.

When you use DSNAME for a temporary data set, code the name as two ampersands (&&) followed by a character string 1 to 8 characters in length:

- The first character following the ampersands must be alphabetic or national.
- The remaining characters must be alphanumeric or national.

The format of the qualified name the system generates depends on whether or not you specified a data set name on the DSNAME parameter:

- All temporary data set names begin as follows:

  `SYSyyddd.Thhmmss.RA000.jjobname`

  where:

  **yy** indicates the year

  **ddd** indicates the Julian day

  **hh** indicates the hour

  **mm** indicates the minute

  **ss** indicates the second

  **jjobname**
  indicates the name of the job

  Date fields and time fields in the system-generated name reflect:

  – For JCL allocations, when the job was processed by the Interpreter. (For JES2 this is when an initiator selects the job for execution. For JES3 this is at CI time.)

  – For dynamic allocations, when the dynamic allocation request was issued.

- If you do not specify a data set name, or TEMPDSFORMAT(UNIQUE) is in effect, the full format of the temporary data set name is:

  ```
  SYSyyddd.Thhmmss.RA000.jjobname.Rggnnnnn
  ```

  where:

  **gg** 01, or, in a sysplex:
  - for JCL allocations, the system identifier of the system that interpreted the job.
  - for dynamic allocations, the system identifier of the system on which the job executed.

  **nnnnn**
  a number that is unique within a system
- If you do specify a data set name and TEMPDSFORMAT(INCLUDELABEL) is in effect, the full format of the temporary data set name is:

  ```
  SYSyyddd.Thhmmss.RA000.jjobname.dsetname.Hgg
  ```

  where:

  **dsetname**
  the 1 to 8 character DSNAME coded following the two ampersands (&&)

  **gg** 01, or, in a sysplex:
  - for JCL allocations, the system identifier of the system which interpreted the job.
  - for dynamic allocations, the system identifier of the system on which the job executed.

If you use DSNAME, note that the system-generated qualified name for the temporary data set will **not** be unique under the following conditions:

- Multiple tasks or APPC transactions having identical jobnames execute at exactly the same time, and
- The tasks or transactions contain DD statements with identical temporary data set names.

To ensure that a temporary data set name is unique, do not code a temporary data set name. Allow the system to assign one.

Only the job that creates a temporary data set has access to it to read and write data and to scratch the data set.

**Note:**

1. In general, the system treats a single ampersand (&) followed by a character string of 1 to 8 characters as a symbolic parameter. (See "Using system symbols and JCL symbols" on page 38.) However, if you code a data set name as a symbolic parameter (by coding DSNAME=&xxxxxxxx), and do not assign a value to or nullify the symbolic parameter, the system will process it as a temporary data set name.
2. The SYSTEM TEMPDSFORMAT(UNIQUE|INCLUDELABEL) option in the parmlib member ALLOCxx enables allocation to use a more unique format for the data set name when DSN=&&mydsn is specified. The unique data set name allows jobs with the same data set names specified to run at the same time, without requiring the JCL programmer to remove the DSN=&&mydsn or to add data set name referback syntax. The system setting for this option may affect the data set name generated for a temporary data set.

**&&dsname**
Specifies the name of a temporary data set.

**&&dsname(member)**
Specifies the name of a temporary partitioned data set (PDS) or partitioned data set extended (PDSE) and a member within that data set.

**member**
1 - 8 alphanumeric or national characters, or a character X'C0'. The first character must be alphabetic or national.

## Data set name for in-stream or sysout data set

Use the DSNAME parameter to assign a data set name to an in-stream data set (defined with the DD * or DD DATA parameter) and to a sysout data set (defined with the DD SYSOUT parameter). When defining an in-stream or sysout data set, you can code the DSNAME parameter or omit it; if omitted, the system generates a name for the data set.

The data set name for in-stream and sysout data sets consists of two ampersands (&&) followed by one through eight 8 alphanumeric or national ($, #, @,) characters, a hyphen, or a character X'C0'. The first character following the ampersands must be alphabetic or national ($, #, @).

The system generates a qualified name for the in-stream or sysout data set. The qualified name contains:

- The userid of the job
- The jobname
- The jobid
- A system-assigned identifier
- The data set name from the DSNAME parameter (if DSNAME is coded), or a question mark (?) if DSNAME is not coded.

The format of the name is:

```
userid.jobname.jobid.Ddsnumber.name
```

where name is the dsname or a question mark (?).

When the system checks a user's authority to access a SYSOUT data set, the check is made against the JESSPOOL class using the fully qualified name, preceded by the node name and a period:

```
nodename.userid.jobname.jobid.Ddsnumber.name
```

Profiles of this format may be defined in your security system to allow other users access to your SYSOUT data sets. A profile is not necessary for you to access your own data sets.

**Note:** A single ampersand before a data set name in a cataloged or in-stream procedure signifies a symbolic parameter. However, if no value is assigned to the name on either the EXEC statement that calls the procedure, a PROC statement in the procedure, or a previous SET statement, the system treats the name as the last qualifier of the data set name for an in-stream or sysout data set.

**&&dsname**
Specifies the last qualifier of the system-generated data set name for an in-stream or sysout data set.

### Data set name copied from earlier dd statement

A backward reference is a reference to an earlier statement in the job or in a cataloged or in-stream procedure called by this or an earlier job step. A backward reference can be coded in the DSNAME parameter to copy a data set name from an earlier DD statement.

When copying the data set name, the system also copies the following from the DD statement:

- Whether or not the data set is a PDS or a PDSE.
- Whether or not the data set is a temporary data set.

***.ddname**
Asks the system to copy the data set name from earlier DD statement ddname.

***.stepname.ddname**
Asks the system to copy the data set name from DD statement, ddname, in an earlier step, stepname, in the same job.

***.stepname.procstepname.ddname**
Asks the system to copy the data set name from a DD statement in a cataloged or in-stream procedure. Stepname is the name of this job step or an earlier job step that calls the procedure, procstepname is the name of the procedure step that contains the DD statement, and ddname is the name of the DD statement.

### Data set name for dummy data set

**NULLFILE**
Specifies a dummy data set. NULLFILE has the same effect as coding the DD DUMMY parameter. NULLFILE must be coded as a single-word parameter. For instance, IBM does not support the use of NULLFILE to obtain a dummy data set for these (or other) formats:

- When followed by a member name
- As a qualifier in a qualified data set name
- As a temporary data set name.

## Relationship to other parameters

Do not code the following parameters with the DSNAME parameter.

```
DDNAME
DYNAM
QNAME
```

Do not code the DCB IPLTXID subparameter with the DSNAME parameter.

***Reserved Data Set Names:*** Do not code the following data set names on the DSNAME parameter with the *, DATA, or SYSOUT parameter (an in-stream or sysout data set); the names are reserved for system use.

```
JESJCL
JESJCLIN
JESMSGLG
JESYSMSG
```

***With DD AMP parameter:*** When you code an AMP parameter for a VSAM data set, do not code a DSNAME:

- That contains parentheses, a minus (hyphen), or a plus (+) sign.
- That is in the form for ISAM.
- That is in the form for PAM (partitioned access method).

- That names a generation data group.

***With DD DISP parameter:*** You can create a permanent data set by specifying a qualified or unqualified data set name, the disposition must be other than DELETE.

The following example illustrates how to create a permanent data set:

```
//REPORT   DD DSN=DEHART.APAR.REPORT,SPACE=(CYL,(5,5)),
//            DISP=(NEW,CATLG),UNIT=SYSALLDA,
//            DCB=(LRECL=121,RECFM=FBA,BLKSIZE=1210)
```

You can create a temporary data set by specifying a:

- &&dsname or by omitting the DSNAME parameter
- Qualified or unqualified data set name and specifying, either explicitly or implicitly, DISP=(NEW,DELETE).

The following two examples illustrate how to create a temporary data set:

```
//MYDD1    DD DSN=TEMP1,UNIT=3480,DISP=(,DELETE),SPACE=(TRK,(1,1))
//DD2      DD UNIT=SYSALLDA,SPACE=(TRK,1),DISP=(NEW,PASS)
```

**Note:** When you code a disposition of CATLG for a data set, do not code a DSNAME name in apostrophes.

## Examples of the DSNAME parameter

***Example 1***

```
//DD1   DD  DSNAME=ALPHA,DISP=(,KEEP),
//        UNIT=3390,VOLUME=SER=389984
```

DD statement DD1 defines a new data set and names it ALPHA. DD statements in later job steps or jobs may retrieve this data set by specifying ALPHA in the DSNAME parameter, unit information in the UNIT parameter, and volume information in the VOLUME parameter.

***Example 2***

```
//DDSMS1   DD  DSNAME=ALPHA.PGM,DISP=(NEW,KEEP),DATACLAS=DCLAS1,
//          MGMTCLAS=MCLAS1,STORCLAS=SCLAS1
```

DD statement DDSMS1 defines a new SMS-managed data set and names it ALPHA.PGM. DD statements in later job steps or jobs may retrieve this data set by specifying ALPHA.PGM in the DSNAME parameter.

***Example 3***

```
//DD2   DD  DSNAME=LIB1(PROG12),DISP=(OLD,KEEP),UNIT=3390
//       VOLUME=SER=882234
```

DD statement DD2 retrieves member PROG12 from the partitioned data set named LIB1.

***Example 4***

```
//DDIN  DD  DATA,DSNAME=&&PAYIN1
        .
        data
        .
/*
```

DD statement DDIN defines PAYIN1 as the last qualifier of the system-generated data set name for the in-stream data set. This generates a data set name such as userid.jobname.jobid.Ddsnumber.PAYIN1.

***Example 5***

```
//DDOUT  DD  DSNAME=&&PAYOUT1,SYSOUT=P
```

DD statement DDOUT defines PAYOUT1 as the last qualifier of the system-generated data set name for the sysout data set. This generates a data set name such as userid.jobname.jobid.Ddsnumber.PAYOUT1.

***Example 6***

```
//DD3   DD  DSNAME=&&WORK,UNIT=3390
```

DD statement DD3 defines a temporary data set. Because the data set is deleted at the end of the job step, the DSNAME parameter can be omitted. The following example shows why a temporary data set should be named.

## DSNTYPE parameter

***Parameter Type***

Keyword, optional

***Purpose***

Use the DSNTYPE parameter to specify:

- A new partitioned data set (PDS)
- A new partitioned data set extended (PDSE), which is also called a *library*, and an optional version level for the new PDSE
- A new hierarchical file system (HFS) data set
- A first-in first-out (FIFO) special file, which is also called a *named pipe*
- A new basic format data set
- A new large format data set
- A new extended format data set and an optional version level for the new sequential data set.

Also use the DSNTYPE parameter to override the DSNTYPE attribute defined in the data class of the new data set.

Serialization of the data set can exist at both the data set (library) level and the member level. If you specify DISP=SHR on the DD statement for a PDSE, sharing of the data set applies to the data set and the individual member specified. Multiple jobs can access different members of the data set and create new members of the data set concurrently. However, concurrent update access to a specific member (or update and read by other jobs) is not allowed. Dispositions of DISP=OLD, NEW, or MOD result in exclusive use of the entire data set. A PDSE can be created through the BPAM, BSAM, and QSAM access methods.

If SMS is not active, the system checks the syntax and then ignores the DSNTYPE parameter.

An HFS data set is a data set used by z/OS UNIX System Services (z/OS UNIX) programs. It contains a mountable file system. It is a partitioned format data set, similar to a PDSE.

A FIFO special file is a type of file with the property that data written to such a file is read on a first-in-first-out basis. A FIFO special file defined in a DD statement provides a connection filled with data among programs. One or more programs can write data into the file; one or more programs can read the data.

***References***

For information on partitioned data sets and PDSEs, see *z/OS DFSMS Using Data Sets*. For information on HFS data sets and FIFO special files, see *z/OS UNIX System Services Planning* and the *z/OS V2R2.0 UNIX System Services User's Guide*.

## Syntax

```
DSNTYPE= {LIBRARY}
         {(LIBRARY,1)}
         {(LIBRARY,2)}
         {HFS    }
         {PDS    }
         {PIPE   }
         {EXTREQ }
         {EXTPREF}
         {LARGE  }
         {BASIC  }
```

## Subparameter definition

**LIBRARY**
Specifies a partitioned data set extended (PDSE). A PDSE can contain data and program object members. LIBRARY uses the PDSE_VERSION parameter in IGDSMS*xx* or its default to determine which version of PDSE to allocate.

**(LIBRARY,1)**
Specifies a version 1 partitioned data set extended (PDSE). A PDSE version 1 can contain data and program object members.

**(LIBRARY,2)**
Specifies a version 2 partitioned data set extended (PDSE). A PDSE version 2 can contain data and program object members. Version 2 offers more efficient directory usage.

**HFS**
Specifies an HFS data set. Specify HFS only when the DD statement also specifies a DSNAME parameter.

**PDS**
Specifies a partitioned data set (PDS). A PDS can contain data and load module members.

**PIPE**
Specifies a FIFO special file. Specify PIPE only when the DD statement also specifies a PATH parameter.

**EXTREQ | (EXTREQ,1) | (EXTREQ,2)**
Specifies for the data set to be extended format if the data set is VSAM or sequential, or if DSORG is omitted from all sources. (EXTREQ,1) specifies a version 1 extended format data set. (EXTREQ,2) specifies a version 2 extended

format data set. If the data set is not striped and has multiple volumes, the system uses FlashCopy for the data set. EXTREQ specifies for the system to select version 1 or 2 based on the PS_EXT_VERSION parameter in the IGDSMSxx member of SYS1.PARMLIB. The default value for PS_EXT_VERSION is 1.

**EXTPREF | (EXTPREF,1) | (EXTPREF,2)**
Specifies for the data set to be extended format if the data set is VSAM or sequential, or if DSORG is omitted from all sources. If extended format is not possible, the system selects basic format. (EXTPREF,1) specifies a version 1 extended format data set. (EXTREQ,2) specifies a version 2 extended format data set. If DFSMSdss is used to copy a version 1 extended format data set that is not striped and has multiple volumes, the system cannot use FlashCopy for the data set. If the extended format data set is version 2 and FlashCopy is available, the system can use FlashCopy. EXTPREF specifies for the system to select version 1 or 2 based on the PS_EXT_VERSION parameter in the IGDSMSxx member of SYS1.PARMLIB. The default value for PS_EXT_VERSION is 1.

**LARGE**
The system will select large format if the data set is sequential (DSORG=PS or PSU) or DSORG is omitted from all sources and the data set is not VSAM.

**BASIC**
The system will select basic format if the data set is sequential (DSORG=PS or PSU) or DSORG is omitted from all sources and the data set is not VSAM.

## Defaults

If you do not specify DSNTYPE, the type of data set is determined by other data set attributes, the data class for the data set, or an installation default.

DSNTYPE cannot default to HFS or PIPE. You must explicitly specify these attributes.

## Overrides

DSNTYPE overrides the DSNTYPE attribute in the DATACLAS parameter for the data set. See “Overrides” on page 134.

DSNTYPE on the DD statement overrides:

- The DSNTYPE for the data set referenced by LIKE
- The DD statement that is referenced by REFDD

The source of the DSNTYPE value is the DD statement or dynamic allocation, the data class, the data set referenced by the LIKE= parameter or the DD statement referenced by the REFDD= parameter. If the resulting DSNTYPE value is HFS or PIPE, conflicting parameters are overridden. If the DSNTYPE value is not HFS or PIPE, other parameters are not overridden.

If the new data set is sequential and on DASD, then a DSNTYPE value of BASIC, LARGE, EXTPREF or EXTREQ is effective. If the new data set is partitioned (by DSORG=PO or a directory size on the SPACE parameter), then a value of LIBRARY or PDS is effective. If you did not specify DSORG=PO or a directory size on SPACE, you will get a PDS or PDSE only if you set the HONOR_DSNTYPE_PDSE(YES) parameter in the IGDSMSxx member in SYS1.PARMLIB. The default value for this parameter is HONOR_DSNTYPE_PDSE(NO).

## Relationship to other parameters

Do not code the following DD parameters with the DSNTYPE parameter.

```
*
AMP
DATA
DDNAME
DYNAM
QNAME
```

You can only code the RECORG DD parameter with the DSNTYPE=EXTREQ or DSNTYPE=EXTPREF parameter. This is because RECORG is used to define VSAM data sets and all other DSNTYPE specifications imply other (non-VSAM) formats.

## Examples of the DSNTYPE parameter

***Example 1***

```
//NEWPDSE  DD  DSNAME=FILEA.ABC(REC1),DISP=(NEW,KEEP)
```

In the example, the NEWPDSE DD statement defines member REC1 in the new PDSE named FILEA.ABC. Note that installation-written ACS routines select the data class (which specifies LIBRARY for DSNTYPE), management class, and storage class for the data set.

***Example 2***

```
//NEWA   DD  DSNAME=REPORT.ONE(WEEK1),DISP=(NEW,KEEP),
//       DATACLAS=DCLAS09,DSNTYPE=LIBRARY
```

In the example, the NEWA DD statement defines member WEEK1 in the new PDSE named REPORT.ONE. DSNTYPE=LIBRARY overrides the DSNTYPE attribute in data class DCLAS09 but uses other data set attributes in DCLAS09. A version 1 or version 2 PDSE will be created based on the PDSE_VERSION specification or omission in the IGDSMS*xx* parmlib member. Note that installation-written ACS routines select the management class and storage class for the data set.

***Example 3***

```
//NEWB   DD  DSNAME=REPORT.ONE(WEEK2),DISP=SHR,
//       DATACLAS=DCLAS09,DSNTYPE=LIBRARY
```

In the example, the NEWB DD statement adds a new member named WEEK2 to the existing PDSE named REPORT.ONE. DSNTYPE=LIBRARY overrides the DSNTYPE attribute in data class DCLAS09 but uses other data set attributes in DCLAS09. Other jobs can be concurrently processing existing members of PDSE named REPORT. Note that installation-written ACS routines select the management class and storage class for the data set.

***Example 4***

```
//NEWC   DD  DSNAME=REPORT.THREE(WEEK3),DISP=(NEW,KEEP),
//       DATACLAS=DCLAS09,DSNTYPE=(LIBRARY,2)
```

In the example, the NEWC DD statement defines member WEEK3 in the new PDSE named REPORT.THREE. DSNTYPE=(LIBRARY,2) overrides the DSNTYPE attribute in data class DCLAS09 but uses other data set attributes in DCLAS09. The "2" as the second DSNTYPE value causes the new PDSE to be in the version 2

format, which can produce performance benefits in some environments. Note that installation-written ACS routines select the management class and storage class for the data set.

*Example 5*

```
//FILESYS DD DSNAME=OPENDS.USRJOE,DATACLAS=DCLAS05,DISP=(NEW,KEEP),
//        DSNTYPE=HFS,SPACE=(CYL,(100,100,1))
```

The FILESYS DD statement creates an HFS data set to contain an HFS file system. The DCLAS05 in DATACLAS specifies allocation characteristics. The number of directory blocks must be specified to indicate that this is an HFS data set but the value has no effect on allocation.

*Example 6*

```
//PIPE  DD PATH='/u/payroll/buffer',DSNTYPE=PIPE,
//     PATHOPTS=(OWRONLY,OEXCL,OCREAT),PATHMODE=(SIWUSR,SIRGRP),
//     PATHDISP=(KEEP,DELETE)
```

The PIPE DD statement creates a file named **/u/payroll/buffer** for use as a FIFO special file. The PATHOPTS parameter specifies that the user intends that the program open the FIFO special file for writing. The PATHMODE parameter specifies that the file owner can write in the FIFO special file and that users in the file group class can read from the FIFO special file. The PATHDISP parameter requests that the file be kept when the program ends normally and deleted when it ends abnormally.

Pathnames are case-sensitive. If you are specifying a pathname containing a special character, including a lowercase character, enclose it in apostrophes. For more information, refer to "PATH parameter" on page 237.

*Example 7*

```
//SYSUT2  DD UNIT=SYSDA,DSNAME=BIGPROJ.BIGDATA,DSNTYPE=LARGE,
//           SPACE=(TRK,(80000,40000))
```

The SYSUT2 DD statement creates a single-volume data set that contains more than 65535 tracks. Space units such as cylinders, megabytes, or average record size can be used instead of counting tracks. A data class (DATACLAS) with a DSNTYPE value of LARGE can be coded instead of DSNTYPE=LARGE. While SMS is running, you can use a data class for a new data set that will not be SMS-managed.

## DUMMY parameter

***Parameter type***

Positional, optional

***Purpose***

Use the DUMMY parameter to specify that:

- No device or external storage space is to be allocated to the data set.
- No disposition processing is to be performed on the data set.
- For BSAM and QSAM, no input or output operations are to be performed on the data set.

One use of the DUMMY parameter is in testing a program. When testing is finished and you want input or output operations performed on the data set, replace the DD DUMMY statement with a DD statement that fully defines the data set.

Another use of the DUMMY parameter is in a cataloged or in-stream procedure. Code on the DD DUMMY statement all the required parameters. When the procedure is called, nullify the effects of the DUMMY parameter by coding on the DD statement that overrides the DD DUMMY statement a DSNAME parameter that matches the DSNAME parameter on the DD DUMMY statement. For example, procedure step PS contains the following:

```
//DS1     DD    DUMMY,DSNAME=A,DISP=OLD
```

Nullify the DUMMY parameter by coding:

```
//JS      EXEC  PROC=PROC1
//PS.DS1  DD    DSNAME=A
```

## Syntax

```
//ddname  DD  DUMMY[,parameter]...
```

All parameters coded on a DD DUMMY statement must be syntactically correct. The system checks their syntax.

## Parameters on DD DUMMY statements

- Code the DUMMY parameter by itself or follow it with all the parameters you would normally code when defining a data set, except the DDNAME parameter.
- Code the DCB parameter, if needed. If the program does not supply all the data control block information, make sure that the DCB parameter supplies the missing information.
- Code AMP=AMORG if you are using VSAM's ISAM interface.
- If you code either VOLUME=REF=dsname or DCB=dsname with DUMMY, the referenced dsname must be cataloged or passed; otherwise, the job is terminated.
- Because no I/O is performed to the dummy data set, the system checks the SPACE and DISP parameters, if coded, for syntax, then ignores them. If you code UNIT with DUMMY, the system will ignore it if the specified unit name is syntactically correct and defined to the system. Otherwise the system terminates the job.

## Relationship to other parameters

Do not code the following parameters with the DUMMY parameter.

```
*
DATA
DDNAME
DYNAM
QNAME
```

## Relationship to other control statements

***Backward references:*** If a later DD statement in a job refers to a DD DUMMY statement when requesting unit affinity (UNIT=AFF=ddname) or volume affinity (VOLUME=REF=*.stepname.ddname), the system assigns a dummy status to the later DD statement.

***Overriding a procedure DD statement:*** Coding DUMMY on a DD statement that overrides a DD statement in a procedure does not nullify symbolic parameters on the overridden DD statement. You must assign values to, or nullify, symbolic parameters on the overridden DD statement as described in "Using system symbols and JCL symbols" on page 38.

If the overriding DD statement contains a DSNAME parameter other than NULLFILE, a PATH parameter other than **/dev/null**, or a SUBSYS, SYSOUT, *, or DATA parameter, the system nullifies a DUMMY parameter on the overridden DD statement.

**Note:** If you code SYSOUT= on an overriding statement, without specifying a subparameter value, the system does not nullify the DUMMY parameter. You must code a subparameter value for SYSOUT to nullify the DUMMY parameter.

***Data sets concatenated to dummy data sets:*** The system treats data sets concatenated to a DUMMY data set as dummy data sets in that I/O operations are bypassed. However, the system performs disposition processing and allocates devices and storage for any concatenated data sets.

## Relationship to access methods

Use one of the following access methods with the DUMMY parameter:

- Basic sequential access method (BSAM)
- Virtual storage access method (VSAM)
- Queued sequential access method (QSAM)
- BDAM load mode (BSAM with MACRF=WL in the data control block)

If you use any other access method, the job is terminated.

**Note:** The ISAM/VSAM interface does not support the DUMMY parameter. For more information on the ISAM/VSAM interface, see *z/OS DFSMS Using Data Sets*.

## Examples of the DUMMY parameter

***Example 1***

```
//OUTDD1  DD  DUMMY,DSNAME=X.X.Z,UNIT=3390,
//            SPACE=(TRK,(10,2)),DISP=(,CATLG)
```

DD statement OUTDD1 defines a dummy data set. The other parameters coded on the statement are checked for syntax but not used.

***Example 2***

```
//IN1  DD  DUMMY,DCB=(BLKSIZE=800,LRECL=400,RECFM=FB)
```

DD statement IN1 defines a dummy data set. The DCB parameter supplies data control block information not supplied in the program. Without it, the step might be abnormally terminated.

***Example 3***

```
//IN2 DD  DUMMY,DSNAME=ELLN,DISP=OLD,VOLUME=SER=11257,UNIT=3390
```

When calling a cataloged procedure that contains DD statement IN2 in procedure step STEP4, you can nullify the effects of the DUMMY parameter by coding:

```
//STEP4.IN2  DD  DSNAME=ELLN
```

*Example 4*

```
//TAB  DD  DSNAME=APP.LEV12,DISP=OLD
```

If you call a cataloged procedure that contains DD statement TAB in procedure step STEP1, you can make this DD statement define a dummy data set by coding:

```
//STEP1.TAB  DD  DUMMY
```

*Example 5*

```
//MSGS  DD  SYSOUT=A
```

If you call a cataloged procedure that contains the DD statement MSGS in procedure step LOCK, you can make this DD statement define a dummy data set by coding:

```
//LOCK.MSGS  DD  DUMMY
```

# DYNAM parameter

***Parameter type***

Positional, optional

***Purpose***

Use the DYNAM parameter to increase by one the control value for dynamically allocated resources held for reuse. Even when DYNAM is not coded, the system normally holds resources in anticipation of reuse. The DYNAM parameter is supported to provide compatibility with older systems.

A DD DYNAM statement is a dummy request.

## Syntax

```
//ddname  DD  DYNAM   [comments]
```

## Relationship to other parameters

Do not code any parameters with the DYNAM parameter.

Do not code on a DD DYNAM statement a ddname that is meaningful to the system; for example, JOBLIB, SYSCHK.

## Relationship to other control statements

- Do not refer to a DD DYNAM statement in a DDNAME parameter.
- To nullify the DYNAM parameter on a DD statement in a cataloged or in-stream procedure, code a SYSOUT or DSNAME parameter in the overriding DD statement. DSNAME=NULLFILE does not nullify a DYNAM parameter.
- Do not code a backward reference to a DD DYNAM statement.
- Do not code the DYNAM parameter on the first DD statement for a concatenation.

## Example of the DYNAM parameter

```
//INPUT  DD  DYNAM
```

This DD statement increases by one the control value for dynamically allocated resources held for reuse.

## EATTR parameter

*Parameter type*

Keyword, optional

*Purpose*

Use the EATTR parameter to indicate whether the data set can support extended attributes (format 8 and 9 DSCBs) or not. To create such data sets, you can include extended address volumes (EAVs) in specific storage groups or specify an EAV on the request or direct the Allocation to an esoteric containing EAV devices.

By definition, a data set with extended attributes can reside in the extended address space (EAS) on an extended address volume (EAV). This parameter can be specified for non-VSAM data sets as well as for VSAM data sets.

The EATTR value has no affect for DISP=OLD processing, even for programs that might open a data set for OUTPUT, INOUT, or OUTIN processing. The value on the EATTR parameter is used for requests when the data set is newly created.

### Syntax

```
EATTR=[OPT|NO]
```

### Subparameter definition

**EATTR = OPT**
Extended attributes are optional. The data set *can* have extended attributes and reside in EAS. This is the default value for VSAM data sets.

**EATTR = NO**
No extended attributes. The data set *cannot* have extended attributes (format 8 and 9 DSCBs) or reside in EAS. This is the default value for non-VSAM data sets.

### Examples of the EATTR parameter

```
//DD2  DD  DSNAME=PDS12,DISP=(,KEEP),UNIT=SYSALLDA,
//         VOLUME=SER=25143,SPACE=(CYL,(10000,,100),,CONTIG),
//         EATTR=OPT
```

The DD statement defines a new partitioned data set. The system allocates 10000 cylinders to the data set, of which one hundred 256-byte records are for a directory. When the CONTIG subparameter is coded, the system allocates 10000 contiguous cylinders on the volume. EATTR=OPT indicates that the data set might be created with extended attributes. With this option, the data set can reside in the extended address space (EAS) of the volume.

# EXPDT parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the EXPDT parameter to specify the expiration date for a new data set. On and after the expiration date, the data set can be deleted or written over by another data set.

**Note:** You may specify a past date; this would not be an error condition.

If the DD statement contains DISP=(NEW,DELETE) or the DISP parameter is omitted and defaults to NEW and DELETE, the system deletes the data set when the step terminates, either normally or abnormally, even if you have specified an expiration date.

Do not specify EXPDT for a temporary data set.

The EXPDT parameter achieves the same result as the RETPD parameter.

Code the EXPDT parameter when you want to specify an expiration date for the data set, or, with SMS, override the expiration date defined in the data class for the data set.

## Syntax

```
EXPDT= {yyddd   }
       {yyyy/ddd}
```

The EXPDT parameter can have a null value only when coded on a DD statement that is either added to a procedure or overrides a DD statement in a procedure.

## Subparameter definition

**EXPDT=yyddd**
**EXPDT=yyyy/ddd**

Specifies an expiration date for the data set.

**yyddd**

This form of the expiration date specifies a two-digit year number *yy* (from 00 through 99) and a three-digit day number *ddd* (from 001 through 365 for a non-leap year date, from 001 through 366 for a leap year date). For example, code February 2, 1999 as EXPDT=99033, and code December 31, 1996 as EXPDT=96366.

**Note:** For expiration dates of January 1, 2000 and later, you MUST use the form EXPDT=yyyy/ddd.

**Note:** Expiration dates of 99365 and 99366 are considered "never-scratch" dates. Data sets with these expiration dates are not deleted or written over.

**yyyy/ddd**

This form of the expiration date specifies a four-digit year number *yyyy* (from 1900 through 2155) and a three-digit day number *ddd* (from 001

through 365 for a non-leap year date, from 001 through 366 for a leap year date). For example, code February 2, 1999 as EXPDT=1999/033, and code December 31, 2000 as EXPDT=2000/366.

**Note:** Expiration dates of 1999/365 and 1999/366 are considered "never-scratch" dates. Data sets with these expiration dates are not deleted or written over.

You may specify the years from 1900. However, if you specify the current date or an earlier date, the data set is immediately eligible for replacement.

## Overrides

With SMS, EXPDT overrides the expiration date defined in the DATACLAS parameter for the data set. See "Overrides" on page 134.

With SMS, both the expiration date specified on EXPDT and defined in the data class for an SMS-managed data set can be limited by a maximum expiration date defined in the management class for the data set.

## Relationship to other parameters

Do not code the following DD parameters with the EXPDT parameter.

```
*
DATA
DDNAME
DYNAM
RETPD
SYSOUT
```

## Deleting a data set before its expiration date

To delete a data set (and make the space occupied by the data set available for reallocation) before the expiration date has passed, use one of the following methods:

- For data sets cataloged in an integrated catalog facility catalog, use the DELETE command, as described in *z/OS DFSMS Access Method Services Commands*.
- For data sets not cataloged in an integrated catalog facility catalog, use the IEHPROGM utility, as described in *z/OS DFSMSdfp Utilities*.
- Use the SCRATCH macro with the OVRD parameter, as described in *z/OS DFSMSdfp Advanced Services*. If the data set is SMS-managed, this also uncatalogs the data set.
- If the storage administrator specified OVRD_EXPDT(YES) in the IGDSMSxx member of SYS1.PARMLIB, you can override the expiration date for SMS-managed DASD data sets by specifying DELETE on the DD DISP parameter. The system will delete the data set whether or not it has expired. See *z/OS MVS Initialization and Tuning Reference* for information about the IGDSMSxx parmlib member.

## Examples of the EXPDT parameter

***Example 1***

```
//DD7  DD  DSNAME=TOM1,DISP=(NEW,KEEP),EXPDT=2006/033,
//         UNIT=SYSDA,SPACE=(TRK,(1,1)),VOLUME=SER=663344
```

In this example, the data set is not eligible for being deleted or written over until February 2, 2006.

*Example 2*

```
//SMSDS2  DD  DSNAME=MYDS2.PGM,DATACLAS=DCLAS02,DISP=(NEW,KEEP),
//            EXPDT=2001/033
```

In this example, the expiration date of February 2, 2001 overrides the expiration date defined in the data class for the data set.

## FCB parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the FCB parameter to specify:

- The forms control buffer (FCB) image JES is to use to guide printing of this sysout data set by a 1403 Printer, 3211 Printer, 3203 Printer Model 5, 3800 Printing Subsystem, 4245 Printer, 4248 Printer, or by a printer supported by systems network architecture (SNA) remote job entry (RJE).
- The carriage control tape JES is to use to control printing of this sysout data set by a 1403 Printer or by a printer supported by SNA RJE.
- The data-protection image JES is to use to control output of this sysout data set by a 3525 Card Punch.
- The name of a page definition to be used by PSF in formatting a print data set.

The FCB image specifies how many lines are to be printed per inch and the length of the form. JES loads the image into the printer's forms control buffer. The FCB image is stored in SYS1.IMAGELIB. IBM provides three standard FCB images:

- STD1, which specifies 6 lines per inch on an 8.5-inch-long form. (3211 and 3203-2 only)
- STD2, which specifies 6 lines per inch on an 11-inch-long form. (3211 and 3203-2 only)
- STD3, which specifies 8 lines per inch for a dump on an 11-inch form. (3800 only)

***References***

For more information on the forms control buffer, see *z/OS DFSMSdfp Advanced Services* or *PSF for z/OS: User's Guide*.

## Syntax

```
FCB=   {fcb-name                  }
       {(fcb-name[,ALIGN|,VERIFY])}
```

- You can omit the parentheses if you code only the fcb-name.
- Code the fcb-name as STD1 or STD2 only to request the IBM-supplied images.
- Code the fcb-name as STD3 for a high-density dump.
- Null positions in the FCB parameter are invalid.

## Subparameter definition

**fcb-name**
Identifies the FCB image. The name is 1 through 4 alphanumeric or national ($, #, @) characters and is the last characters of a SYS1.IMAGELIB member name:

- FCB2xxxx member for a 3211, a 3203 model 5, or a printer supported by SNA.
- FCB3xxxx member for a 3800.
- FCB4xxxx member for a 4248.

**ALIGN**
Requests that the system ask the operator to check the alignment of the printer forms before the data set is printed.

**Note:**

- ALIGN is ignored for a sysout data set.
- ALIGN is ignored for a data set printed on an AFP printer. AFP printers do not use the ALIGN subparameter.

**VERIFY**
Requests that the system ask the operator to verify that the image displayed on the printer is for the desired FCB image. The operator can also take this opportunity to align the printer forms.

**Note:** VERIFY is ignored for a sysout data set.

## Defaults

If you do not code the FCB parameter, the system checks the FCB image in the printer's forms control buffer; if it is a default image, as indicated by its first byte, JES uses it. If it is not a default image, JES loads the FCB image that is the installation default specified at JES initialization.

## Overrides

An FCB parameter on a sysout DD statement overrides an OUTPUT JCL FCB parameter.

If both an FCB parameter and a PAGEDEF parameter are coded in your JCL, PSF ignores the FCB parameter. For more information, see *PSF for z/OS: User's Guide*.

## Relationship to other parameters

Do not code the following parameters with the FCB parameter.

```
*
AMP
DATA
DDNAME
DYNAM
KEYOFF
PROTECT
QNAME
```

Do not code the following DCB subparameters with the FCB parameter.

```
CYLOFL
RKP
INTVL
```

For output to the 3525, do not code the SYSOUT parameter and the FCB parameter; the system ignores the FCB parameter.

## Relationship to other control statements

You can also code the FCB parameter on the following:

- The OUTPUT JCL statement.
- The JES2 /*OUTPUT statement.
- The JES3 //*FORMAT PR statement.

## Defining an FCB image for a work station

When a work station uses a peripheral data set information record (PDIR), the FCB image is defined in the work station. The DD statement FCB fcb-name subparameter must match the FCB name defined in the PDIR work station.

When a work station does not use a PDIR, add an FCB member to SYS1.IMAGELIB. At setup time, JES3 translates the FCB into a set vertical format (SVF).

## Requesting a high-density dump

You can request a high-density dump on the 3800 through two parameters on the DD statement for the dump data set or on an OUTPUT JCL statement referenced by the dump DD statement:

- FCB=STD3. This parameter produces dump output at 8 lines per inch.
- CHARS=DUMP. This parameter produces 204-character print lines.

You can code one or both of these parameters. You can place both on the same statement or one on each statement.

## Examples of the FCB parameter

***Example 1***

```
//DD1   DD  UNIT=3211,FCB=(IMG1,VERIFY)
```

In this example, the DD statement defines an output data set to be printed by a 3211. The FCB parameter requests that the data set be printed under control of the FCB2IMG1 member in SYS1.IMAGELIB. Because VERIFY is coded, the system displays the FCB image on the printer before printing the data set.

***Example 2***

```
//DD2   DD  SYSOUT=A,FCB=IMG2
```

This sysout DD statement specifies output class A. If output class A routes output to a printer having the forms control buffer feature, JES loads the FCB image IMG2 into the forms control buffer. If the printer does not have the forms control buffer feature, the operator receives a message to mount the carriage control tape IMG2 on the printer.

***Example 3***

```
//OUTDDS  DD  UNIT=3211,FCB=(6,ALIGN)
```

In this example, the DD statement defines an output data set to be printed by a 3211. The FCB parameter requests that the data set be printed under control of the

FCB image named 6. Because ALIGN is coded, the system issues a message to the operator requesting that the alignment of the printer forms be checked before the data set is printed.

***Example 4***

```
//PUNCH   DD  UNIT=3525,FCB=DP2
```

In this example, the DD statement requests output on a 3525. Therefore, the FCB parameter defines the data protection image to be used for the 3525.

***Example 5***

```
//SYSUDUMP  DD  SYSOUT=A,FCB=STD3
```

In this example, the DD statement requests that the 3800 print a dump at 8 lines per inch.

# FILEDATA parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the FILEDATA keyword to describe the content type of a z/OS UNIX file so that the system can determine how to process the file.

***References***

For more information on network file protocols, see *z/OS Network File System Guide and Reference*.

## Syntax

```
FILEDATA= {BINARY}
          {TEXT  }
          {RECORD}
```

## Subparameter definition

**BINARY**
The file described by the DD statement is a byte-stream file and does not contain record delimiters. The access method does not insert or delete record delimiters, <newline> character X'15'.

**TEXT**
When you copy MVS data sets to text files in the z/OS UNIX file system, a <newline> character X'15' is appended to the end of each record.

**RECORD**
Indicates that the data consists of records with prefixes. The record prefix contains the length of the record that follows. On output, the access method inserts a record prefix at the beginning of each record. On input, the access method uses the record prefix to determine the length of each record. The access method does not return the prefix as part of the record. Code

FILEDATA=RECORD when you cannot code FILEDATA=TEXT because your data might contain bytes that are considered delimiters.

**Note:**
The record prefix for FILEDATA=RECORD is mapped by the IGGRPFX macro. This is different from the record descriptor word (RDW) that is in z/OS physical sequential format-V data sets.

## Defaults

If you do not code the FILEDATA parameter, the system assigns a default value of BINARY.

## Overrides

The FILEDATA parameter does not override the specification of any other JCL keyword or system parameter.

## Relationship to other parameters

You can code the FILEDATA parameter only on a DD statement that contains a PATH parameter.

You can code the following parameters with the FILEDATA parameter.

| | | |
|---|---|---|
| BLKSIZE | LRECL | PATHMODE |
| BUFNO | NCP | PATHOPTS |
| DSNTYPE | PATH | RECFM |
| DUMMY | PATHDISP | TERM |

## Example of the FILEDATA parameter

```
//DD1    DD PATH='/u/d89pek1/new',FILEDATA=TEXT,
//          PATHMODE=(SIRWXU,SISUID),PATHOPTS=(ORDONLY,OCREAT)
```

In this example, the DD statement identifies a hierarchical file and informs the system that this file contains records delimited by the newline character.

# FLASH parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the FLASH parameter to identify the forms overlay to be used in printing this sysout data set on a 3800 Printing Subsystem and, optionally, to specify the number of copies on which the forms overlay is to be printed.

**Note:** FLASH applies only for a data set printed on a 3800.

***References***

For information on forms overlays, see the *Forms Design Reference Guide for the 3800*.

## Syntax

```
        {overlay-name          }
FLASH=  {(overlay-name[,count])}
        {NONE                  }
```

The count subparameter is optional. If you omit it, you can omit the parentheses. However, if you omit it, you must not code it as a null; for example, FLASH=(ABCD,) is invalid.

## Subparameter definition

**overlay-name**
Identifies the forms overlay frame that the operator is to insert into the printer before printing begins. The name is 1 through 4 alphanumeric or national ($, #, @) characters.

**count**
Specifies the number, 0 through 255, of copies that JES is to flash with the overlay, beginning with the first copy printed. Code a count of 0 to flash **all** copies.

**NONE**
Suppresses flashing for this sysout data set.

If FLASH=NONE is on a DD statement in a job to be executed at a remote node, JES3 sets the overlay-name to zero before sending the job to the node.

## Defaults

If you do not code a FLASH parameter and an installation default was not specified at JES2 or JES3 initialization, forms are not flashed.

If you specify an overlay-name without specifying a count or with a count of 0, all copies are flashed. That is, the default for count is 255.

## Overrides

A FLASH parameter on a sysout DD statement overrides an OUTPUT JCL FLASH parameter.

**Note:** A null first subparameter is invalid in a FLASH parameter on a DD statement, but is permitted on an OUTPUT JCL statement.

## Relationship to other parameters

Do not code the following parameters with the FLASH parameter.

| | | |
|---|---|---|
| * | DISP | PROTECT |
| AMP | DSID | QNAME |
| DATA | DYNAM | VOLUME |
| DDNAME | LABEL | |

***Relationship to COPIES parameter:*** If this DD statement or a referenced OUTPUT JCL statement also contains a COPIES parameter, JES prints with the forms overlay the number of copies specified in one of the following:

- COPIES=nnn, if the FLASH count is larger than nnn. For example, if COPIES=10 and FLASH=(LTHD,12) JES prints 10 copies, all with the forms overlay.

- The sum of the group-values specified in the COPIES parameter, if the FLASH count is larger than the sum. For example, if COPIES=(,(2,3,4)) and FLASH=(LTHD,12) JES prints nine copies in groups, all with the forms overlay.
- The count subparameter in the FLASH parameter, if the FLASH count is smaller than nnn or the sum from the COPIES parameter. For example, if COPIES=10 and FLASH=(LTHD,7) JES prints seven copies with the forms overlay and three copies without.

### Relationship to other control statements

FLASH can also be coded on the following:

- The OUTPUT JCL statement.
- The JES3 //*FORMAT PR statement.
- The JES2 /*OUTPUT statement.

### Verification of forms overlay frame

Before printing starts, the system requests the operator to load the specified forms overlay frame in the printer. A frame must be loaded, but the system cannot verify that it is the correct frame.

### Printing without flashing

To print without flashing, specify one of the following:

- FLASH=NONE on the DD or OUTPUT JCL statement.
- Omit the FLASH parameter on all of the statements for the data set and on all JES initialization statements.
- For a sysout data set, omit the FLASH parameter on the DD statement and specify FLASH=(,0) on a referenced OUTPUT JCL statement.

### Example of the FLASH parameter

```
//DD1 DD  SYSOUT=A,COPIES=10,FLASH=(ABCD,5)
```

In this example, JES issues a message to the operator requesting that the forms-overlay frame named ABCD be inserted into the printer. Then JES prints the first five copies of the data set with the forms-overlay and the last five copies without.

## FREE parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the FREE parameter to specify when the system is to unallocate the resources used for this DD statement's data set. The resources can be devices, volumes, or exclusive use of a data set.

**Note:** Specifying FREE will not release the enqueue on the data set until the last step that requires the data set completes processing.

## Syntax

```
FREE= {END  }
      {CLOSE}
```

## Subparameter definition

**END**
Requests that the system unallocate the data set at the end of the last step that references the data set.

**CLOSE**
Requests that the system unallocate the data set when it is closed.

## Defaults

If no FREE parameter is specified, the default is END. Also, if the FREE parameter is incorrectly coded, the system substitutes END and issues a warning message.

## Overrides

FREE=CLOSE is ignored when:

- The data set is a member of a concatenated group.
- The task using the data set abnormally terminates.

  If you specify FREE=CLOSE and the job step abnormally terminates before the data set is closed, the system uses the abnormal termination disposition from the DISP parameter to process the data set. If a recovery routine, such as an ESTAE routine, gets control and closes the data set, then the system uses the normal termination disposition.

  If the job step abnormally terminates after the data set is closed, then the system has already processed the data set using the normal termination disposition.
- The data set is referenced by another DD statement in the same or subsequent step.
- The data set is a VSAM data set.
- The DDname on the DD statement is JOBLIB or STEPLIB.

## Relationship to other parameters

Do not code the following parameters with the FREE parameter.

| | | |
|---|---|---|
| * | DDNAME | QNAME |
| AMP | DYNAM | RECORG |
| DATA | KEYOFF | RLS |

If the DD statement specifies FREE=END and a DISP subparameter of PASS, the data set is not unallocated until the end of the job or until used for a later DD statement with a disposition of other than PASS.

Do not specify FREE=CLOSE on a DD statement with a ddname of JOBLIB or STEPLIB; CLOSE is ignored.

If you specify SPIN=NO with FREE=CLOSE, the sysout data set will be unallocated, but not printed until the end of the job.

When you specify SPIN=UNALLOC with FREE=CLOSE, the sysout data set is available for printing immediately when you explicitly close or dynamically unallocate the data set. If you do not explicitly close or dynamically unallocate the data set, it will be available for printing at the end of the step.

If you specify SPIN=UNALLOC with FREE=END, the sysout data set is unallocated at the end of the step, and is made available for printing then. If you dynamically unallocate the sysout data set, the system makes it available for printing immediately.

If you specify SPIN=NO with FREE=END, the system makes the sysout data set available for printing at the end of the job, regardless of when the data set is unallocated or closed.

## Relationship to other control statements

If a DD statement requests unit affinity in a UNIT=AFF parameter or volume affinity in a VOLUME=REF parameter with an earlier DD statement, do not code FREE=CLOSE on the earlier statement.

If you code FREE=CLOSE on a sysout DD statement that references an OUTPUT JCL statement containing a GROUPID parameter, JES2 will not group the data sets into one output group. Instead, JES2 produces one copy of the sysout data set for each OUTPUT JCL statement that the DD statement references.

## Relationship to the CLOSE macro instruction

When FREE=CLOSE is specified for a data set that is opened and closed more than once during a job step:

- The data set is unallocated after it is closed if the assembler CLOSE macro instruction specifies DISP, REWIND, or FREE. If the data set is reopened after the system has unallocated it, the job step abnormally terminates, unless the data set is dynamically allocated in the interval.

The data set is not unallocated until the end of the job if the assembler CLOSE macro instruction specifies LEAVE or REREAD. Then the data set can be reopened.

## Examples of the FREE parameter

***Example 1***

```
//EA33  DD  SYSOUT=D,FREE=CLOSE
```

In this example, the FREE=CLOSE parameter makes JES unallocate this output class D data set when it is closed, rather than at the end of the job step. JES schedules the data set for printing.

***Example 2***

```
//EA33  DD  DSNAME=SYBIL,DISP=OLD,FREE=CLOSE
```

In this example, the FREE=CLOSE parameter makes JES unallocate the data set, dequeue it, and make it available to other jobs as soon as it is closed.

***Example 3***

```
//STEP1   EXEC PGM=ABLE1
//DD1     DD   DSNAME=A,DISP=(,PASS),FREE=END
//STEP2   EXEC PGM=ABLE2
//DD2     DD   DSNAME=A,DISP=(OLD,CATLG),FREE=END
```

In this example, data set A is passed by STEP1 to STEP2. FREE=END on DD statement DD1 is ignored because the disposition is PASS. FREE=END on DD statement DD2 causes data set A to be unallocated at the end of STEP2, when it is also cataloged.

*Example 4*

```
//STEP1   EXEC PGM=BAKER1
//DD      DD   DSNAME=A,DISP=(NEW,PASS),FREE=END
//STEP2   EXEC PGM=BAKER2
```

In this example, data set A is a new data set. Because PASS is specified, FREE=END is ignored and the data set remains allocated.

# FREEVOL parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Use the FREEVOL parameter to specify whether to allow other jobs to read freed volumes of a multivolume tape file as the volume is dismounted by the job.

When multiple DD statements in JOB1 specify one or more of the same volume serial numbers, and also FREEVOL=EOV, it becomes possible for JOB2 to obtain the ENQ on the volume serial number and begin processing as soon as JOB1 releases the ENQ. This results in JOB1 abending when attempting to process a subsequent DD with the same volume serial number, because the volume is no longer available to JOB1. In addition, FREEVOL=EOV has no effect on the allocation requirement that all specified volume serial numbers be available for enqueue; if any are not available, allocation issues IEF235D and waits for volume availability or for the job to be cancelled.

**Note:** FREEVOL is most effective when a second job waits for a single volume. If it waits for multiple volumes, it will still need to wait for all volumes to become available.

## Syntax

```
FREEVOL={EOV | END}
```

## Subparameter definition

**EOV**
Requests that when reading a multivolume data set, the system finish reading the current volume and then dequeue the volume serial number and demount the volume. This makes the volume immediately available to another job in another system. An attempt by the same task to reprocess the volume using the same JCL DD statement will result in an abnormal end.

**END**
Requests that volumes be dequeued at the end of the job step.

## Defaults

If no FREEVOL parameter is specified, the default is END. Also, if the FREEVOL parameter is incorrectly coded, the system substitutes END and issues a warning message.

## Overrides

FREEVOL=EOV is not honored when the tape volume disposition is not REQIND (for example, if the tape disposition is RETAIN or CLOSE with the LEAVE option).

## Relationship to other parameters

Do not code the FREEVOL parameters with PATH related keywords, such as the following:

| | |
|---|---|
| PATH | PATHOPTS |
| PATHMODE | PATHDISP |
| FILEDATA | |

## Relationship to other control statements

None

# GDGORDER parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Use the GDGORDER parameter for a DD that specifies the base name of a GDG data set (a GDG-all request). This keyword specifies the order in which the individual generation data sets (GDSs) will be concatenated.

## Syntax

```
GDGORDER=USECATLG | LIFO | FIFO
```

## Subparameter definition

**USECATLG**
The GDS concatenation is ordered as specified in the GDG data set catalog entry.

**LIFO**
The GDS concatenation is ordered with the newest GDS defined first and the oldest GDS last.

**FIFO**
The GDS concatenation is ordered with the oldest GDS defined first and the newest GDS last.

## Defaults

USECATLG is the default value. When the GDGORDER keyword is not specified, the concatenation order from the catalog definition of the GDG data set is used. However, the default setting in the catalog entry for a GDG base data set definition is LIFO, to match the setting for releases prior to z/OS V2.1.

## Example of the GDGORDER parameter

***Example 1:*** The following example GDG has a base name D24PP1.SMF.DATA and three generation data sets:

```
D24PP1.SMF.DATA.G0001V00
D24PP1.SMF.DATA.G0002V00
D24PP1.SMF.DATA.G0003V00
```

The following DD statement overrides the default LIFO specification and creates a FIFO data set concatenation to be read by the job step program:

```
//SMFDATA DD DSN=D24PP1.SMF.DATA,DISP=SHR,GDGORDER=FIFO
```

Data is read from the data sets in the specified order.

The following concatenation is used:

```
//SMFDATA DD DSN=D24PP1.SMF.DATA(-2),DISP=SHR
//        DD DSN=D24PP1.SMF.DATA(-1),DISP=SHR
//        DD DSN=D24PP1.SMF.DATA(0),DISP=SHR
```

The concatenation is resolved as the following code:

```
//SMFDATA DD DSN=D24PP1.SMF.DATA.G0001V00,DISP=SHR
//        DD DSN=D24PP1.SMF.DATA.G0002V00,DISP=SHR
//        DD DSN=D24PP1.SMF.DATA.G0003V00,DISP=SHR
```

# HOLD parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the HOLD parameter to tell the system to hold a sysout data set until it is released by the system operator. When the data set is ready for processing, notify the system operator to release it via a TSO/E NOTIFY parameter, a JES2 /*MESSAGE statement, or a JES3 //*OPERATOR statement.

A TSO/E user can specify HOLD=YES to retrieve a sysout data set and display it on a terminal. For JES3, the TSO/E user can process only work on the hold queue.

**Note:**

1. HOLD is supported only for sysout data sets. If HOLD appears on a DD statement that does not contain a SYSOUT parameter, it is ignored.
2. HOLD allows the sysout data set to be the internal reader. If the sysout data set is the internal reader, the job being submitted will be held.
3. In a JES2 system, SYSOUT held by specifying HOLD=YES may be selected via a SAPI (Sysout Application Process Interface) application. JES3 systems are not allowed to select the held SYSOUT via SAPI until the hold is released via operator command.

## Syntax

```
HOLD= {YES}
      {Y  }
      {NO }
      {N  }
```

## Subparameter definition

**YES**

Requests that the system hold the sysout data set until the data set is released by the system operator. You can also code this subparameter as Y.
**NJE Notes**:

- In a JES2 NJE environment, the system does not hold the data set until it reaches its ultimate destination node.
- If the destination node is a JES3 node, the system may still not hold the data set if the class of output being transmitted is not defined as a hold class.

  If the sending node is JES3, the system holds the output data set at that node on the BDT queue (when transmitting to an SNA-attached node) or the WTR queue (when transmitting to a BSC-attached node) if all of the following are true:

  – The "// DD SYSOUT=" JCL statement does not contain a DEST=(node,userid) parameter.
  – The SYSOUT= parameter does not contain the WRITER-NAME subparameter and the output class is not defined as a hold class.
  – No WRITER= parameter is coded on the OUTPUT JCL statement.

  Example 1.

  The following job executes on NODE1 and results in the SYSUT2 output data set being held on the BDT queue on NODE1. (NODE5 is attached to NODE1 via SNA and output class A is not defined as a hold class.)

```
//S1       EXEC PGM=IEBGENER
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   DUMMY
//SYSUT1   DD   DSN=SYS1.PROCLIB(JES3),DISP=SHR
//SYSUT2   DD   SYSOUT=A,HOLD=YES,DEST=NODES
```

  Example 2.

  The following job executes on NODE1 and results in the SYSUT2 output data set being held on the WTR queue on NODE1. (NODE5 is attached to NODE1 via BSC and output class A is not defined as a hold class.)

```
//S1       EXEC PGM=IEBGENER
//O1       OUTPUT CLASS=A,DEST=NODE2.MYWRITR
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   DUMMY
//SYSUT1   DD   DSN=SYS1.PROCLIB(JES3),DISP=SHR
//SYSUT2   DD   SYSOUT=(,),HOLD=YES,OUTPUT=(*.O1)
```

**NO**
Requests that the output data set **not** be held. You can also code this subparameter as N.

## Defaults

None.

If you do not specify the HOLD parameter, or if the value specified on the HOLD parameter is incorrectly coded, the output data set will be held or not held based on how your installation specified the SYSOUT class. Examples of incorrect values on the hold parameter include:

- 'HOLD=' without a value specified
- A value for HOLD other than YES, Y, NO, or N

## Overrides

HOLD=NO is overridden by the unallocation verb of dynamic allocation or the TSO/E FREE command.

Either HOLD=YES or HOLD=NO on the DD statement overrides the sysout data set disposition specified on the OUTDISP parameter of the OUTPUT JCL statement.

## Relationship to other parameters

Code the HOLD parameter only on a DD statement with the SYSOUT parameter.

For JES3, be aware that if the SYSOUT is associated with an output descriptor that is defined by the OUTPUT JCL statement, then the output characteristics are merged for SYSOUT on the HOLD queue.

JES3 ignores HOLD=YES when

- DEST=(node,userid) is coded on the SYSOUT= DD statement. Example 1 shows this case. (JES3 does not ignore the HOLD=YES when DEST= is coded on the OUTPUT DD statement. Example 2 shows this case.) or
- the sysout data set is placed on the hold queue, for example, if SYSOUT=(,writer-name) is coded.

**Ignored but permitted DD parameter**: If you specify the SUBSYS DD parameter, the system checks it for syntax and then ignores it.

## Relationship to other control statements

Code a NOTIFY parameter on the JOB statement to ask the system to send a message to your TSO/E userid when job processing is complete.

JES2 users can use the /*NOTIFY control statement to direct job notification messages and to override a JOB NOTIFY parameter.

## Examples of the HOLD parameter

### Example 1

```
//JOB01    JOB    ,'HAROLD DUQUETTE',MSGLEVEL=1
//STEP1    EXEC   PGM=MJCOSCO
//DD1      DD     SYSOUT=B,DEST=RMT6,HOLD=YES
```

In this example, sysout data set DD1 from JOB01 is held on a queue until the TSO/E user at RMT6 asks the system operator to release the data set.

### Example 2

```
//$JOBxx   JOB    ,'OSWALD CHALMERS',MSGLEVEL=1
//OUT1     OUTPUT DEST=NODE2.printer,CLASS=A,...
//STEP1    EXEC   PGM=IEBGENER
//SYSPRINT DD     SYSOUT=*
//SYSUT1   DD     DISP=SHR,DSN=A.DATA.SET
//SYSUT2   DD     SYSOUT=(,),OUTPUT=(*.OUT1),HOLD=YES
```

In this example, if the job is submitted on NODE1, JES3 does not ignore the HOLD=YES. The SYSOUT data set is held at NODE1 and is not transmitted to NODE2 to be held there.

# KEYLABL1 parameter

***Parameter Type***

Keyword, optional

***Purpose***

Use the KEYLABL1 to specify the label for the key encrypting key used by the Encryption Key Manager. The key encrypting key is used to encrypt the data (encryption) key.

Code the KEYLABL1 parameter when you want to:

- Specify the label for the key encrypting key used by the Encryption Key Manager, or
- Override the label for the key encrypting key defined in the data class of the data set.

Specification of the key labels does not by itself enable encryption. Encryption must be enabled by a data class that specifies an encryption format, for example EEFMT2.

At least one of KEYLABL1 or KEYLABL2 must have a private key associated with it.

For complete documentation on using tape encryption, see *z/OS DFSMS Software Support for IBM System Storage TS1140, TS1130, and TS1120 Tape Drives (3592)*.

## Syntax

```
KEYLABL1='mykeylabel1'
```

## Subparameter definition

*mykeylabel1*
Specifies the KEYLABL1 for the key encrypting key used by the Encryption Key Manager. The key label can be up to 64 characters in length.

## Defaults

One or both key labels may be specified. If only one key label is specified, specification of either of KEYLABL1 or KEYLABL2 is allowed; it does not have to be specified as KEYLABL1.

If only one key label and encoding mechanism is specified, the same key label and encoding mechanism is used for both the key label and the encoding parameters.

If no key label is specified, either through the DD statement or through data class, externally specified key manager defaults will be used.

## Overrides

Coding KEYLABL1 or KEYLABL2 overrides both labels for the key encrypting key defined in the DATACLAS parameter for the data set. See “Overrides” on page 134.

## Relationship to other parameters

If you specify the KEYLABL1 parameter on a DD statement, you must also code the KEYENCD1 parameter.

## Examples of the KEYLABL1 parameter

***Using label encoding:***

```
//DD1  DD  DSN=DSN5,DISP=(NEW,CATLG),STORCLAS=ATL,
// KEYLABL1='LABELQ1.LABELQ2.LABELQ3',KEYENCD1=L
```

***Using hash encoding:***

```
//DD1  DD  DSN=DSN5,DISP=(NEW,CATLG),STORCLAS=ATL,
// KEYLABL1='LABELQ1.LABELQ2.LABELQ3',KEYENCD1=H
```

# KEYLABL2 parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the KEYLABL2 to specify the label for the key encrypting key used by the Encryption Key Manager. The key encrypting key is used to encrypt the data (encryption) key.

Code the KEYLABL2 parameter when you want to:

- Specify the label for the key encrypting key used by the Encryption Key Manager, or
- Override the label for the key encrypting key defined in the data class of the data set.

Specification of the key labels does not by itself enable encryption. Encryption must be enabled by a data class that specifies an encryption format, for example EEFMT2.

At least one of KEYLABL1 or KEYLABL2 must have a private key associated with it.

For complete documentation on using tape encryption, see *z/OS DFSMS Software Support for IBM System Storage TS1140, TS1130, and TS1120 Tape Drives (3592).*

## Syntax

```
KEYLABL2='mykeylabel2'
```

## Subparameter definition

*mykeylabel2*
Specifies the KEYLABL2 for the key encrypting key used by the Encryption Key Manager. The key label can be up to 64 characters in length.

## Defaults

One or both key labels may be specified. If only one key label is specified, specification of either KEYLABL1 or KEYLABL2 is allowed; it does not have to be specified as KEYLABL1.

If only one key label and encoding mechanism is specified, the same key label and encoding mechanism is used for both key label and encoding parameters.

If no key label is specified, either through the DD statement or through data class, externally specified key manager defaults will be used.

## Overrides

Coding KEYLABL1 or KEYLABL2 overrides both labels for the key encrypting key defined in the DATACLAS parameter for the data set. See "Overrides" on page 134.

## Relationship to other parameters

If you specify the KEYLABL2 parameter on a DD statement, you must also code the KEYENCD2 parameter.

## Examples of the KEYLABL2 parameter

***Using label encoding:***

```
//DD1  DD  DSN=DSN5,DISP=(NEW,CATLG),STORCLAS=ATL,
// KEYLABL2='LABELQ1.LABELQ2.LABELQ3',KEYENCD2=L
```

***Using hash encoding:***

```
//DD1  DD  DSN=DSN5,DISP=(NEW,CATLG),STORCLAS=ATL,
// KEYLABL2='LABELQ1.LABELQ2.LABELQ3',KEYENCD2=H
```

# KEYENCD1 parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the KEYENCD1 parameter to specify how the label for the key encrypting key specified by the key label 1 is encoded by the Encryption Key Manager and stored on the tape cartridge.

Code the KEYENCD1 parameter when you want to specify how the label for the key encrypting key specified by the key label 1 is encoded by the Encryption Key Manager and stored on the tape cartridge.

Specification of the key encoding does not by itself enable encryption. Encryption must be enabled by a data class that specifies an encryption format, for example EEFMT2.

For complete documentation on using tape encryption, see *z/OS DFSMS Software Support for IBM System Storage TS1140, TS1130, and TS1120 Tape Drives (3592)*.

## Syntax

```
KEYENCD1=L|H
```

## Subparameter definition

**L** Indicates that the key label 1 will be stored as part of the EEDK structure on the tape cartridge.

**H** Indicates that a hash of the public key referenced by key label 1 will be stored on the cartridge rather than the key label.

## Overrides

KEYENCD1 overrides the encoding mechanism of the label for the key encrypting key defined in the DATACLAS parameter for the data set. See "Overrides" on page 134.

## Relationship to other parameters

If you specify the KEYENCD1 parameter on a DD statement, you must also code the KEYLABL1 parameter.

## Example of the KEYENCD1 parameter

***Using label encoding:***

```
//DD1  DD  DSN=DSN5,DISP=(NEW,CATLG),STORCLAS=ATL,
// KEYLABL1='LABELQ1.LABELQ2.LABELQ3',KEYENCD1=L
```

***Using hash encoding:***

```
//DD1  DD  DSN=DSN5,DISP=(NEW,CATLG),STORCLAS=ATL,
// KEYLABL1='LABELQ1.LABELQ2.LABELQ3',KEYENCD1=H
```

# KEYENCD2 parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the KEYENCD2 parameter to specify how the label for the key encrypting key specified by the key label 2 is encoded by the Encryption Key Manager and stored on the tape cartridge.

Code the KEYENCD2 parameter when you want to specify how the label for the key encrypting key specified by the key label 2 is encoded by the Encryption Key Manager and stored on the tape cartridge.

Specification of the key encoding does not by itself enable encryption. Encryption must be enabled by a data class that specifies an encryption format, for example EEFMT2.

For complete documentation on using tape encryption, see *z/OS DFSMS Software Support for IBM System Storage TS1140, TS1130, and TS1120 Tape Drives (3592).*

## Syntax

```
KEYENCD2=L|H
```

## Subparameter definition

**L** Indicates that the key label 2 will be stored as part of the EEDK structure on the tape cartridge.

**H** Indicates that a hash of the public key referenced by the key label 2 will be stored on the cartridge rather than the key label.

## Overrides

KEYENCD2 overrides the encoding mechanism of the label for the key encrypting key defined in the DATACLAS parameter for the data set. See "Overrides" on page 134.

## Relationship to other parameters

If you specify the KEYENCD2 parameter on a DD statement, you must also code the KEYLABL2 parameter.

## Example of the KEYENCD2 parameter

***Using label encoding:***

```
//DD1  DD  DSN=DSN5,DISP=(NEW,CATLG),STORCLAS=ATL,
// KEYLABL2='LABELQ1.LABELQ2.LABELQ3',KEYENCD2=L
```

***Using hash encoding:***

```
//DD1   DD  DSN=DSN5,DISP=(NEW,CATLG),STORCLAS=ATL,
// KEYLABL2='LABELQ1.LABELQ2.LABELQ3',KEYENCD2=H)
```

# KEYLEN parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the KEYLEN parameter to specify the length of the keys used in a new data set.

Code the KEYLEN parameter when you want to:

- Specify a key length for the data set or
- With SMS, override the key length defined in the data class of the data set.

The key length can be supplied from the data set label (or data class with SMS). If a key length is not specified or supplied, input or output requests must not require keys.

KEYLEN applies to data sets with the BDAM, BPAM, BSAM, EXCP, and QISAM access methods, and, with SMS, to VSAM data sets.

## Syntax

```
KEYLEN=bytes
```

## Subparameter definition

**bytes**
Specifies the length, in bytes, of the keys used in the data set.

The number of bytes is:

- 0 to 255 for non-VSAM data sets. The key length must be less than or equal to the record length.

  **Note:** Use only 0 for a member of a partitioned data set extended (PDSE). Use 0 or 8 to perform input operations on the directory of a PDSE.
- 1 to 255 for VSAM key-sequenced (RECORG=KS) data sets. A key length must be specified, either explicitly with the KEYLEN or LIKE parameter, or in the data class for the data set. The key length must be less than the record length.

## Overrides

KEYLEN overrides the key length specified in the data set label, and with SMS, KEYLEN overrides the key length defined in the DATACLAS parameter for the data set. See “Overrides” on page 134.

## Relationship to other parameters

Do not code the following DD parameters with the KEYLEN parameter.

| | |
|---|---|
| * | DCB=STACK |
| DATA | DCB=TRTCH |
| DCB=KEYLEN | DDNAME |
| DCB=MODE | DYNAM |
| DCB=PRTSP | |

## Examples of the KEYLEN parameter

***Example 1***

```
//DD4   DD    DSNAME=JST,DISP=(NEW,KEEP),UNIT=3390,
//            SPACE=(CYL,(12,2)),DCB=(A.B.C),KEYLEN=8
```

DD statement DD4 defines a new data set named JST and requests that the system copy the DCB information from the data set label of the cataloged data set named A.B.C. If the data set label contains a key length specification, it is overridden by the KEYLEN coded on this DD statement.

***Example 2***

```
//SMSDS3  DD  DSNAME=MYDS3.PGM,DATACLAS=VSAM1,DISP=(NEW,KEEP),
//          KEYLEN=6
```

In the example, where the data class VSAM1 defines a key-sequenced VSAM data set, the key length of 6 overrides the key length defined in the data class.

# KEYOFF parameter

***Parameter type***

Keyword, optional–use this parameter only with SMS.

Without SMS, use the RKP subparameter of the DCB parameter described in "DCB subparameters" on page 140.

***Purpose***

Use the KEYOFF parameter to specify the key offset, the position of the first byte of the record key in each logical record of a new VSAM data set. The first byte of a logical record is position 0.

If SMS is not installed or is not active, the system syntax checks and then ignores the KEYOFF parameter.

Code the KEYOFF parameter only for a VSAM key-sequenced data set (RECORG=KS).

Code the KEYOFF parameter when you want to (1) specify a key offset for the data set or (2) override the key offset defined in the data class of the data set.

***References***

See *z/OS DFSMS Using Data Sets* for information on VSAM key-sequenced data sets.

## Syntax

```
KEYOFF=offset-to-key
```

## Subparameter definition

**offset-to-key**
Specifies the position (offset), in bytes, of the first byte of the key in each record. The offset is 0 to the difference between the record length (LRECL) and key length (KEYLEN), in the range 0 to 32,760.

## Overrides

KEYOFF overrides the key offset defined in the DATACLAS parameter for the data set. See "Overrides" on page 134.

## Relationship to other parameters

Do not code the following DD parameters with the KEYOFF parameter.

*
DATA
DCB=RESERVE
DCB=RKP
DDNAME
DYNAM
FCB
FREE=CLOSE
UCS

### Example of the KEYOFF parameter

```
//SMSDS3  DD  DSNAME=MYDS3.PGM,DATACLAS=VSAM1,DISP=(NEW,KEEP),
//            KEYOFF=2
```

In the example, the data class VSAM1 defines a key-sequenced VSAM data set. The key offset of 2 overrides the key offset defined in the data class and specifies that the first byte of the key is in the third position of each record.

## LABEL parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the LABEL parameter to specify for a tape or direct access data set:

- The type and contents of the label or labels for the data set.
- If a password is required to access the data set.
- If the system is to open the data set only for input or output.
- The expiration date or retention period for the data set.

Although subparameters RETPD and EXPDT are shown in the syntax of the LABEL parameter, you should use the RETPD or EXPDT DD parameter to specify a retention period or expiration date for the data set.

For a tape data set, this parameter can also specify the relative position of the data set on the volume.

***References***

For details on tape labels, see *z/OS DFSMS Using Magnetic Tapes*. For details on direct access labels, see *z/OS DFSMS Using Data Sets*. For information on protecting a data set with a password, see *z/OS DFSMSdfp Advanced Services*.

### Syntax

```
LABEL=([data-set-sequence-number][,label] [,PASSWORD][,IN ][,RETPD=nnnn      ])
                                  [,     ] [,NOPWREAD][,OUT][,EXPDT={yyddd   }]
                                           [,        ]      [       {yyyy/ddd}]

         label is one of the following:
             AL
             AUL
             BLP
             LTM
             NL
             NSL
             SL
             SUL
```

The first four subparameters are positional; the last subparameter is keyword. If you omit any positional subparameters but code a following **positional** subparameter, indicate each omitted subparameter by a comma. If the following subparameter is one of the keyword subparameters (EXPDT or RETPD), you do not need commas to indicate omitted subparameters. For example:

- LABEL=(0001,SUL,PASSWORD,IN)
- LABEL=(,SUL,PASSWORD)
- LABEL=(,SUL,,IN,EXPDT=97033)
- LABEL=(,,PASSWORD,EXPDT=1997/033)
- LABEL=(,SUL,EXPDT=1997/033)
- LABEL=(0001,,,IN)
- LABEL=(0001,EXPDT=1997/033)

If you specify only the data-set-sequence-number or only the retention period or only the expiration date, you can omit the parentheses. For example, code LABEL=data-set-sequence-number, LABEL=RETPD=nnnn, LABEL=EXPDT=yyddd, or LABEL=EXPDT=yyyy/ddd.

***Alternate syntax for RETPD and EXPDT:*** RETPD and EXPDT should be specified as DD parameters rather than subparameters of the LABEL parameter. This allows you to specify a retention period or expiration date without the need to code LABEL.For example, code RETPD and EXPDT on the DD statement as:

- RETPD=366 or EXPDT=2006/033

See the DD RETPD parameter described in "RETPD parameter" on page 258, and the DD EXPDT parameter described in "EXPDT parameter" on page 193.

## Subparameter definition

### Data-set-sequence-number

**`data-set-sequence-number`**

Identifies the relative position of a data set on a tape volume. Before z/OS V1R5 the value of the data-set-sequence-number on the LABEL parameter was limited to 9999. However, starting with z/OS V1R5, the system allows data set sequence numbers up to 65535 for the following media:

- Standard label (SL) tapes, including standard user label tape (SUL) and leading tape mark (LTM).
- Unlabled (NL) tapes.
- Bypass label processing (BLP)

For data set sequence numbers greater than 4 decimal digits, up to 65535, you must use the OPEN macro instruction in your application as follows:

- For uncataloged data sets, update the data set sequence number in the JFCB and use the OPEN,TYPE=J macro.
- For catalogued data sets, use the OPEN macro. The catalog provides the data set sequence number.

If you do take advantage of data set sequence numbers above 4 decimal digits, you should modify any non-IBM applications that print tape labels or access the tape labels directly, because the data set sequence number in the SL label might not be in EBCDIC format.. For additional information on data set sequence numbers, see *z/OS DFSMS Macro Instructions for Data Sets*.

Omit this subparameter or code 0 or 1 to indicate the first data set on the tape volume. Also omit this subparameter for the following:

- Cataloged data sets. The system obtains the data-set-sequence-number from the catalog.
- A DD DSNAME parameter that requests all members of a generation data group (GDG). The system retrieves the data-set-sequence-number from the catalog.
- A data set passed from a preceding step. The system obtains the data-set-sequence-number from the passing step.

## Label

The system does not retain label type information for cataloged data sets; if the label type is not coded in the LABEL parameter for a cataloged data set, the system assumes SL.

For a data set on a direct access device, the system obtains the label type from the DD statement; the label type is not obtained from any other source referred to in the DD statement. Only two label types are valid for direct access devices: SL and SUL.

**SL**
Indicates that a data set has IBM standard labels. If this subparameter is omitted, SL is the default.

Code only SL or SUL for data sets on direct access devices.

If the LABEL parameter is coded on a SYSCKEOV DD statement, code LABEL=(,SL).

**SUL**
Indicates that a data set has both IBM standard and user labels.

Code only SL or SUL for data sets on direct access devices.

Do not code SUL for partitioned or indexed sequential data sets.

**AL**
Indicates that a tape data set has ISO/ANSI Version 1 or ISO/ANSI/FIPS Version 3 labels.

If you specify AL for a tape generation data set for output, the ending .GnnnnVnn (where n=0 through 9) **will not** appear as part of the file identifier (data set name field) of the HDR1 label. Instead, the data is placed in the generation and version number fields of the HDR1 label.

**AUL**
Indicates that a tape data set has user labels and ISO/ANSI Version 1 or ISO/ANSI/FIPS Version 3 labels.

**NSL**
Indicates that a tape data set has nonstandard labels.

Before you code NSL, ensure that your installation has created and installed non-standard label processing routines, described in *z/OS DFSMS Installation Exits*.

**NL**
Indicates that a tape data set has no labels.

When retrieving two or more data sets from several NL or BLP tape volumes, concatenate the DD statements and repeat the LABEL parameter on each DD statement.

If you are processing ASCII data on unlabeled tapes, the data control block must specify OPTCD=Q.

**BLP**
Requests that the system bypass label processing for a tape data set.

If the installation did not specify the BLP feature in the reader cataloged procedure, BLP has the same effect as NL.

If you code BLP and the tape volume has labels, a tapemark delimits the data set. To let the system position a tape with labels to the proper data set, code the data-set-sequence-number subparameter; the number must reflect all labels and data sets that precede the desired data set.

Do not specify BLP when the DD DSNAME parameter requests all members of a generation data group (GDG); the system obtains the data-set-sequence-number from the catalog. Therefore, coding BLP might result in incorrect tape positioning.

When retrieving two or more data sets from several NL or BLP tape volumes, or when retrieving a data set from several BLP tape volumes and those volumes have labels, concatenate the DD statements and repeat the LABEL parameter on each DD statement.

**LTM**
Indicates that the data set has a leading tapemark.

**Note:** You may use the LABEL parameter when allocating a system-managed tape volume, but you cannot use the NSL or LTM subparameters. If the ACS routine does not exclude these subparameters, the job will fail with JCL errors.

System-managed tape volumes must be IBM standard label or ANSI standard tapes.

## Password protection

For an SMS-managed data set (one with an assigned storage class), SMS sets the password indicators in the VTOC and catalog but ignores the indicators and does not use password protection for the data set. See the DD SECMODEL parameter description in “SECMODEL parameter” on page 262.

Password protecting data sets requires the following:

- Data set names no longer than 17 characters. MVS retains in the tape label only the rightmost 17 characters of the data set name. Consequently, longer names could be identical in password checks.
- Volumes with IBM standard labels or ISO/ANSI/FIPS Version 3 labels.
- A password assigned in the PASSWORD data set. If a password is not assigned, the system will abnormally terminate a job step when it attempts to open the data set for output, if NOPWREAD is coded, or for input or output, if PASSWORD is coded.

To create a password-protected data set following an existing password-protected data set, code the password of the existing data set. The password must be the same in both the existing and the new data set.

To password-protect a data set on a tape volume containing other data sets, you must password-protect all the data sets on the volume and the passwords must be the same for all data sets.

To password-protect an existing data set using PASSWORD or NOPWREAD, open the data set for output the first time it is used during the job step.

**PASSWORD**
Indicates that a data set cannot be read, changed, deleted, or written to unless the system operator or TSO/E user supplies the correct password.

**NOPWREAD**
Indicates that a data set cannot be changed, deleted, or written to unless the system operator or TSO/E user supplies the correct password. No password is necessary for reading the data set.

### Input or output processing

**IN**
One of the following

- Indicates that a BSAM data set opened for INOUT or a BDAM data set opened for UPDAT is to be read only. The IN subparameter overrides the processing option in the assembler OPEN macro instruction. Any attempt by the processing program to write in the data set makes the system give control to the error analysis (SYNAD) routine.
- In a system-managed tape library environment LABEL=(,,,IN) indicates that the allocated volume will be used for read-only purposes and that a read-compatible device can be allocated. For example, if the volume was written using 128-track recording technology on a 3590 Model B, a 3590 Model E or 3590 Model H device can also be allocated for this request. For more information about read-compatibility in a system-managed tape library environment and with other device types, see the tape device selection information in *z/OS DFSMS OAM Planning, Installation, and Storage Administration Guide for Tape Libraries*.

**OUT**
Indicates that a BSAM data set opened for OUTIN or OUTINX is to be written in only. The OUT subparameter overrides the processing option in the assembler OPEN macro instruction. Any attempt by the processing program to read the data set makes the system give control to the error analysis (SYNAD) routine.

### Retention period or expiration date for data set

Avoid using the RETPD and EXPDT subparameters on the LABEL parameter to specify a retention period or expiration date for the data set. Use the DD RETPD parameter ("RETPD parameter" on page 258) or the DD EXPDT parameter ("EXPDT parameter" on page 193), which allow you to specify a retention period or expiration date without coding the LABEL parameter.

## Defaults

- If no data-set-sequence-number subparameter is specified or if the number is coded as 0 or 1, the default is the first data set on the tape volume, unless the data set is passed or cataloged.
- If no label type subparameter is specified, the default is only IBM standard labels (SL).

## Relationship to other parameters

Do not code the following parameters with the LABEL parameter.

| | | |
|---|---|---|
| * | DATA | MODIFY |

| | | |
|---|---|---|
| BURST | DDNAME | QNAME |
| CHARS | DYNAM | SYSOUT |
| COPIES | FLASH | |

Do not specify the LABEL parameter with the FUNC subparameter of the DCB parameter. The results are unpredictable.

ISO/ANSI/FIPS Version 3 tape data sets can be protected by use of the ACCODE parameter.

If you specify a LABEL parameter on a SYSCKEOV DD statement, code LABEL=(,SL).

## Relationship to other control statements

When a VOLUME=REF subparameter refers to an earlier DD statement to use the same volume(s):

- For tape, the system copies the LABEL label type subparameter from the referenced DD statement; the copied label type overrides the label type on the referencing DD statement.
- For direct access, the system uses a LABEL=(,SL) or LABEL=(,SUL) subparameter from the referencing DD statement. If the referencing DD statement specifies any other label type, the system copies the LABEL label type subparameter from the referenced DD statement; the copied label type overrides the label type on the referencing DD statement.
- You do not need to provide a data set sequence number when the DD DSNAME parameter references all the members of a GDG or a single member through a relative generation number; the system obtains the data from the catalog. For all other data set names, however, you must provide the data set sequence number on the LABEL parameter.

## Data conversion

AL or AUL in the LABEL parameter requests conversion between EBCDIC and ASCII. You can also request conversion by specifying OPTCD=Q in the data control block. If the tape is not labeled, LABEL=(,NL), you *must* specify OPTCD=Q for conversion to occur.

## Examples of the LABEL parameter

***Example 1***

```
//DD1  DD  DSNAME=HERBI,DISP=(NEW,KEEP),UNIT=TAPE,
//         VOLUME=SER=T2,LABEL=(3,NSL,RETPD=188)
```

DD statement DD1 defines a new data set. The LABEL parameter tells the system:

- This data set is to be the third data set on the tape volume.
- This tape volume has nonstandard labels.
- This data set is to be kept for 188 days.

Although LABEL=(3,NSL,RETPD=188) is valid, it is better practice to use the DD RETPD parameter as follows:

```
//DD1  DD  DSNAME=HERBI,DISP=(NEW,KEEP),UNIT=TAPE,
//         VOLUME=SER=T2,LABEL=(3,NSL),RETPD=188
```

***Example 2***

```
//DD2  DD  DSNAME=A.B.C,DISP=(,CATLG,DELETE),UNIT=3390,LABEL=(,NL)
```

DD statement DD2 defines a new data set, requests that the system catalog it, and indicates that the data set has no labels. Each time this data set is used by a program, the DD statement must include LABEL=(,NL).

***Example 3***

```
//DD3  DD  DSNAME=SPECS,UNIT=3390,VOLUME=SER=10222,
//         DISP=OLD,LABEL=4
```

DD statement DD3 indicates an existing data set. The LABEL parameter indicates that the data set is fourth on the tape volume.

***Example 4***

```
//STEP1  EXEC PGM=FIV
//DDX    DD   DSNAME=CLEAR,DISP=(OLD,PASS),UNIT=3390,
//            VOLUME=SER=1257,LABEL=(,NSL)
//STEP2  EXEC PGM=BOS
//DDY    DD   DSNAME=*.STEP1.DDX,DISP=OLD,LABEL=(,NSL)
```

DD statement DDX in STEP1 indicates an existing data set with nonstandard labels and requests that the system pass the data set. DD statement DDY in STEP2 receives the data set. DDY contains the label type, because the system does not obtain the label type through the backward reference in the DSNAME parameter.

***Example 5***

```
//DDZ  DD  DSNAME=CATDS,DISP=OLD,LABEL=(,SUL)
```

DD statement DDZ indicates an existing, cataloged data set on direct access. The data set has IBM standard labels and user labels. The LABEL parameter is required; otherwise, if the DD statement does not contain a LABEL parameter, the system assumes that a direct access data set has SL labels.

***Example 6***

```
//DD7  DD  DSNAME=TOM1,DISP=(NEW,KEEP),LABEL=EXPDT=2006/033,
//         UNIT=3390,SPACE=(TRK,(1,1)),VOLUME=SER=663344
```

DD statement DD7 defines a new data set, requests the system to keep the data set, and indicates that the data set cannot be deleted or written over until the expiration date of February 2, 2006.

Although LABEL=EXPDT=2006/033 is valid, it is better practice to use the DD EXPDT parameter as follows:

```
//DD7  DD  DSNAME=TOM1,DISP=(NEW,KEEP),EXPDT=2006/033,
//         UNIT=3390,SPACE=(TRK,(1,1)),VOLUME=SER=663344
```

## LGSTREAM parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the LGSTREAM parameter to specify the prefix of the name of the log stream for an SMS-managed VSAM data set. Use it only for allocating SMS-managed VSAM data sets that will be accessed using record level sharing (RLS).

## Syntax

```
LGSTREAM=name
```

The name, up to a maximum of twenty-six characters, consists of one or more segments. Each segment may contain one to eight characters, which may be alphabetic, numeric, or national ($, #, @) characters. Segments are joined by periods, with periods being counted as characters towards the limit of twenty-six. The first character of each segment must be non-numeric.

## Subparameter definition

**name**
Specifies the name of the prefix the system logger uses for the forward recovery log stream for recording changes made to the data set when accessed in the RLS mode. The system logger adds other qualifiers to the end of the LGSTREAM name to generate the data set name where it keeps the forward recovery logs.

## Defaults

If you do not code a LGSTREAM parameter the system will assign the value specified in the SMS data class assigned to the data set, if applicable.

## Overrides

The system ignores LGSTREAM specifications for non-SMS-managed and non-VSAM data sets and for VSAM linear data sets.

The LGSTREAM name on a DD statement can override the LOGSTREAMID name specified in the SMS data class.

## Relationship to other parameters

Code a disposition of NEW or of MOD treated as NEW. (The system ignores the LGSTREAM parameter for existing data sets.)

Do not code the following DD parameters with the LGSTREAM parameter.

| | | |
|---|---|---|
| * | DLM | QNAME |
| BURST | DYNAM | SEGMENT |
| CHARS | FLASH | SPIN |
| COPIES | MODIFY | SYSOUT |
| DATA | OUTPUT | TERM |
| DCB=DSORG | PATHOPTS | UCS |
| DCB=RECFM | PATHMODE | |
| DDNAME | PATHDISP | |

**Note:** If you code the DSNTYPE parameter with the LGSTREAM parameter, the DSNTYPE value must be EXTREQ or EXTPREF.

### Example of the LGSTREAM parameter

```
//FRED    DD  DSN=VSAM.DATASET,LGSTREAM=SSAB1234.NEW,RECORG=KS,
//            KEYLEN=8,KEYOFF=0,DISP=(,KEEP)
```

In this example, the system will create an SMS-managed VSAM key-sequenced data set if the storage administrator assigns a data class that provides other parameters such as SPACE and LOG=ALL, and assigns a POOL storage group. The system logger will use the name SSAB1234.NEW as the prefix to generate the data set name where it will keep the forward recovery logs.

## LIKE parameter

***Parameter type***

Keyword, optional — use this parameter only with SMS

Without SMS, use the DCB=dsname form of the DCB parameter described in "Subparameter definition" on page 137.

***Purpose***

Use the LIKE parameter to specify the allocation attributes of a new data set by copying the attributes of a model data set, which must be an existing cataloged data set and reside on a direct access volume.

The following attributes are copied from the model data set to the new data set:

- Data set organization
  - Record organization (RECORG) or
  - Record format (RECFM)
- Record length (LRECL)
- Key length (KEYLEN)
- Key offset (KEYOFF)
- Type, PDS, PDSE, basic format, extended format, large format, or HFS (DSNTYPE)
- Space allocation (AVGREC and SPACE)

  Unless you explicitly code the SPACE parameter for the new data set, the system determines the space to be allocated for the new data set by adding up the space allocated in the first three extents of the model data set. Therefore, the space allocated for the new data set will generally not match the space that was specified for the model data set. Note that regardless of the units in which the model data set was allocated, the new data set will be allocated in tracks. This assumes that space was not specified on the JCL and is being picked up from the model data set.

  If a data class with OVERRIDE SPACE(YES) is explicitly specified on the JCL, the SPACE attributes in the data class will take president over any other SPACE attributes specified either through JCL or in the modeled data set. See "DATACLAS parameter" on page 132 for more details.

  **Note:** Directory quantity is picked up as part of the space allocation attribute except when the model data set is a PDSE. When you create a PDS, the directory blocks must be specified directly on the JCL by using the SPACE parameter.

There is no requirement that either the new data set or the model data set must be SMS-managed. If the new data set is to reside on tape:

- The model data set must be a sequential DASD data set
- Only the record format (RECFM) and the record length (LRECL) attributes are copied to the new data set.

For VSAM data set compression, the LIKE parameter copies existing data set attributes. That is, LIKE processing on a model data set that is compressed will pass the attribute to the new data set. This means that specifying compaction in DATACLAS is not the only way compression can be achieved.

When you specify the LIKE parameter on a JCL DD statement, the SMS read-only variable values that correspond to the attributes copied from the model data set are not available as input to the ACS routines. For more information on SMS read-only variables, see *z/OS DFSMSdfp Storage Administration*.

If SMS is not installed or is not active, the system syntax checks and then ignores the LIKE parameter.

The retention period (RETPD) or expiration date (EXPDT) is not copied to the new data set.

**Note:** Do not use the LIKE parameter to copy attributes from a temporary data set (&&dsname), partitioned data set if a member name is included, and relative generation number for a GDG.

## Syntax

```
LIKE=data-set-name
```

## Subparameter definition

**data-set-name**
Specifies the data set name (dsname) of the model data set whose attributes are to be used as the attributes of the new data set.

## Overrides

Any attributes obtained using the LIKE parameter override the corresponding attributes in the DATACLAS parameter.

Any attributes you specify on the same DD statement with the following parameters override the corresponding attributes obtained from the model data set.

- AVGREC (record request and space quantity)
- DSNTYPE (type, PDS, PDSE, basic format, extended format, large format, or HFS)
- KEYLEN (key length)
- KEYOFF (key offset)
- LRECL (record length)
- RECORG (record organization) or RECFM (record format)
- SPACE (average record length, primary, secondary, and directory quantity)

## Relationship to other parameters

Do not code the following DD parameters with the LIKE parameter.

DYNAM
REFDD
SYSOUT

## Examples of the LIKE parameter

***Example 1***

```
//SMSDS6  DD  DSNAME=MYDS6.PGM,LIKE=MYDSCAT.PGM,DISP=(NEW,KEEP)
```

In the example, the data set attributes used for MYDS6.PGM are obtained from the cataloged model data set MYDSCAT.PGM.

***Example 2***

```
//SMSDS7  DD  DSNAME=MYDS7.PGM,LIKE=MYDSCAT.PGM,DISP=(NEW,KEEP),
//              LRECL=1024
```

In the example, the data set attributes used for MYDS7.PGM are obtained from the cataloged model data set MYDSCAT.PGM. Also, the logical record length of 1024 overrides the logical record length obtained from the model data set.

# LRECL parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the LRECL parameter to specify the length of the records in a new data set.

Code the LRECL parameter when you want to

- Specify the logical record length for the data set, or
- With SMS, override the record length defined in the data class of the data set.

LRECL applies to data sets with the BPAM, BSAM, EXCP, QISAM, and QSAM access methods, and with SMS, to VSAM data sets.

## Syntax

```
LRECL=(bytes)
```

## Subparameter definition

**bytes**
Specifies (1) the length, in bytes, for fixed length records or (2) the maximum length, in bytes, for variable-length records.

The value of bytes is:

- 1 to 32,760 for non-VSAM data sets.

- 1 to 32,761 for VSAM key-sequenced (KS), entry-sequenced (ES), or relative record (RR) data sets. (LRECL does not apply to VSAM linear space, RECORG=LS, data sets.)

  For VSAM key-sequenced (KS) data sets, a record length must be specified, either explicitly with the LRECL or LIKE parameter, or in the data class for the data set. The record length must be greater than the key length.

**Note:** When RECFM is F or U, the length must not exceed DCB BLKSIZE. For RECFM=D or V, the length must be a minimum of 5 and a maximum of BLKSIZE minus 4 to account for the 4 byte record descriptor word (RDW) preceding the data in every record. For RECFM=VS, the length can exceed BLKSIZE. For unblocked records when DCB RKP=0, the length is for only the data portion of the record. LRECL=0 is valid only for RECFM=U.

### Additional syntax for LRECL=(bytes)

**LRECL=nnnnnK**
Specifies the length in kilobytes for variable-length spanned records in ISO/ANSI/FIPS Version 3 tape data sets that are processed by the Data Facility Product using the extended logical record interface (XLRI). **nnnnn** is from 1 through 16,383 and indicates multiples of 1024 bytes. The value in the DCB macro must already be coded as LRECL=0K or LRECL=nnnnnK. If a **K** is coded for any other type of data set, only the numeric value of LRECL is recognized.

**LRECL=X**
For QSAM only, specifies that the logical record length exceeds 32,760 bytes for variable-length spanned records. This option is not valid for ISO/ANSI/FIPS Version 3 variable-length records.

## Overrides

LRECL overrides the record length specified in the data set label, and with SMS, LRECL overrides the record length defined in the DATACLAS parameter for the data set. See “Overrides” on page 134.

## Relationship to other parameters

Do not code the following DD parameters with the LRECL parameter.

DCB=LRECL
DDNAME
DYNAM

## Examples of the LRECL parameter

***Example 1***

```
//DD1B    DD  DSNAME=EVER,DISP=(NEW,KEEP),UNIT=3380,
//              RECFM=FB,LRECL=326,SPACE=(23472,(200,40))
```

In the example, the logical record length of 326 is used for the new data set EVER.

***Example 2***

```
//SMSDS2  DD  DSNAME=MYDS2.PGM,DATACLAS=DCLAS02,DISP=(NEW,KEEP),
//              LRECL=256
```

In the example, the logical record length of 256 overrides the logical record length defined in the data class for the data set.

# MAXGENS parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the MAXGENS parameter to specify the maximum number of generations for members in a Version 2 PDSE.

Use MAXGENS only with Version 2 PDSEs.

## Syntax

```
MAXGENS=maximum-generations
```

## Subparameter definition

**maximum-generations**
Specifies the maximum number of generations for members in a Version 2 PDSE.

The value is 0 to 2,000,000,000. The default is 0.

The value may be limited by MAXGENS_LIMIT in the IGDSMSxx member of PARMLIB.

MAXGENS is ignored when specified for a non Version 2 PDSE data set.

## Relationship to other parameters

Do not code the MAXGENS parameter if DSNTYPE= (LIBRARY,2) is not in effect.

## Examples of the MAXGENS parameter

***Example 1 MAXGENS Parameter***

```
//SAM00001 DD DISP=(NEW,CATLG),DSN=IBMUSER.TEST1.PDSE00,
//            DSNTYPE=(LIBRARY,2),LRECL=800,BLKSIZE=8000,RECFM=FB,
//            VOL=SER=338001,
//            UNIT=SYSDA,SPACE=(CYL,(50,50,1)),MAXGENS=10
```

# MGMTCLAS parameter

***Parameter type***

Keyword, optional

This parameter is useful only with SMS-managed data sets.

Without SMS, there are no DD parameters that provide this function.

***Purpose***

Use the MGMTCLAS parameter to specify a management class for a new SMS-managed data set. The storage administrator at your installation defines the names of the management classes you can code on the MGMTCLAS parameter.

After the data set is allocated, the attributes in the management class control:

- Migration of the data set, including migration from primary storage to DFSMShsm-owned storage to archival storage
- Backup of the data set, including frequency of backup, number of versions, and retention criteria for backup versions
- Automatic deletion of data sets
- Automatic release of unused space in data sets

The Hierarchical Storage Manager (DFSMShsm) or a functionally equivalent program performs these functions.

If SMS is not installed or is not active, the system syntax checks and then ignores the MGMTCLAS parameter.

SMS ignores the MGMTCLAS parameter if you specify it for an existing data set.

The use of a management class can be protected by RACF.

***References***

See *z/OS DFSMS Using the Interactive Storage Management Facility* for information on how to use ISMF to view your installation-defined management classes.

## Syntax

```
MGMTCLAS=[management_class_name]
```

**Note:** If you specify a null MGMTCLAS, the JCL parser accepts it but ignores it.

## Subparameter definition

**management-class-name**
Specifies the name of a management class to be used for management of the SMS-managed data set after the data set is allocated.

The name, one to eight alphanumeric or national ($ # @) characters, is defined by the storage administrator at your installation.

## Defaults

If you do not specify MGMTCLAS for a new data set and the storage administrator has provided an installation-written automatic class selection (ACS) routine, the ACS routine may select a management class for the data set. Check with your storage administrator to determine if an ACS routine will select a management class for the new data set, in which case you do not need to specify MGMTCLAS.

## Overrides

You cannot override management class attributes via JCL parameters. With SMS, MGMTCLAS overrides the attributes defined in the DATACLAS parameter for the data set. See “Overrides” on page 134.

The management class for a data set defines a maximum value for the expiration date or retention period of the data set. This maximum limits the values that are specified on the EXPDT or RETPD parameter, or defined in the data class for the data set.

An ACS routine can override the management class that you specify on the MGMTCLAS parameter.

## Relationship to other parameters

Do not code the following DD parameters with the MGMTCLAS parameter.

| | | | |
|---|---|---|---|
| * | DYNAM | DATA | QNAME |
| DDNAME | | | |

Code MGMTCLAS only when you specify a storage class for the data set (via the STORCLAS parameter) or an ACS routine selects a storage class.

## Example of the MGMTCLAS parameter

```
//SMSDS1  DD  DSNAME=MYDS1.PGM,DATACLAS=DCLAS1,STORCLAS=SCLAS1,
//          MGMTCLAS=MCLAS01,DISP=(NEW,KEEP)
```

In the example, SMS uses the attributes in the management class named MCLAS01 to handle the migration and backup of the SMS-managed data set. Note that installation-written ACS routines may override the specified management class, storage class, and data class.

# MODIFY parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the MODIFY parameter to specify a copy-modification module that tells JES how to print this sysout data set on a 3800 Printing Subsystem. The module can specify the following:

- Legends.
- Column headings.
- Where and on which copies the data is to be printed.

The module is defined and stored in SYS1.IMAGELIB using the IEBIMAGE utility program.

**Note:** MODIFY applies only for the 3800 Printing Subsystem Models 1 and 2 and the 3800 Printing Subsystem Models 3, 6, and 8 in compatibility mode.

***References***

For more information on the copy modification module and the IEBIMAGE utility program, see *z/OS DFSMSdfp Utilities*.

## Syntax

```
MODIFY= {module-name        }
        {(module-name[,trc])}
```

- You must code the module-name.
- The trc subparameter is optional. If you omit it, you can omit the parentheses. However, if you omit it, you must not code it as a null; for example, MODIFY=(TAB1,) is invalid.

## Subparameter definition

**module-name**
Identifies a copy-modification module in SYS1.IMAGELIB. The module-name is 1 through 4 alphanumeric or national ($, #, @) characters.

**trc**
Identifies which table-name in the CHARS parameter is to be used. This **table reference character** is 0 for the first table-name specified, 1 for the second, 2 for the third, or 3 for the fourth. The CHARS parameter is on the following, in override order:

1. This DD statement.
2. A referenced OUTPUT JCL statement.
3. A statement in the library member specified on the OUTPUT JCL PAGEDEF parameter.
4. A statement in the SYS1.IMAGELIB member obtained by default.
5. A JES3 initialization statement.

## Defaults

If no MODIFY parameter is specified, JES3 uses an installation default specified at initialization. JES2 provides no installation default at initialization.

If you do not specify **trc** or if the **trc** value is greater than the number of table-names in the CHARS parameter, JES2 uses the first table named in the CHARS parameter and JES3 uses the default character arrangement table.

## Overrides

A MODIFY parameter on a sysout DD statement overrides an OUTPUT JCL MODIFY parameter.

**Note:** A null first subparameter is invalid in a MODIFY parameter on a DD statement, but is permitted on an OUTPUT JCL statement.

## Relationship to other parameters

Do not code the following parameters with the MODIFY parameter.

| | | |
|---|---|---|
| * | DISP | PROTECT |
| AMP | DSID | QNAME |
| DATA | DYNAM | SUBSYS |
| DDNAME | LABEL | VOLUME |

## Relationship to other control statements

MODIFY can also be coded on the following:

- The OUTPUT JCL statement.
- The JES3 //*FORMAT PR statement.
- The JES2 /*OUTPUT statement.

The second character of each logical record can be a TRC code, so that each record can be printed in a different font. This way of specifying fonts is indicated by the OUTPUT JCL TRC parameter.

## Example of the MODIFY parameter

```
//DD1 DD  UNIT=3800,MODIFY=(A,0),CHARS=(GS15,GS10)
```

In this example, the MODIFY parameter requests that the data in the copy-modification module named A replace variable data in the data set to be printed by the 3800. Module A defines which positions are to be replaced and which copies are to be modified. The second subparameter in MODIFY specifies that the first character arrangement table in the CHARS parameter, GS15, be used.

# OUTLIM parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the OUTLIM parameter to limit the number of logical records in the sysout data set defined by this DD statement. When the limit is reached, the system exits to the SYSOUT limit exit routine. If the installation supplies an installation-written routine, the routine can determine whether to terminate the job or increase the limit. If the installation does not supply a routine, the system terminates the job.

**Note:** OUTLIM is valid only on a DD statement with a SYSOUT parameter.

***References***

For more information on the SYSOUT limit exit routine, see *z/OS MVS Installation Exits* .

## Syntax

```
OUTLIM=number
```

## Subparameter definition

**number**
Specifies the maximum number of logical records. The number is 1 through 8 decimal digits from 1 through 16777215.

## Default

(1) If no OUTLIM parameter is specified or OUTLIM=0 is coded and (2) if output is not limited by JES control statements, JES3 uses an installation default specified at initialization; JES2 provides no installation default at initialization.

## Relationship to other parameters

Code the OUTLIM parameter only on a DD statement with the SYSOUT parameter.

Do not code the OUTLIM parameter with the DCB subparameters CPRI or THRESH; these subparameters can alter the OUTLIM value.

***On dump DD statements:*** On a SYSABEND or SYSUDUMP DD statement:

- JES3 ignores the OUTLIM parameter.
- JES2 limits the output as specified on the OUTLIM parameter.

Not only can JECL statement limit output, but the OUTLIM parameter is applied independently of other limits.

## Relationship to other control statements

Output can also be limited by the following:

- The LINES, BYTES, PAGES, or CARDS parameter of the JES2 /*JOBPARM statement.
- The LINES, BYTES, PAGES, or CARDS parameter of the JES3 //*MAIN statement.
- The LINES, BYTES, PAGES, or CARDS parameter of the JOB statement.

## Example of the OUTLIM parameter

```
//OUTDD  DD  SYSOUT=F,OUTLIM=1000
```

The limit for the number of logical records is 1000.

# OUTPUT parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the OUTPUT parameter with the SYSOUT parameter to associate a sysout data set explicitly with an OUTPUT JCL statement. JES processes the sysout data set using the options from this DD statement combined with the options from the referenced OUTPUT JCL statement.

When the OUTPUT parameter references more than one OUTPUT JCL statement, the system produces separate output for each OUTPUT JCL statement.

**Note:** Code the OUTPUT parameter only on a DD statement with either a SYSOUT or SUBSYS parameter. If you code the OUTPUT parameter without SYSOUT, the system checks the OUTPUT parameter for syntax and ignores it, unless you also code the SUBSYS parameter. If you code the SUBSYS parameter, the system passes the OUTPUT parameter to the subsystem identified in the

SUBSYS parameter. The subsystem might support the OUTPUT parameter or might ignore it. The Infoprint Server subsystem, for example, uses the OUTPUT parameter to process a sysout data set. For more information about the Infoprint Server subsystem, refer to *z/OS Infoprint Server User's Guide*.

## Syntax

```
OUTPUT= {reference                }
        {(reference[,reference]...)}

A reference is one of the following:

    *.name
    *.stepname.name
    *.stepname.procstepname.name
```

- You can omit the parentheses if you code only one reference.
- You must not code a null in an OUTPUT parameter. For example, OUTPUT=(,*.name) is invalid.
- You can reference a maximum of 128 OUTPUT JCL statements on one OUTPUT parameter.
- You can code references in any combination. For example, the following are valid:

```
//EXA  DD   SYSOUT=A,OUTPUT=(*.name,*.name,*.stepname.name)
//EXB  DD   SYSOUT=A,OUTPUT=(*.stepname.name,
//          *.stepname.procstepname.name,*.name)
```

- You can code the references to OUTPUT JCL statements in any order.

## Subparameter definition

***.name**
Refers to an earlier OUTPUT JCL statement with **name** in its name field. The system searches for the OUTPUT JCL statement first in the same step, then before the first EXEC statement of the job.

***.stepname.name**
Refers to an earlier OUTPUT JCL statement, **name**, in this step or an earlier step, stepname, in the same job.

***.stepname.procstepname.name**
Refers to an OUTPUT JCL statement in a cataloged or in-stream procedure. Stepname is the name of this job step or an earlier job step that calls the procedure, procstepname is the name of the procedure step that contains the OUTPUT JCL statement, and name is the name field of the OUTPUT JCL statement.

## Defaults

If you do not code an OUTPUT parameter on a sysout DD statement, JES obtains processing options for the sysout data set in the following order:

1. From each OUTPUT JCL statement containing DEFAULT=YES in the same step.
2. From each OUTPUT JCL statement containing DEFAULT=YES before the first EXEC statement in the job, provided that the step contains no OUTPUT JCL statements with DEFAULT=YES.
3. Only from the sysout DD statement, provided that neither the step nor job contains any OUTPUT JCL statements with DEFAULT=YES.

If you do not specify a SYSOUT class on the DD statement, JES3 uses the truncation value associated with the first referenced (or defaulted) OUTPUT statement that does specify a class. If this DD statement specifies an OUTPUT class, JES3 accepts that class and its associated truncation value.

## Overrides

When an OUTPUT JCL statement is used with the sysout DD statement to specify processing, JES handles parameters as follows:

- If a parameter appears on the DD statement, JES uses the parameter.
- If a parameter appears only on the OUTPUT JCL statement, JES uses the parameter.
- If the same parameter appears on both statements, JES uses the DD parameter.

JES uses the whole overriding parameter, ignoring the whole overridden parameter. If a subparameter is left off the overriding parameter, the system does not pick up that subparameter from the overridden parameter. For example:

```
//EXAMP2  OUTPUT   FLASH=(ABCD,3)
//FVZ2    DD       SYSOUT=F,OUTPUT=*.EXAMP2,FLASH=(EFGH)
```

Only EFGH is used. The system ignores all of the FLASH parameter on the OUTPUT JCL statement, including the second parameter.

## Relationship to other parameters

Code the OUTPUT parameter only on a DD statement with the SYSOUT or SUBSYS parameter.

***With INTRDR subparameter in SYSOUT parameter:*** Do not code an OUTPUT parameter when the SYSOUT parameter specifies a JES2 internal reader by an INTRDR parameter.

***Null subparameters:*** A null first subparameter is invalid in a FLASH or MODIFY parameter on a DD statement, but is permitted on an OUTPUT JCL statement. For example, MODIFY=(,3) is valid only on an OUTPUT JCL statement.

***SYSOUT third subparameter:*** You cannot reference a JES2 /*OUTPUT statement using the third subparameter of the SYSOUT parameter if either of the following is also coded:

- The OUTPUT parameter on the same DD statement.
- An OUTPUT JCL statement containing DEFAULT=YES in the same step or before the EXEC statement of the job, when the DD statement does not contain an OUTPUT parameter.

***DEFAULT parameter on OUTPUT JCL statement:*** If you code DEFAULT=YES on an OUTPUT JCL statement, you can still refer to that OUTPUT JCL statement in the OUTPUT parameter of a sysout DD statement.

## Location in the JCL

All referenced OUTPUT JCL statements must precede the DD statement that refers to them. If the referencing DD statement appears in an in-stream or cataloged procedure, the referenced OUTPUT JCL statement must precede the DD statement in the procedure. A sysout DD statement in a procedure cannot refer to an OUTPUT JCL statement in the calling step.

## No match for OUTPUT name

If the system finds no match for the name coded in the OUTPUT parameter, the system issues a JCL error message and fails the job.

## Processing options in multiple references

A sysout DD statement can refer to more than one OUTPUT JCL statement, either explicitly in an OUTPUT parameter containing more than one reference or implicitly when several default OUTPUT JCL statements apply. The processing options for a sysout data set come from one sysout DD statement and one OUTPUT JCL statement. In multiple references, each combination of sysout DD statement and one of the referenced OUTPUT JCL statements produces a separate set of printed or punched output.

Processing options are **not** cumulative across a group of OUTPUT JCL statements.

Note that in JES3, when TYPE=DSISO and/or TRUNC=YES|NO are specified on the SYSOUT initialization statement, and a sysout DD statement that does not specify a class references multiple OUTPUT statements, the data set DSISO/TRUNC characteristics are derived from the first class specification encountered in the OUTPUT statements. If the DD statement does specify a class, the DSISO/TRUNC characteristics are derived from that class.

## Examples of the OUTPUT parameter

***Example 1***

```
//J1     JOB    ,'MARY LUDWIG'
//JOUT   OUTPUT CLASS=C,FORMS=RECP,INDEX=6
//STEP1  EXEC   PGM=XYZ
//SOUT   OUTPUT CLASS=H,BURST=YES,CHARS=GT12,FLASH=BLHD
//ALL    DD     SYSOUT=(,),OUTPUT=(*.JOUT,*.SOUT),COPIES=5
//IN     DD     *
         .
        (data)
         .
/*
```

The OUTPUT parameter references two OUTPUT JCL statements. Therefore, the system prints the single sysout data set twice:

- For DD ALL combined with OUTPUT JOUT, the sysout data set is printed in class C. In the installation, output class C is printed on a 3211 Printer. Combining the parameters from the DD and OUTPUT JCL statements, the system prints 5 copies of the data set on form RECP and indents the left margin 5 spaces.
- For DD ALL combined with OUTPUT SOUT, the sysout data set is printed in class H. In the installation, output class H is printed on a 3800 Printing Subsystem. Combining the parameters from the DD and OUTPUT JCL statements, the system prints 5 copies of the data set with the forms-overlay frame named BLHD using character-arrangement table GT12 and bursts the output.

***Example 2***

```
//J6     JOB    ,'SUE THACKER'
//OUTA   OUTPUT DEST=HQ
//STEP1  EXEC   PGM=RDR
//OUTB   OUTPUT CONTROL=DOUBLE
//DS1    DD     SYSOUT=A,OUTPUT=(*.OUTA,*.OUTB)
```

```
//STEP2  EXEC   PGM=WRT
//OUTC   OUTPUT DEST=ID2742
//DS2    DD     SYSOUT=A,OUTPUT=(*.OUTC,*.STEP1.OUTB)
```

The OUTPUT parameter on DS1 references:

- The job-level OUTPUT JCL statement OUTA to send the sysout data set to HQ.
- The step-level OUTPUT JCL statement OUTB to print the sysout data set double-spaced on the local 3800 Printing Subsystem used for output class A.

The OUTPUT parameter on DS2 references:

- OUTPUT JCL statement OUTB in the first step to print the sysout data set double-spaced on the local 3800 Printing Subsystem used for output class A.
- OUTPUT JCL statement OUTC in the same step to send the sysout data set to userid ID2742, which is attached to the local system.

**Note:** The references to OUTPUT JCL statements are in no particular order.

## PATH parameter

***Parameter type:*** Keyword, optional — use this parameter only with a UNIX file.

***Purpose:*** Use the PATH parameter to specify the name of the UNIX file.

***Reference:*** For information on UNIX files, see *z/OS V2R2.0 UNIX System Services User's Guide*.

**Note:** Allocation verifies the validity of the pathname. However, there is no ENQ or locking of the pathname, so it is possible to modify a pathname component, even in an asynchronous process. Doing this may cause errors in OPEN or unexpected results with no errors reported.

### Syntax

```
PATH=pathname
```

- Enclose the pathname value in single quotation marks if it contains a character other than:
  - Uppercase letters
  - Numbers
  - National characters
  - Slash (/)
  - Asterisk (*)
  - Plus (+)
  - Hyphen (-)
  - Period (.)
  - Ampersand (&)
- Enclose the pathname value in single quotation marks if you continue it on another statement. For example:

  ```
  //EXA  DD  PATH='/u/payroll/directory171/DEPT64directory/accountingDIR/-
  //             personhoursfile'
  ```

  See Chapter 3, "Format of statements," on page 13 for the rules on continuing parameters in apostrophes.

## Subparameter definition

**pathname**

Identifies a file in a ***z/OS UNIX file system***. The pathname consists of the names of the directories from the root to the file being identified, and then the name of the file.

Each directory or filename:

- Is preceded by a slash (/). The system treats any consecutive slashes as a single slash.
- Can contain symbolic parameters.
- Has a length of 1 through 254 characters, not including the slash.
- Consists of printable characters from X'40' through X'FE'. These printable characters include all the characters that can be used in a portable filename, plus additional characters. For a portable filename, use only the portable filename character set, which is listed in *z/OS V2R2.0 UNIX System Services User's Guide*. A filename can contain characters outside this range, but it cannot be specified in JCL.
- Is subject to symbolic substitution. An ampersand (&) (X'50'), followed by a character string that matches a valid symbolic parameter in the JCL, causes a substitution to occur, based on the syntax rules for symbolic parameters.
- Is case-sensitive. Thus, **/u/joe** and **/u/JOE** and **/u/Joe** define three different files.

The pathname:

- Has the form:

  ```
  /name1/name2/name3/.../namen
  ```

- Begins with a slash.
- Has a length of 1 through 255 characters. The system checks the length after substituting for any symbols and before compressing any consecutive slashes.

## Defaults

Defaults for a DD statement with a PATH parameter are:

- If the PATHDISP parameter is not specified, the normal and abnormal disposition is KEEP.
- If the PATHOPTS parameter is not specified, the status is OLD.

## Relationship to other parameters

Only the following JCL parameters can be used with the PATH parameter:

BLKSIZE
BUFNO
DSNTYPE=PIPE
DUMMY
FILEDATA
LRECL
NCP
PATHDISP
PATHMODE
PATHOPTS
RECFM
TERM

Do not code PATHDISP, PATHMODE, or PATHOPTS on a DD statement without a PATH parameter.

Do not code a PATH parameter on the following DD statements:

JOBLIB
STEPLIB
SYSABEND
SYSMDUMP
SYSUDUMP

Coding the PATH parameter is useful only when the following is true:

- The program being run has been coded to recognize and process the PATH specification. Programs designed to use such DD statements must either:
  - Use dynamic allocation information retrieval to obtain the information specified for PATH, PATHOPTS, and PATHMODE, and pass it to the **open()** callable service. See *z/OS V2R2.0 UNIX System Services User's Guide* for details on using **open()**.
  - Use the C/370™ **fopen(//dd: )** function. **fopen()** handles the differences between DD statements with PATH and DSN specified. See *z/OS V2R2.0 UNIX System Services User's Guide* for details on using **fopen()**.

***If***:

- You specify **either**:
  - OCREAT alone

    **or**:
  - Both OCREAT and OEXCL

  on the PATHOPTS parameter,

  ***And if***:
- The file does not exist,

Then MVS performs an **open()** function. The options from PATHOPTS, the pathname from the PATH parameter, and the options from PATHMODE (if specified) are used in the **open()**. MVS uses the **close()** function to close the file before the application program receives control.

For status group options other than OCREAT and OEXCL, the description in this documentation assumes that the application passes the subparameters to the **open()** function without modification. That is, this application uses dynamic allocation information retrieval (the DYNALLOC macro) to retrieve the values specified for PATHOPTS and passes the values to the **open()** function. The application program can ignore or modify the information specified in the JCL.

## Relationship to other statements

A PATH parameter other than **/dev/null** on a DD statement that overrides a procedure statement nullifies the DUMMY parameter on the overridden statement.

Backward and forward references to a DD statement containing a PATH parameter are not permitted. For backward references, the referring DD statement is treated as an error. For forward references, the DD statement referred to is treated as an error.

## Dummy z/OS UNIX files

The following DD statements define a dummy z/OS UNIX file. The statements are equivalent; for DUMMY3, the extra slashes (/) are compressed to single slashes.

```
//DUMMY1  DD PATH='/dev/null'
//DUMMY2  DD DUMMY,PATH=/ANYNAME
//DUMMY3  DD PATH='//dev///null'
```

The system checks the syntax of pathnames specified with DUMMY. In the DD statement DUMMY2, the pathname must be a valid name.

## Example of the PATH parameter

```
//DD1  DD  PATH='/usr/applics/pay.time',PATHOPTS=ORDONLY
```

The DD statement specifies the z/OS UNIX file **pay.time** that is listed in the directory **applics**. The directory **applics** is listed in the directory **usr**. The PATHOPTS parameter specifies that the program can only read the file.

The effects of the missing PATH parameters are:

- The file must already exist, because the statement does not specify PATHOPTS=OCREAT.
- The system will keep the file for both normal and abnormal step terminations, because the statement does not contain a PATHDISP parameter.
- The access permissions were set with a PATHMODE parameter when the file was created.

# PATHDISP parameter

***Parameter type:*** Keyword, optional — use this parameter only with a UNIX file.

***Purpose:*** Use the PATHDISP parameter to specify the disposition of a UNIX file when the job step ends normally or abnormally.

***Reference:*** For information on UNIX files, see *z/OS V2R2.0 UNIX System Services User's Guide*.

## Syntax

```
PATHDISP={normal-termination-disposition                                    }
        ={(normal-termination-disposition,abnormal-termination-disposition)}

PATHDISP=([KEEP  ][,KEEP  ])
        =([DELETE][,DELETE])

A normal-termination-disposition or abnormal-termination-disposition is
one of the following:

     KEEP
     DELETE
```

- If you omit the normal-termination-disposition parameter, you must code a comma to indicate its absence. For example: PATHDISP=(,DELETE)
- If you code only the normal-termination-disposition parameter, you may omit the enclosing parentheses.

## Subparameter definition

**KEEP**
Specifies that the file should be kept:

- When the step ends normally, KEEP is the first subparameter.
- When the step ends abnormally, KEEP is the second subparameter.

**DELETE**
Specifies that the file should be deleted:

- When the step ends normally, DELETE is the first subparameter.
- When the step ends abnormally, DELETE is the second subparameter.

Deleting a file deletes the name for the file. If the file has other names created by **link()** functions, DELETE does not delete the file itself. The file persists until all of its names are deleted.

## Defaults

The system uses KEEP for both the normal and abnormal dispositions:

- If you do not code a value on the PATHDISP parameter — for example, PATHDISP=(,)
- If you do not code a PATHDISP on a DD statement with a PATH parameter

If you code only a normal-termination-disp, such as PATHDISP=DELETE, the abnormal disposition is the same as the normal disposition.

If you code only an abnormal-termination-disp, such as PATHDISP=(,DELETE), the system uses KEEP for the normal disposition.

## Relationship to other parameters

Code the PATHDISP parameter only on a DD statement that contains a PATH parameter.

You can code the following parameters with the PATHDISP parameter:

BLKSIZE
BUFNO
DSNTYPE=PIPE
DUMMY
FILEDATA
LRECL
NCP
PATH
PATHMODE
PATHOPTS
RECFM
TERM

## Example of the PATHDISP parameter

```
//DD1  DD  PATH='/usr/applics/pay.time',PATHDISP=(KEEP,DELETE)
```

The DD statement identifies a file that already exists. The DD statement requests that the system keep the file, if the step ends normally. If the step ends abnormally, the system deletes the filename and, if no other names were set using **link()**, deletes the file itself.

# PATHMODE parameter

***Parameter type:*** Keyword, optional — use this parameter only with a UNIX file.

***Purpose:*** Use the PATHMODE parameter to specify the file access attributes when the system is creating the UNIX file named on the PATH parameter. Creating the file is specified by a PATHOPTS=OCREAT parameter.

***Reference:*** For information on UNIX files, see the *z/OS V2R2.0 UNIX System Services User's Guide*.

## Syntax

```
PATHMODE={file-access-attribute                              }
         {(file-access-attribute[,file-access-attribute]...)}

A file-access-attribute is one of the following:

For file owner class:        SIRUSR
                             SIWUSR
                             SIXUSR
                             SIRWXU

For file group class:        SIRGRP
                             SIWGRP
                             SIXGRP
                             SIRWXG

For file other class:        SIROTH
                             SIWOTH
                             SIXOTH
                             SIRWXO

To set user and group IDs:   SISUID
                             SISGID
```

- You can specify up to 14 file-access-attributes.
- The file-access-attributes can be in any order.
- Duplicate file-access-attributes are treated as one specification.
- Do not code null positions. For example, do not code PATHMODE=(,file-access-attribute) or PATHMODE=(file-access-attribute,,file-access-attribute).

## Subparameter definition

### For file owner class

The file owner class consists of the user who created the file or who currently owns the file. The user is identified by an OMVS user ID (UID).

**SIRUSR**
Specifies permission for the file owner to read the file.

**SIWUSR**
Specifies permission for the file owner to write the file.

**SIXUSR**
Specifies permission for the file owner either:

- To search, if the file is a directory
- To execute the program in the file, for a file other than a directory

**SIRWXU**
Specifies permission for the file owner either:

- To read, write, and search, if the file is a directory
- To read, write, and execute, for a file other than a directory

This value has the same effect as specifying all three parameters (SIRUSR, SIWUSR, and SIXUSR).

## For file group class

The file group class contains the users who are in the same group as the file. The group is identified by an OMVS group ID (GID).

**SIRGRP**
Specifies permission for users in the file group class to read the file.

**SIWGRP**
Specifies permission for users in the file group class to write the file.

**SIXGRP**
Specifies permission for users in the file group class either:

- To search, if the file is a directory
- To execute the program in the file, for a file other than a directory

**SIRWXG**
Specifies permission for users in the file group class either:

- To read, write, and search, if the file is a directory
- To read, write, and execute, for a file other than a directory

This value has the same effect as specifying all three parameters (SIRGRP, SIWGRP, and SIXGRP).

## For file other class

The ***file other class*** consists of all users other than the file owner or the members of the file's group who can access z/OS UNIX resources on the MVS system.

**SIROTH**
Specifies permission for users in the file other class to read the file.

**SIWOTH**
Specifies permission for users in the file other class to write the file.

**SIXOTH**
Specifies permission for users in the file other class either:

- To search, if the file is a directory
- To execute the program in the file, for a file other than a directory

**SIRWXO**
Specifies permission for users in the file other class either:

- To read, write, and search, if the file is a directory
- To read, write, and execute, for a file other than a directory

This value has the same effect as specifying all three parameters (SIROTH, SIWOTH, and SIXOTH).

## To set user and group IDs in a program

These controls allow users to run a program with the user ID of the file owner or the group ID of the file owner of the program file. They control access authorization a particular program is running. The file owner can set the controls any time, not just in the DD statement.

Do not specify these controls in JCL, because they will be reset when the file is written.

The system overrides the SISUID and SISGID parameters and sets the controls so that no users can run the program when either:

- The DD statement creates the file
- A user writes in the file, thus changing the program

Then, for the program to be run, the file owner must reset the controls.

**SISUID**
Specifies that the system set the user ID of the process to be the same as the user ID of the file owner when the file is run as a program.

**SISGID**
Specifies that the system set the group ID of the process to be the same as the group ID of the file owner when the file is run as a program. The group ID is taken from the directory in which the file resides.

## Defaults

When creating a new z/OS UNIX file, if you do not code a PATHMODE on a DD statement with a PATH parameter, the system sets the permissions to 0, which prevents access by all users. If the z/OS UNIX file already exists, PATHMODE is checked for syntax but ignored. The permission bits are left as they are set.

## Relationship to other parameters

Code the PATHMODE parameter only on a DD statement that contains both a PATH parameter and a PATHOPTS parameter with OCREAT.

If OCREAT is not on the statement, the PATHMODE parameter is checked for syntax and then ignored.

You can code the following parameters with the PATHMODE parameter:

BLKSIZE
BUFNO
DSNTYPE=PIPE
DUMMY
FILEDATA
LRECL
NCP
PATH
PATHMODE
PATHOPTS
RECFM
TERM

*If*:

- You specify **either**:
  - OCREAT alone

    **or**:
  - Both OCREAT and OEXCL

  on the PATHOPTS parameter,

  ***And if***:

- The file does not exist,

Then MVS performs an **open()** function. The options from PATHOPTS, the pathname from the PATH parameter, and the options from PATHMODE (if specified) are used in the **open()**. MVS uses the **close()** function to close the file before the application program receives control.

For status group options other than OCREAT and OEXCL, the description in this documentation assumes that the application passes the subparameters to the **open()** function without modification. That is, this application uses dynamic allocation information retrieval (the DYNALLOC macro) to retrieve the values specified for PATHOPTS and passes the values to the **open()** function. The application program can ignore or modify the information specified in the JCL.

## Example of the PATHMODE parameter

```
//DD1  DD  PATH='/usr/applics/pay.time',PATHDISP=(KEEP,DELETE),
//         PATHOPTS=(OWRONLY,OCREAT,OEXCL),PATHMODE=(SIRWXU,SIRGRP)
```

The DD statement requests that the file named in the PATH parameter be created. The PATHMODE parameter specifies that the file owner can read, write, and search or execute the file and that users in the file group can read the file.

# PATHOPTS parameter

***Parameter type:*** Keyword, optional — use this parameter only with aUNIX file.

***Purpose:*** Use the PATHOPTS parameter to specify the access and status for the UNIX file named in the PATH parameter.

***Reference:*** For information on UNIX files, see *z/OS V2R2.0 UNIX System Services User's Guide*.

## Syntax

```
PATHOPTS={file-option                    }
         {(file-option[,file-option]...)}

A file-option can be in the access or status group and is one of the following:

Access group:  ORDONLY
               OWRONLY
               ORDWR

Status group:  OAPPEND
               OCREAT
               OEXCL
               ONOCTTY
               ONONBLOCK
               OSYNC
               OTRUNC
```

- You can specify up to 7 file-options.
- The file-options can be in any order.
- Code only one file-option from the access group. If you specify more than one file-option from the access group, the system uses ORDWR as the access.
- Code any combination of file-options from the status group.
- Duplicate file-options are treated as one specification.
- Do not code null positions. For example, do not code PATHOPTS=(,file-option) or PATHOPTS=(file-option,,file-option).

## Subparameter definition

### Access group

**ORDONLY**
Specifies that the program should open the file for reading.

**OWRONLY**
Specifies that the program should open the file for writing.

**ORDWR**
Specifies that the program should open the file for reading and writing. Do not use this option for a FIFO special file.

### Status group

**OAPPEND**
Specifies that MVS sets the file offset to the end of the file before each write, so that data is written at the end of the file.

**OCREAT**
Specifies that:

- If the file does not exist, the system is to create it. If a directory specified in the pathname does not exist, one is not created, and the new file is not created.
- If the file already exists and OEXCL was **not** specified, the system allows the program to use the existing file.
- If the file already exists and OEXCL was specified, the system fails the allocation and the job step.

**OEXCL**
Specifies that:

- If the file does not exist, the system is to create it.
- If the file already exists, the system fails the allocation and the job step.

The system ignores OEXCL if OCREAT is not also specified.

**ONOCTTY**
Specifies that if the PATH parameter identifies a terminal device, opening of the file does not make the terminal device the controlling terminal for the process.

**ONONBLOCK**
Specifies the following, depending on the type of file:

- For a FIFO special file:
  - With ONONBLOCK specified and ORDONLY access: An **open()** function for reading-only returns without delay.

  - With ONONBLOCK *not* specified and ORDONLY access: An **open()** function for reading-only blocks (waits) until a process opens the file for writing.
  - With ONONBLOCK specified and OWRONLY access: An **open()** function for writing-only returns an error if no process currently has the file open for reading.
  - With ONONBLOCK *not* specified and OWRONLY access: An **open()** function for writing-only blocks (waits) until a process opens the file for reading.
- For a character special file that supports nonblocking open:
  - If ONONBLOCK is specified: An **open()** function returns without blocking (waiting) until the device is ready or available. Device response depends on the type of device.
  - If ONONBLOCK is *not* specified: An **open()** function blocks (waits) until the device is ready or available.

Specification of ONONBLOCK has no effect on other file types.

**OSYNC**
Specifies that the system is to move data from buffer storage to permanent storage before returning control from a callable service that performs a write.

**OTRUNC**
Specifies that the system is to truncate the file length to zero if all the following are true:

- The file specified on the PATH parameter exists.
- The file is a regular file.
- The file successfully opened with ORDWR or OWRONLY.

The system does not change the mode and owner. OTRUNC has no effect on FIFO special files or character special files.

## Defaults

If you do not code a value on the PATHOPTS parameter or if you do not code a PATHOPTS on a DD statement with a PATH parameter, the system assumes that the pathname exists, searches for it, and issues a message if the pathname does not exist.

If the file exists and you specify PATHOPTS without a file-option for the access group, the allocation succeeds assuming ORDONLY. If the file does not exist and you specify PATHOPTS without a file-option from the access group, the system fails to open the file and issues a message.

## Relationship to other parameters

Code the PATHOPTS parameter only on a DD statement that contains a PATH parameter.

You can code the following parameters with the PATHOPTS parameter:

BLKSIZE
BUFNO
DSNTYPE=PIPE
DUMMY
FILEDATA
LRECL
NCP
PATH
PATHMODE
PATHOPTS
RECFM
TERM

*If*:

- You specify **either**:
  - OCREAT alone

    **or**:
  - Both OCREAT and OEXCL

  on the PATHOPTS parameter,

  *And if*:
- The file does not exist,

Then MVS performs an **open()** function. The options from PATHOPTS, the pathname from the PATH parameter, and the options from PATHMODE (if specified) are used in the **open()**. MVS uses the **close()** function to close the file before the application program receives control.

For status group options other than OCREAT and OEXCL, the description in this documentation assumes that the application passes the subparameters to the **open()** function without modification. That is, this application uses dynamic allocation information retrieval (the DYNALLOC macro) to retrieve the values specified for PATHOPTS and passes the values to the **open()** function. The application program can ignore or modify the information specified in the JCL.

## File status

The MVS system uses the PATHOPTS parameter to determine the status for the file, as follows:

- **OLD** status:
  - PATHOPTS is not on the DD statement.
  - PATHOPTS does not contain a file option.
  - PATHOPTS does not contain OCREAT.
- **MOD** status: PATHOPTS contains OCREAT and OAPPEND, but not OEXCL.
- **NEW** status: PATHOPTS contains both OCREAT and OEXCL.

**Note:**

1. The DISP parameter cannot appear on a DD statement containing the PATH parameter.
2. There is no direct correspondence between the various PATHOPTS settings and the DISP status parameter.

## Example of the PATHOPTS parameter

```
//DD1  DD  PATH='/usr/applics/pay.time',PATHDISP=(KEEP,DELETE),
//         PATHOPTS=(OWRONLY,OCREAT,OEXCL),PATHMODE=(SIRWXU,SIRGRP)
```

OCREAT in the PATHOPTS parameter specifies that the file named in the PATH parameter be created. OWRONLY requests that the system open the file only for writing. OEXCL specifies that, if the file already exists, the system will not create a file and the job step will fail.

# PROTECT parameter

***Parameter Type***

Keyword, optional

Use the PROTECT parameter only if RACF is installed and active.

With SMS, use the SECMODEL parameter to protect data sets; SECMODEL is described in "SECMODEL parameter" on page 262.

***Purpose***

Use the PROTECT parameter to tell the z/OS Security Server, which includes RACF, to protect:

- One data set on a direct access volume.
- One data set on a tape volume with one of the following types of labels:
  - IBM standard labels, LABEL=(,SL) or LABEL=(,SUL)
  - ISO/ANSI/FIPS Version 3 labels, LABEL=(,AL) or LABEL=(,AUL)
  - Nonstandard labels, LABEL=(,NSL), if the installation provides support
- An entire tape volume with one of the following:
  - IBM standard labels, LABEL=(,SL) or LABEL=(,SUL)
  - ISO/ANSI/FIPS Version 3 labels, LABEL=(,AL) or LABEL=(,AUL)
  - Nonstandard labels, LABEL=(,NSL), if the installation provides support
  - No labels, LABEL=(,NL)
  - Bypassed label processing, LABEL=(,BLP)
  - Leading tapemarks, LABEL=(,LTM)

***References***

For more information on RACF, see http://www-03.ibm.com/systems/z/os/zos/features/racf/.

## Syntax

```
PROTECT= {YES}
         {Y  }
```

## Subparameter definition

**YES**
Requests RACF to protect a direct access data set, tape data set, or tape volume. This parameter can also be coded as Y.

## Overrides

With SMS, the DD SECMODEL parameter overrides the PROTECT=YES parameter.

## Relationship to other parameters

Do not code the following parameters with the PROTECT parameter.

| | | |
|---|---|---|
| * | DLM | QNAME |
| BURST | DYNAM | SYSOUT |
| CHARS | FCB | TERM |
| DATA | FLASH | UCS |
| DDNAME | MODIFY | |

***DSNAME parameter for RACF-protected data sets:*** RACF expects the data set name specified in the DSNAME parameter to have a high-level qualifier that is defined to RACF. See the *z/OS Security Server RACF Security Administrator's Guide* for details.

## Requirements for protecting a tape data set

A DD statement that contains a PROTECT parameter to establish RACF protection for a tape data set must:

- Specify or imply VOLUME=PRIVATE.
- Specify or imply DISP=NEW, DISP=OLD, or DISP=SHR; it must not specify or imply DISP=MOD.
- Specify in the LABEL parameter a label type of:
  - SL or SUL for IBM standard labels.
  - AL or AUL for ISO/ANSI Version 1 or ISO/ANSI/FIPS Version 3 tape labels.
  - NSL for nonstandard labels. In this case, the NSL installation exit routine must issue a RACDEF or RACROUTE TYPE=DEFINE macro instruction. See *z/OS Security Server RACROUTE Macro Reference* for a description of these macro instructions.
- If the data set is not the first on the volume, specify a data-set-sequence-number in the LABEL parameter, which requires that the RACF TAPEDSN option be active.

## Requirements for protecting a tape volume

A DD statement that contains a PROTECT parameter to establish RACF protection for a tape volume must:

- Specify or imply VOLUME=PRIVATE.
- Specify or imply DISP=NEW.
- Specify in the LABEL parameter a label type of:
  - SL or SUL for IBM standard labels.
  - AL or AUL for ISO/ANSI Version 1 or ISO/ANSI/FIPS Version 3 tape labels.
  - NSL for nonstandard labels. In this case, the NSL installation exit routine must issue a RACDEF or RACROUTE TYPE=DEFINE macro instruction.

- NL for no labels.
- BLP for bypass label processing.
- LTM for leading tapemark.

Note that RACF cannot fully protect unlabeled tapes because RACF cannot verify the volume serial number directly; the operator must verify the volume serial number when mounting the tape volume.

## Requirements for protecting a direct access data set

A DD statement that contains a PROTECT parameter to establish RACF protection for a direct access data set must:

- Name a permanent data set in the DSNAME parameter.
- Specify a status of DISP=NEW or MOD treated as NEW. RACF can establish protection only when the data set is being created.

## Examples of the PROTECT parameter

***Example 1***

```
//DASD  DD  DSNAME=USER37.MYDATA,DISP=(,CATLG),
//     VOLUME=SER=333000,UNIT=3390,SPACE=(TRK,2),PROTECT=YES
```

This DD statement requests RACF protection for the new direct access data set USER37.MYDATA.

***Example 2***

```
//TAPEVOL  DD  DSNAME=MHB1.TAPEDS,DISP=(NEW,KEEP),LABEL=(,NL),
//          VOLUME=SER=T49850,UNIT=3390,PROTECT=YES
```

This DD statement requests RACF protection for tape volume T49850. Because a specific tape volume is requested, it automatically has the PRIVATE attribute. The volume has no labels.

***Example 3***

```
//TAPEDS  DD  DSNAME=INST7.NEWDS,DISP=(NEW,CATLG),LABEL=(2,SUL),
//         VOLUME=SER=223344,UNIT=3390,PROTECT=YES
```

This DD statement requests RACF protection for INST7.NEWDS, which is the second data set on tape volume 223344. Because a specific tape volume is requested, it automatically has the PRIVATE attribute. The volume has IBM standard and user labels; the RACF TAPEDSN option must be active.

# RECFM parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the RECFM parameter to specify the format and characteristics of the records in a new data set. All the format and characteristics must be completely described in one source, that is, in the data set label of an existing data set, in the DCB macro, in the DD DCB parameter, or in the DD RECFM parameter. However, the processing program can modify the RECFM field in the DCB.

Code the RECFM parameter when you want to (1) specify the record format for the data set or (2) with SMS, override the record format defined in the data class of the data set.

The syntax of the RECFM parameter is described in the following topics:

- Coding RECFM for BDAM Access Method
- Coding RECFM for BPAM Access Method
- Coding RECFM for BSAM, EXCP, and QSAM Access Methods

## Coding RECFM for BDAM access method

***Syntax: BDAM access method:***

```
RECFM=     {U  }
           {V  }
           {VS }
           {VBS}
           {F  }
           {FT }
```

**U** indicates that the records are undefined length.

**V** indicates that the records are variable length.

**VS** indicates that the records are variable length and spanned.

**VBS** indicates that the records are variable length, blocked, and spanned, and that the problem program must block and segment the records.

**F** indicates that the records are fixed length.

**T** indicates that the records may be written using the track-overflow feature.

**Default: undefined-length, unblocked records.**

## Coding RECFM for BPAM access method

***Syntax: BPAM Access Method***

```
RECFM=     {U   } [A]
           {UT  } [M]
           {V   }
           {VB  }
           {VS  }
           {VT  }
           {VBS }
           {VBT }
           {VBST}
           {F   }
           {FB  }
           {FT  }
           {FBT }
```

A or M can be coded with any record format, such as: RECFM=FBA

**A** indicates that the records contain ISO/ANSI control characters.

**B** indicates that the records are blocked.

**F** indicates that the records are fixed length.

**M** indicates that the records contain machine code control characters.

**T** indicates that the records may be written using the track-overflow feature.

**U** indicates that the records are undefined length.

**V** indicates that the records are variable length.

**Default: U**

## Coding RECFM for BSAM, EXCP, and QSAM access methods

*Syntax: BSAM, EXCP, and QSAM Access Methods*

```
RECFM=      {U   } [A]
            {UT  } [M]
            {F   }
            {FB  }
            {FS  }
            {FT  }
            {FBS }
            {FBT }
            {V   }
            {VB  }
            {VS  }
            {VT  }
            {VBS }
            {VBT }
            {VBST}
```

A or M can be coded with any record format, such as: RECFM=FBA

For BSAM, EXCP, and QSAM using ISO/ANSI/FIPS data sets on tape:

```
RECFM=      {D  } [A]
            {DB }
            {DS }
            {DBS}
            {U  }
            {F  }
            {FB }
```

A can be coded with any record format, such as: RECFM=FBA

A or M cannot be specified if the PRTSP subparameter is specified.

**A** indicates that the record contains ISO/ANSI device control characters.

**B** indicates that the records are blocked.

**D** indicates that the records are variable-length ISO/ANSI tape records.

**F** indicates that the records are fixed length.

**M** indicates that the records contain machine code control characters.

**S** (1) For fixed-length records, indicates that the records are written as standard blocks, that is, no truncated blocks or unfilled tracks within the data set, with the exception of the last block or track. (2) For variable-length records, indicates that a record can span more than one block.

**T** indicates that the records can be written using the track-overflow feature, if required.

**U** indicates that the records are undefined length. U is invalid for an ISO/ANSI/FIPS Version 3 tape data set.

**V** indicates that the records are variable length. V cannot be specified for (1)

a variable-length ISO/ANSI tape data set (specify D for this data set), (2) a card reader data set, or (3) a 7-track tape unless the data conversion feature (TRTCH=C) is used.

**Default: U**

## Overrides

RECFM overrides the record format specified in the data set label, and with SMS, RECFM overrides the record format defined in the DATACLAS parameter for the data set. See "Overrides" on page 134. RECFM also overrides any contradictory information from the LIKE data set, the REFDD DD statement and the data class. Examples include RECORG, reuse, and initial load.

## Relationship to other parameters

Do not code the following DD parameters with the RECFM parameter.

| | |
|---|---|
| * | DDNAME |
| AMP | DYNAM |
| DATA | RECORG |
| DCB=DSORG | |
| DCB=RECFM | |

## Examples of the RECFM parameter

***Example 1***

```
//DD1B  DD    DSNAME=EVER,DISP=(NEW,KEEP),UNIT=3380,
//            RECFM=FB,LRECL=326,SPACE=(23472,(200,40))
```

In the example, the record format of fixed block (FB) is used for the new data set EVER.

***Example 2***

```
//SMSDS6  DD  DSNAME=MYDS6.PGM,DATACLAS=DCLAS06,DISP=(NEW,KEEP),
//          RECFM=FB
```

In the example, the record format of fixed block (FB) overrides the record format defined in the data class for the data set.

# RECORG parameter

***Parameter type:*** Keyword, optional — use this parameter only with SMS

Without SMS, see the AMP parameter described in "AMP parameter" on page 105.

***Purpose:*** Use the RECORG parameter to specify the organization of the records in a new VSAM data set.

Code the RECORG parameter when you want to (1) specify the record organization for the data set or (2) override the record organization defined in the data class of the data set.

If SMS is not installed or is not active, the system syntax checks and then ignores the RECORG parameter.

***References:*** See *z/OS DFSMS Using Data Sets* for information on VSAM data sets.

## Syntax

```
         {KS}
RECORG=  {ES}
         {RR}
         {LS}
```

## Subparameter definition

**KS**
Specifies a VSAM key-sequenced data set.

**ES**
Specifies a VSAM entry-sequenced data set.

**RR**
Specifies a VSAM relative record data set.

**LS**
Specifies a VSAM linear space data set.

## Defaults

If you do not specify RECORG, SMS assumes a physical sequential (PS) or partitioned (PO) data set.

## Overrides

The RECORG parameter overrides the record organization defined in the DATACLAS parameter for the data set. See "Overrides" on page 134. RECORG also overrides any contradictory information from the LIKE data set, the REFDD DD statement and the data class. Examples include RECFM, BLKSIZE, and DSNTYPE other than EXTPREF or EXTREQ.

## Relationship to other parameters

Do not code the following DD parameters with the RECORG parameter.

| | |
|---|---|
| * | DDNAME |
| DATA | DSNTYPE |
| DCB=DSORG | DYNAM |
| DCB=RECFM | RECFM |
| FREE=CLOSE | |

## Example of the RECORG parameter

```
//SMSDS3  DD  DSNAME=MYDS3.PGM,DATACLAS=VSAM1,DISP=(NEW,KEEP),
//            RECORG=KS
```

In the example, the record organization of key-sequenced (KS) overrides the record organization defined in the data class.

## REFDD parameter

*Parameter type*

Keyword, optional — use this parameter only with SMS

Without SMS, use the DCB=*.ddname form of the DCB parameter described in "Subparameter definition" on page 137.

*Purpose*

Use the REFDD parameter to specify attributes for a new data set by copying attributes of a data set defined on an earlier DD statement in the same job.

The following attributes are copied to the new data set from (1) the attributes specified on the referenced DD statement, and (2) for attributes not specified on the referenced DD statement, from the data class of the data set specified by the referenced DD statement:

- Data set organization
  - Record organization (RECORG) or
  - Record format (RECFM)
- Maximum number of generations for a version 2 PDSE (MAXGENS).
- Record length (LRECL)
- Key length (KEYLEN)
- Key offset (KEYOFF)
- DSNTYPE (type, PDS, PDSE, basic format, extended format, large format or HFS)

  For PDSE (LIBRARY) data sets, the VERSION attribute value is also copied.
- Space allocation (AVGREC and SPACE)

  If a data class with OVERRIDE SPACE(YES) is explicitly specified on the JCL, the SPACE attributes in the data class will take president over any other SPACE attributes specified either through JCL or in the modeled data set. See "DATACLAS parameter" on page 132 for more details.

Only RECFM and LRECL apply to tape data sets.

REFDD does not copy DCB attributes from the data set label. See the DD LIKE parameter.

If SMS is not installed or is not active, the system checks the syntax and then ignores the REFDD parameter.

The retention period (RETPD) or expiration date (EXPDT) is not copied to the new data set.

**Note:** Do not use the REFDD parameter to copy attributes from a temporary data set (&&dsname), partitioned data set if a member name is included, and relative generation number for a GDG.

## Syntax

```
        {*.ddname                       }
REFDD=  {*.stepname.ddname              }
        {*.stepname.procstepname.ddname}
```

## Subparameter definition

***.ddname**
***.stepname.ddname**
***.stepname.procstepname.ddname**
Specify a backward reference to an earlier DD statement. The referenced DD statement cannot name a cataloged data set or refer to another DD statement.

***.ddname**
Specifies the ddname of an earlier DD statement in the same step.

***.stepname.ddname**
Specifies the ddname of a DD statement in an earlier step, stepname, in the same job.

***.stepname.procstepname.ddname**
Specifies the ddname of a DD statement in a cataloged or in-stream procedure called by an earlier job step. Stepname is the name of the job step that calls the procedure and procstepname is the name of the procedure step that contains the DD statement.

Do not reference a DD * or a DD DATA statement.

## Overrides

Any attributes specified on the referenced DD statement override the corresponding data class attributes of the referenced data set.

Any attributes you specify on the referencing DD statement with the following parameters override the corresponding attributes obtained from the referenced DD statement and the data class attributes of the referenced data set.

- RECORG (record organization) or RECFM (record format)
- LRECL (record length)
- KEYLEN (key length)
- KEYOFF (key offset)
- DSNTYPE (type, PDS, PDSE, basic format, extended format, large format, or HFS)
- AVGREC (record request and space quantity)
- SPACE (average record length, primary, secondary, and directory quantity)
- MAXGENS (maximum number of generations for a version 2 PDSE)

## Relationship to other parameters

Do not code the following DD parameters with the REFDD parameter.

DYNAM
LIKE

## Examples of the REFDD parameter

*Example 1*

```
//SMSDS6  DD  DSNAME=MYDS6.PGM,DATACLAS=DCLAS01,DISP=(NEW,KEEP),
//          LRECL=512,RECFM=FB
//SMSDS7  DD  DSNAME=MYDS7.PGM,REFDD=*.SMSDS6,DISP=(NEW,KEEP)
```

In the example, the data set attributes used for MYDS7.PGM are obtained from the referenced data set MYDS6.PGM.

*Example 2*

```
//SMSDS6  DD  DSNAME=MYDS6.PGM,DATACLAS=DCLAS01,DISP=(NEW,KEEP),
//          LRECL=512,RECFM=FB
//SMSDS8  DD  DSNAME=MYDS8.PGM,REFDD=*.SMSDS6,DISP=(NEW,KEEP),
//          LRECL=1024
```

In the example, the data set attributes used for MYDS8.PGM are obtained from the referenced data set MYDS6.PGM. Also, the logical record length of 1024 overrides the logical record length obtained from the referenced data set.

# RETPD parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the RETPD parameter to specify the retention period for a new data set to help reduce the chance of later accidental deletion. After the retention period, the data set can be deleted or written over by another data set.

If the DD statement contains DISP=(NEW,DELETE) or the DISP parameter is omitted to default to NEW and DELETE, the system deletes the data set when the step terminates normally or abnormally, even though a retention period is also specified.

Do not specify RETPD for a temporary data set.

The RETPD parameter achieves the same result as the EXPDT parameter.

Code the RETPD parameter when you want to (1) specify a retention period for the data set or (2) with SMS, override the retention period defined in the data class for the data set.

## Syntax

```
RETPD=nnnn
```

- The RETPD parameter can have a null value only when coded on a DD which either:
  - Overrides a DD in a procedure
  - Is added to a procedure.

## Subparameter definition

**nnnn**

Specifies the retention period, in days, for the data set. The nnnn is one through five decimal digits ( 0-93000).

**Note:** At releases z/OS 1.10 through z/OS 1.12, the system will accept RETPD values up to 93000, however, when the value coded is above 9999, then 9999 will be used.

The system adds nnnn to the current date to produce an expiration date. For SMS data sets, the system adds nnnn to the data set creation date to produce an expiration date. The calculated expiration date uses 365-day years and 366-day leap years. However, if the produced expiration date exceeds December 31, 2155, then the expiration date will be set to December 31, 2155.

**Note:** If you code RETPD and the calculated expiration date is December 31, 1999, the expiration date is set to January 1, 2000.

## Overrides

With SMS, RETPD overrides the retention period defined in the DATACLAS parameter for the data set. See “Overrides” on page 134.

With SMS, both the retention period specified on RETPD and defined in the data class for an SMS-managed data set can be limited by a maximum retention period defined in the management class for the data set.

## Relationship to other parameters

Do not code the following DD parameters with the RETPD parameter.

| | |
|---|---|
| * | DYNAM |
| DATA | EXPDT |
| DDNAME | SYSOUT |

## Deleting a data set before its retention period passes

To delete a data set before the retention period has passed, use one of the following:

- For data sets cataloged in an integrated catalog facility catalog, use the DELETE command, as described in *z/OS DFSMS Access Method Services Commands*.
- For data sets not cataloged in an integrated catalog facility catalog, use the IEHPROGM utility, as described in *z/OS DFSMSdfp Utilities*.
- For a non-VSAM data set, use the SCRATCH macro with the OVRD parameter, as described in *z/OS DFSMSdfp Advanced Services*.
- The system operator can reply "u" to the IEC507D message prompt to delete unexpired data sets.
- You can override the retention period for SMS-managed DASD data sets by specifying OVRD_EXPDT(YES) in the IGDSMSxx SYS1.PARMLIB member and specifying DELETE on the DD DISP statement. The data set will be deleted whether or not the retention period has passed. See *z/OS MVS Initialization and Tuning Reference* for information about the IGDSMSxx parmlib member.

## Examples of the RETPD parameter

***Example 1***

```
//DD1  DD  DSNAME=HERBI,DISP=(NEW,KEEP),UNIT=TAPE,
//         VOLUME=SER=T2,LABEL=(3,NSL),RETPD=188
```

In the example, the data set is not eligible for being deleted or written over for 188 days.

***Example 2***

```
//SMSDS2  DD  DSNAME=MYDS2.PGM,DATACLAS=DCLAS02,DISP=(NEW,KEEP),
//          RETPD=732
```

In the example, the retention period of 732 days overrides the retention period defined in the data class for the data set.

# RLS parameter

***Parameter type***

Keyword, optional

***Purpose***

You can, on a system that includes MVS/DFSMS Version 1 Release 3 or higher, use the RLS parameter to specify the level of record sharing, or **sharing protocol**, for a VSAM data set containing records that must be shared. See *z/OS DFSMS Using Data Sets* for a description of the sharing protocols and to determine whether your application can run in a shared data environment without modification.

**Note:** RLS is most useful for an existing application. For a new or heavily-modified application, you can request record-level sharing in application code and do not need to specify RLS on the DD statement.

## Syntax

```
RLS= {NRI}
     {CR }
     {CRE }
```

## Subparameter definition

**NRI**
Specifies "no read integrity" (NRI). The application can read all records. Use this subparameter if the application can read uncommitted changes made to a data set by another application. NRI provides better performance than the CR subparameter because it avoids the overhead of obtaining a lock when reading a record from the data set.

**CR**
Specifies "consistent read" (CR). This subparameter requests VSAM to obtain a SHARE lock on each record the application reads. This ensures the application will not read uncommitted changes made to a data set by another application. VSAM obtains the lock while processing a GET NUP request, and releases the lock before completing the GET request. An application that processes a data set allocated with RLS=CR may require modification if it tries to read changes to the data set.

**CRE**

Specifies "consistent read explicit" (CRE). This subparameter requests serialization of the record access with update or erase of the record by another unit of recovery. CRE read provides the application a means of ensuring that records read by a unit of recovery are not changed by other units of recovery until the reading unit of recovery issues a syncpoint.

For VSAM record-level sharing (RLS), CRE is valid only for an application that supports commit and backout. With DFSMS Transactional VSAM Services (DFSMStvs), if a batch (non-CICS) application specifies CRE as the value of the JCL RLS keyword or specifies CRE through the ACB, the data set is opened for DFSMStvs access.

After a batch application opens a data set for RLS access with the NRI or CR subparameter, it is an error for the application to request CRE through the RPL.

CRE does not inhibit update or erase of the record by the unit of recovery that issued the CRE request. The consistent-read explicit GET request obtains a SHARE lock on the record. The SHARE lock remains held until a commit or backout request is processed.

CRE readers should issue frequent commits to reduce contention with updaters. CRE read does not affect update locking rules. Deadlocks are possible when a mixture of CRE read and update is used.

## Overrides

Specifying RLS does not override any other JCL parameter. See *z/OS DFSMS Using Data Sets* for a description of how to override the RLS value specified in the JCL.

## Relationship to other parameters

Do not code any of the following DD parameters with the RLS parameter:

| | | |
|---|---|---|
| * (an asterisk) | DSNTYPE | PATHDISP |
| AMP | DYNAM | QNAME |
| BURST | FLASH | SEGMENT |
| CHARS | FREE | SPIN |
| COPIES | MODIFY | SYSOUT |
| DATA | OUTPUT | TERM |
| DCB (see Note) | PATH | UCS |
| DDNAME | PATHOPTS | |
| DLM | PATHMODE | |

**Note:** You can code RLS with DCB as long as the only DCB subparameters you specify are KEYLEN and LRECL.

## Examples of the RLS parameter

***Example 1***

```
//      EXEC PGM=BATCHPRG
//DD1   DD DSN=A,RLS=NRI,DISP=SHR
```

When the program BATCHPRG opens DD1, the data set is to be processed as a shared resource. NRI specifies that an application can read uncommitted changes made by other applications.

***Example 2***

```
//      EXEC PGM=BATCHPRG
//DD2   DD DSN=B,RLS=CR,DISP=SHR
```

When the program BATCHPRG opens DD2, the data set is to be processed as a shared resource. CR specifies that an application can read only committed changes made by other applications.

## SECMODEL parameter

***Parameter type:*** Keyword, optional — use this parameter only with SMS

Without SMS, use the DD PROTECT parameter described in .

***Purpose:*** Use the SECMODEL parameter to specify the name of an existing RACF data set profile that is copied to the discrete data set profile that RACF builds for the new data set.

The following information from the RACF data set profile, which RACF uses to control access to the data set, is copied to the discrete data set profile of the new data set:

- OWNER - indicates the user or group assigned as the owner of the data set profile.
- ID - indicates the access list of users or groups authorized to access the data set.
- UACC - indicates the universal access authority associated with the data set.
- AUDIT/GLOBALAUDIT - indicates which access attempts are logged.
- ERASE - indicates that the data set is to be erased when it is deleted (scratched).
- LEVEL - indicates the installation-defined level indicator.
- DATA - indicates installation-defined information.
- WARNING - indicates that an unauthorized access causes RACF to issue a warning message but allow access to the data set.
- SECLEVEL - indicates the name of an installation-defined security level.

Use the SECMODEL parameter (1) when you want a different RACF data set profile than the default profile selected by RACF or (2) when there is no default profile.

If SMS is not installed or is not active, the system syntax checks and then ignores the SECMODEL parameter.

***References:*** For information about RACF, see *z/OS Security Server RACF Command Language Reference*.

### Syntax

```
SECMODEL=(profile-name[,GENERIC])
```

### Subparameter definition

**profile-name**
Specifies the name of a RACF model profile, discrete data set profile, or generic data set profile. The named profile is copied to the discrete data set profile of the new data set.

If a generic data set profile is named, GENERIC must also be coded.

**GENERIC**
Identifies that the profile-name refers to a generic data set profile.

## Overrides

The SECMODEL parameter overrides the PROTECT=YES parameter.

## Relationship to other parameters

Do not code the following DD parameters with the SECMODEL parameter.

| | |
|---|---|
| * | DDNAME |
| DATA | DYNAM |

## Examples of the SECMODEL parameter

***Example 1***

```
//SMSDS4  DD  DSNAME=MYDS4.PGM,SECMODEL=(GROUP4.DEPT1.DATA),
//          DISP=(NEW,KEEP)
```

In the example, RACF uses the previously defined model data set profile named GROUP4.DEPT1.DATA to control access to the new data set.

***Example 2***

```
//SMSDS5  DD  DSNAME=MYDS5.PGM,SECMODEL=(GROUP5.*,GENERIC),
//          DISP=(NEW,KEEP)
```

In the example, RACF uses the previously defined generic data set profile named GROUP5.* to control access to the new data set.

# SEGMENT parameter

***Parameter type:*** Keyword, optional

***Purpose:*** In a JES2 system, use the SEGMENT parameter to allow part of a job's output to be printed while the job is still executing, or to allow multiple segments of a job's output to be printed simultaneously on multiple printers. With SEGMENT, portions of a data set are spun, one segment at a time. You determine the size of the portion with the SEGMENT parameter. SEGMENT allows you to specify the number of pages produced for a sysout data set before the system processes the segment of the data set. To count pages, JES2 uses the carriage control characters in the data that skip to channel 1.

SEGMENT is supported by JES2 only. The SEGMENT parameter applies only to line mode data sets with RECFM=A or RECFM=M. The system might suspend segmentation if it reaches the threshold for segmentation allowed by JES2. For more information on the segmentation threshold, see *z/OS JES2 Initialization and Tuning Reference*.

## Syntax

```
SEGMENT=page-count
```

## Subparameter definition

**page-count**
Indicates the number of pages produced for the sysout data set for the current segment. When the number is reached, the system spins-off the data segment for output processing.

## Overrides

The system spins the sysout regardless of SPIN, FREE, and OUTDISP specifications.

## Relationship to other parameters

Do not code the following parameters with the SEGMENT parameter.

| | | | |
|---|---|---|---|
| * | DDNAME | EXPDT | QNAME |
| AMP | DISP | LABEL | RETPD |
| CHKPT | DSNAME | LIKE | SUBSYS |
| DATA | DYNAM | PROTECT | VOLUME |

Page mode data is not counted for segmentation.

The system might suspend segmentation if it reaches the threshold for segmentation allowed by JES.

## Example of the segment parameter

```
//DD1   DD  SYSOUT=A,SEGMENT=100
```

In this example, if the sysout data set produced 400 pages, then four separate segments, 100 pages in each, are produced for output processing.

# SPACE parameter

*Parameter type:* Keyword, optional

**Note:** With SMS, code the SPACE parameter when you want to

- Request space for a new data set, or
- Override the space allocation defined in the DATACLAS parameter for the data set.

See the DATACLAS parameter (described in ) and the AVGREC parameter (described in ).

*Purpose:* Use the SPACE parameter to request space for a new data set on a direct access volume. You can request space in two ways:

- Tell the system how much space you want and let the system assign specific tracks.
- Tell the system the specific tracks to be allocated to the data set.

Letting the system assign the specific tracks is most frequently used. You specify only how space is to be measured — in tracks, cylinders, blocks, or records — and how many of those tracks, cylinders, blocks, or records are required.

The SPACE parameter has no meaning for tape volumes; however, if you assign a data set to a device class that contains both direct access devices and tape devices, for example, UNIT=SYSSQ, you should code the SPACE parameter.

If you code the SPACE parameter on a DD statement that defines an existing data set, the SPACE value you specify temporarily overrides the SPACE value used to create the data set. For example, a data set created with SPACE=(CYL,(5,1)) causes 5 cylinders to be allocated to the data set, and, if it needs more space, it can obtain 1 additional cylinder.

Suppose, though, that there is one particular job that specifies DISP=MOD and will write many records to this data set. JCL for this job can define, for example, SPACE=(CYL,(5,10)) to obtain an additional 10 cylinders instead of just 1 cylinder. The override, however, is in effect only for the single step of the job which specifies the overriding SPACE parameter. Any other job or step within the same job that requires a secondary extent and does not have a SPACE parameter override gets just the 1 additional cylinder specified in the JCL that created the dataset.

**Note:**

- When creating VSAM data sets, be aware that there is no direct one-to-one correspondence between 'define cluster' parameters and JCL keyword parameters.
- The average value in the SPACE keyword is meant to be an average block length value for space calculations and is not meant to represent an LRECL value.
- The AVGREC keyword is only to be used as a multiplier in determining how much space is to be allocated.
- When defining VIO data sets, be aware that a SPACE parameter in the JCL or the SPACE value defined for a data class will override the system default space value.
- The size of a data set is limited to 65,535 tracks per volume except for the following types of data sets:
  - Hierarchical File System (HFS)
  - Extended format sequential
  - Partitioned data set extended (PDSE)
  - VSAM
  - Large format

## Syntax

For system assignment of space:

```
SPACE= ({TRK,}(primary-qty[,second-qty][,directory])[,RLSE][,CONTIG][,ROUND])
       ({CYL,}            [,          ]             [,    ][,MXIG  ]
       ({blklgth,}                                        [,ALX   ]
       ({reclgth,}                                        [,      ]
```

To request specific tracks:

```
SPACE= (ABSTR,(primary-qty,address [,directory])
```

To request only directory space:

```
SPACE=(,(,,directory))
```

- You can omit the parentheses around the primary quantity if you do not code secondary, directory, or index quantities. For example,

  `SPACE=(TRK,20,RLSE,CONTIG)  or  SPACE=(TRK,20).`

  Note that if you omit these inner parentheses, you also omit the commas within them.

- All the subparameters are positional. Code a comma to indicate an omitted subparameter if any others follow. Thus:
  - If you code primary and directory quantities and omit a secondary quantity, code a comma inside the inner parentheses to indicate the omission. For example, SPACE=(TRK,(20,,2)).
  - If you omit RLSE but code a following subparameter, code a comma to indicate the omission. For example, SPACE=(TRK,(20,10),,CONTIG) or SPACE=(TRK,20,,CONTIG).
  - If you omit CONTIG, MXIG, or ALX and ROUND follows, code a comma to indicate the omission. For example, SPACE=(400,30,RLSE,,ROUND). If you also omit RLSE, this example becomes SPACE=(400,30,,,ROUND).

## Subparameter definition

The following DD: SPACE subparameters are supported:

### System assignment of space

**TRK**
Requests that space be allocated in tracks.

**CYL**
Requests that space be allocated in cylinders.

**blklgth — (only if AVGREC is not coded)**
Specifies the average block length of the data, in bytes. The blklgth is a decimal number from 0 through 65535. This parameter indicates that the values specified for primary-qty and second-qty are block quantities, and directs the system to compute the number of tracks to allocate using a block length. The value specified for block size uses block length in this computation, with the exception of the value zero. See primary-qty and second-qty descriptions for how a zero block size is handled.

**reclgth — (only if AVGREC is coded and SMS is active)**
With SMS, specifies the average record length of the data, in bytes. The reclgth is a decimal number from 0 through 65535. This parameter indicates that the values specified for primary-qty and second-qty are record quantities, whose average record length is reclgth. If you specify zero, no space will be allocated.

The system allocates DASD space in whole tracks. The number of tracks required depends on how the records are blocked. The system uses one of the following as the block length to compute the number of tracks to allocate, in the order indicated:

1. The block size from the DCB parameter, if specified.
2. The system determined block size, if available.
3. A default value of 4096.

**primary-qty**

Syntax allows for values of 0-16777215. Actual allowances will vary depending on physical and other environmental variables.

Specifies one of the following:

- For TRK, the number of tracks to be allocated.

- For CYL, the number of cylinders to be allocated.
- For a block length, the number of data blocks in the data set.
- For a record length, the number of records in the new data set. Use the AVGREC parameter to specify that the primary quantity represents units, thousands, or millions of records.

**Note:** When you specify TRK or CYL for a partitioned data set (PDS or PDSE), the primary quantity includes the space for the directory. When you specify a block length or record length for a partitioned data set (PDS or PDSE), the primary quantity does not include the directory space; the system assigns the directory to space outside the primary space assignment.

If the data set does not have the space constraint relief option, one volume must have enough available space for the primary quantity. If you request a particular volume and it does not have enough space available for your request, the system terminates the job step. In order for a data set to have the space constraint relief option, it must be SMS-managed and the data class must specify the option.

If you specify a blklgth of zero for the first subparameter, the system uses one of the following as the block length to compute the number of tracks to allocate, in the order indicated:

1. The block size from the DCB parameter, if specified
2. The block size determined from RECFM and LRECL on the DD statement or data class, if available
3. A default value of 4096.

To request an entire volume, either code the ALX parameter or specify in the primary quantity the number of tracks or cylinders on the volume minus the number used by the volume table of contents (VTOC), volume label track, VTOC index, and VVDS (if any). The volume must not contain other data sets.

**second-qty**

Syntax allows for values of 0-16777215. Actual allowances will vary depending on physical and other environmental variables.

Specifies the number of additional tracks, cylinders, blocks, or records to be allocated, if more space is needed. The system does not allocate additional space until it is needed.

With SMS, use the AVGREC parameter to specify that the secondary quantity represents units, thousands, or millions of records. The system computes the number of tracks to allocate using a block length as indicated in the following order:

1. The block size from the DCB parameter, if specified
2. The system determined block size, if available
3. A default value of 4096.

If the first subparameter specifies the average block length, the system computes the number of tracks for the secondary quantity from the second-qty number and one of the following, in order:

1. The blklgth subparameter of the SPACE parameter.
2. The saved average block length value specified when the data set was created, if no SPACE parameter was specified for an existing data set.
3. The block length in the BLKSIZE field of the data control block.

When you specify a secondary quantity and the data set requires additional space, the system allocates the specified quantity:

- In contiguous tracks or cylinders, if available.
- If not available:
  - If the data set does not have the space constraint relief option, in up to five extents.
  - With the space constraint relief option, the system might have to allocate more than five new extents. A data set has this option only if it is SMS-managed and the data class specifies the option.

The system can allocate up to 123 extents for a data set on a volume if it is a PDSE, an HFS data set, an extended format data set, or a VSAM data set in a catalog. For other types of data sets the system can allocate up to 16 extents for each data set on each volume. An extent is space that may or may not be contiguous to other space allocated to the data set. The extents for a data set include the primary quantity space and user-label space.

**Note:** BDAM data sets cannot be extended.

When your program has filled a sequential data set's allocated space on a volume, the system determines where the following data is written as follows:

- If the disposition of the data set is NEW or MOD and the limit on the number of extents on a volume has not been reached, the system attempts to allocate the secondary quantity on the same volume.
- If the disposition of the data set is OLD or SHR, the system examines the next volume specified for the data set.
  - If space has been allocated on the next volume for the data set, the next volume is used for the data set.
  - If space has not been allocated on the next volume for the data set, secondary space is allocated on the next volume for the data set.

  If there is not another volume specified for the data set, the system attempts to allocate the secondary quantity on the current volume.

  Note that your program should not write with a disposition of DISP=SHR unless you take precautions to prevent other programs from writing at the same time.

If the requested volumes have no more available space and if at least one volume is demountable, the system asks the operator to mount scratch (nonspecific) volumes until the secondary allocation is complete. If none of the volumes are demountable, the system abnormally terminates the job step.

**`directory`**
Specifies the number of 256-byte records needed in the directory of a partitioned data set (PDS). Syntax allows for values of 0-16777215.

**Note:**

1. When creating a partitioned data set (PDS), you must request space for a directory.
2. When creating a partitioned data set extended (PDSE), the size of the directory grows dynamically as needed. SMS uses the size requested for a PDSE directory only if you later convert the PDSE to a PDS.
3. When creating a hierarchical file system (HFS) data set, you must specify the number of directory blocks to indicate that this is an HFS data set, but the value has no effect on allocation.

The PDS directory must fit in the first extent of the data set. If the primary quantity is too small for the directory, or if the system has allocated the primary quantity over multiple extents and the first extent is too small for the directory, then the allocation fails.

With SMS, you can specify the number of directory records on the SPACE parameter without specifying any other subparameters. For example:

```
//DD12  DD   DSNAME=PDS.EXMP,DATACLAS=DCLAS12,SPACE=(,(,,20)),
//         DISP=(NEW,KEEP)
```

specifies 20 directory records for the data set. In this example, the number of specified directory records (20) overrides the number of directory records defined in the data class of the data set. (SMS uses all other space allocation attributes defined in the data class of the data set.)

**RLSE (Partial Release)**

Requests that space allocated to an output data set, but not used, is to be released when the data set is closed. This **partial release** parameter causes the close function to release unused space only if the data set is open to allow writing and the last operation was not a read or a POINT macro.

For a multi-volume sequential data set, RLSE releases unused space on the current volume and any subsequent volumes when the data set is closed. This is also valid if the data set is GUARANTEED SPACE.

If you specify RLSE and an abnormal termination occurs, the system does not release unused space even though the data set is open.

RLSE is supported only for sequential, partitioned, and VSAM extended format data sets.

Coding RLSE for primary allocation does not prohibit use of secondary allocation. The secondary request for space is still in effect.

The system ignores a request to release unused space when closing a data set if it cannot immediately obtain exclusive control of the data set. Circumstances that would preclude obtaining exclusive control include:

- Another job is sharing the data set.
- Another task in the same multitasking job is processing an OPEN, CLOSE, EOV, or FEOV request for any other data set.
- Another data control block is open for the data set.

The RLSE subparameter is ignored when TYPE=T is coded in the CLOSE macro instruction.

When coding RLSE for an existing data set, code the unit of measurement and primary quantity as they appeared in the original request. For example, if the original request was:

```
SPACE=(TRK,(100,50))
```

you can release unused tracks when you retrieve the data set by coding:

```
SPACE=(TRK,(100),RLSE)
```

You can release space in the following additional ways other than by deleting the data set:

- Partial release option in the management class
- DFSMShsm space management cycle
- PARTREL macro issued by an authorized program.

**CONTIG**
Requests that space allocated to the data set must be contiguous. This subparameter affects only primary space allocation.

If CONTIG is specified and contiguous space is not available, the system terminates the job step.

**MXIG**
Requests that space allocated to the data set must be (1) the largest area of available contiguous space on the volume and (2) equal to or greater than the primary quantity. This subparameter affects only primary space allocation.

**Caution:** IBM recommends that you use extreme care when coding this parameter. Large amounts of storage could be allocated, depending on how much free space is available at the time the request is made. If you code this parameter, IBM recommends that you also code the RLSE parameter to release any unused space.

**ALX**
Requests that space allocated to the data set is to be up to 5 of the largest areas of available contiguous space on the volume, and each area must be equal to or greater than the primary quantity. The system allocates fewer than 5 areas only when 5 areas of sufficient size are not available. ALX affects only primary space allocation.

For example, assume the following space extents (in tracks) are available: 910, 435, 201, 102, 14, 12, and 8.

If your job requests 14 tracks as its primary allocation, and ALX is in effect, the job receives the following 5 extents: 910, 435, 201, 102, and 14.

However, if the job requests 15 tracks as its primary allocation, it would receive 4 extents: 910, 435, 201, and 102. The job does not receive the 14-track extent because it is less than the primary space allocation.

**Caution:** IBM recommends that you use extreme care when coding this parameter. Large amounts of storage could be allocated, depending on how much free space is available at the time the request is made. If you code this parameter, IBM recommends that you also code the RLSE parameter to release any unused space.

**ROUND**
When the first subparameter specifies the average block length, requests that space allocated to the data set must be equal to an integral number of cylinders. If the first subparameter specifies TRK, or CYL, the system ignores ROUND.

## Request for specific tracks

For an SMS-managed data set (one with an assigned storage class), do not code ABSTR.

**ABSTR**
Requests that the data set be allocated at the specified location on the volume.

**primary-qty**
Specifies the number of tracks to be allocated to the data set.

The volume must have enough available space for the primary quantity. If it does not, the system terminates the job step.

**address**
Specifies the track number of the first track to be allocated. Count the first

track of the first cylinder on the volume as 0. Count through the tracks on each cylinder until you reach the track on which you want the data set to start.

**address**
Specifies the track number of the first track to be allocated. Count the first track of the first cylinder on the volume as 0. Count through the tracks on each cylinder until you reach the track on which you want the data set to start. The absolute track address must be a decimal number equal to or less than 65535.

**Note:** Do not request track 0.

**directory**
Specifies the number of 256-byte records needed in the directory of a partitioned data set.

**Note:** When creating a partitioned data set, you must request space for a directory.

## Overrides

With SMS, the SPACE parameter overrides the space allocation attributes defined in the data class for the data set.

Explicit specification of SPACE on the DD statement overrides both the SPACE and the AVGREC values specified in the data class.

One exception is if a data class with OVERRIDE SPACE(YES) specified is assigned to the allocation. In this case, the space allocation attributes defined in the data class will override what is specified on JCL even with explicit specification of SPACE on the DD statement.

## Relationship to other parameters

Do not code the following parameters with the SPACE parameter.

| | |
|---|---|
| * | DYNAM |
| DATA | QNAME |
| DDNAME | SUBSYS |

***With KEYLEN for block requests:*** If space is requested in blocks and the blocks have keys, code the DD parameter KEYLEN (or the DCB subparameter KEYLEN) on the DD statement and specify the key length.

## SPACE for new data sets with SMS

With SMS, code the SPACE parameter with or without the AVGREC parameter when you want to (1) request space for the data set or (2) override the space allocation attributes defined in the data class for the data set.

## Examples of the SPACE parameter

***Example 1***

```
//DD1  DD  DSNAME=&&TEMP,UNIT=MIXED,SPACE=(CYL,10)
```

The DD statement defines a temporary data set. The UNIT parameter requests any available tape or direct access volume; MIXED is the installation's name for a group of tape and direct access devices. If a tape volume is assigned, the SPACE

parameter is ignored; if a direct access volume is assigned, the SPACE parameter is used to allocate space to the data set. The SPACE parameter specifies only the required subparameters: the type of allocation and a primary quantity. It requests that the system allocate 10 cylinders.

***Example 2***

```
//DD2  DD  DSNAME=PDS12,DISP=(,KEEP),UNIT=3390,
//         VOLUME=SER=25143,SPACE=(CYL,(10,,10),,CONTIG)
```

The DD statement defines a new partitioned data set. The system allocates 10 cylinders to the data set, of which ten 256-byte records are for a directory. Since the CONTIG subparameter is coded, the system allocates 10 contiguous cylinders on the volume.

***Example 3***

```
//REQUEST1 DD  DSNAME=EXM,DISP=NEW,UNIT=3390,VOLUME=SER=606674,
//          SPACE=(1024,75),DCB=KEYLEN=8
//REQUESTA DD  DSNAME=EXQ,DISP=NEW,UNIT=3390,
//          SPACE=(1024,75),DCB=KEYLEN=8
```

These DD statements request space in block lengths. The average block length of the data is 1024 bytes. 75 blocks of data are expected as output. Each block is preceded by a key eight bytes long. The system computes how many tracks are needed, depending on the device requested in the UNIT parameter.

***Example 4***

```
//REQUEST2 DD  DSNAME=PET,DISP=NEW,UNIT=3390,VOLUME=SER=606674,
//          SPACE=(ABSTR,(5,1))
```

In this example, the SPACE parameter asks the system to allocate 5 tracks, beginning on the second track of the volume.

***Example 5***

```
//DD3    DD  DSNAME=MULTIVOL,UNIT=3390,DISP=(,CATLG),
//           VOLUME=SER=(223344,223345),SPACE=(CYL,(554,554))
```

This example shows how to create a multivolume data set on two complete volumes. The two volumes do not contain other data sets. A volume on 3350 Direct Access Storage contains 555 cylinders. The unrequested cylinder contains the volume table of contents (VTOC).

***Example 6***

```
//SMSDS3  DD  DSNAME=MYDS3.PGM,DATACLAS=DCLAS03,DISP=(NEW,KEEP),
//          SPACE=(128,(5,2)),AVGREC=K
```

In this example, the space allocation defined in the DCLAS03 data class is overridden by the SPACE and AVGREC parameters, which indicate an average record length of 128 bytes, a primary quantity of 5K (5,120) records, and a secondary quantity of 2K (2,048) records.

# SPIN parameter

*Parameter type*

Keyword, optional

*Purpose*

Use the SPIN parameter to specify that the output for the SYSOUT data set is to be made available for processing:

- Immediately upon unallocation; or
- At the end of the job.

If you specify the output to be immediately available upon unallocation, you can also specify for the data set to be capable of being spun via operator command, when the data set reaches a certain size, or when the data set has been active for a specified time period.

## Syntax

```
SPIN= {NO                              }
      {UNALLOC                         }
      {(UNALLOC,'hh:mm')  }
      {(UNALLOC,'+hh:mm') }
      {(UNALLOC,nnn [K|M])}
      {(UNALLOC,NOCMND)   }
      {(UNALLOC,CMNDONLY) }
```

**Note:** UNALLOC is supported on JES2 only.

## Subparameter definition

**NO** Indicates that the system makes the sysout data set available for processing as a part of the output at the end of the job, regardless of when the data set is unallocated.

**UNALLOC**
JES2 only. Indicates that the system makes the data set available for processing immediately when the data set is unallocated. If you dynamically unallocate the sysout data set, either explicitly or by specifying FREE=CLOSE, the system makes the data set available for processing immediately. If you do not dynamically unallocate it, the sysout data set is unallocated at the end of the step, and the system will make it available for processing then.

**(UNALLOC,'hh:mm')**
JES2 only. Indicates that the data set is to be spun at time 'hh:mm' each 24 hour period. hh is hours and has a range of 00 through 23. mm is minutes and has a range of 00 through 59. Note that the time must be specified within apostrophes.

**(UNALLOC,'+hh:mm')**
JES2 only. Indicates that the data set is to be spun every **hh:mm**' time interval, where **hh** is hours and has a range of 00-23 and **mm** is minutes and has a range of 00-59. The minimum interval that can be specified is 10 minutes (**mm**). Hours **hh** must be specified even if zero. For example,

SPIN=(UNALLOC,'+00:20') specifies that the data set be spun at 20 minute intervals. Note that the time interval must be specified within apostrophe characters.

**(UNALLOC,nnn[K|M])**
JES2 only. Indicates that the data set is to be spun when it has the specified number of lines, where **nnn** is lines. A minimum of 500 lines must be specified. Specify the optional characters **K** for thousands of lines and **M** for millions of lines.

**(UNALLOC,NOCMND)**
JES2 only. Indicates that the data set cannot be spun before it is unallocated.

**(UNALLOC,CMNDONLY)**
JES2 only. Indicates data set is only to be spun when an operator issues a command to spin the data set.

## Defaults

If you dynamically unallocate the SYSOUT data set, the default is that the data set is immediately available for processing. If you unallocate the SYSOUT data set at the end of the step, the default is that the data set is available for processing at the end of the job.

If you specify SPIN=UNALLOC, the following defaults apply:

- A data set that is closed by the application program is available for processing immediately.
- A data set that is closed as part of the end-of-step cleanup, such as for a program abend, is available for processing at the end of the job.
- A data set can be spun as the result of an operator command. This is the same processing as SPIN=(UNALLOC,CMNDONLY)

If you specify SPIN=NO the default is that the data set is available for processing at the end of the job

## Overrides

The SEGMENT parameter overrides the SPIN parameter.

The SPIN parameter overrides the FREE parameter for SYSOUT data sets.

**Note:** Another way for a program to control when the SYSOUT data set becomes available for processing is to issue a SETPRT macro. For more information, see *z/OS DFSMS Macro Instructions for Data Sets*.

## Relationship to other parameters

Do not code the following parameters with the SPIN parameter.

| | | | |
|---|---|---|---|
| * | DDNAME | LABEL | RETPD |
| AMP | DISP | LIKE | SUBSYS |
| CHKPT | DYNAM | PROTECT | VOLUME |
| DATA | EXPDT | QNAME | |

## Examples of the SPIN parameter

***Example 1***

```
//DD1   DD  SYSOUT=A,FREE=CLOSE,SPIN=UNALLOC
```

In this example, if you explicitly close or dynamically unallocate the SYSOUT data set, the system makes it available for printing immediately. If you do not explicitly close or dynamically unallocate the SYSOUT data set, the system makes it available for printing at the end of the step. If a JES2 command is issued requesting a spin operation ($TJnnn,SPIN or $TJnnn,SPIN,DD=ddname), the data set is made available for printing immediately.

***Example 2***

```
//DD2   DD  SYSOUT=A,FREE=CLOSE,SPIN=NO
```

In this example, the system makes the SYSOUT data set available for printing at the end of the job, regardless of when it is unallocated or closed.

***Example 3***

```
//DD3   DD  SYSOUT=A,FREE=END,SPIN=UNALLOC
```

If a JES2 command is issued requesting a spin operation ($TJnnn,SPIN or $TJnnn,SPIN,DD=ddname), the data set is made available for printing immediately.

In this example, the SYSOUT data set is unallocated at the end of the step, and made available for printing then. If you dynamically unallocate the SYSOUT data set, the system makes it available for printing immediately.

***Example 4***

```
//DD4   DD  SYSOUT=A,FREE=END,SPIN=NO
```

In this example, the system makes the SYSOUT data set available for printing at the end of the job, regardless of whether the data set is unallocated or closed.

***Example 5***

```
//DD5 DD SYSOUT=A,SPIN=(UNALLOC,5K)
```

In this example, the system splits the data set into 5000 record segments and makes the SYSOUT data set available for printing every 5000 records. Whatever remains in the data set at the end of the STEP is available for printing at the end of step.

## STORCLAS parameter

***Parameter type***

Keyword, optional — this parameter is useful only with SMS-managed data sets.

Without SMS or for non-SMS-managed data sets, use the UNIT parameter (described in ) and the VOLUME parameter (described in ).

***Purpose***

Use the STORCLAS parameter to specify a storage class for a new SMS-managed data set. The storage administrator at your installation defines the names of the storage classes you can code on the STORCLAS parameter.

The storage class contains the attributes that identify a storage service level to be used by SMS for storage of the data set. It replaces the storage attributes that are specified on the UNIT and VOLUME parameters for non-SMS-managed data sets.

An **SMS-managed data set** is defined as a data set that has a storage class assigned. A storage class is assigned when either (1) you specify the STORCLAS parameter or (2) an installation-written automatic class selection (ACS) routine selects a storage class for a new data set.

If SMS is not installed or is not active, the system syntax checks and then ignores the STORCLAS parameter.

SMS ignores the STORCLAS parameter if you specify it for an existing data set.

The use of a storage class can be protected by RACF.

*References*

See *z/OS DFSMS Using the Interactive Storage Management Facility* for information on how to use ISMF to view your installation-defined storage classes.

## Syntax

```
STORCLAS=[storage_class_name]
```

**Note:** If you specify a null STORCLAS, the JCL parser accepts it but ignores it.

## Subparameter definition

**storage-class-name**
Specifies the name of a storage class to be used for storage of the data set.

The name, one to eight characters, is defined by the storage administrator at your installation.

## Defaults

If you do not specify STORCLAS for a new data set and the storage administrator has provided an installation-written automatic class selection (ACS) routine, the ACS routine may select a storage class for the data set. Check with your storage administrator to determine if an ACS routine will select a storage class for the new data set, in which case you do not need to specify STORCLAS.

## Overrides

No attributes in the storage class can be overridden by JCL parameters.

An ACS routine can override the storage class that you specify on the STORCLAS parameter.

## Relationship to other parameters

If the storage administrator has specified GUARANTEED_SPACE=YES in the storage class, then volume serial numbers you specify on the VOLUME=SER parameter override the volume serial numbers used by SMS. Otherwise, volume serial numbers are ignored.

**Note:** The UNIT parameter, with a specific device number or esoteric device type and the SMSHONOR keyword, can be used to trim the set of devices assigned to the request through the storage class.

Do not code the following DD parameters with the STORCLAS parameter.

| | | |
|---|---|---|
| * | DYNAM | UNIT=AFF |
| DATA | QNAME | VOLUME=REF |
| DDNAME | | |

## Examples of the STORCLAS parameter

***Example 1***

```
//SMSDS1  DD  DSNAME=MYDS1.PGM,STORCLAS=SCLAS01,DISP=(NEW,KEEP)
```

In the example, SMS uses the attributes in the storage class named SCLAS01 for the storage service level of the data set. Note that installation-written ACS routines may select a management class and data class and can override the specified storage class.

***Example 2***

```
//SMSDS2  DD  DSNAME=MYDS2.PGM,STORCLAS=SCLAS02,DISP=(NEW,KEEP),
//          VOLUME=SER=(223344,224444)
```

In the example, SMS uses the attributes in the storage class named SCLAS02 for the storage service level of the data set. Also, if the storage administrator has specified GUARANTEED_SPACE=YES in the storage class, VOLUME=SER can be coded and the data set will reside on the specified volumes. (However, if space is not available on the volumes, the job step fails.) Note that installation-written ACS routines may select a management class and data class and can override the specified storage class.

# SUBSYS parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the SUBSYS parameter to request a subsystem to process this data set and, optionally, to specify parameters defined by the subsystem.

Do not use the SUBSYS parameter for an SMS-managed data set (one with an assigned storage class).

In a loosely-coupled multiprocessing environment, the requested subsystem must be defined on all processors that could interpret this DD statement. If sub-parameters are supplied, or if the job is to execute on the system, the subsystem must be ACTIVE.

***Considerations for an APPC Scheduling Environment***

In an APPC scheduling environment, avoid coding the system symbolic SYSUID on the SUBSYS parameter. Symbolic substitution is inconsistent when you code SYSUID as a subparameter of SUBSYS parameter.

***References***

For more information on the SUBSYS parameter and subsystem-defined parameters, refer to the documentation for the requested subsystem.

## Syntax

```
SUBSYS= {subsystem-name                                  }
        {(subsystem-name[,subsystem-subparameter]...)}
```

**Single Subparameter**: You can omit the parentheses if you code only the subsystem-name.

**Number of Subparameters** : If needed, you can code up to 254 subsystem-subparameters on a JES2 system (the length of all parm-statements cannot exceed about 8 KB on a JES2 system), or up to 1020 bytes of data on a JES3 system.

**Multiple Subparameters**: When the parameter contains more than the subsystem-name, separate the subparameters by commas and enclose the subparameter list in parentheses. For example, SUBSYS=(XYZ,1724,DT25).

**Positional Subparameters**: If you omit a subparameter that the subsystem considers positional, code a comma in its place.

**Special Characters**: When a subparameter contains special characters, enclose the subparameter in apostrophes. For example, SUBSYS=(XYZ,1724,'KEY=Y').

Code each apostrophe that is part of a subparameter as two consecutive apostrophes. For example, code O'Day as SUBSYS=(XYX,1724,'NAME=O''DAY').

If you code a symbolic parameter on the SUBSYS parameter, you can code the symbolic parameter in apostrophes.

**Continuation onto Another Statement**: Enclose the subparameter list in only one set of parentheses. End each statement with a comma after a complete subparameter. For example:

```
//DS1  DD  DSNAME=DATA1,SUBSYS=(XYZ,1724,'KEY=Y',
//       DT25,'NAME=O''DAY')
```

**Note:** The SUBSYS parameter can have a null value only when coded on a DD which either:

- Overrides a DD in a procedure
- Is added to a procedure.

## Subparameter definition

**subsystem-name**
Identifies the subsystem. The subsystem name is 1 through 4 alphanumeric or national ($, #, @) characters; the first character must be alphabetic or national ($, #, @). The subsystem must be available in the installation.

**subsystem-subparameter**
Specifies information needed by the subsystem. A subparameter consists of alphanumeric, national ($, #, @), or special characters.

## Relationship to other parameters

Do not code the following DD parameters with the SUBSYS parameter:

| | | |
|---|---|---|
| * | DDNAME | QNAME |
| AMP | DYNAM | SEGMENT |
| DATA | MODIFY | SYSOUT |

The specified subsystem can define other parameters that you must not code with the SUBSYS parameter:

***Ignored but permitted DD parameters:*** If you specify any of the following DD parameters, the system checks them for syntax and then ignores them:

HOLD UNIT

If you specify the SPACE parameter, the system checks its syntax and then ignores it, but the subsystem designated on the SUBSYS parameter may use this information when it allocates the DD.

***DISP parameter:*** The system checks the DISP status subparameter for syntax, but always indicates a status of MOD to the subsystem. If the DISP normal or abnormal termination subparameter is CATLG or UNCATLG, the system allocates the appropriate catalog to the subsystem.

***DUMMY parameter:*** If DUMMY is specified with SUBSYS, the subsystem checks the syntax of the subsystem subparameters. If they are acceptable, the system treats the data set as a dummy data set.

***When this statement overrides a procedure statement:*** If SUBSYS appears on a DD statement that overrides a DD statement in a cataloged or in-stream procedure, the following occurs:

- The system ignores a UNIT parameter, if specified, on the overridden DD statement.
- The system nullifies a DUMMY parameter, if specified, on the overridden DD statement.

## Subsystem support for JCL parameters

The specified subsystem might not support all parameters on the DD and OUTPUT JCL statements. Refer to the documentation for the subsystem to determine the JCL parameters that the subsystem supports. For information about the JCL parameters supported by the Infoprint Server subsystem, see *z/OS Infoprint Server User's Guide*.

## Examples of the SUBSYS parameter

***Example 1***

```
//DD1  DD  DSNAME=ANYDS,DISP=OLD,SUBSYS=ABC
```

The DD statement asks subsystem ABC to process data set ANYDS.

***Example 2***

```
//DD1 DD  DSNAME=ANYDS,DISP=OLD,SUBSYS=(XYZ2,
//        'KEYWORD=DATA VALUE1')
```

The DD statement asks subsystem XYZ2 to process data set ANYDS. The system passes the subparameter KEYWORD=DATA VALUE1 to the subsystem. The parameter is enclosed in apostrophes because it contains an equal sign and a blank, which are special characters.

***Example 3***

```
//DD1 DD DSNAME=ANYDS,DISP=OLD,SUBSYS=(XYZ2,IKJ2,
//       'NAME=''MODULE1''','DATE=4/11/86')
```

The DD statement asks subsystem XYZ2 to process the data set ANYDS. The system passes three subparameters to the subsystem: IKJ2, NAME='MODULE1' and DATE=4/11/86. Note that the character string MODULE1 is passed to the subsystem enclosed in apostrophes.

***Example 4***

```
//DD1 DD SUBSYS=(AOP1,'MyPrinter')
```

The DD statement asks the Infoprint Server subsystem named AOP1 to process a sysout data set. The system passes the subparameter MyPrinter to the Infoprint Server subsystem. The subparameter is enclosed in apostrophes because it contains lowercase letters. For more information about the Infoprint Server subsystem, the printer subparameter, and other subparameters that the Infoprint Server subsystem supports, see *z/OS Infoprint Server User's Guide*.

## SYMBOLS parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the SYMBOLS parameter to request JES to perform symbol substitution within in-stream data.

### Syntax

```
SYMBOLS=({JCLONLY|EXECSYS|CNVTSYS} [,logging-DDname])
```

**Valid values**: Specify one of three **SYMBOLS=** values:

**JCLONLY**
Substitute JCL symbols that have been made available by the EXPORT statement and JES Symbols dynamically created by the IAZSYMBL JES symbol service, which is described in *z/OS JES Application Programming*.

**EXECSYS**
Substitute symbols as described for JCLONLY. In addition, substitute system symbols from the system where this job is executing.

**CNVTSYS**
Substitute symbols as described for JCLONLY. In addition, substitute system symbols from the system where this job completed JCL conversion.

*logging-DDname*
Optional parameter that indicates a valid DD name for the data set to use for logging results of the symbol substitution. Rules for DD names are described in "DDNAME parameter" on page 150. Logging is not performed in the following cases:

- if *logging-DDname* is specified on the DD statement which describes an in-stream data set that is the target on the PARMDD keyword. See "PARMDD parameter" on page 354.
- if *logging-DDname* is specified on the SYSTSIN DD statement which describes input data for the TMP (Terminal Monitoring Program).
- if in-stream data set is opened by one task and then read by a different task.
- if data set specified by *logging-DDname* cannot be successfully opened.

## Relationship to other parameters

Do not code the following DD parameters with the SYMBOLS parameter:

| | | |
|---|---|---|
| PATH | PATHOPTS | PATHMODE |
| PATHDISP | RLS | FILEDATA |
| LGSTREAM | | |

## Example of the SYMBOLS parameter

In the following example, a data set is deleted and then reallocated, and two JCL symbols–DSNAME and VOLSER–are exported and used for symbol substitution in the in-stream data sets:

```
//REALLOC  JOB 1,TESTJOB,
//          MSGLEVEL=(1,1),CLASS=A
//E1  EXPORT SYMLIST=(DSNAME)
//E2  EXPORT SYMLIST=(VOLSER)
//S1  SET    DSNAME=HASP.TEST.MACLIB
//S2  SET    VOLSER=J2COM1
//DEALLIB EXEC PGM=IDCAMS,REGION=300K
//DD1       DD UNIT=3390,DISP=OLD,VOL=SER=&VOLSER
//SYSPRINT  DD SYSOUT=*
//SYSIN     DD *,SYMBOLS=JCLONLY
     DELETE &DSNAME -
            NONVSAM PURGE SCRATCH FILE(DD1)
     ALLOCATE DSNAME('&DSNAME.') -
         UNIT(3390) VOLUME(&VOLSER.) -
         NEW CATALOG DSNTYPE(LIBRARY) SPACE(65,15) DIR(56) TRACKS
```

# SYMLIST parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the SYMLIST parameter to list the symbols that pass to the internal reader (INTRDR). The internal reader enables your job to submit another job for later execution. Ensure that the symbols are defined when your job is submitted to INTRDR. Symbols are either JCL symbols that are made available to the job execution phase using the EXPORT statement, or JES symbols that are dynamically created using the JES Symbol service (IAZSYMBL). The special asterisk character (*) indicates that all JCL and JES symbols that meet JCL requirements are passed to the internal reader.

**Note:** This does not include symbols that are defined by JES (such as the SYSUID symbol), unless they are explicitly exported by using the **EXPORT JCL** statement.

## Syntax

```
SYMLIST= { (sym1,sym2,sym3,...) }
         { * }
```

**Single subparameter**: You can omit the parentheses if you are exporting only one symbol.

**Length**: The entire symbol string must not exceed 142 characters. The total character count:

- Includes any commas, which are considered part of the information.
- Excludes any enclosing parentheses or apostrophes, which are not considered part of the information.

**Multiple subparameters**: When exporting more than one symbol, you must separate the symbols by commas and enclose the information within parentheses or apostrophes. For example, SYMLIST=(SYM86,SYM87) or SYMLIST='SYM86,SYM87'.

Code an apostrophe that is part of a symbol name as two consecutive apostrophes. For example, code SYM89'90 as SYMLIST='SYM89''90'

**Continuation onto another statement**: Enclose the symbol name string in parentheses, and end each statement with a comma after a complete subparameter. For example:

```
//EXPORT86  EXPORT  SYMLIST=(SYM86,'SYM87/88','SYM=89',
//  'SYM90')
```

## Relationship to other parameters

Do not code the following DD parameters with the SYMLIST parameter:

| | | |
|---|---|---|
| PATH | PATHOPTS | PATHMODE |
| PATHDISP | RLS | FILEDATA |
| LGSTREAM | | |

**Note:** The SYMLIST parameter has an effect only on a DD statement for an internal reader.

## Example of the SYMLIST parameter

In the following example, FIRSTJOB submits SECJOB. The symbolic parameters DSNAME and VOLSER are passed by the SYSUT2 SYMLIST parameter in FIRSTJOB. These symbols are used in the SYSUT1 DD statement in SECJOB:

```
//FIRSTJOB JOB    MSGLEVEL=(1,1),MSGCLASS=A,NOTIFY=IBMUSER
//MYEXPRT  EXPORT SYMLIST=(DSNAME,VOLSER)
//MYSET1   SET DSNAME=HASP.TEST.MACLIB
//MYSET2   SET VOLSER=J2COM1
//STEP1    EXEC   PGM=IEBGENER
//SYSPRINT DD     SYSOUT=*
//SYSUT2   DD     SYSOUT=(A,INTRDR),SYMLIST=(DSNAME,VOLSER)
//SYSIN    DD     DUMMY
//SYSUT1   DD     DISP=SHR,DSN=TEST.JCL(SECJOB)
```

The previous job assumes that the following job is in the SECJOB member in the TEST.JCL data set:

```
//SECJOB   JOB    MSGLEVEL=(1,1),MSGCLASS=A,NOTIFY=IBMUSER
//STEP1    EXEC   PGM=IEBGENER
//SYSPRINT DD     SYSOUT=*
//SYSUT2   DD     SYSOUT=*
//SYSIN    DD     DUMMY
//*        &DSNAME and &VOLSER from FIRSTJOB are used
//SYSUT1   DD     DISP=SHR,DSN=&DSNAME,VOL=SER=&VOLSER.,
//                UNIT=3390
```

## SYSOUT parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the SYSOUT parameter to identify this data set as a system output data set, usually called a sysout data set.

Do not use the SYSOUT parameter for an SMS-managed data set (one with an assigned storage class).

The SYSOUT parameter also:

- Assigns this sysout data set to an output class. The attributes of each output class are defined during JES initialization.
- Optionally requests an external writer to process the sysout data set rather than JES. An external writer is an IBM- or installation-written program.
- Optionally identifies the forms on which the data set is to be printed or punched.
- Optionally refers to a JES2 /*OUTPUT statement for processing parameters.

The sysout data set is processed according to the following processing options, in override order:

1. The options specified on this sysout DD statement.
2. The options specified on a referenced OUTPUT JCL statement.
3. The options specified on a referenced JES2 /*OUTPUT statement or on a JES3 //*FORMAT statement.
4. The installation default options for the requested output class.

**Note:**

1. If a sysout data set has the same class as the JOB statement MSGCLASS parameter, the job log appears on the same output listing as this sysout data set.
2. An installation should maintain a list of available output classes and their attributes. Some classes should be used for most printing and punching, but others should be reserved for special processing. Each class is processed by an output writer. The system operator starts the output writers for the commonly used output classes. If you plan to specify a special output class, ask the operator to start the output writer for that class. If the writer is not started before the job produces the sysout data set, the data set is retained until the writer is started.
3. If the automatic restart manager (ARM) restarts a job, JES discards all non-spin sysout data sets created during the previous execution. (You can avoid losing that output by adding SPIN=UNALLOC to the DD statement for the SYSOUT data set.)

***References***

For information on output writers and external writers, see *z/OS MVS Using the Subsystem Interface*.

## Syntax

```
SYSOUT= { class                                  }
        { *                                      }
        { ([class] [,writer-name] [,form-name]) }
                   [,INTRDR     ] [,code-name]

SYSOUT=(,)
```

- You can omit the parentheses if you code only a class.
- All of the subparameters are positional. Code a comma to indicate an omitted subparameter as follows:
  - If you omit the class, code a comma to indicate the omission. For example, when other subparameters follow, code SYSOUT=(,INTRDR,FM26). When other subparameters do not follow, code a null class as SYSOUT=(,).
  - If you omit a writer-name but code a form-name or code-name, code a comma to indicate the omission. For example, SYSOUT=(A,,FM26).
  - Omission of the third subparameter does not require a comma. For example, SYSOUT=A or SYSOUT=(A,INTRDR).

## Subparameter definition

**class**
Identifies the output class for the data set. The class is one character: A through Z or 0 through 9, which you may optionally include in quotation marks. The attributes of each output class are defined during JES initialization; specify the class with the desired attributes.

The CLASS value specified on the DD statement when used with the INTRDR option is the output class (MSGCLASS) of the job going through the internal reader, assuming that the job (going through the internal reader) does not specify MSGCLASS= on its job statement. If it does, the MSGCLASS from the JOB statement overrides the class on the SYSOUT=(A,INTRDR).

***** Requests the output class in the MSGCLASS parameter on the JOB statement.

In a JES2 system you can also use the dollar-sign ($) to request the output class in the MSGCLASS parameter on the JOB statement.

**(,)**
Specifies a null class. A null class must be coded to use the CLASS parameter on a referenced OUTPUT JCL statement.

Specifying SYSOUT=(,) nullifies the SYSOUT class, but designates the data set as a sysout data set that JES will process.

Specifying SYSOUT=, nullifies the entire SYSOUT parameter, and causes the system to process the data set as a normal non-subsystem data set. Because there is no DSNAME parameter, the system treats the data set as a temporary data set. To avoid allocation failures, you might need to supply UNIT or SPACE information.

**writer-name**
Identifies the member name (1 to 8 alphanumeric characters) of an installation-written program.

An external writer is a started task used to process output. Because the external writer is a started task, it has a userid associated with it. Process output with an external writer by naming the writer on the DD statement that defines the output:

```
//MYOUTPUT  DD SYSOUT=(A,XTWTR)
```

In order for the writer to process that output, the writer's userid must be in a RACF access list. The access list permits the writer's userid to the SYSOUT data set. The writer's userid is the userid specified in the started procedure table for the writer task. If your installation's policy requires security labels, the security label associated with the external writer must be equal to or greater than the security label associated with the SYSOUT. For more information, see your security administrator.

Do not code STDWTR as a writer-name. STDWTR is reserved for JES and used as a parameter in the MVS operator's MODIFY command.

In a JES3 system, do not code NJERDR as a writer-name. NJERDR is reserved for JES3.

**INTRDR**
Tells JES that this sysout data set is to be sent to the internal reader as an input job stream.

**form-name**
Identifies the print or punch forms. form-name is 1 through 4 alphanumeric or national ($, #, @) characters.

**code-name**
Identifies an earlier JES2 /*OUTPUT statement from which JES2 is to obtain processing characteristics. The code-name must be the same as the **code** parameter on the JES2 /*OUTPUT statement.

**Note:**

- code-name is supported only on JES2 systems.
- Do not specify the code-name subparameter when the job or job step contains a default OUTPUT JCL statement.

## Defaults

In a JES2 system, if you do not specify a class on this DD statement or a referenced OUTPUT JCL statement, JES2 assigns the sysout data set to the output class defined by the MSGCLASS value of the JOB statement. See the override order shown under "Purpose" for how this default is established.

If you do not code a writer-name subparameter on this DD statement or a referenced OUTPUT JCL statement, the installation's job entry subsystem processes the sysout data set.

If you do not code a form-name subparameter on this DD statement or a referenced OUTPUT JCL statement, JES uses an installation default specified at initialization.

## Overrides

The class subparameter of the DD statement SYSOUT parameter overrides an OUTPUT JCL CLASS parameter. On the DD statement, you must code a null class in order to use the OUTPUT JCL CLASS parameter; for example:

```
//OUTDS DD SYSOUT=(,),OUTPUT=*.OUT1
```

The writer-name subparameter of the DD statement SYSOUT parameter overrides an OUTPUT JCL WRITER parameter.

The form-name subparameter of the DD statement SYSOUT parameter overrides an OUTPUT JCL FORMS parameter. Note that the SYSOUT form-name subparameter can be only four characters maximum while both the OUTPUT JCL FORMS form-name and the JES initialization default form names can be eight characters maximum.

## Relationship to other parameters

Do not code the following DD parameters with the SYSOUT parameter.

| | | |
|---|---|---|
| * | DDNAME | LIKE |
| AMP | DISP | PROTECT |
| CHKPT | DYNAM | QNAME |
| DATA | EXPDT | RETPD |
| DATACLAS | LABEL | SUBSYS |
| | | VOLUME |

***Ignored Parameters:*** Because JES allocates sysout data sets, the UNIT and SPACE parameters are ignored, if coded on a sysout DD statement.

***Parameters on Procedure DD Statements that are Overridden:*** When an overriding DD statement contains a SYSOUT parameter, the system ignores a UNIT parameter on the overridden DD statement in the cataloged or in-stream procedure.

***Naming a Sysout Data Set:*** Code the DSNAME parameter with the SYSOUT parameter if you wish to assign the last qualifier of the system-generated name to a sysout data set.

***SYSOUT and DEST Subparameters:*** Do not code the SYSOUT writer-name subparameter when coding a DEST userid subparameter. These subparameters are mutually exclusive. You can code:

```
//VALID1  DD  SYSOUT=D,DEST=(node,userid)
//VALID2  DD  SYSOUT=(D,writer-name),DEST=(node)
```

***With DCB Subparameters:*** JES2 ignores DCB=PRTSP=2 on a DD statement that also contains a SYSOUT parameter.

For JES, it is not necessary to select a specific BLKSIZE on the DCB parameter for performance reasons because the subsystem selects its own blocking.

***INTRDR with OUTPUT Parameter:*** Do not code an OUTPUT parameter when the writer-name subparameter is INTRDR.

## Relationship to other control statements

A sysout DD statement can directly or indirectly reference an OUTPUT JCL statement. The parameters on the referenced OUTPUT JCL statement combine with the parameters on the sysout DD statement to control the processing of the sysout data set. See “OUTPUT parameter” on page 233 and Chapter 23, “OUTPUT JCL statement,” on page 469.

SYSOUT cannot specify a code-name subparameter in a job or job step that contains an OUTPUT JCL statement; in this case, JES2 treats the third subparameter as a form-name, instead of a reference to a JES2 /*OUTPUT statement.

***Backward references:*** Do not refer to an earlier DD statement that contains a SYSOUT parameter.

## Starting an external writer when requested

When a statement supplying processing options for a sysout data set specifies an external writer, the writer must be started before it can print or punch the data set. The writer is started by a system command from the operator or in the input stream. If the writer is not started before the job produces the sysout data set, the data set is retained until the writer is started.

## Held classes in a JES2 system

A sysout data set is held if the sysout DD statement contains HOLD=YES or the OUTPUT JCL statement specifies OUTDISP=HOLD.

## Held classes in a JES3 system

If CLASS specifies a class-name that is defined to JES3 as a held class for the output service hold queue (Q=HOLD), all of the new output characteristics might not be included in the data set on the writer queue when (1) the data set is moved from the hold queue to the output service writer queue (Q=WTR), (2) the data set includes an OUTPUT JCL statement, and (3) the NQ= or NCL= keyword is used.

For more information, see *z/OS JES3 Initialization and Tuning Guide*.

## Significance of output classes

To print this sysout data set and the messages from your job on the same output listing, code one of the following:

- The same output class in the DD SYSOUT parameter as in the JOB MSGCLASS parameter.
- DD SYSOUT=* to default to the JOB MSGCLASS output class.
- DD SYSOUT=(,) to default to one of the following:
  1. The CLASS parameter in an explicitly or implicitly referenced OUTPUT JCL statement. In this case, the OUTPUT JCL CLASS parameter should specify the same output class as the JOB MSGCLASS parameter.
  2. The JOB MSGCLASS output class, if no OUTPUT JCL statement is referenced or if the referenced OUTPUT JCL statement contains either CLASS= or CLASS=*.

## Examples of the SYSOUT parameter

***Example 1:***

```
//DD1    DD     SYSOUT=P
```

In this example, the DD statement specifies that JES is to write the sysout data set to the device handling class P output.

***Example 2:***

```
//DD2    DD     DSNAME=&&PAYOUT1,SYSOUT=P
```

In this example, DD statement DD2 defines PAYOUT1 as the last qualifier of the system-generated name for the sysout data set. The system generates a name such as userid.jobname.jobid.Ddsnumber.PAYOUT1. The DD statement specifies that JES is to write the data set to the device handling class P output.

***Example 3:***

```
//JOB50  JOB    ,'C. BROWN',MSGCLASS=C
//STEP1  EXEC   PGM=SET
//DDX    DD     SYSOUT=C
```

In this example, DD statement DDX specifies that JES is to write the sysout data set to the device handling class C output. Because the SYSOUT parameter and the MSGCLASS parameter specify the same class, the messages from this job and the sysout data set can be written to the same device.

***Example 4:***

```
//STEP1  EXEC   PGM=ANS
//OT1    OUTPUT DEST=NYC
//OT2    OUTPUT DEST=LAX
//OT3    OUTPUT COPIES=5
//DSA    DD     SYSOUT=H,OUTPUT=(*.OT2,*.OT1,*.OT3)
```

In this example, the DD statement combines with the three referenced OUTPUT JCL statements to create three separate sets of output:

1. DSA combines with OT1 to send the sysout data set to NYC.
2. DSA combines with OT2 to send the sysout data set to LAX.
3. DSA combines with OT3 to print five copies of the data set locally on the printer used for output class H.

Note that the output references can be in any order.

***Example 5:***

```
//DD5    DD     SYSOUT=(F,,2PRT)
```

In this example, the DD statement specifies that JES is to write the sysout data set to the device handling class F output. The data set is to be printed or punched on forms named 2PRT.

## TERM parameter

***Parameter type***

Keyword, optional

Do not use the TERM parameter for an SMS-managed data set (one with an assigned storage class).

***Purpose***

Use the TERM parameter to indicate to the system that a data set is coming from or going to a terminal for a TSO/E user.

***Considerations for an APPC scheduling environment***

The TERM parameter has no function in an APPC scheduling environment. If you code TERM, the system will check it for syntax and ignore it.

## Syntax

```
TERM=TS
```

## Subparameter definition

**TS**

In a **foreground job** submitted by a TSO/E user, indicates that the input or output data set is coming from or going to a TSO/E userid.

In a **background or batch job**, the system either:

- Ignores the TERM=TS parameter, when it appears with other parameters.
- Fails the TERM=TS parameter with an allocation error, when the parameter appears by itself. (The system bypasses this error if SYSOUT=* is coded with TERM=TS.)

## Relationship to other parameters

Do not code the following DD parameters with the TERM parameter.

| | |
|---|---|
| * | DYNAM |
| AMP | PROTECT |
| DATA | QNAME |
| DDNAME | |

Code only the DCB and SYSOUT parameters with the TERM parameter. The system ignores any other DD parameters.

## Location in the JCL

To ensure that the system uses the desired OUTPUT JCL statement, code all referenced OUTPUT JCL statements in the input stream before the DD statement that refers to them. For example, if the referencing DD statement appears in an in-stream or cataloged procedure, the referenced OUTPUT JCL statement should precede the DD statement in the procedure.

In a foreground TSO/E job, a DD statement containing TERM=TS and a SYSOUT parameter begins an in-stream data set.

When concatenating DD statements, the DD statement that contains TERM=TS must be the last DD statement in a job step.

## Examples of the TERM parameter

***Example 1***

```
//DD1  DD  TERM=TS
```

In a foreground job submitted from a TSO/E userid, this DD statement defines a data set coming from or going to the TSO/E userid.

***Example 2***

```
//DD1  DD  TERM=TS,SYSOUT=*
```

In a background or batch job, the system ignores TERM=TS and recognizes a sysout data set. (An allocation error occurs if SYSOUT=* is not coded with TERM=TS.)

***Example 3***

```
//DD3  DD  UNIT=3400-5,DISP=(MOD,PASS),TERM=TS,LABEL=(,NL),
//         DCB=(LRECL=80,BLKSIZE=80)
```

In a foreground job, the system ignores all of the parameters in this example except TERM and DCB. In a batch job, the system ignores only the TERM parameter.

## UCS parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the UCS (universal character set) parameter to identify:

- The UCS image JES is to use in printing this sysout data set.
- A print train (print chain or print band) JES is to use in printing this sysout data set on an impact printer.
- A font for this sysout data set printed on an AFP printer in a JES2 system. In this use, the UCS parameter acts like a CHARS parameter.

The UCS image specifies the special character set to be used. JES loads the image into the printer's buffer. The UCS image is stored in SYS1.IMAGELIB. IBM provides the special character set codes in Table 18 on page 291.

***References***

For more information on the UCS parameter, see *z/OS DFSMSdfp Advanced Services*.

### Syntax

```
UCS=  {character-set-code                     }
      {(character-set-code [,FOLD] [,VERIFY]) }
                           [,    ]
```

- You can omit the parentheses if you code only a character-set-code.
- All of the subparameters are positional. If you omit FOLD but code VERIFY, code a comma to indicate the omission. For example, UCS=(AN,,VERIFY).
- Null positions in the UCS parameter are invalid.

### Subparameter definition

**character-set-code**
Identifies a universal character set. The character-set-code is 1 through 4 alphanumeric or national ($, #, @) characters. See Table 18 on page 291 for IBM standard special character set codes.

**FOLD**
Requests that the chain or train for the universal character set be loaded in fold mode. Fold mode is described in *2821 Component Description*. Fold mode is most often used when upper- and lower-case data is to be printed only in uppercase.

**Note:** JES2 and JES3 do not support the FOLD subparameter. For JES2, the FOLD option is specified in the UCS image for JES2-controlled printers. See *z/OS DFSMSdfp Advanced Services*.

**VERIFY**
Requests that, before the data set is printed, the operator verify visually that the character set image is for the correct chain or train. The character set image is displayed on the printer before the data set is printed.

*Table 18. Special Character Sets for the 1403, 3203 Model 5, and 3211 Printers*

| 1403 | 3203 Model 5 | 3211 | Characteristics |
|---|---|---|---|
| AN | AN | A11 | Arrangement A, standard EBCDIC character set, 48 characters |
| HN | HN | H11 | Arrangement H, EBCDIC character set for FORTRAN and COBOL, 48 characters |
| | | G11 | ASCII character set |
| PCAN | PCAN | | Preferred alphanumeric character set, arrangement A |
| PCHN | PCHN | | Preferred alphanumeric character set, arrangement H |
| PN | PN | P11 | PL/I alphanumeric character set |
| QN | QN | | PL/I preferred alphanumeric character set for scientific applications |
| QNC | QNC | | PL/1 preferred alphanumeric character set for commercial applications |
| RN | RN | | Preferred character set for commercial applications of FORTRAN and COBOL |
| SN | SN | | Preferred character set for text printing |
| TN | TN | T11 | Character set for text printing, 120 characters |
| XN | | | High-speed alphanumeric character set for 1403, Model 2 |
| YN | | | High-speed preferred alphanumeric character set for 1403, Model N1 |
| **Note:** Where three values exist (for the 1403, 3211, and 3203 Model 5 printers), code any one of them. JES selects the set corresponding to the device on which the data set is printed. Not all of these character sets may be available at your installation. Also, an installation can design character sets to meet special needs and assign a unique code to them. Follow installation procedures for using character sets. | | | |

## Defaults

If you do not code the UCS parameter, the system checks the UCS image in the printer's buffer; if it is a default image, as indicated by its first byte, JES uses it. If it is not a default image, JES loads the UCS image that is the installation default specified at JES initialization.

On an impact printer, if the chain or train does not contain a valid character set, JES asks the operator to specify a character set and to mount the corresponding chain or train.

## Overrides

For printing on a printer with the UCS feature, the UCS parameter on a sysout DD statement overrides an OUTPUT JCL UCS parameter. For printing on a 3800 Model 1, a CHARS parameter on the sysout DD statement or the OUTPUT JCL statement overrides all UCS parameters.

For a data set scheduled to the Print Services Facility (PSF), the PSF uses the following parameters, in override order, to select the font list:

1. Font list in the library member specified by an OUTPUT JCL PAGEDEF parameter.
2. DD CHARS parameter.
3. OUTPUT JCL CHARS parameter.
4. DD UCS parameter.
5. OUTPUT JCL UCS parameter.
6. JES installation default for the device.
7. Font list on the PAGEDEF parameter in the PSF cataloged procedure.

See "PAGEDEF parameter" on page 539 for more information.

## Relationship to other parameters

Do not code the following DD parameters with the UCS parameter.

| | |
|---|---|
| * | DYNAM |
| AMP | KEYOFF |
| DATA | PROTECT |
| DDNAME | QNAME |

Do not code the UCS parameter with the DCB subparameters CYLOFL, INTVL, RESERVE, and RKP.

The FOLD and VERIFY subparameters are meaningful only when you specify a printing device directly on a DD statement, for example, UNIT=00E, thus bypassing JES sysout processing.

## Using special character sets

To use a special character set, SYS1.IMAGELIB must contain an image of the character set, and the chain or train for the character set must be available. IBM provides standard special character sets, and the installation may provide user-designed special character sets.

## Examples of the UCS parameter

***Example 1***

```
//DD1 DD UNIT=1403,UCS=(YN,,VERIFY)
```

In this example, the DD statement requests a 1403 Printer. The UCS parameter requests the chain or train for special character set code YN. Because VERIFY is coded, the system will display the character set image on the printer before the data set is printed.

***Example 2***

```
//DD2  DD  SYSOUT=G,UCS=PN
```

In this example, the DD statement requests the device for output class G. If the device is a printer with the UCS feature, the system loads the UCS image for code PN. If the device is an impact printer, the system asks the operator to mount the chain or train for PN, if it is not already mounted. If the device is a 3800, the system uses the UCS subparameter to select the character-arrangement table. Otherwise, the system ignores the UCS parameter.

## UNIT parameter

***Parameter Type:*** Keyword, optional

**Note:** With SMS, you do not need to use the UNIT parameter to specify a device for an SMS-managed data set. Use the STORCLAS parameter (described in ) or let an installation-written automatic class selection (ACS) routine select a storage class for the data set.

Also with SMS, for a non-SMS-managed data set, if your storage administrator has set a system default unit under SMS, you do not need to specify UNIT. Check with your storage administrator.

***Purpose:*** Use the UNIT parameter to ask the system to place the data set on:

- A specific device.
- A certain type or group of devices.
- The same device as another data set.

The UNIT parameter can also tell the system how many devices to assign and request that the system defer mounting the volume until the data set is opened.

### Syntax

```
{UNIT=([ddd          ] [,unit-count] [,DEFER][,SMSHONOR]) }
       [/ddd         ] [,P         ] [,     ]
       [/dddd        ] [,          ]
       [device-type
       [group-name

{UNIT=AFF=ddname                                 }
```

- You can omit the parentheses if you code only the first subparameter.
- All of the subparameters are positional. If you omit unit-count or P but code DEFER, code a comma to indicate the omission; one device is assigned to the data set. For example, UNIT=(3490,,DEFER,).
- Device-type is mutually exclusive with SMSHONOR.

### Subparameter definition

**device-number**

Identifies a specific device by a 3-digit or 4-digit hexadecimal number. Precede a 4-digit number with a slash (/). A 3-digit number can be specified with or without a slash.

**Attention**: Specify a device number only when necessary. When you specify a device number, the system can assign only that specific device. If the device is already being used, the job must be delayed or canceled. If the device number is changed or logically moved, the allocation request will fail.

However, for a permanently mounted direct access device, such as a 3390 Direct Access Storage, specifying a device type (UNIT=3390) and a volume serial number in the VOLUME=SER parameter has the same result as specifying a device number in the UNIT parameter.

In a JES3 system, if any DD UNIT parameter in a job specifies a device-number for a device that is JES3-managed or jointly JES3/MVS managed, either the JOB statement must contain a SYSTEM or SYSAFF parameter, or the JES3 //*MAIN statement must contain a SYSTEM parameter.

For an SMS-managed tape library request where SMSHONOR is not specified, or an SMS-managed DASD request, the device number is ignored. For an SMS-managed tape library request where SMSHONOR is specified, the system attempts to allocate the specified device, provided it is in the storage group(s) assigned to the request by SMS.

**device-type**
Requests a device by its generic name, which is an IBM-supplied name that identifies a device by its machine type and model. For example, UNIT=3390.

When a device-type name contains a hyphen, do not enclose it in apostrophes, for example, UNIT=3400-5.

Obtain the list of device types you can specify from your installation.

If you specify the device-type subparameter, SMS ignores it.

For a 3480 Magnetic Tape Subsystem in compatibility mode, code UNIT=3400-9 or a group-name.

**group-name**
Requests a group of devices by a symbolic name. The installation must have assigned the name to the device(s) during system initialization or IBM must have assigned the name. The group-name is 1 through 8 alphanumeric characters.

For an SMS-managed tape library request where SMSHONOR is not specified, or an SMS-managed DASD request, the group-name is ignored. For an SMS-managed tape library request, if SMSHONOR is specified along with the group-name, the system attempts to allocate to the subset of devices in the group-name, if they are also selected by SMS.

**Group names**: A group-name can identify a single device or a group of devices. A group can consist of devices of the same or different types. For example, a group can contain two or more different types of direct access storage devices (DASD) or two or more different types of tape devices, or even a mixture of both direct access and tape devices. Note, however, that IBM does not recommend that a group contain both direct access storage devices (DASD) and tape devices.

**Note:** A group name is called an esoteric name in Hardware Configuration Definition (HCD) terminology.

**Allocation from groups**: The system assigns a device from the group. If a group consists of only one device, the system assigns that device. If the group consists of more than one device type, the units requested are allocated from the same device type. For example, if GPDA contains 3380 Disk Storage and 3390 Direct Access Storage devices, a request for two units would be allocated to two 3380s or to two 3390s.

**Extending data set**: If a data set that was created using the group-name subparameter is to be extended, the system allocates additional devices of the same type as the original devices. However, the additional devices might not necessarily be from the same group.

**SYSALLDA**: IBM assigned group-names include SYSALLDA, which contains all direct access devices defined to the system.

**SYS3480R and SYS348XR**: SYS3480R and SYS348XR are IBM-assigned group names. SYS3480R contains 3480, 3480X, and 3490 Magnetic Tape Subsystems. SYS348XR contains 3480X and 3490 Magnetic Tape Subsystems.

Use these group names to override the device type eligibility retrieved by the system when referencing existing 3480- or 3480 XF-formatted data sets. Specifically, use SYS3480R when you want to read 3480-formatted data sets and use SYS348XR when you want to read 3480 XF-formatted data sets.

**Note:** LABEL=(n,,,IN) is the system-managed tape library equivalent of either UNIT=SYS3480R or UNIT=SYS348XR.

**unit-count**
Specifies the number of devices for the data set. "Unit-count" is a decimal number from 1 through 59.

**Number of devices allocated**: The system uses the unit-count to determine how many devices to allocate. For tapes, the system uses the unit-count subparameter to allocate the specified number of system-managed or non-system-managed units. If you also specify P (for parallel mount) in the UNIT parameter, and for SMS-managed DASD, the system uses the **highest** of the following numbers to determine how many devices and volumes to allocate:

- Unit-count specified in the UNIT parameter
- Volume-count specified in the VOLUME parameter
- Number of volume serial numbers implicitly or explicitly specified

You may receive more devices than the unit-count requests if you specify VOLUME=REF or a permanently resident or reserved volume. And, if two DD statements in a step request the same volume and either DD statement requests any other volume(s), the system assigns an additional device.

**Unit count for received or VOLUME=REF data sets**: The system assigns one device when the DD statement receives a passed data set or refers in a VOLUME=REF subparameter to a cataloged data set or earlier DD statement for volume and unit information. Code a unit-count subparameter if the data set needs more than one device.

**Unit count when device number specified**: When the first subparameter requests a specific device, the unit count must be 1 or omitted. Only when the device is a communication device can the unit count be higher than 1.

**Unit count when SMSHONOR specified**: When SMSHONOR is specified, only the subset of devices that are selected by SMS and are within the specified group-name are eligible. When a unit count is specified, the requested number of devices are selected from the subset of devices. If the subset does not contain enough devices to satisfy the requested unit count, the request fails.

**P** Asks the system to allocate the same number of devices as requested in the VOLUME volume-count or SER subparameter, whichever is higher. Thus, all volumes for the data set are mounted in parallel.

If you specify the P subparameter for system-managed DASD, the system ignores it. If you specify the P subparameter for system-managed tape libraries, the system honors it.

**DEFER**
Asks the system to assign the data set to device(s) but requests that the volume(s) not be mounted until the data set is opened. To defer mounting, DEFER must be specified or implied for all DD statements that reference the volume.

If you specify the DEFER subparameter for system-managed DASD, the system ignores it. If you specify the DEFER subparameter for system-managed tape libraries, the system honors it.

**DEFER when data set is never opened:** If you request deferred mounting of a volume and the data set on that volume is never opened by the processing program, the volume is never mounted during the job step.

**Restrictions on DEFER**: Do not code DEFER:

- For a new data set on direct access. The system ignores DEFER.
- On a SYSCKEOV DD statement.

**SMSHONOR**
Asks the system to honor the device number or group-name and allocate to the device number or group-name specified in the case of an SMS tape library request.

Use only device number or group-name (user-defined esoteric) when you use SMSHONOR. The following subset of devices must be consistent with libraries assigned to the storage class specified on the request or selected by ACS routines of the installation:

1. Subset of devices that are in the specified group name.
2. Subset of devices that are with the specified device number, which are selected by the storage class assigned to the request.

**Note:** If the device number or group-name does not intersect with the storage class, the request fails.

If you specify the SMSHONOR subparameter on a non-SMS system, the system ignores the keyword.

If you specify a group name in a JES3 environment, all devices in the group must be of the same device type and from the same tape library. There are no such restrictions in a non-JES3 environment.

**AFF=ddname**
Requests that the system allocate different data sets residing on different, removable volumes to the same device during execution of the step. This request is called **unit affinity**, where "ddname" is the ddname of an earlier DD statement in the same step. Use unit affinity to reduce the number of devices that are used in a job step; request that an existing data set be assigned to the same device(s) as another existing data set.

If you specify the UNIT=AFF subparameter for system-managed DASD, the system ignores it. If you specify the UNIT=AFF subparameter for system-managed tape libraries, the system attempts to honor it. For system-managed tape libraries with OA44357 installed, SMS construct and job-related information is associated with the DD rather than a device. This allows for multiple DDs with different SMS construct information to be allocated to the same device using AFF=ddname.

Under certain conditions the system ignores unit affinity. See *z/OS MVS JCL User's Guide* for more information.

In a JES3 environment, UNIT=AFF=ddname may not be honored. See *z/OS MVS JCL User's Guide* and *z/OS HCD Planning* for information about device eligibility and unit affinity.

**Restrictions on UNIT=AFF**: Do not code UNIT=AFF=ddname:

- With DISP=NEW if the data set referenced in the AFF subparameter resides on a direct access device. This restriction applies only to non-SMS-managed DASD. If coded, the system terminates the job. If the referenced data set can be allocated to either tape or DASD, the system allocates both requests to tape devices.
- On a DD * or DD DATA statement or on a DD statement containing a SUBSYS parameter. The system ignores the UNIT=AFF and defaults the device to SYSALLDA.
- When the DD statement referenced in the AFF subparameter contains FREE=CLOSE.
- With the STORCLAS parameter.
- With an affinity specification to an earlier DD statement that requests SYS3480R or SYS348XR on the group-name subparameter, unless volume affinity also exists. Volume affinity exists when two DD statements both reference a data set on the same volume. Do not also specify DISP=OLD or DISP=MOD; attempting to write 3480 data to a 3490 drive, or 3490 data to a 3480 drive, fails during OPEN processing with ABEND 813-04 accompanied by message IEC149I.

## Overrides

If you code SYSOUT and UNIT on the same statement, the SYSOUT parameter overrides the UNIT parameter.

The system also obtains device information when the system obtains volume serial information from:

- A VOLUME=REF=dsname reference to an earlier data set.
- A VOLUME=REF=ddname reference to an earlier DD statement.
- The volume(s) for a passed data set.
- The catalog for a cataloged data set.

However, you can override the retrieved device information if the device you specify is a subset of the retrieved device information; otherwise the system ignores the overriding device information. For example, if the retrieved unit grouping is 3390, and the specified unit subparameter is 3390A (a subset of 3390), then the system allocates from the devices contained in 3390A.

If you have 3490 Magnetic Tape Subsystem models A10 and A20 defined to your system and you use one of the IBM-generated group names SYS3480R or SYS348XR, the system overrides the device type retrieved from the catalog with a device from the esoteric device group.

For more about how the system uses device information it retrieves from the catalog, see the text about the relationship of the UNIT and VOLUME parameters for non-SMS-managed data sets in *z/OS MVS JCL User's Guide*.

**Note:** LABEL=(n,,,IN) is the system-managed tape library equivalent of either UNIT=SYS3480R or UNIT=SYS348XR.

You can mount 3480-formatted or 3480X-formatted (18-track formatted) tape volumes, that are not extended, on a 3490 tape device (36-track write, 18-track or 36-track read).

## Relationship to other parameters

Do not code the following DD parameters with the UNIT parameter.

| | |
|---|---|
| * | DYNAM |
| DATA | QNAME |
| DDNAME | |

Do not code the UNIT DEFER subparameter on a SYSCKEOV DD statement.

To allocate a device, such as a printer or telecommunications device, that does not involve a data set, do not code the DISP parameter.

See also "Restrictions on UNIT=AFF."

## Relationship to other control statements

When SMSHONOR is specified, a device number or a group name must be specified.

## Location in the JCL

When a DD statement contains a UNIT=AFF=ddname parameter, the DD statement referenced in the AFF subparameter must be defined earlier in the job step; otherwise, the system treats the DD statement containing UNIT=AFF as a DD DUMMY statement.

The following example illustrates a case where the system treats the DD statement containing the UNIT=AFF as a DD DUMMY statement:

```
//STEP  EXEC  PGM=TKM
//DD1   DD    DDNAME=DD5
//DD2   DD    DSNAME=A,DISP=OLD
//DD3   DD    DSNAME=C,DISP=SHR,UNIT=AFF=DD1
//DD5   DD    DSNAME=B,DISP=SHR
```

DD3 requests unit affinity to DD1. Although DD1 occurs earlier in the job step than DD3, it refers to DD5 that is located after DD3. Because DD1 is not completely defined, the system treats DD3 as a dummy statement.

## Examples of the UNIT parameter

***Example 1:***

```
//STEP2  EXEC PGM=POINT
//DDX  DD  DSNAME=EST,DISP=MOD,VOLUME=SER=(42569,42570),
//     UNIT=(3480,2)
//DDY  DD  DSNAME=ERAS,DISP=OLD,UNIT=3480
//DDZ  DD  DSNAME=RECK,DISP=OLD,
//     VOLUME=SER=(40653,13262),UNIT=AFF=DDX
```

DD statement DDX requests two 3480 tape devices, DD statement DDZ requests the same two devices as DDX. Note that the operator will have to change volumes on the two 3480 devices during execution of the job step.

DD statement DDY requests one 3480 tape device.

*Example 2:*

```
//DD1  DD  DSNAME=AAG3,DISP=(,KEEP),
//       VOLUME=SER=13230,UNIT=3400-5
```

This DD statement defines a new data set and requests that the system assign any 3420 Magnetic Tape Unit that can operate in 6250 BPI NRZI nine-track format.

*Example 3:*

```
//DD2  DD  DSNAME=X.Y.Z,DISP=OLD,UNIT=(,2)
```

This DD statement defines a cataloged data set and requests that the system assign two devices to the data set. The system obtains the device type from the catalog.

*Example 4:*

```
//DD3  DD  DSNAME=COLLECT,DISP=OLD,
//       VOLUME=SER=1095,UNIT=(3490,,DEFER)
```

This DD statement defines an existing data set that resides on a tape volume and requests that the system assign a 3490 tape device. Because DEFER is coded, the volume will not be mounted until the data set is opened.

*Example 5:*

```
//STEPA  DD  DSNAME=FALL,DISP=OLD,UNIT=237
```

For this data set, the system retrieves the volume and device type from the catalog. The UNIT parameter, by specifying device 237, overrides the catalog device type; however, device 237 must be the same type as the device stated in the catalog.

***Example 6:*** This example shows the use of the ALLOCxx UNITAFF default.

This example assumes the following environment:

- UNITAFF(3490) was specified in parmlib member ALLOC05, defining a 3490 as the default unit-affinity-ignored unit name. This default is used when unit affinity is ignored, the referenced DD is an SMS-managed request and the referencing DD is a NEW non-SMS-managed request, and the system is unable to obtain a unit from the primary DD in the unit affinity chain.
- The SMS ACS routines are defined so that:
  - Data set L is to be redirected from tape to an SMS-managed DASD volume, SD3.
  - Data set M is not to be redirected and is, therefore, still intended to go to a non-SMS managed tape volume.

```
//JOB2  JOB  ......
//STEP1 EXEC ......
//DD11  DD   DSN=L,DISP=(NEW),UNIT=3480,.....
//STEP2 EXEC ......
//DD21  DD   DSN=L,DISP=OLD,......
//DD22  DD   DSN=M,DISP=(NEW,CATLG),UNIT=AFF=DD21
```

In STEP1, DD11, data set L is created and cataloged on SD3, SMS-managed DASD (redirected using SMS ACS routines).

In STEP2, DD21, data set L is an existing data set and is cataloged on SD3, SMS-managed DASD. DD21 is both the referenced DD (referenced by the UNIT=AFF on DD22) and the primary DD.

In STEP2, DD22 is the referencing DD, which requests unit affinity to DD21. Because data set L is on SMS-managed DASD, the system cannot honor the unit affinity for DD22 which is intended to go to tape. With the unit affinity ignored, the system must determine a unit to be used for DD22.

The system is not able to rely on the unit information in the catalog for data set L, because the catalog reflects a DASD unit (as a result of being redirected). Because data set L was created in a prior step and there is no unit specified on DD21, the system is not able to use the JCL for DD21 as a source of unit information. The system will, therefore, use the unit-affinity-ignored unit name of 3490 for DD22.

***Example 7:***

```
//JOB7 JOB ......
//STEP1 EXEC ......
//DD01 DD DSN=A,DISP=SHR,UNIT=(/B5B8,,,SMSHONOR)
//DD02 DD DSN=B,DISP=SHR,UNIT=(MYTAPE,,DEFER,SMSHONOR)
```

To allocate to this Tape Library Request, the DD statement UNIT=(/B5B8,,,SMSHONOR) requires to use the device number B5B8, which is in the list of devices selected by SMS .

For this Tape Library Request, the DD statement UNIT=(MYTAPE,,DEFER,SMSHONOR) requires the following devices:

- the devices that are in the esoteric MYTAPE.
- the devices that are in the list selected by SMS.

## VOLUME parameter

***Parameter type:*** Keyword, optional

***Terminology:*** Data sets on system-managed tape volumes exhibit both SMS and non-SMS characteristics. When necessary, **data sets on a system-managed tape volume** are distinguished from **system-managed DASD data sets**. Otherwise, the term **system-managed data sets** refers to both data sets on a system-managed tape volume and system-managed DASD data sets.

To cause multiple data sets to be stacked on the same volume, see *z/OS MVS JCL User's Guide* for recommendations and examples.

With SMS, consider the following:

- All volumes in a multi-volume data set should reside in the same system-managed tape library and must belong to the same tape storage group. If all of the volumes do not reside in the same tape library, the installation can enter the volumes through the DFSMS installation exit, CBRUXVNL.
- You cannot make a specific volume reference to a scratch volume.
- You do not need to use the VOLUME parameter to specify volumes for new data sets. See the DATACLAS parameter described in and the STORCLAS parameter described in .
- You cannot override the volume count for an existing system-managed DASD data set (but you can specify a volume count when you create a new system-managed DASD data set).
- If the storage administrator has specified a system default unit name and you do not code a UNIT name for non-system-managed data sets, then the system uses the volumes associated with the default unit name. In this case, you do not need

to code the VOLUME parameter. Check with your storage administrator to determine whether a default unit name has been specified.

***Purpose:*** Use the VOLUME parameter to identify the volume or volumes on which a data set resides or will reside. You can request:

- A private volume
- Retention of the volume
- A specific volume by serial number
- The same volume that another data set uses

You can also specify which volume of a multivolume data set is to be processed first and, for an output data set, the number of volumes required.

A **nonspecific volume request** is a DD statement for a new data set that can be assigned to any volume or volumes. To make a nonspecific volume request for a new data set, either:

- Omit the VOLUME parameter.
- Code a VOLUME parameter but omit a SER or REF subparameter.

## Syntax

```
{VOLUME} = ([PRIVATE] [,RETAIN] [,volume-sequence-number] [,volume-count]
{VOL   }              [,      ] [,                       ]

              [SER=serial-number                     ]
              [SER=(serial-number[,serial-number]...)]
          [,] [REF=dsname                            ]
              [REF=*.ddname                          ]
              [REF=*.stepname.ddname                 ]
              [REF=*.stepname.procstepname.ddname    ]
```

**Single subparameter**: You can omit the parentheses if you code only PRIVATE or only a keyword subparameter. For example, VOLUME=PRIVATE or VOLUME=SER=222001 or VOLUME=REF=DS1.

**Null REF subparameter**: The REF subparameter of the VOLUME parameter can have a null value only when coded on a DD that either overrides a DD in a procedure or is added to a procedure.

**Null positional subparameters**: Null positions in the VOLUME=SER parameter are invalid.

**Positional subparameters**: The first four subparameters are positional. The last subparameter, SER or REF, is a keyword subparameter and must follow all positional subparameters. Code a comma to indicate an omitted positional subparameter as follows:

- If you omit PRIVATE and code RETAIN, code a comma before RETAIN. For example, VOLUME=(,RETAIN,2,3,SER=(222001,222002,222003)).
- Code a comma when RETAIN is omitted and the volume sequence number and volume count subparameters follow. For example, VOLUME=(PRIVATE,,2,3,SER=(222001,222002,222003)), and if PRIVATE is also omitted, VOLUME=(,,2,3,SER=(222001,222002,222003)).
- Code a comma when the volume sequence number is omitted and the volume count subparameter follows. For example, VOLUME=(,RETAIN,,3,SER=(222001,222002,222003)), and VOLUME=(PRIVATE,,,3,SER=(222001,222002,222003)), and VOLUME=(,,,3,SER=(222001,222002,222003)).
- Code a comma when the volume count is omitted, at least one other subparameter precedes it, and a keyword subparameter follows. For example, VOLUME=(,RETAIN,2,,SER=(222001,222002,222003)), and VOLUME=(,,2,,SER=(222001,222002,222003)), and VOLUME=(,RETAIN,REF=*.stepname.ddname)

**Single SER subparameter**: You can omit the parentheses in the SER subparameter if you code only one serial number. For example, VOLUME=SER=222001.

**Special characters**: When a serial number in the SER subparameter contains special characters, other than hyphens, enclose it in apostrophes. For example, VOLUME=SER=(222001,222-02,'222/03').

When the dsname in the REF subparameter contains special characters, other than the periods used in a qualified name, enclose it in apostrophes. For example, VOLUME=REF='DS/284'.

Code each apostrophe that is part of the serial number or data set name as two consecutive apostrophes. For example, VOLUME=SER='O''HARE' or VOLUME=REF='DS''371'.

## Subparameter definition

**PRIVATE**

Requests a private volume. Private means that:

- The system is not to allocate an output data set to the volume unless the volume is specifically requested, such as in a VOLUME=SER subparameter.
- If tape, the volume is to be demounted after the data set is closed, unless RETAIN is also coded or the DD DISP parameter specifies PASS.
- If a demountable direct access volume, the volume is to be demounted after the data set is closed.

**RETAIN**

For a private tape volume, RETAIN requests that this volume is not to be demounted or rewound after the data set is closed or at the end of the step. For a public tape volume, RETAIN requests that this volume is to be retained at the device if it is demounted during the job.

RETAIN has no effect on the handling of direct access volumes.

For JES3-managed tape devices, JES3 does not honor the RETAIN parameter after reaching the end of the job. However, if RETAIN is coded and the tape

volume is to be shared with a later step, JES3 designates the volume as retained. JES3 also ignores the RETAIN parameter when issuing its KEEP/RETAIN messages.

**volume-sequence-number**
Identifies the volume of an existing multivolume data set to be used to begin processing the data set. The volume sequence number is a decimal number from 1 through 255; the first volume is identified as 1. The volume sequence number must be less than or equal to the number of volumes on which the data set exists; otherwise, the job fails.

If the volume sequence number is not specified the system will process the first volume.

For new data sets, the system ignores the volume sequence number.

**volume-count**
Specifies the maximum number of volumes that an **output** data set requires. The volume count is a decimal number from 1 through 255. The total volume count for all DD statements in one job step cannot exceed 4095.

**Note:** The system uses the unit count to determine how many devices to allocate. However, if you also specify P (for parallel mount) in the UNIT parameter, the system might use the value specified for the volume count to determine how many devices and volumes to allocate. See the unit-count description in "Subparameter definition" on page 293.

You can use the volume count to override the maximum volume count in a data class:

- The maximum volume count for an SMS-managed mountable tape data set or a non-managed tape data set is 255.
- The maximum volume count for a non-VSAM or non-SMS-managed data set is 255.
- The maximum volume count for a VSAM or SMS-managed data set is 59.
- SMS has an upper limit of 59 volsers being specified in the JCL. If more than 59 volsers are specified in the JCL, SMS processes the allocation as non-SMS managed.

**Volume count and serial numbers**: When the volume count is greater than:

- The number of volume serials coded in the SER subparameter
- The number of volume serials the system retrieved from the catalog
- The number of volume serials the system retrieved from VOL=REF
- The number of volume serials the system retrieved from a passed data set,

the system assigns other volumes to the remaining devices. If the volume count is smaller than the number of volume serials, the system ignores the volume count.

If a data set may need more volumes than the number of volume serials coded, specify a volume count equal to the total number of volumes that might be used. Requesting more volumes in the volume count will make sure that the data set can be written on more volumes if it exceeds the requested volumes.

If you do not code a volume count and volume serial number, the system can extend an existing cataloged data set that resides on a removable volume up to 20 volumes.

**Volume count for nonspecific requests**: If the request is for a nonspecific, public volume on a direct access device, the system ignores the volume count and allocates the number of volumes in the UNIT count subparameter.

If the request is for a nonspecific, private volume, the system treats it like a specific request if the volume count is more than one and allocates the number of volumes in the volume count.

**Volume count for system-managed DASD data sets**: You cannot specify a volume count for an existing system-managed DASD data set. (If you do, the system will ignore it.) When you create a new system-managed DASD data set, the volume count defined in the data class may be overridden by using the volume-count subparameter. However, if the volume-count subparameter specifies a value greater then 59, the system will set the volume count to a maximum of 59. The maximum volume count for a VSAM or System-Managed DASD data set is 59.

**Volume count for system-managed tape data sets**: If you specify a volume count and DISP=PASS on a DD statement, the system will pass the volume count to subsequent receiving steps within the job. This may cause the system to allocate more devices than expected to the receiving DD. Coding UNIT=AFF in the receiving step's DD will result in the optimum number of devices being allocated to the receiving DD. For more information about the number of devices allocated, refer to the *z/OS MVS JCL User's Guide*. When you create a new system-managed TAPE data set, the volume count defined in the data class may be overridden by using the volume-count subparameter. If you do not provide a volume count parameter than the volume count specified in the data class would be used.

**SER=serial-number**
**SER=(serial-number[,serial-number]...)**

Identifies by serial number the volume(s) on which the data set resides or will reside. A volume serial number is 1 through 6 alphanumeric, national ($, #, @), or special characters; enclose a serial number that contains special characters, other than hyphens, in apostrophes. If the number is shorter than 6 characters, it is padded with trailing blanks.

You can code a maximum of 255 volume serial numbers on a DD statement. The maximum number of volume serial numbers for a VSAM or SMS-managed data set is 59.

Do not specify duplicate volume serial numbers in a SER subparameter. Each volume must have a unique volume serial number, regardless of whether it is a tape or disk volume.

Do not code a volume serial number as SCRTCH, PRIVAT, or Lnnnnn (L with five numbers); these are used in messages to ask the operator to mount a volume. SCRTCH is used when the dataset being created on the non-specific volume is temporary [DISP=(NEW,DELETE) or DSN=&&tempname]. PRIVAT is used for all other cases of non-specific volumes. Lnnnnn is used as the "volser" of an NL (unlabeled) tape.

Do not code a volume serial number as MIGRAT, which is used by the Hierarchical Storage Manager DFSMShsm for migrated data sets. When using some typewriter heads or printer chains, a volume serial number may be unrecognizable if you code certain special characters.

For a permanently mounted direct access device, such as a 3390 Direct Access Storage, specifying a volume serial number and UNIT=3390 has the same result as specifying a device number in the UNIT parameter. **For new**

**SMS-managed data sets:** For an SMS-managed data set, code the SER subparameter only if the storage administrator has specified GUARANTEED_SPACE=YES in the storage class of the data set. In this case, SMS uses the volumes you explicitly specify. If it is unable to do so, the allocation fails. The volume serial numbers must be assigned to the same storage group. If GUARANTEED_SPACE=YES is not in effect, SMS ignores any volume serial numbers you specify for new SMS-managed data sets.

**For SMS-managed library tape volumes:** For SMS-managed Library Tape volume, the Guaranteed Space storage class attribute is ignored and the system will allocate to the specified volume.

**For existing data sets:**

- **If you do not specify a volume serial number and you specify an SMS-managed or cataloged data set**: the system will allocate the data set to the volume on which it resides.
- **If you specify a non-SMS-managed volume serial number**: the system will allocate the data set on the volume specified, regardless of whether there is a cataloged or SMS-managed data set of the same name elsewhere. If there is no data set with the specified name on the volume specified, the allocation request will complete but a later request to OPEN the DD will fail. The request will fail.
- **If you specify an SMS-managed volume serial number:** the system will find and allocate the data set to the volume on which it resides, even if that is different from the volume specified. If there is no SMS-managed data set with the specified name, the allocation request will fail.
- **When multiple DD statements in the same step for the same SMS-managed DASD data set are specified:** if DISP=MOD is specified, or the OPEN or OPENJ macro is issued with the EXTEND or OUTINX option, a data integrity exposure occurs when the data set is extended on additional volume(s). This new volume information is not available to the other DD statements in the job step for the same data set. The data on the new volume(s) will be overlayed if the data set is opened for output processing using one of the other DD statements in the same job step and the data set is again extended.

  **Recommendation:** Have only one DD statement per step for a data set that may need to extend to a new volume.
- **When two data sets, one that is SMS-managed and one that is not, share the same data set name:**
  - If you specify the non-SMS-managed volume, the system will allocate the non-SMS-managed data set.
  - If you do not specify the volume information, or you specify an SMS-managed volume, the system will allocate the SMS-managed data set.

**REF=dsname**
**REF=*.ddname**
**REF=*.stepname.ddname**
**REF=*.stepname.procstepname.ddname**

Tells the system to obtain volume serial numbers from another data set or an earlier DD statement.

**Note:**

If the referencing data set is an existing relative generation data set (GDG data set) and the VOL=REF specifies one of the three forms of REF=*.ddname, then

the VOL=REF is ignored and the volume serial numbers are taken from the catalog entry for the named generation of the data set. For example:

```
// ddname2 DD DSN=TSOUSR2.TAPE.GDGNAME(0),DISP=SHR,VOL=REF=*.
stepname.ddname1
```

The volume serial numbers used would be for the current generation of TSOUSR2.TAPE.GDGNAME, not for the data set referenced by *.stepname.ddname1.

VOL=REF obtains ONLY the volume serial numbers from the referenced data set or earlier DD statement. In particular it does not obtain the volume sequence number, volume count, label type, or data set sequence number.

**dsname**

Names a cataloged or passed data set. The system assigns this data set to the same volumes containing the cataloged or passed data set.

When dsname names a passed data set, the reference must appear on a DD statement before the receiving DD statement. (After a passed data set is received, the passed data set information is no longer available.)

When the dsname contains special characters, other than the periods used in a qualified name, enclose it in apostrophes.

The dsname can be an alias name or a catalog name. The dsname cannot be a generation data group (GDG) base name or a member name of a non-GDG data set.

The dsname can be a GDG relative generation member, but since it contains special characters (the parentheses) it must be enclosed in apostrophes. It is important to be aware that while a GDG relative generation name in the DSNAME parameter always resolves to the state of the GDG index as of the beginning of the job, a GDG relative generation name in the dsname sub-parameter of the VOLUME parameter always resolves to the state of the GDG index as of the beginning of the step. For example if, at the beginning of a job, the latest generation of a dataset was G0007V00, and steps 1 and 2 of the job each created a new generation, then the following JCL for step 3 of the job would be resolved as shown:

```
//STEP3  EXEC PGM=pgmname
//DD1    DD   DSN=gdgname(0),
//            DISP=SHR,
//            VOL=REF='gdgname(0)'
```

DSN=gdgname(0) would resolve to generation G0007V00, since that was the zeroeth generation at the beginning of the job.

VOL=REF=gdgname(0) would resolve to generation G0009V00, since that was the zeroeth generation at the beginning of the step.

By the same token,

```
//STEP3  EXEC PGM=pgmname
//DD1    DD   DSN=gdgname(+3),
//            DISP=NEW,
//            VOL=REF='gdgname(+2)'
```

DSN=gdgname(+3) would resolve to generation G0010V00, since that would be the +3 generation at the beginning of the job.

VOL=REF=gdgname(+2) would resolve to generation G0011V00, since that would be the +2 generation at the beginning of the step. Since G0011V00 would not yet exist, it would not be possible to refer to it as an existing dataset, and so this JCL would fail.

***.ddname**
Asks the system to obtain the volume serial numbers from earlier DD statement ddname in the same job step.

***.stepname.ddname**
Asks the system to obtain the volume serial numbers from DD statement, ddname, in an earlier step, stepname, in the same job.

***.stepname.procstepname.ddname**
Asks the system to obtain the volume serial numbers from a DD statement in a cataloged or in-stream procedure. Stepname is the name of the job step that calls the procedure, procstepname is the name of the procedure step that contains the DD statement, and ddname is the name of the DD statement.

**Referenced data set not opened**: When REF refers to a DD statement in a previous step and the data set was not opened, the system allocates a device that has the widest range of eligibility to meet both DD statement requests. Thus, the system might allocate a device for which the referring data set is not eligible. To prevent this problem for tape data sets, always code the DCB DEN subparameter or the DCB TRTCH subparameter on a DD statement that you plan to reference.

**References to multivolume tape data sets**: When REF refers to a data set residing on more than one tape volume, the system allocates all volumes to the referencing DD when it represents an OLD data set, that is, a data set that existed prior to the current job step. For a NEW tape data set the system allocates only the last volume of a referenced multivolume tape data set.

If an *earlier* job step extends the referenced data set to more volumes, or adds or extends an earlier data set so that the referenced data set resides on a later volume, the new volume information *is* available to the referencing DD statement.

If the *current* job step extends the referenced data set to more volumes, or adds or extends an earlier data set so that the referenced data set resides on a later volume, the new volume information is available to the referencing DD statement ONLY when the referenced data set is a new data set with no volume serial numbers explicitly or implicitly specified, which means only if the entire collection of data sets on the volumes was created in the current step. In other words, if the current job step extends the referenced data set to more volumes, or adds or extends an earlier data set so that the referenced data set resides on a later volume, the new volume information *is not* available to the referencing DD statement when either of the following conditions is true:

- The data set that is referenced (directly or through a chain of references) existed before the start of the step containing the reference.
- The data set that is referenced (directly or through a chain of references) is a new data set requested with specific volume serial numbers. However the new volume will be resolved if one unit is allocated when writing multifile, multivolume labelled data sets leaving the tape positioned at the end of each data set created.

If the referenced data set already exists *and* has volume serial numbers explicitly specified, then the last listed volume serial is used even if the earlier data set actually exists on or is written to fewer volumes.

If the referenced data set is new and has specific volume serials, then the last listed volume serial is used even if the data set is written with fewer volumes.

In either of these cases, the allocation of the referencing data set is likely to fail.

**References to multivolume direct access data sets**: When REF refers to a data set that resides on more than one direct access volume, the system allocates all of the volumes.

If a DD statement that is requesting a new data set has a unit count and volume count greater than one but specifies no volume serial numbers, one volume is allocated. If a second DD statement within the same step requests the same data set, the same volume is allocated to it. If this job step extends the data set to more volumes, this new volume information is not available to the second DD statement.

Two or more DD statements in the same step can request the same data set. However, if the data set is extended to additional volumes in that step, the additional volume information is not available to the second or succeeding DD statements within the step.

**References to DD statements with UNIT group names**: When REF refers to a DD statement containing a UNIT group-name subparameter, the system allocates a device of the same type actually used for the referenced data set, but not necessarily a device in the referenced group-name.

**References to VSAM data sets**: When REF refers to a multivolume VSAM data set, the system allocates a device of the same type as the first device type used for the referenced VSAM data set.

**References to SMS-managed data sets**: When REF refers to an SMS-managed data set, SMS manages the new data set using the storage class of the referenced data set, if it is available, and applies these rules:

- If the reference is to a data set on one or more SMS-managed tape volumes, then the two data sets must be assigned to the same storage group. If the automatic class selection (ACS) routine does not assign the same storage group to the referenced and referencing data sets, the allocation fails with message IGD304I.
- For references to data sets on SMS-managed media other than tape, the two data sets must be assigned to compatible types of storage groups. This ensures the consistency for locate requests. For example, if the referenced data set is on DASD, allocating the referencing data set to be allocated on tape could result in potential locate request errors. If the ACS routine does not assign compatible types of storage groups to both data sets, the allocation fails with message IGD318I.

**References to non-SMS-managed data sets**: When REF refers to a non-SMS-managed data set, the ACS routine receives control and can do one of two things:

- Allow the allocation to proceed as a non-SMS-managed data set.
- Fail the allocation by exiting with a non-zero return code.

If the ACS routine attempts to make the referencing data set SMS-managed, SMS fails the allocation with message IGD305I.

**Do not refer to in-stream data sets**: Do not refer to a DD *, DD DATA, or DD SYSOUT statement. The system ignores the reference and defaults the device name to SYSALLDA, which is the group name for all direct access devices defined to the system.

**References to DUMMY data sets**: If ddname refers to a DD DUMMY statement, the data set for this DD statement is also assigned a dummy status.

**Label type picked up from referenced statement**: When REF is coded, the system also copies the LABEL label type subparameter from the referenced DD statement.

## Overrides

The volume sequence number overrides a DISP=MOD parameter. Thus, instead of starting at the end of the data set on the last volume, according to the MOD subparameter, processing of the data set begins with the volume indicated by the volume sequence number.

## Relationship to other parameters

Do not code the following parameters with the VOLUME parameter.

| | | |
|---|---|---|
| BURST | DDNAME | MODIFY |
| CHARS | DYNAM | QNAME |
| COPIES | FLASH | SYSOUT |

Do not code VOLUME=REF with the STORCLAS parameter.

***Other DD parameter picked up from referenced statement:*** When REF is coded, the system also copies the LABEL label type subparameter from the referenced DD statement.

***For 3540 Diskette Input/Output Units:*** The VOLUME=SER, DCB=BUFNO, and DSID parameters on a DD * or DD DATA statement are ignored except when they are detected by a diskette reader as a request for an associated data set. See the *3540 Programmer's Reference*.

## VOLUME parameter in a JES3 system

When you do not code a volume serial number, code PRIVATE if you want JES3 to manage the allocation. Otherwise, MVS manages the allocation.

RETAIN is ignored in a JES3 system.

## VOLUME parameter for optical readers

For optical readers, if no volume serial number is specified, the system assumes VOLUME=SER=OCRINP.

## VOLUME parameter for nonspecific volume requests

A nonspecific volume request can appear on a DD statement for a new data set; the data set is assigned to any volume or volumes. The nonspecific request is made through a VOLUME parameter that does not contain a SER or REF subparameter. The parameter can contain the following subparameters:

```
VOLUME=(PRIVATE,RETAIN,,volume-count)
```

**Note:** The use of PRIVATE on nonspecific requests eligible to permanently resident DASD devices is not recommended. Operator intervention is required to allow the system to allocate such a request to a private volume.

## VOLUME parameter for specific multi-volume tape requests

When allocating a specific, multi-volume tape data set, if the data set resides on multiple tape volumes that are:

- System-managed, then all volumes should reside in the same system-managed tape library and the same tape storage group. (If all of the volumes do not reside in the same tape library, the installation can enter the volumes through the DFSMS installation exit, CBRUXVNL.) These volumes must also be part of the same SMS storage group.
- Non-system-managed, then all volumes must be outside of any system-managed tape library.

## Examples of the VOLUME parameter

***Example 1***

```
//DD1  DD  DSNAME=DATA3,UNIT=SYSDA,DISP=OLD,
//       VOLUME=(PRIVATE,SER=548863)
```

The DD statement requests an existing data set, which resides on the direct access volume, serial number 548863. Since PRIVATE is coded, the system will not assign to the volume another data set for which a nonspecific volume request is made and will demount the volume at the end of the job.

***Example 2***

```
//DD2  DD  DSNAME=QUET,DISP=(MOD,KEEP),UNIT=(3390,2),
//       VOLUME=(,,,4,SER=(96341,96342))
```

The DD statement requests an existing data set, which resides on two volumes, serial numbers 96341 and 96342. The VOLUME volume count subparameter requests four volumes, if required. Thus, if more space is required, the system can assign a third and fourth volume.

***Example 3***

```
//DD3  DD  DSNAME=QOUT,UNIT=3390
```

The DD statement defines a data set that is created and deleted in the job step. By omission of the VOLUME parameter, the statement makes a nonspecific volume request, thereby asking the system to assign a suitable volume to the data set.

***Example 4***

```
//DD4  DD  DSNAME=NEWDASD,DISP=(,CATLG,DELETE),UNIT=3390,
//      VOLUME=SER=335006,SPACE=(CYL,(10,5))
```

This new data set is assigned to volume serial number 335006, which is a permanently mounted volume on a particular 3350 Direct Access Storage. You can obtain the same space on the same volume in another way: instead of specifying the volume serial number and UNIT=3350, you can specify the device number of the particular 3350 device in the UNIT parameter.

***Example 5***

```
//OUTDD  DD  DSNAME=TEST.TWO,DISP=(NEW,CATLG),
//         VOLUME=(,,,3,SER=(333001,333002,333003)),
//         SPACE=(TRK,(9,10)),UNIT=(3390,P)
//NEXT   DD  DSNAME=TEST.TWO,DISP=(OLD,DELETE)
```

DD statement OUTDD creates a multivolume data set and catalogs it. If the data set does not require three volumes, it will reside on fewer volumes. DD statement NEXT then deletes the data set.

If the data set resides on fewer volumes than the number of volumes on which it is cataloged, the following messages appear in the job log when the system deletes the data set:

```
IEF285I   TEST.TWO                                    DELETED
IEF285I   VOL SER NOS=333001,333003.
IEF283I   TEST.TWO                                    NOT DELETED
IEF283I   VOL SER NOS=333002 1.
IEF283I   TEST.TWO                                    UNCATALOGED
IEF283I   VOL SER NOS=333001,333002,333003.
```

If the data set resides on all specified volumes, the following messages appear in the job log when the system deletes the data set:

```
IEF285I   TEST.TWO                                    DELETED
IEF285I   VOL SER NOS=333001,333002,333003.
```

***Example 6***

```
//SMSDS2  DD  DSNAME=MYDS2.PGM,STORCLAS=SCLAS02,DISP=(NEW,KEEP),
//          VOLUME=SER=(223344,224444)
```

For new system-managed DASD data setsor data sets on a system-managed tape volume, the system uses the attributes in the storage class named SCLAS02 for the storage service level of the data set. Also, if the storage administrator has specified GUARANTEED_SPACE=YES in the storage class for DASD VOLUME=SER can be coded and the data set will reside on the specified volumes. (However, if space is not available on the volumes, the job step fails. Allocation also fails if the requested volumes aren't in any of the possible storage groups for the data set. For tape requests, the system always gets the tape request specified with a specific volume serial.) Installation-written automatic class selection (ACS) routines select the data class and management class.

***Example 7***

```
//STEP1  EXEC PGM=....
//DD1    DD DSN=OLD.SMS.DATASET,DISP=SHR
//DD2    DD DSN=FIRST,DISP=(NEW,CATLG,DELETE),VOL=REF=*.DD1

//STEP2  EXEC PGM=...
//DD3    DD DSN=SECOND,DISP=(NEW,CATLG,DELETE),VOL=REF=*.STEP1.DD1
```

DD1 in STEP1 identifies the original SMS-managed data set OLD.SMS.DATASET. DD2 in STEP1 and DD3 in STEP2 each create an SMS-managed data set using the attributes in the storage class associated with the original data set OLD.SMS.DATASET in DD1.

# Chapter 13. Special DD statements

Use special DD statements to specify private catalogs, private libraries, and data sets for storage dumps and checkpoints. This topic provides descriptions of these special statements.

## Description

### Syntax

```
//ddname  DD keyword-parameter[,keyword-parameter]...  [comments]
```

### Special ddnames

The special data sets are identified by the following ddnames:

JOBLIB
STEPLIB
SYSABEND
SYSCHK
SYSCKEOV
SYSIN
SYSMDUMP
SYSUDUMP

Except for SYSIN, code these ddnames only when you want the special data sets.

## JOBLIB DD statement

***Purpose:*** Use the JOBLIB DD statement to:

- Create a private library.
- Identify a private library that the system is to search for the program named in each EXEC statement PGM parameter in the job. Only if the system does not find the program in the private library, does it search the system libraries.

A private library is a partitioned data set or partitioned data set extended on a direct access device. Each member is an executable, user-written program.

### Syntax

```
//JOBLIB DD parameter[,parameter]...  [comments]
```

### Parameters on JOBLIB DD statements

***When retrieving a cataloged library:***

- Code the DSNAME parameter.

© Copyright IBM Corp. 1988, 2015

- Code the DISP parameter. The status subparameter must be OLD or SHR. The disposition subparameters should indicate what you want done with the private library after its use in the job.
- Do not code VOLUME or UNIT.

*When retrieving a library that is not cataloged:*

- Code the DSNAME parameter.
- Code the DISP parameter. The DISP parameter must be DISP=(OLD,PASS) or DISP=(SHR,PASS). SHR indicates that the data set is old, but allows other jobs to use the library.
- Code the UNIT parameter.
- Code the VOLUME parameter.

*When creating a library:*

- Code the DSNAME parameter to assign the library a name.
- Code the UNIT parameter. The library must be allocated to a direct access device.
- Code a VOLUME parameter, unless a nonspecific request is to be made for any volume.
- Code the SPACE parameter, allowing enough space for the entire library on one direct access volume. Specify space for the PDS directory.
- Code a DISP parameter. The status is NEW. Code CATLG as the disposition, if you intend to keep the library you are creating. Code PASS as the disposition, if you wish the library to be available throughout the job, but deleted at job termination. Note that you must code a disposition; otherwise, the system assumes DELETE and deletes the library at the end of the first step.

**Note:** Do not use VSAM for a JOBLIB library.

*When adding members to the library:*

- In the DSNAME parameter, follow the library name with the name of the program being added to the library. For example, DSNAME=LIBRARY(PROGRAM).
- Code the status in the DISP parameter as MOD. If you cataloged the library when you created it, do not code a disposition. Otherwise, code PASS or CATLG.
- If the JOBLIB library is being created in the job, the JOBLIB DD DISP specified CATLG, and the first step adds a member to it, supply unit and volume information in the first step by coding: VOLUME=REF=*.JOBLIB. This parameter is needed because the library is not actually cataloged until the first step completes execution. Otherwise, unit and volume information should not be supplied for a cataloged library.
- Do not code a SPACE parameter. The JOBLIB DD statement requests space for the entire library.

***Other parameters:*** Code the DCB parameter if complete data control block information is not contained in the data set label. Do not specify FREE=CLOSE; CLOSE is ignored.

Do not code a UNIT=AFF parameter on a JOBLIB statement where the object of the affinity is the same JOBLIB statement. In other words, a JOBLIB statement should not have an affinity back to itself.

## Relationship to other control statements

***Concatenating job libraries:*** To specify more than one private library for a job:

- Code a JOBLIB DD statement.
- Immediately follow this statement with DD statements that define other private libraries. Omit a ddname from these subsequent DD statements.

The system searches the libraries for the program in the same order as the DD statements.

***Overriding a JOBLIB:*** If you want the system to ignore the JOBLIB for a particular job step and the step does not require another private library, define the system library on a STEPLIB DD statement. For example, specify:

```
//STEPLIB DD  DSNAME=SYS1.LINKLIB,DISP=SHR
```

For this particular job step, the system will search SYS1.LINKLIB, as specified on the STEPLIB DD statement, for the program requested in the EXEC statement. The system will not search the JOBLIB.

***EXEC statement COND parameter:*** If COND=ONLY is specified on the EXEC statement of a job step and a JOBLIB DD statement is being used, the system does not pass the unit and volume information to any succeeding steps, and the system must search the catalog for the JOBLIB data set's unit and volume information.

## Location in the JCL

- The JOBLIB DD statement must immediately follow the JOB statement and any JES statements. There must be no intervening EXEC or other DD statements between the JOBLIB DD statement and the JOB statement.
- If libraries are concatenated to the JOBLIB library, the concatenated DD statements must immediately follow the JOBLIB DD statement.
- Do not include a JOBLIB DD statement in an in-stream or cataloged procedure.

## Relationship of a JOBLIB to a STEPLIB

Use a STEPLIB DD statement to define a private library for one job step in a job. If you include a STEPLIB DD statement for a job step and a JOBLIB DD statement for the entire job, the system first searches the step library and then the system library for the requested program. The system ignores the job library for a step that has a STEPLIB DD statement.

## Examples of the JOBLIB DD statement

***Example 1:***

```
//PAYROLL JOB  JONES,CLASS=C
//JOBLIB  DD   DSNAME=PRIVATE.LIB4,DISP=(OLD,PASS)
//STEP1   EXEC PGM=SCAN
//STEP2   EXEC PGM=UPDATE
//DD1     DD   DSNAME=*.JOBLIB,DISP=(OLD,PASS)
```

The private library requested on the JOBLIB DD statement is cataloged. The system passes catalog information to subsequent job steps. The system searches for the programs SCAN and UPDATE first in PRIVATE.LIB4, then in SYS1.LINKLIB. DD statement DD1 refers to the private library requested in the JOBLIB DD statement.

***Example 2:***

```
//PAYROLL JOB  FOWLER,CLASS=L
//JOBLIB  DD   DSNAME=PRIV.DEPT58,DISP=(OLD,PASS),
//           UNIT=3390,VOLUME=SER=D58PVL
//STEP    EXEC PGM=DAY
//STEP2   EXEC PGM=BENEFITS
//DD1     DD   DSNAME=*.JOBLIB,VOLUME=REF=*.JOBLIB,DISP=(OLD,PASS)
```

The private library requested on the JOBLIB DD statement is not cataloged; therefore, unit and volume information is specified. The system searches for the programs DAY and BENEFITS first in PRIV.DEPT58, then in SYS1.LINKLIB. DD statement DD1 refers to the private library requested in the JOBLIB DD statement.

***Example 3:***

```
//TYPE   JOB  MSGLEVEL=(1,1)
//JOBLIB DD   DSNAME=GROUP8.LEVEL5,DISP=(NEW,CATLG),
//          UNIT=3390,VOLUME=SER=148562,SPACE=(CYL,(50,3,4))
//STEP1  EXEC PGM=DISC
//DDA    DD   DSNAME=GROUP8.LEVEL5(RATE),DISP=MOD,
//          VOLUME=REF=*.JOBLIB
//STEP2  EXEC PGM=RATE
```

The private library requested on the JOBLIB DD statement does not exist yet; therefore, the JOBLIB DD statement contains all the parameters required to define the library. The library is created in STEP1, when DD statement DDA defines the new member RATE for the library. Therefore, the system searches SYS1.LINKLIB for the program named DISC. In STEP2, the system searches for the program RATE first in GROUP8.LEVEL5.

***Example 4:***

```
//PAYROLL JOB  BIRDSALL,TIME=1440
//JOBLIB  DD   DSNAME=KRG.LIB12,DISP=(OLD,PASS)
//        DD   DSNAME=GROUP31.TEST,DISP=(OLD,PASS)
//        DD   DSNAME=PGMSLIB,UNIT=3390,
//           DISP=(OLD,PASS),VOLUME=SER=34568
```

The three DD statements concatenate the three private libraries. The system searches the libraries for each program in this order:

- KRG.LIB12
- GROUP31.TEST
- PGMSLIB
- SYS1.LINKLIB

## STEPLIB DD statement

***Purpose:*** Use the STEPLIB DD statement to:

- Create a private library.
- Identify a private library that the system is to search for the program named in the EXEC statement PGM parameter. If the system does not find the program in the private library, only then does the system search the system libraries.

The private library is a partitioned data set (PDS) or partitioned data set extended (PDSE) on a direct access device. Each member is an executable, user-written program.

Subsequent job steps in the same job may refer to or receive a private library defined on a STEPLIB DD statement. Also, you can place a STEPLIB DD statement in an in-stream or cataloged procedure.

## Syntax

```
//STEPLIB DD parameter[,parameter]...  [comments]
```

## Parameters on STEPLIB DD statements

***When retrieving a cataloged library:***

- Code the DSNAME parameter.
- Code the DISP parameter. The status subparameter must be OLD or SHR. The disposition subparameters should indicate what you want done with the private library after its use in the job step.
- Do not code VOLUME or UNIT.

***When retrieving a library passed from a previous step:*** In the passing job step, code a DISP disposition subparameter of PASS when a step library is to be used by subsequent steps in the job.

In a receiving step:

- Code in the DSNAME parameter either the name of the step library or a backward reference of the form *.stepname.STEPLIB. If the step library is defined in a procedure, the backward reference must include the procedure step name: *.stepname.procstepname.STEPLIB.
- Code the DISP parameter. The status subparameter must be OLD. The disposition subparameters should indicate what you want done with the private library after its use in the receiving step.

***When retrieving a library that is neither cataloged nor passed:***

- Code the DSNAME parameter.
- Code the DISP parameter. The status subparameter must be OLD or SHR. The disposition subparameters should indicate what you want done with the private library after its use in the job step.
- Code the UNIT parameter.
- Code the VOLUME parameter.

***When creating a library:***

- Code the DSNAME parameter to assign the library a name.
- Code the UNIT parameter. The library must be allocated to a direct access device.
- Code a VOLUME parameter, unless a nonspecific request is to be made for any volume.
- Code the SPACE parameter, allowing enough space for the entire library on one direct access volume. Specify space for the PDS directory.
- Code a DISP parameter. The status is NEW. Code CATLG as the disposition, if you intend to keep the library you are creating. Code PASS as the disposition, if you wish the library to be available to a following step. Note that you must code a disposition; otherwise, the system assumes DELETE and deletes the library at the end of the step.

**Note:** Do not use VSAM for a STEPLIB library.

***When adding members to the library:***

- In the DSNAME parameter, follow the library name with the name of the program being added to the library. For example, DSNAME=LIBRARY(PROGRAM).
- Code the status in the DISP parameter as MOD. If the library is cataloged, do not code a disposition. Otherwise, code PASS or CATLG.
- If the library is cataloged, do not code unit and volume information. Otherwise, code UNIT and VOLUME.
- Do not code a SPACE parameter. The STEPLIB DD statement requests space for the entire library.

***Other parameters:*** Code the DCB parameter if complete data control block information is not contained in the data set label. Do not specify FREE=CLOSE; CLOSE is ignored.

## Relationship to other control statements

***Concatenating step libraries:*** To specify more than one private library for a step:

- Code a STEPLIB DD statement.
- Immediately follow this statement with DD statements that define other private libraries. Omit a ddname from these subsequent DD statements.

The system searches the libraries for the program in the same order as the DD statements.

***Overriding a JOBLIB:*** If you want the system to ignore the JOBLIB for a particular job step and the step does not require another private library, define the system library on a STEPLIB DD statement. For example, specify:

```
//STEPLIB DD  DSNAME=SYS1.LINKLIB,DISP=SHR
```

For this particular job step, the system will first search SYS1.LINKLIB, as specified on the STEPLIB DD statement, for the program requested in the EXEC statement. The system will not search the JOBLIB.

## Location in the JCL

Place a STEPLIB DD statement in any position among the DD statements for a step.

If libraries are concatenated to the STEPLIB library, the concatenated DD statements must immediately follow the STEPLIB DD statement.

## Relationship of a STEPLIB to a JOBLIB

Use a JOBLIB DD statement to define a private library that the system is to use for an entire job. If you include a JOBLIB DD statement for the job and a STEPLIB DD statement for an individual job step, the system first searches the step library and then the system library for the program requested in the EXEC statement. The system ignores the JOBLIB library for that step.

## Examples of the STEPLIB DD statement

***Example 1***

```
//PAYROLL JOB  BROWN,MSGLEVEL=1
//STEP1   EXEC PROC=LAB14
//STEP2   EXEC PGM=SPKCH
```

```
//STEPLIB DD   DSNAME=PRIV.LIB5,DISP=(OLD,KEEP)
//STEP3   EXEC PGM=TIL80
//STEPLIB DD   DSNAME=PRIV.LIB12,DISP=(OLD,KEEP)
```

The system searches PRIV.LIB5 for the program SPKCH and PRIV.LIB12 for TIL80. The system catalogs both private libraries.

***Example 2***

```
//PAYROLL JOB  BAKER,MSGLEVEL=1
//JOBLIB  DD   DSNAME=LIB5.GROUP4,DISP=(OLD,PASS)
//STEP1   EXEC PGM=SNZ12
//STEP2   EXEC PGM=SNAP10
//STEPLIB DD   DSNAME=LIBRARYP,DISP=(OLD,PASS),
//           UNIT=3390,VOLUME=SER=55566
//STEP3   EXEC PGM=A1530
//STEP4   EXEC PGM=SNAP11
//STEPLIB DD   DSNAME=*.STEP2.STEPLIB,
//           DISP=(OLD,KEEP)
```

The system searches LIBRARYP for program SNAP10; LIBRARYP is passed to subsequent steps of this job. The STEPLIB DD statement in STEP4 refers to the LIBRARYP library defined in STEP2; the system searches LIBRARYP for SNAP11. Since a JOBLIB DD statement is included, the system searches for programs SNZ12 and A1530 first in LIB5.GROUP4, then in SYS1.LINKLIB.

***Example 3***

```
//PAYROLL JOB  THORNTON,MSGLEVEL=1
//JOBLIB  DD   DSNAME=LIB5.GROUP4,DISP=(OLD,PASS)
//STEP1   EXEC PGM=SUM
//STEPLIB DD   DSNAME=SYS1.LINKLIB,DISP=OLD
//STEP2   EXEC PGM=VARY
//STEP3   EXEC PGM=CALC
//STEPLIB DD   DSNAME=PRIV.WORK,DISP=(OLD,PASS)
//        DD   DSNAME=LIBRARYA,DISP=(OLD,KEEP),
//           UNIT=3390,VOLUME=SER=44455
//        DD   DSNAME=LIB.DEPT88,DISP=(OLD,KEEP)
//STEP4   EXEC PGM=SHORE
```

For STEP2 and STEP4, the system searches the private library named LIB5.GROUP4 defined in the JOBLIB DD statement first for programs VARY and SHORE. For STEP1, the system searches SYS1.LINKLIB first for program SUM, because the STEPLIB DD statement names the system library.

A concatenation of private libraries is defined in STEP3. The system searches for the program named CALC in this order: PRIV.WORK, LIBRARYA, LIB.DEPT88, SYS1.LINKLIB. If a later job step refers to the STEPLIB DD statement in STEP3, the system will search for the program in the private library named PRIV.WORK and, if it is not found there, in SYS1.LINKLIB; the concatenated libraries are not searched.

## SYSABEND, SYSMDUMP, and SYSUDUMP DD statements

***Purpose:*** Use a SYSABEND, SYSMDUMP, or SYSUDUMP DD statement in a job step to direct the system to produce a dump. The system produces the requested dump:

- If the step terminates abnormally.
- If the step starts to terminate abnormally, but system recovery procedures enable the step to terminate normally.

The dump DD statements for requesting dumps are:

**SYSABEND DD statement**
Produces a dump of user and system areas; this dump contains all the areas dumped in a SYSUDUMP, plus:

- The local system queue area (LSQA), including subpools 229, 230, and 249
- The input/output system (IOS) control blocks for the failing task.

The dump is formatted, so that it can be printed directly.

**SYSMDUMP DD statement**
Produces a dump of the system areas and the program's address space. The dump is unformatted and machine-readable. It must be processed by the interactive problem control system (IPCS) and therefore should not be directed to SYSOUT. z/OS R6 introduced system-determined BLKSIZE support for SYSMDUMPs. If you wish to control the BLKSIZE for compatibility with tools developed for earlier releases, add the following DCB attributes to your SYSMDUMP DD statement.

```
DCB=RECFM=FB,LRECL=4160,BLKSIZE=4160
```

**SYSUDUMP DD statement**
Produces a dump of user areas. The dump is formatted, so that it can be printed directly.

The dump contents are as described only when the installation uses the IBM-supplied defaults for the dumps. The contents of these dumps can be set during system initialization and/or can be changed for an individual dump in the ABEND macro instruction, in a CHNGDUMP command, and by a SLIP command. For details, see *z/OS MVS Initialization and Tuning Guide*.

Dumps are optional; use a dump DD statement only when you want to produce a dump.

***References:*** For information on how to interpret dumps, see *z/OS MVS Diagnosis: Tools and Service Aids*.

## Syntax

```
//SYSABEND  DD parameter[,parameter]...  [comments]
//SYSMDUMP  DD parameter[,parameter]...  [comments]
//SYSUDUMP  DD parameter[,parameter]...  [comments]
```

## Location in the JCL

Do not place in the same job step two DD statements with the same dump ddname.

## Storing a dump

If you wish to store a dump instead of having it printed, code the following parameters on the dump DD statement:

- The DSNAME parameter.
- The UNIT parameter.
- The VOLUME parameter. This parameter is optional and not recommended. The system will select a volume.
- The DISP parameter. The data set's status is NEW. Because you want to store the data set, make the data set's abnormal termination disposition KEEP or CATLG.

- The SPACE parameter, if the dump is written on direct access.

**Tip:** SYSABEND, SYSUDUMP, and SYSMDUMP can use extended format sequential data sets to exploit striping, compression, or both. Striping speeds the writing process and compression reduces the space consumed and speeds the I/O bound dump process. Extended format sequential data set hold more than 64K tracks per volume making it an attractive destination for dumps. DSNAME=LARGE also allows data sets to use more than 64K tracks per volume.

**Note:** Do not use VSAM for dump data sets.

***SYSMDUMP Requirements:*** The SYSMDUMP DD statement must specify a magnetic tape unit or a direct access device. Do not direct SYSMDUMP to SYSOUT.

With the exception of the following facility, the system processes dump data sets according to the disposition to which they are allocated. To keep only the first SYSMDUMP dump written to a dump data set, specify the following on the SYSMDUMP DD statement:

- DSNAME=SYS1.SYSMDPxx, where xx is 00 through FF and indicates the specific dump data set to be used. SYSMDPxx is a preallocated data set that must have end-of-file (EOF) mark as its first record.
- DISP=SHR
- FREE=CLOSE for multiple job steps

**Note:** This restriction is not enforced. If SYSOUT is used, the resulting dump will be unusable for diagnosis.

See *z/OS MVS Diagnosis: Tools and Service Aids* for a description of the SYS1.SYSMDPxx naming convention and an explanation of how the system manages the dump data sets.

## Printing a dump

To print a dump for either a SYSABEND or SYSUDUMP DD statement, code one of the following on the DD statement for the output data set:

- A UNIT parameter that specifies a printer.
- The SYSOUT parameter that specifies a print output class.

To print a dump for a SYSMDUMP DD statement, use the following program:

**IPCS**
: This program is described in *z/OS MVS IPCS User's Guide*. When using IPCS, the data set disposition affects the collection of events.

If you print the dump in a JES3 system on a 3800 Printing Subsystem, code CHARS=DUMP for a dump with 204 characters per line and FCB=STD3 for 8 lines per inch.

## Overriding dump DD statements

To change the type of dump requested in a dump DD statement in a cataloged or in-stream procedure, the ddname of the overriding DD statement in the calling step must be different from the dump ddname of the procedure DD statement.

## Duplicate dump requests

You can code more than one dump request in a job step using DD statements that have **different** ddnames. When you do this, the system uses the last dump DD statement it encounters.

When the system finds dump DD statements with duplicate ddnames, processing is as follows:

- **In a JES2 system**, the job fails with message IEA912I.
- **In a JES3 system**:
  - If both DD statements request JES3- or jointly-managed devices, the job is cancelled during JES3 interpretation.
  - If only one or neither statement requests JES3- or jointly-managed devices, the job fails with message IEA912I.

## Examples of the SYSABEND, SYSMDUMP, and SYSUDUMP DD statements

*Example 1*

```
//STEP2    EXEC PGM=A
//SYSUDUMP DD   SYSOUT=A
```

The SYSUDUMP DD statement specifies that you want the dump routed to system output class A.

*Example 2*

```
//SYSMDUMP DD  DSNAME=DUMP,DISP=(NEW,KEEP),
//          UNIT=3390,VOLUME=SER=147958
```

The SYSMDUMP DD statement specifies that the dump is to be stored on a tape. Because the LABEL parameter is not coded, the tape must have IBM standard labels.

*Example 3*

```
//STEP1    EXEC PGM=PROGRAM1
//SYSABEND DD   DSNAME=DUMP,UNIT=3390,DISP=(,PASS,KEEP),
//          VOLUME=SER=1234,SPACE=(TRK,(40,20))
//STEP2    EXEC PGM=PROGRAM2
//SYSABEND DD   DSNAME=*.STEP1.SYSABEND,DISP=(OLD,DELETE,KEEP)
```

Both SYSABEND DD statements specify that the dump is to be stored. The space request in STEP1 is ample and will not inhibit dumping due to insufficient space. If STEP1 does not abnormally terminate but STEP2 does, the system writes the dump for STEP2 in the space allocated in STEP1. In both steps, an abnormal termination disposition of KEEP is specified so that the dump is stored if either of the steps abnormally terminates. If both of the steps successfully execute, the second DISP subparameter, DELETE, in STEP2 instructs the system to delete the data set and free the space acquired for dumping.

*Example 4*

```
//STEP     EXEC PGM=EXSYSM
//SYSMDUMP DD   UNIT=3390,VOLUME=SER=123456,SPACE=(CYL,(0,1)),
//           DISP=(NEW,DELETE,KEEP),DSNAME=MDUMP
```

The SYSMDUMP DD statement allocates dump data set MDUMP to a direct access device.

*Example 5*

```
//JOB1     JOB
//STEP     EXEC PGM=EXSYSMDP
//SYSMDUMP DD   DSNAME=SYS1.SYSMDP00,DISP=SHR

//JOB2     JOB
//STEP     EXEC PGM=EXSYSMDP
//SYSMDUMP DD   DSNAME=SYS1.SYSMDP00,DISP=SHR
```

Only the SYSMDUMP dump written by the first job will be in data set SYS1.SYSMDP00. All subsequent jobs receive message IEA849I, indicating that the data set is full.

**Note:** When you specify a DSNAME of SYS1.SYSMDPxx with DISP=SHR, the system writes the first SYSMDUMP dump on the data set. You must offload this first SYSMDUMP dump and write an EOF mark at the beginning of the SYS1.SYSMDPxx data set before subsequent dumps can be written to that data set.

# SYSCHK DD statement

***Purpose:*** Use the SYSCHK DD statement to define a checkpoint data set that the system is to write during execution of a processing program. Use this statement again when the step is restarted from a checkpoint written in the data set.

**Note:** If restart is to begin at a step, as indicated by the RD parameter on the EXEC statement, do not use a SYSCHK DD statement.

***References:*** For detailed information about the checkpoint/restart facilities, see *z/OS DFSMSdfp Checkpoint/Restart*.

## Syntax

```
//SYSCHK DD parameter[,parameter]...  [comments]
```

## Parameters on SYSCHK DD statements

***When creating a checkpoint data set:***

- Code a SPACE parameter, but do not request secondary space.
  - The **primary space** request must be large enough to hold all checkpoints. Although your program or the system can write checkpoints in secondary space, the system **cannot** perform a restart from checkpoints in secondary space.
  - If you do **not** request **secondary space** and the primary space fills up, the job abnormally terminates. You can successfully restart the job at the last checkpoint; however, when the processing program or system writes the next checkpoint the job abnormally terminates again.
  - If you **do** request **secondary space** and the primary space fills up, the processing program or the system writes one invalid checkpoint followed by successful checkpoints. An attempt to restart from one of the checkpoints following the invalid checkpoint results in abnormal termination.
- Code the RLSE subparameter of the SPACE parameter only if the processing program opens the checkpoint data set and the checkpoint data set remains open until the end of the program. If you specify RLSE, the system releases unused space after the first CLOSE macro instruction.

Do **not** code the RLSE subparameter:

- If the processing program opens the checkpoint data set before writing each checkpoint and closes the checkpoint data set after writing each checkpoint. The system releases all unused space while closing the data set after the first checkpoint, leaving no space for additional checkpoints.
- If the system opens the checkpoint data set. The system opens and closes the checkpoint data set before it writes the first checkpoint. With RLSE specified, the system would release all space before the first checkpoint could be written.

- Code the **CONTIG** subparameter of the SPACE parameter to request contiguous space. The system otherwise provides additional primary space using extents. If the extents are **not** contiguous, any checkpoints in these extents cannot be used for a successful restart.

***When retrieving a cataloged checkpoint data set:***

- Code the DSNAME parameter.
- Code the DISP parameter to specify a status of OLD and a disposition of KEEP.
- Code the VOLUME parameter. If the checkpoint entry is on a tape volume other than the first volume of the checkpoint data set, code the volume serial number or volume sequence number to identify the correct volume. The serial number of the volume on which a checkpoint entry was written appears in the console message issued after the checkpoint entry is written.
- Code the UNIT parameter, if you coded the VOLUME parameter, because the system will not look in the catalog for unit information.

***When retrieving a checkpoint data set that is not cataloged:***

- Code the DSNAME parameter. If the checkpoint data set is a partitioned data set (PDS), do not code a member-name in the DSNAME parameter.
- Code the DISP parameter to specify a status of OLD and a disposition of KEEP.
- Code the VOLUME parameter. The serial number of the volume on which a checkpoint entry was written appears in the console message issued after the checkpoint entry is written.
- Code the UNIT parameter.

***Other parameters:***

- Code the LABEL parameter if the checkpoint data set does not have standard labels.
- Code DCB=TRTCH=C if the checkpoint data set is on 7-track magnetic tape with nonstandard labels or no labels.
- If the volume containing the checkpoint data set is to be mounted on a JES3-managed device, do not code the DEFER subparameter of the UNIT parameter on the SYSCHK DD statement.

**Note:** Do not use VSAM for a checkpoint data set, and do not use a partitioned data set extended (PDSE) for a checkpoint data set.

## Relationship to other control statements

Code the RESTART parameter on the JOB statement; without it, the system ignores the SYSCHK DD statement.

## Location in the JCL

- When writing checkpoints, place the SYSCHK DD statement after any JOBLIB DD statements, if coded; otherwise, after the JOB statement.
- When restarting a job from a checkpoint, place the SYSCHK DD statement immediately before the first EXEC statement of the resubmitted job.

## Examples of the SYSCHK DD statement

***Example 1***

```
//JOB1   JOB  RESTART=(STEP3,CK3)
//SYSCHK DD   DSNAME=CHLIB,UNIT=3390,
//            DISP=OLD,VOLUME=SER=456789
//STEP1  EXEC PGM=A
```

The checkpoint data set defined on the SYSCHK DD statement is not cataloged.

***Example 2***

```
//JOB2   JOB  RESTART=(STEP2,NOTE2)
//JOBLIB DD   DSNAME=PRIV.LIB3,DISP=(OLD,PASS)
//SYSCHK DD   DSNAME=CHECKPTS,DISP=(OLD,KEEP),
//            UNIT=3390,VOLUME=SER=438291
//STEP1  EXEC PGM=B
```

The checkpoint data set defined on the SYSCHK DD statement is not cataloged. Note that the SYSCHK DD statement follows the JOBLIB DD statement.

# SYSCKEOV DD statement

***Purpose:*** Use the SYSCKEOV DD statement to define a checkpoint data set for checkpoint records from the checkpoint at end-of-volume (EOV) facility. The checkpoint at EOV facility is invoked by a DD CHKPT parameter.

***References:*** For information on the DD CHKPT parameter, see "CHKPT parameter" on page 123. For information on checkpoint/restart facilities, see *z/OS DFSMSdfp Checkpoint/Restart*.

## Syntax

```
//SYSCKEOV DD parameter[,parameter]...  [comments]
```

## Parameters on SYSCKEOV DD statements

***When creating a checkpoint data set:***

- Code a SPACE parameter, but do not request secondary space. The **primary space** request must be large enough to hold all checkpoints; if not, the job abnormally terminates.
- Do not code the RLSE subparameter of the SPACE parameter.
- Code the CONTIG subparameter of the SPACE parameter to request contiguous space. The system otherwise provides additional primary space using extents.
- The SYSCKEOV DD statement must define a BSAM data set, but cannot define a partitioned data set extended (PDSE).
- Code DISP=MOD to reduce loss of checkpoint data in case of a system failure during checkpointing.

*Other parameters:*

- Do not code on the SYSCKEOV DD statement the following:
  - CHKPT=EOV parameter.
  - DCB parameter. All DCB information is provided by the checkpoint at EOV facility.
  - DEFER subparameter of the UNIT parameter.
- If you code the LABEL parameter, you must specify LABEL=(,SL) for IBM standard labels.
- If the SYSCKEOV data set resides on a direct access storage device, that device cannot be shared with another processor.

## Location in the JCL

If you code a CHKPT parameter on any DD statements in a job step, place a SYSCKEOV DD statement in the DD statements for the step.

## Example of the SYSCKEOV DD statement

```
//SYSCKEOV DD  DSNAME=CKPTDS,UNIT=TAPE,DISP=MOD
```

This statement defines a checkpoint data set for checkpoint at EOV records.

# SYSIN DD statement

***Purpose:*** By convention, people often use a SYSIN DD statement to begin an in-stream data set. In-stream data sets begin with a DD * or DD DATA statement; these DD statements can have any valid ddname, including SYSIN. If you omit a DD statement before input data, the system provides a DD * statement with the ddname of SYSIN.

## Syntax

```
//SYSIN DD parameter[,parameter]...  [comments]
```

## Parameters on SYSIN DD statements

The first parameter is an * or DATA, to signal that an in-stream data set follows immediately.

Do not code a symbolic in place of (one that would resolve to) the * or DATA positional parameter on a SYSIN type DD statement.

## Location in the JCL

A SYSIN DD statement appears at the beginning of an in-stream data set.

## Examples of SYSIN DD statements

```
//STEP1  EXEC  PGM=READ
//SYSIN  DD    *
         .
         .
         data
         .
//OUT1   DD    SYSOUT=A
//STEP2  EXEC  PGM=WRITE
```

```
//SYSIN  DD    DATA,DLM=17
         .
         .
         .
17
```

# Chapter 14. Delimiter statement

***Purpose:*** Use the delimiter statement to indicate the end of data or transmittal records in the input stream.

***Considerations for an APPC scheduling environment:*** The delimiter statement has no function in an APPC scheduling environment. If you code a delimiter statement, the system will ignore it.

## Description

### Syntax

```
/* [comments]
xx [comments]
```

A delimiter statement consists of the characters /* or the two characters specified in a DLM parameter in columns 1 and 2 and one field: comments.

Do not continue a delimiter statement.

### Comments field

The comments field follows the delimiter characters.

For JES2, code any comments in columns 4 through 80. (A blank must follow the delimiter characters.)

For JES3, text in columns 3 through 80 is a comment, except when the default delimiter (/*) is used with an //XMIT statement causing the text starting in column 3 to be recognized as a JECL statement (for example, /*ROUTE, /*JOBPARM). This includes JES2 commands (/*$command) except that any command prefix other than $ is considered a comment instead of a command.

To avoid ambiguity in these cases, IBM recommends that you either start comments in column 4 or use a delimiter other than the default on the //XMIT statement.

### Relationship to the DLM parameter

The system recognizes a delimiter other than /* if a DLM parameter is coded on:

- A DD * or DD DATA statement that defines an in-stream data set.
- An XMIT JCL statement that precedes input stream records to be transmitted to another node.
- A JES2 /*XMIT statement that precedes input stream records to be transmitted to another node.

A delimiter statement is optional:

- If the data is preceded by a DD * statement without a DLM parameter.
- If transmitted records are preceded by an /*XMIT statement without a DLM parameter.

© Copyright IBM Corp. 1988, 2015

## Location in the JCL

A delimiter statement must appear:

- At the end of an in-stream data set that begins with a DD DATA statement.
- At the end of an in-stream data set that begins with a DD statement containing a DLM parameter.
- At the end of records to be transmitted to another node when the records are preceded by an /*XMIT statement containing a DLM parameter.
- At the end of records to be transmitted to another node when the records are preceded by an XMIT JCL statement.

## Examples of the delimiter statement

***Example 1***

```
//JOB54   JOB   ,'C BROWN',MSGLEVEL=(2,0)
//STEPA   EXEC  PGM=SERS
//DD1     DD    *
          .
          .
          data
          .
/*        END OF DATA FOR DATA SET DD1
//DD2     DD    DATA,DLM=AA
          .
          .
          data
          .
AA        END OF DATA FOR DATA SET DD2
```

***Example 2***

```
//JOB54   JOB   ,'C BROWN',MSGLEVEL=(2,0)
//        XMIT  DEST=NODEA,DLM=BB
//JOB55   JOB   ,'C BROWN',MSGLEVEL=(2,0)
//STEPA   EXEC  PGM=SERS
//DD1     DD    *
          .
          .
          data
          .
/*        END OF DATA FOR DATA SET DD1
//DD2     DD    DATA,DLM=AA
          .
          .
          data
          .
AA        END OF DATA FOR DATA SET DD2
BB        END OF TRANSMITTED JOB
```

This example shows nested delimiter statements.

# Chapter 15. ENDCNTL statement

***Purpose:*** Use the ENDCNTL statement to mark the end of the program control statements following a CNTL statement.

## Description

### Syntax

```
//[label] ENDCNTL [comments]
```

The ENDCNTL statement consists of the characters // in columns 1 and 2, and three fields: label, operation (ENDCNTL), and comments.

### Label field

Code a label on the ENDCNTL statement, as follows:

- Each label must be unique within the job.
- The label must begin in column 3.
- The label is 1 through 8 alphanumeric or national ($, #, @) characters.
- The first character must be alphabetic or national ($, #, @).
- The label must be followed by at least one blank.

### Operation field

The operation field consists of the characters ENDCNTL and must be preceded and followed by at least one blank. It can begin in any column.

### Comments field

The comments field follows the ENDCNTL after at least one intervening blank.

### Location in the JCL

The ENDCNTL statement immediately follows the one or more program control statements following a CNTL statement. Thus, the ENDCNTL statement can appear in a job step or in a cataloged or in-stream procedure.

### Example of the ENDCNTL statement

```
//STEP1     EXEC      PGM=PRINT
//ABLE      CNTL
//STATE1    PRINTDEV  BUFNO=20,PIMSG=YES,DATACK=BLOCK
//BAKER     ENDCNTL
//CALLER    DD        UNIT=3800-3,CNTL=*.ABLE
```

(For information about the PRINTDEV JCL statement see *PSF for z/OS: Customization*.)

© Copyright IBM Corp. 1988, 2015

# Chapter 16. EXEC statement

***Purpose:*** Use the EXEC (execute) statement to identify the program or cataloged or in-stream procedure that this job step is to execute and to tell the system how to process the job step. The EXEC statement marks the beginning of each step in a job or a procedure.

A job can have a maximum of 255 job steps. This maximum includes all steps in any procedures the EXEC statements call.

The parameters you can specify for step processing are arranged alphabetically.

***References:*** For information about the JES initialization parameters that provide installation defaults, see *z/OS JES2 Initialization and Tuning Reference* and *z/OS JES3 Initialization and Tuning Reference*.

## Description

### Syntax

```
//[stepname]  EXEC  positional-parm[,keyword-parm]...[,symbolic-parm=value]...
                                    [comments]
```

The EXEC statement consists of the characters // in columns 1 and 2 and four fields: name, operation (EXEC), parameter, and comments.

An EXEC statement is required for each job step.

### Name field

A stepname is optional, but is needed for the following. When a stepname is needed, it must be unique within the job, including stepnames in any procedures called by the job. If stepnames are not unique within the job, results might be unpredictable; but in most cases, references to non-unique stepnames will resolve to the first occurrence of that stepname.

- Referring to the step in later job control statements.
- Overriding parameters on an EXEC statement or DD statement in a cataloged or in-stream procedure step.
- Adding DD statements to a cataloged or in-stream procedure step. However, a stepname is not required when adding to the first step in a procedure.
- Performing a step or checkpoint restart at or in the step.
- Identifying a step in a cataloged or in-stream procedure.

Code a stepname as follows:

- The stepname must begin in column 3.
- The stepname is 1 through 8 alphanumeric or national ($, #, @) characters.
- The first character must be alphabetic or national ($, #, @).
- The stepname must be followed by at least one blank.

© Copyright IBM Corp. 1988, 2015

- The stepname may be preceded by up to 8 alphanumeric or national characters and then separated by a period. If the stepname is coded in this way, the characters up to and including the period are ignored.

### Stepnames for started tasks

When JCL runs as a started task, the system assigns a stepname of `ssssssss` (when the START command was `S membername.ssssssss`) or `STARTING` (when the START command was `S membername`). Embedded procedures that refer back to the invoking procedure, as on a COND parameter, need to specify the stepname the system assigns.

## Operation field

The operation field consists of the characters EXEC and must be preceded and followed by at least one blank. It can begin in any column.

## Parameter field

An EXEC statement has two kinds of parameters: positional and keyword.

Do not use EXEC statement parameter keywords as symbolic parameters, names, or labels.

***Positional Parameters:*** An EXEC statement must contain **one** of the positional parameters: PGM, PROC, or procedure name. This positional parameter must precede all keyword parameters.

*Table 19. Positional parameters*

| POSITIONAL PARAMETERS | VALUES | PURPOSE |
|---|---|---|
| `PGM= {program-name }`<br>`{*.stepname.ddname }`<br>`{*.stepname.procstepname.ddname}`<br>`{JCLTEST }`<br>`{JSTTEST }`<br><br>See section “PGM parameter” on page 358 | program-name: 1 - 8 alphanumeric or $, #, @ characters member containing program<br><br>stepname: DD in named step<br><br>procstepname: step in named procedure<br><br>JCLTEST and JSTTEST: scan for syntax without executing the job (JES3 only) | Names the program the system is to execute or, for JES3 only, requests syntax check without execution |
| `{PROC=procedure-name}`<br>`{procedure-name }`<br><br>See section “PROC and procedure name parameters” on page 360 | procedure-name: 1 - 8 alphanumeric or $, #, @ characters | Names the cataloged or in-stream procedure the system is to call and execute. |

***Keyword parameters:*** An EXEC statement can contain the following keyword parameters. You can code any of the keyword parameters in any order in the parameter field after the positional parameter.

*Table 20. Keyword parameters*

| KEYWORD PARAMETERS | VALUES | PURPOSE |
|---|---|---|
| ACCT[.procstepname]=(accounting-information)<br><br>See section "ACCT parameter" on page 338 | accounting-information: up to 142 characters<br><br>[.procstepname]: name of procedure EXEC containing ACCT to be affected | Specifies accounting information for the step |
| ADDRSPC[.procstepname]= {VIRT}<br>{REAL}<br><br>See section "ADDRSPC parameter" on page 340 | VIRT: virtual (pageable) storage<br><br>REAL: central (nonpageable) storage<br>[.procstepname]: name of procedure EXEC containing ADDRSPC to be affected | Indicates the type of storage required for the step. |
| CCSID=*nnnnn* | *nnnnn*: 1 - 65535 | Specifies the coded character set identifier indicating the character code conversion performed on reads from and writes to tapes accessed in ISO/ANSI Version 4 format. |
| COND[.*procstepname*]=<br>((*code*,*operator*[,*stepname*][.*procstepname*]) )<br>([,(*code*,*operator*[,*stepname*][.*procstepname*])]...)<br>([,EVEN] )<br>([,ONLY]<br>)<br><br>See section "COND parameter" on page 342 | | |
| | *code*: 0 - 4095<br>operator: GT Code from<br>GE chart on<br>EQ section<br>Table 21 on page 347<br>LT<br>LE<br>NE<br><br>EVEN: execute step even if preceding step ended abnormally<br><br>ONLY: execute step only if preceding step ended abnormally<br><br>*stepname*: step issuing return code<br><br>*procstepname*: step is in named procedure<br><br>[.*procstepname*]: name of procedure EXEC containing COND to be affected | Specifies the return code tests used to determine if this step is to be executed or bypassed. |
| DYNAMNBR[.*procstepname*]=*n*<br><br>See section "DYNAMNBR parameter" on page 350 | n: 0 - 3273 minus number of DD statements in step<br><br>[.procstepname]: name of procedure EXEC containing DYNAMNBR to be affected | Holds a number of data set allocations for reuse and sets the control limit. |

*Table 20. Keyword parameters (continued)*

| KEYWORD PARAMETERS | VALUES | PURPOSE |
|---|---|---|
| MEMLIMIT={*nnnnn*M}<br>{*nnnnn*G}<br>{*nnnnn*T}<br>{*nnnnn*P}<br>{NOLIMIT}<br><br>See section "MEMLIMIT parameter" on page 351 | *n*: 0 - 99999 | Specifies the limit on the total number of usable virtual pages above the bar in a single address space. |
| PARM[.procstepname]=subparameter<br>PARM[.*procstepname*]=(*subparameter*,*subparameter*)<br>PARM[.*procstepname*]=('*subparameter*',*subparameter*)<br>PARM[.*procstepname*]='*subparameter*,*subparameter*'<br><br>See section "PARM parameter" on page 352 | | |
| | *subparameter*: up to 100 characters<br><br>[.procstepname]: name of procedure EXEC containing PARM to be affected | Passes variable information to the processing program. |
| PERFORM[.procstepname]=n<br><br>See section "PERFORM parameter" on page 356 | n: 1 - 999<br><br>[.procstepname]: name of procedure EXEC containing PERFORM to be affected | In WLM compatibility mode (not available on z/OS V1R3 or later systems), specifies the step's performance group. In WLM goal mode, PERFORM on the EXEC statement is ignored except for the TSO logon procedure, where it can be used to classify the TSO user to a service class or report class. |
| RD[.procstepname]= {R }<br>{RNC}<br>{NR }<br>{NC }<br><br>See section "RD parameter" on page 361 | R: restart, checkpoints allowed<br><br>RNC: restart, no checkpoints<br><br>NR: no restart, checkpoints allowed<br><br>NC: no restart, no checkpoints<br><br>[.procstepname]: name of procedure EXEC containing RD to be affected | In a non-APPC scheduling environment, indicates whether the operator should perform automatic step restart, if the step fails, and controls whether checkpoints are written for CHKPT macros or DD statement CHKPT parameters. |
| REGION[.procstepname]={valueK}<br>{valueM}<br><br>See section "REGION parameter" on page 364 | valueK: 1 - 7 digits from 1 - 2096128<br><br>valueM: 1 - 4 digits from 1 - 2047<br><br>[.procstepname]: name of procedure EXEC containing REGION to be affected | Specifies the amount of space in kilobytes or megabytes required by the step. |

*Table 20. Keyword parameters (continued)*

| KEYWORD PARAMETERS | VALUES | PURPOSE |
|---|---|---|
| RLSTMOUT[.*procstepname*]={*nnn*}<br>{0 } | *nnn*: a value in seconds ranging from 0 to 9999.<br><br>0: this value means that the request has no time out value. | Specifies the maximum time in seconds that a VSAM RLS or DFSMStvs request is to wait for a required lock before the request is assumed to be in deadlock. |
| TIME[.procstepname]= {([minutes][,seconds])}<br>{1440 }<br>{NOLIMIT }<br>{MAXIMUM }<br><br>See section "TIME parameter" on page 370 | minutes: 1 - 357912<br><br>seconds: 1 - 59<br><br>[.procstepname]: name of procedure EXEC containing TIME to be affected<br><br>NOLIMIT: specifies that the step can use the processor for an unlimited amount of time<br><br>MAXIMUM: specifies that the step can use the processor for the maximum amount of time | Specifies the maximum time the step is to use the processor and requests messages giving the time used. |

***Keyword parameters on an EXEC statement that calls a procedure:*** When an EXEC statement positional parameter calls a cataloged or in-stream procedure, all of the EXEC statement's keyword parameters override matching EXEC keyword parameters in the called procedure. If a keyword parameter is to override a parameter on only one EXEC statement in the procedure, code **.procstepname** immediately following the keyword:

```
keyword.procstepname=value
```

The procstepname is the name field on the procedure EXEC statement containing the keyword parameter to be overridden. For example:

```
//STEP1  EXEC  PROC=WKREPORT,ACCT.PSTEPWED=5670
```

The accounting information **5670** applies only to step PSTEPWED in the procedure WKREPORT.

***Symbolic parameters on an EXEC statement that calls procedures:*** An EXEC statement can assign values to, or nullify, symbolic parameters. See "Using system symbols and JCL symbols" on page 38 for more information about symbolic parameters.

## Comments field

The comments field follows the parameter field after at least one intervening blank.

## Location in the JCL

An EXEC statement must be the first statement in each job step or cataloged or in-stream procedure step.

## Examples of EXEC statements

***Example 1***

```
//STEP4  EXEC PGM=DREC,PARM='3018,NO'
```

The EXEC statement named STEP4 invokes a program named DREC and passes the value in the PARM parameter to DREC.

***Example 2***

```
//       EXEC PGM=ENTRY,TIME=(2,30)
```

This EXEC statement, which does not have a stepname, invokes a program named ENTRY and specifies the maximum processor time for execution of the step.

***Example 3***

```
//FOR    EXEC PROC=PROC489,ACCT=DB1528,RD.PSTEP2=RNC,DEV=3390
```

The EXEC statement named FOR invokes a cataloged or in-stream procedure named PROC489. The ACCT parameter applies to all steps in the procedure. The RD parameter applies to only the step named PSTEP2. The DEV parameter assigns the value 3350 to the symbolic parameter Device Support Bucket in a procedure statement.

## ACCT parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the ACCT parameter to specify one or more subparameters of accounting information that apply to this step. The system passes the accounting information to the installation's accounting routines.

***References***

For more information on how to add accounting routines, see *z/OS MVS System Management Facilities (SMF)*.

### Syntax

```
ACCT[.procstepname]=(accounting-information)
```

**Single subparameter**: You can omit the parentheses if the accounting information consists of only one subparameter.

**Length**: The entire accounting-information must not exceed 142 characters:

- Including any commas, which are considered part of the information.
- Excluding any enclosing parentheses or apostrophes, which are not considered part of the information.

**Multiple subparameters**: When the accounting-information consists of more than one subparameter, separate the subparameters by commas and enclose the information in parentheses or apostrophes. For example, ACCT=(5438,GROUP6) or ACCT='5438,GROUP6'.

**Special characters**: When a subparameter contains special characters, other than hyphens or plus zero (+0, an overpunch), enclose it in apostrophes and the information in parentheses or enclose all of the information in apostrophes. For example, ACCT=(387,'72/159') or ACCT='387,72/159'.

Code each apostrophe that is part of the accounting-information as two consecutive apostrophes. For example, code DEPT'D58 as ACCT='DEPT''D58'

If you code a symbolic parameter on the ACCT parameter, you can code the symbolic parameter in apostrophes.

**Continuation onto another statement**: Enclose the accounting-information in parentheses. End each statement with a comma after a complete subparameter. For example:

```
//STEP1  EXEC  PGM=WRITER,ACCT=(1417,J318,'D58/920','CHG=2',
//    '33.95')
```

## Subparameter definition

**accounting-information**
Specifies one or more subparameters of accounting information, as defined by the installation.

## On an EXEC statement that calls a procedure

If an EXEC statement calls a cataloged or in-stream procedure, the ACCT parameter overrides the ACCT parameter on or is added to:

- The EXEC statement named in the procstepname qualifier. The information applies only to the named procedure step. The EXEC statement can have as many ACCT.procstepname parameters as the procedure has steps; each ACCT parameter must specify a unique procstepname.
- All EXEC statements in the procedure if procstepname is not coded. Then the information applies to all steps in the called procedure.

## Examples of the ACCT parameter

***Example 1***

```
//STEP1  EXEC PGM=JP5,ACCT=(LOCATION8,'CHGE+3')
```

This EXEC statement executes program JP5 and specifies accounting information for this job step.

***Example 2***

```
//STP3  EXEC PROC=LOOKUP,ACCT=('/83468')
```

This EXEC statement calls cataloged or in-stream procedure LOOKUP. The accounting information applies to this job step, STP3, and to all the steps in procedure LOOKUP.

***Example 3***

```
//STP4  EXEC PROC=BILLING,ACCT.PAID=56370,ACCT.LATE=56470,
//      ACCT.BILL='121+366'
```

This EXEC statement calls cataloged or in-stream procedure BILLING. The statement specifies different accounting information for each of the procedure steps: PAID, LATE, and BILL.

# ADDRSPC parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the ADDRSPC parameter to indicate to the system that the job step requires virtual storage (which is pageable) or central storage (also called real storage, which is nonpageable).

## Syntax

```
ADDRSPC[.procstepname]=   {VIRT}
                          {REAL}
```

## Subparameter definition

**VIRT**
Requests virtual storage. The system **can** page the job step.

**REAL**
Requests central storage (also called real storage). The system **cannot** page the job step and must place the job step in central storage.

## Defaults

If no ADDRSPC parameter is specified, the default is VIRT.

## Overrides

The JOB statement ADDRSPC parameter applies to all steps of the job and overrides any EXEC statement ADDRSPC parameters.

Code EXEC statement ADDRSPC parameters when each job step requires different types of storage. The system uses an EXEC statement ADDRSPC parameter only when no ADDRSPC parameter is on the JOB statement and only during the job step.

## Relationship to the EXEC REGION parameter

***When ADDRSPC=REAL:*** Code a REGION parameter to specify how much central storage the job needs. If you omit the REGION parameter, the system uses the default.

***When ADDRSPC=VIRT or ADDRSPC is omitted:*** Code a REGION parameter to specify how much virtual storage the job needs. If you omit the REGION parameter, the system uses the default.

## On an EXEC statement that calls a procedure

If this EXEC statement calls a cataloged or in-stream procedure, the ADDRSPC parameter overrides the ADDRSPC parameter on or is added to:

- The EXEC statement named in the procstepname qualifier. The parameter applies only to the named procedure step. The EXEC statement can have as many ADDRSPC.procstepname parameters as the procedure has steps; each ADDRSPC parameter must specify a unique procstepname.
- All EXEC statements in the procedure if procstepname is not coded. Then the parameter applies to all steps in the called procedure.

## Examples of the ADDRSPC parameter

***Example 1***

```
//CAC1  EXEC PGM=A,ADDRSPC=VIRT
```

This EXEC statement executes program A and requests virtual (pageable) storage. Because the REGION parameter is not specified, the storage available to this job step is the installation default or the region size specified on the JOB statement.

***Example 2***

```
//CAC2  EXEC PROC=B,ADDRSPC=REAL,REGION=80K
```

This EXEC statement calls procedure B and requests central (nonpageable) storage. The REGION parameter specifies 80K of storage.

# CCSID parameter

***Parameter type:*** Keyword, optional

***Purpose:*** You can request the access method to convert data between the coded character set identifier (CCSID) specified on the JOB or EXEC statement and the CCSID specified on the DD statement. Data conversion is supported on access to ISO/ANSI Version 4 tapes using access methods BSAM or QSAM, but not using EXCP.

ISO/ANSI tapes are identified by the LABEL=(,AL) or LABEL=(,AUL) keyword. The CCSID parameter does not apply to ISO/ANSI Version 1 or ISO/ANSI/FIPS Version 3 tapes or to tapes with labels other than AL or AUL. See *z/OS DFSMS Using Data Sets*for selecting ISO/ANSI Version 4 tapes. It also contains a list of supported CCSIDs.

The CCSID value of 65535 has a special meaning: it suppresses conversion.

When CCSID is not specified at the JOB, EXEC, or DD levels, data passed to BSAM and QSAM is converted to 7-bit ASCII when writing to ISO/ANSI tapes.

This might result in data loss on conversion. On READ operations the CCSID (if recorded) on the tape header label is used for conversion.

The CCSID is recorded in the tape header label if conversion is not defaulted.

## Syntax

```
CCSID= nnnnn
```

## Subparameter definition

**nnnnn**
The CCSID as a decimal number from 1 through 65535.

## Default

If no CCSID parameter is specified on the JOB statement, the default is 500.

## Relationship to other parameters

Do not code the following parameters with the CCSID parameter:

| | | |
|---|---|---|
| * | DDNAME | QNAME |
| BURST | DYNAM | SYSOUT |
| CHARS | FCB | TERM |
| COPIES | FLASH | UCS |
| DATA | MODIFY | |

## Examples of the CCSID parameter

For examples of the CCSID parameter see “Examples of the CCSID parameter” on page 120.

# COND parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the COND parameter to test return codes from previous job steps and determine whether to bypass this job step. You can specify one or more tests on the COND parameter, and you can test return codes from particular job steps or from every job step that has completed processing. If any of the test conditions are satisfied, the system evaluates the COND parameter as true and bypasses the job step. If none of the test conditions specified on the COND parameter are satisfied, the system evaluates the COND parameter as false and executes the job step.

The system performs the COND parameter tests against return codes from the current execution of the job. If a test returns a previously bypassed step, the system evaluates the test as false.

Bypassing a step because of a return code test is not the same as abnormally terminating the step. The system abnormally terminates a step following an error so serious that it prevents successful execution. In contrast, bypassing of a step is merely its omission.

If a step abnormally terminates, the system normally bypasses all following steps in the job unless the step(s) are part of an IF/THEN/ELSE/ENDIF construct that specifies the ABEND, ABENDCC, or ¬ABEND keywords, described in Chapter 18, "IF/THEN/ELSE/ENDIF statement construct," on page 379. Another way to make the system execute a following step, for instance, to write a dump, is to code EVEN or ONLY on that step's EXEC statement. The EVEN or ONLY subparameters are interpreted first. If they indicate that the step should be executed, then the return code tests, if specified, are performed. If no return code tests were typed or if none of the coded tests is satisfied, the system executes the step. Finally, steps following a step that terminated abnormally might execute. This occurs if the step that abended contained a recovery routine like ESPIE, ESTAE or FRR that intercepted the abend and requested that normal termination occur.

Instead of coding a JOB statement COND parameter, code an EXEC statement COND parameter when you want to:

- Specify different tests for each job step.
- Name a specific step whose return code the system is to test.
- Specify special conditions for executing a job step.
- Bypass only one step. When a step is bypassed because of a JOB COND parameter, all following steps in the job are bypassed.

**Note:** Depending on the program invoked, a test showing that a return code from a step is zero is not sufficient to verify that the step did not fail. The system can fail a step (or job) even if the return code is zero. For example, this could happen as a result of specifying CATLG_ERR FAILJOB(YES) and incurring a "post execution error". To determine if a step failed due to a "post execution error", the SMF type 30, sub-type 4 record for the job step can be examined. In this record, bit SMF30SYE in the two-byte SMF30STI field will be on if the job failed due to a "post execution error".

## Syntax

```
COND[.procstepname] = (code,operator)
COND[.procstepname] = ((code,operator[,stepname][.procstepname])
                       [,(code,operator[,stepname][.procstepname])]... [,EVEN])
                                                                       [,ONLY]
COND=EVEN
COND=ONLY
```

- One return code test is: (code,operator)
- You can omit the outer parentheses if you code only one return code test or only EVEN or ONLY.
- Specify up to eight return code tests. However, if you code EVEN or ONLY, specify up to seven return code tests.
- You can omit all return code tests and code only EVEN or ONLY.
- Place the EVEN or ONLY subparameters before, between, or after the return code tests.
- Null positional subparameters of the COND parameter are invalid.

## Subparameter definition

**code**
Specifies a number that the system compares to the return codes from all previous steps in the job or from specific steps. ***code*** is a decimal number from 0 through 4095.

**Note:** Specifying a decimal number greater than 4095 could result in invalid return code testing or invalid return codes in messages.

**operator**
Specifies the type of comparison to be made to the return code. If the specified test is true, the step is bypassed. Use Table 21 on page 347 to select the correct operator. Operators and their meanings are:

| Operator | Meaning |
|---|---|
| GT | Greater than |
| GE | Greater than or equal to |
| EQ | Equal to |
| LT | Less than |
| LE | Less than or equal to |
| NE | Not equal to |

**stepname**
Identifies the EXEC statement of a previous job step that issues the return code to be used in the test. If the specified step is in a procedure, this step must be in the same procedure. Otherwise, the specified step must not be in a procedure; the specified step must contain a PGM keyword, rather than invoke a procedure. Note that if stepnames are not unique within the job, such as when the same procedure is executed multiple times, results might be unpredictable; but in most cases, references to non-unique stepnames will resolve to the first occurrence of that stepname.

If you omit stepname, the code you specify is compared to the return codes from all previous steps. If the return code issued by any of those previous steps causes the test condition to be satisfied, the system evaluates the COND parameter as true and bypasses the job step.

If this step is invoked in JCL that runs as a started task, see "Stepnames for started tasks" on page 334 for information about the stepname the system assigns.

**stepname.procstepname**
Identifies a step in a cataloged or in-stream procedure called by an earlier job step. Stepname identifies the EXEC statement of the calling job step; procstepname identifies the EXEC statement of the procedure step that issues the return code to be used in the test. The step identified by procstepname must contain the PGM keyword, rather than invoke a procedure. Note that if stepnames are not unique within the job, such as when the same procedure is executed multiple times, results might be unpredictable; but in most cases, references to non-unique stepnames will resolve to the first occurrence of that stepname.

**EVEN**
Specifies that this job step is to be executed **even if** a preceding job step abnormally terminated. When EVEN is coded, the system:

- Does not test the return code of any steps that terminated abnormally.
- Does test the return code of any steps that terminated normally. If none of the return code tests for these steps is satisfied, this job step is executed.

See "Considerations when using the COND parameter" on page 346 for cautions related to the use of EVEN.

**ONLY**
Specifies that this job step is to be executed **only if** a preceding step abnormally terminated. When ONLY is coded, the system:

- Does not test the return code of any steps that terminated abnormally.
- Does test the return code of any steps that terminated normally. If none of the return code tests for these steps is satisfied, this job step is executed.

See "Considerations when using the COND parameter" on page 346 for cautions related to the use of ONLY.

## Overrides

If you code the COND parameter on the JOB statement and on one or more of the job's EXEC statements, and if a return code test on the JOB statement is satisfied, the job terminates. In this case, the system does not process any subsequent EXEC statement COND parameters.

If the tests on the JOB statement are not satisfied, the system then performs the return code tests on the EXEC statement. If a return code test is satisfied, the step is bypassed.

## Location in the JCL

You can specify the COND parameter on any EXEC statement in the job. However, the system evaluates a COND parameter on the first EXEC statement in a job as false.

## On an EXEC statement that calls a procedure

The COND parameter on an EXEC statement that calls a cataloged or in-stream procedure (a calling EXEC statement) either overrides or is added to the called EXEC statements.

The COND on the calling EXEC statement overrides the COND on the called EXEC statement. If the called EXEC statement does not have COND coded on it, the COND on the calling EXEC statement will be added to the called EXEC statement.

If an EXEC statement calls a cataloged or in-stream procedure, the COND parameter overrides the COND parameter on or is added to:

- The EXEC statement named in the procstepname qualifier, which is to the left of the equals sign. The parameter applies only to the named procedure step. The EXEC statement can have as many COND.procstepname parameters as the procedure has steps; each COND parameter must specify a unique procstepname.
- All EXEC statements in the procedure if procstepname is not coded. Then the parameter applies to this job step and to all steps in the called procedure.

## Considerations when using the COND parameter

Be aware of the following considerations when specifying COND parameters. Some of these considerations relate to errors that prevent step execution, no matter what is specified on the COND parameter, while others are related to the use of the COND parameter.

### Errors that prevent step execution, regardless of COND specifications

Certain error conditions prevent the system from executing a step, regardless of any requests specified through the COND parameter. These conditions are as follows:

***Abnormal termination by the system:*** After certain types of abnormal termination by the system, remaining job steps are not executed, regardless of whether EVEN or ONLY were specified. The completion codes associated with these types of abnormal termination are:

**122** Operator canceled job

**222** Operator or TSO/E user canceled job

You might encounter other system completion codes for which remaining job steps are not executed, regardless of whether EVEN or ONLY was specified. See *z/OS MVS System Codes* for further information about specific system completion codes.

***Backward references to data sets:*** If a step is bypassed because of its COND parameter or if a step abnormally terminates, a data set that was to have been created or cataloged in the step may not exist, may not be cataloged, or may be incomplete. Thus, a job step should not refer to a data set being created or cataloged in a step that could be bypassed or abnormally terminated. If the job step does make such a reference, the system might not be able to execute the step.

***When the program does not have control:*** For the system to act on the COND parameter, the step must abnormally terminate while the program has control. If a step abnormally terminates during scheduling, due to failures such as JCL errors or the inability to allocate space, the system bypasses the remaining steps, no matter what the COND parameter requests.

### JES3 considerations

In both JES2 and JES3 systems, an EXEC COND parameter determines if a step is executed or bypassed. However, JES3 processes all jobs as though each step will execute; therefore, JES3 allocates devices for steps that are bypassed. JES3 will fail jobs that delete a data set in one step and attempt to reference the deleted data set in a later step, even if the step that deletes the data set is bypassed during execution. JES3 does not support conditional JCL, although it does permit conditional statements to be specified.

### COND parameter on the first statement in a Job

The system evaluates a COND parameter on the first EXEC statement in a job as false.

### JOBLIB with COND=ONLY

If the job contains a JOBLIB DD statement and ONLY is specified in a job step, the JOBLIB unit and volume information are not passed to the next step; when the next step is executed, the system searches the catalog for the JOBLIB data set.

### When the JOB statement contains a RESTART parameter

When restarting a job, the restart step becomes, in effect, the first step in the job. Therefore, the system evaluates a COND parameter on the restart step as false and executes the step. Subsequent steps might be executed. When a COND parameter on a step following the restarted step refers to a step that precedes the restarted step, the system evaluates the COND parameter as false. If all other COND parameters on that step are also false, the system executes the step. When the JOB statement contains a RESTART parameter with a checkpoint id, the system evaluates the COND parameter on the designated restart step as false and executes the step.

| Restarted step | COND parameter processing |
|---|---|
| The restarted step does not call a procedure and is not a step within a procedure. | The system evaluates any COND parameters on the restarted step as false, and executes the step. |
| The restarted step calls a procedure and does not contain a COND parameter. | The system evaluates any COND parameters on the first step to be executed within the procedure at restart as false, and executes the step. Subsequent steps containing COND parameters are processed normally. |
| The restarted step is within a procedure, and the step that called the procedure does not contain a COND parameter. | The system evaluates any COND parameters on the first step to be executed within the procedure at restart as false, and executes the step. Subsequent steps containing COND parameters are processed normally. |
| The restarted step calls a procedure, and the restarted step contains a COND parameter without a procstepname qualifier. | The system evaluates any COND parameters on the restarted step as false, and executes the step. The system evaluates any COND parameters on steps within the called procedure as false, regardless of whether they were overridden or added from the COND parameter on the step that called the procedure. |
| The restarted step is within a procedure, and the step that called the procedure contains a COND parameter without a procstepname qualifier. | The system evaluates any COND parameters on the restarted step as false, and executes the step. Any subsequent steps within the procedure that contain COND parameters are processed normally. |
| The restarted step calls a procedure, and the restarted step contains one or more COND parameters with procstepname qualifiers. | The system evaluates any COND parameters on the first step to be executed within the procedure at restart as false, and executes the step. Subsequent steps that contain COND parameters are processed normally. COND parameters on these subsequent steps are added or overridden as specified in the calling step. |
| The restarted step is within a procedure, and the step that called the procedure contains one of more COND parameters with procstepname qualifiers. | The system evaluates any COND parameters on the first step to be executed within the procedure at restart as false, and executes the step. Subsequent steps that contain COND parameters are processed normally. COND parameters on these subsequent steps are added or overridden as specified in the calling step. |

## Summary of COND parameters

*Table 21. Bypassing or Execution of Current Step Based on COND Parameter*

| Test in COND parameter | Return Code (RC) from a previous step | |
|---|---|---|
| | Execute current step | Bypass current step |
| COND=(code,GT) | code <= RC | code > RC |
| COND=(code,GE) | code < RC | code >= RC |
| COND=(code,EQ) | code ^= RC | code = RC |
| COND=(code,LT) | code >= RC | code < RC |
| COND=(code,LE) | code  RC | code <= RC |

*Table 21. Bypassing or Execution of Current Step Based on COND Parameter (continued)*

| Test in COND parameter | Return Code (RC) from a previous step | |
|---|---|---|
| | **Execute current step** | **Bypass current step** |
| COND=(code,NE) | code = RC | code ^= RC |
| **Note:** When the COND parameter does not name a previous step, the system tests all previous steps. If any test is satisfied, the system bypasses the current step. | | |

*Table 22. Effect of EVEN and ONLY Subparameters on Step Execution*

| EVEN or ONLY Specified? | Any Preceding Abend? | Any Tests Satisfied? | Current Step Execute? |
|---|---|---|---|
| EVEN | No | No | Yes |
| EVEN | No | Yes | No |
| EVEN | Yes | No | Yes |
| EVEN | Yes | Yes | No |
| ONLY | No | No | No |
| ONLY | No | Yes | No |
| ONLY | Yes | No | Yes |
| ONLY | Yes | Yes | No |
| Neither | No | No | Yes |
| Neither | No | Yes | No |
| Neither | Yes | No | No |
| Neither | Yes | Yes | No |

## Examples of the COND parameter

***Example 1:***

```
//STEP6  EXEC PGM=DISKUTIL,COND=(4,GT,STEP3)
```

In this example, if the return code from STEP3 is 0 through 3, the system bypasses STEP6. If the return code is 4 or greater, the system executes STEP6. Because neither EVEN nor ONLY is specified, the system does not execute this step if a preceding step abnormally terminates.

***Example 2:***

```
//TEST2  EXEC PGM=DUMPINT,COND=((16,GE),(90,LE,STEP1),ONLY)
```

The system executes this step ONLY if two conditions are met:

1. A preceding job step abnormally terminated.
2. No return code tests are satisfied.

Therefore, the system executes this step only when all three of the following are true:

- A preceding job step abnormally terminated.
- The return codes from all preceding steps are 17 or greater.
- The return code from STEP1 is 89 or less.

The system bypasses this step if any one of the following is true:

- All preceding job steps terminated normally.
- The return code from any preceding step is 0 through 16.
- The return code from STEP1 is 90 or greater.

***Example 3:***

```
//STEP1  EXEC  PGM=CINDY
          .
          .
//STEP2  EXEC  PGM=NEXT,COND=(4,EQ,STEP1)
          .
          .
//STEP3  EXEC  PGM=LAST,COND=((8,LT,STEP1),(8,GT,STEP2))
          .
```

In this example, if STEP1 returns a code of 4, STEP2 is bypassed. Before STEP3 is executed, the system performs the first return code test. If 8 is less than the return code from STEP1, STEP3 is bypassed; or, restated, if the STEP1 return code is less than or equal to 8, STEP3 is executed. Because 4 is less than 8, STEP3 is executed.

The system does not perform the second return code test because STEP2 was bypassed.

***Example 4:***

```
//STP4  EXEC PROC=BILLING,COND.PAID=((20,LT),EVEN),
//     COND.LATE=(60,GT,FIND),
//     COND.BILL=((20,GE),(30,LT,CHGE))
```

This statement calls cataloged or in-stream procedure BILLING. The statement specifies different return code tests for each of the procedure steps: PAID, LATE, and BILL. The system executes step PAID even if a preceding step abnormally terminates unless the accompanying return code is satisfied.

***Example 5:*** The procedure TEST exists in SYS1.PROCLIB:

```
//TEST      PROC
//PROCSTP1  EXEC PGM=IEFBR14,COND=(0,NE)
//PROCSTP2  EXEC PGM=IEFBR14,COND=(0,NE)
//PROCSTP3  EXEC PGM=IEFBR14
//PROCSTP4  EXEC PGM=IEFBR14,COND=(4,LT)
//          PEND
```

The job:

```
//JOB1      JOB...RESTART=JOBSTEP
//JOBSTEP   EXEC PROC=TEST
```

JOB1 restarts at JOBSTEP. PROCSTP1 is the first step in the job because of the RESTART specification, and the COND parameter test is not valid because no previous steps have run. Therefore, the system evaluates the COND parameter for PROCSTP1 as false, and PROCSTP1 runs. PROCSTP3 has no COND parameter. The COND parameters for PROCSTP2 and PROCSTP4 are used.

The job:

```
//JOB1      JOB...RESTART=JOBSTEP.PROCSTP2
//JOBSTEP   EXEC PROC=TEST,COND=(8,GT)
```

JOB1 restarts at PROCSTP2 as called by JOBSTEP. The COND parameter on JOBSTEP does not specify a procstepname qualifier and therefore applies to all steps in procedure TEST. The system evaluates the COND parameter for PROCSTP2, the restart step, as false, and the step runs. However, the COND parameter for steps PROCSTP3 and PROCSTP4 evaluates as true (because 8 is greater than the return code of 0 provided by all previous steps in the job), and the steps are bypassed.

The job:

```
//JOB1      JOB...RESTART=JOBSTEP.PROCSTP2
//JOBSTEP   EXEC PROC=TEST,COND.PROCSTP4=(8,GT)
```

JOB1 restarts at PROCSTP2 as called by JOBSTEP. Because of the RESTART specification, PROCSTP2 is the first step in the job. The system evaluates the COND parameter for PROCSTP2 as false, and the step runs. PROCSTP3 has no COND parameter. PROCSTP4 is overridden as specified on JOBSTEP.

# DYNAMNBR parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the DYNAMNBR parameter to tell the system to hold a number of resources in anticipation of reuse. Code DYNAMNBR instead of several DD statements with DYNAM parameters.

## Syntax

```
DYNAMNBR[.procstepname]=n
```

## Subparameter definition

**n** Specifies a value used to calculate the maximum number of data set allocations that the system can hold in anticipation of reuse. Specify n as a decimal number from 0 through 3273 minus the number of DD statements in the step.

Note that the limit of 3273 is based on the number of single unit DD statements for a 64K TIOT (task input output table). This limit can be different depending on the installation-defined TIOT size. 32K is the default TIOT size. The limit for a 32K TIOT is 1635. (In a JES3 system, the installation might further reduce the limit.)

**Note:** If you specify DISP=(NEW,PASS) but, at the end of the job, one or more data sets were not received by any job step, then the maximum number of DD statements you can specify decreases by one. For example, if the current limit is 1635 DD statements, you can specify DISP=(NEW,PASS), and up to 1634 DD statements. See the *z/OS MVS Initialization and Tuning Reference*, under ALLOCxx, for more TIOT information.

The number of resources that the system actually holds in anticipation of reuse equals n plus the number of DD statements in the step, including any DD statements in a cataloged or in-stream procedure called by the step. The system uses this sum of n plus the number of DD statements in the step to establish a control limit. See the *z/OS MVS JCL User's Guide* for Dynamic Allocation and Control Limit information. See the *z/OS MVS Programming: Authorized Assembler Services Guide* for additional Control Limit information.

## Defaults

If no DYNAMNBR parameter is specified, the default is 0. If you code DYNAMNBR incorrectly, the system uses the default of 0 and issues a JCL warning message.

## On an EXEC statement that calls a procedure

If this EXEC statement calls a cataloged or in-stream procedure, the DYNAMNBR parameter overrides the DYNAMNBR parameter on or is added to:

- The EXEC statement named in the procstepname qualifier. The parameter applies only to the named procedure step. The EXEC statement can have as many DYNAMNBR.procstepname parameters as the procedure has steps; each DYNAMNBR parameter must specify a unique procstepname.
- All EXEC statements in the procedure if procstepname is not coded. Then the parameter applies to all steps in the called procedure.

## Example of the DYNAMNBR parameter

```
//STEP1  EXEC PROC=ACCT,DYNAMNBR.CALC=12
```

For the procedure step CALC, this statement specifies that the system should hold the following data set allocations for reuse: 12 plus the number of DD statements following this EXEC statement and the number of DD statements in procedure ACCT.

# MEMLIMIT parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the MEMLIMIT parameter to specify the limit on the total size of usable virtual storage above the bar in a single address space.

## Syntax

```
MEMLIMIT={nnnnnM}
         {nnnnnG}
         {nnnnnT}
         {nnnnnP}
         {NOLIMIT}
```

## Subparameter definition

**nnnnnM**
**nnnnnG**
**nnnnnT**
**nnnnnP**

Specifies a value to be used as the limit on the total size of usable virtual storage above the bar in a single address space. The value may be expressed in megabytes (M), gigabytes (G), terabytes (T), or petabytes (P). nnnnn may be a value from 0 to 99999, with a maximum value of 16384P.

**NOLIMIT**
Specifies that there is no limit on the virtual storage to be used above the bar.

**Note:** Unlike the REGION parameter, MEMLIMIT=0M (or equivalent in G, T, or P) means that the step can not use virtual storage above the bar.

## Defaults

If no MEMLIMIT parameter is specified, the default is the value defined to SMF, except when REGION=0K/0M is specified, in which case the default is NOLIMIT.

## Overrides

The JOB statement MEMLIMIT parameter applies to all steps of the job and overrides any EXEC statement MEMLIMIT parameter.

If MEMLIMIT is not specified, SMF provides a default value. The IEFUSI installation exit can override any JCL- or SMF-supplied value.

## Relationship to the REGION parameter

A specification of REGION=0K/0M will result in a MEMLIMIT value being set to NOLIMIT, when a MEMLIMIT value has not been specified on either the JOB or EXEC statements, and IEFUSI has not been used to set the MEMLIMIT.

## Considerations When Using the MEMLIMIT parameter

Specifying a REGION size that gives the job all the available storage, such as 0K or any value greater than 16,384K, can cause storage problems if the IBM- or installation-supplied routine IEALIMIT or IEFUSI is not used to establish a limiting value.

## Example of the MEMLIMIT parameter

```
//STEPA  EXEC PGM=ADDER,MEMLIMIT=10000M
```

This job step specifies a limit of 10000 megabytes of usable virtual storage above the bar, depending on other job and installation factors.

# PARM parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the PARM parameter to pass variable information to the processing program executed by this job step. To use the information, the processing program must contain instructions to retrieve the information.

***References***

For details on the format of the passed information and its retrieval, see *z/OS MVS Programming: Assembler Services Guide*.

## Syntax

```
PARM[.procstepname]=subparameter
PARM[.procstepname]=(subparameter,subparameter)
PARM[.procstepname]=('subparameter',subparameter)
PARM[.procstepname]='subparameter,subparameter'
```

**Length**: The length of the subparameters passed must not exceed 100 characters:

- Including any commas, which are passed to the processing program.
- Excluding any enclosing parentheses or apostrophes, which are not passed.

For example, PARM='P1,123,MT5' is received by the program as P1,123,MT5.

**Commas**: When you code more than one subparameter, separate the subparameters by commas and enclose the subparameters in parentheses or apostrophes. For example, PARM=(P1,123,MT5) or PARM='P1,123,MT5'.

**Special characters and blanks**: When a subparameter contains special characters or blanks, enclose it in apostrophes and the other subparameters in parentheses, or enclose all the subparameters in apostrophes. For example, PARM=(P50,'12+80') or PARM='P50,12+80'.

Code each apostrophe and ampersand that is part of the subparameter as two consecutive apostrophes or ampersands. For example, code 3462&5 as PARM='3462&&5'.

However, if a subparameter contains a symbolic parameter, code a single ampersand. You can code the symbolic parameter in apostrophes.

**Continuation onto another statement**: Enclose the subparameters in parentheses. End each statement with a comma after a subparameter. For example:

```
//STEP1  EXEC    PGM=WORK,PARM=(DECK,LIST,'LINECNT=80',
//    '12+80',NOMAP)
```

Do not code an apostrophe in column 71; see “Continuing parameter fields enclosed in apostrophes” on page 17 if you need more information.

## Subparameter definition

**subparameter**
Consists of the information to be passed to the processing program.

## On an EXEC statement that calls a procedure

If an EXEC statement calls a cataloged or in-stream procedure, the PARM parameter overrides the PARM parameter on or is added to:

- The EXEC statement named in the procstepname qualifier. The information applies only to the named procedure step. The EXEC statement can have as many PARM.procstepname parameters as the procedure has steps; each PARM parameter must specify a unique procstepname.
- The **first** EXEC statement in the procedure if procstepname is not coded; the system **nullifies any PARM parameters on any following EXEC statements in the procedure**. The information applies to only the first step in the called procedure.

## Examples of the PARM parameter

*Example 1*

```
//RUN3  EXEC PGM=APG22,PARM='P1,123,P2=5'
```

The system passes P1,123,P2=5 to the processing program named APG22.

***Example 2***

```
//   EXEC PROC=PROC81,PARM=MT5
```

The system passes MT5 to the first step of the procedure named PROC81. If PROC81 contains more steps and their EXEC statements contain PARM parameters, the system nullifies those PARM parameters.

***Example 3***

```
//STP6  EXEC PROC=ASMFCLG,PARM.LKED=(MAP,LET)
```

The system passes MAP,LET to the procedure step named LKED in procedure ASMFCLG. If any other procedure steps in ASMFCLG contain a PARM parameter, those PARM parameters remain in effect.

***Example 4***

```
//RUN4  EXEC PGM=IFOX00,PARM=(NOOBJECT,'LINECNT=50', 'TRUNC(BIN)',
//   DECK)
```

The system passes NOOBJECT,LINECNT=50,TRUNC(BIN),DECK to processing program IFOX00. Because the PARM parameter contains a list of more than one subparameter, the information is enclosed in parentheses.

## PARMDD parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the PARMDD parameter in conjunction with a DD statement to pass variable information to the processing program executed by this job step. To use the information, the processing program must contain instructions to retrieve the information.

Specifying the PARMDD keyword causes the job to be scheduled on systems that are at or above the z/OS 2.1 level.

The DD statement can be a SYSIN DD (a DD coded as DD * or DD DATA) or it can reference a data set, UNIX System Service file or DD statement in a previous step (via DSN=*.stepname.ddname).

The PARMDD keyword is mutually exclusive with the PARM keyword.

***References***

For details on the format of the passed information and its retrieval, see *z/OS MVS Programming: Assembler Services Guide*.

If the program to be invoked is from an APF authorized library or has been bound with the AC(1) attribute, see *z/OS MVS Program Management: User's Guide and Reference* for a discussion of the Binder LONGPARM option.

## Syntax

```
//STEP1   EXEC PGM=pgm,PARMDD=ddname
```

## Relationship to other control statements

The DD name specified on the PARMDD statement must exist on a DD statement within the step.

## Data set requirements

The data set that is associated with the DDname specified on the PARMDD keyword must be a physical sequential (PS) data set. Other data set organizations are rejected and the job is failed. When using PARMDD, you must comply with the following support statements:

- Partitioned data set (PDS and PDSE) members provide the appearance of a physical sequential (PS) data set and are supported.
- z/OS UNIX System Services files also provide the appearance of a physical sequential (PS) data set and are supported.
- Other data sets that provide the appearance of a physical sequential (PS) data set are not supported.

The data set can have a fixed (F), fixed-block (FB), variable (V) or variable-block (VB) record format (RECFM). Spanned (S) and undefined (U) record formats are rejected and the job is failed.

## Record length requirements

The maximum supported record length (LRECL) is 32760 bytes. Fixed record length data sets and in-stream data sets are examined to determine if their records contain sequence numbers. If sequence numbers exist, they are assumed to consist of eight contiguous numeric characters occupying the last eight bytes of the input record. If a record is found to contain a sequence number, the length of the input record is adjusted to remove the sequence number.

**Note:** Ensure that you do not place numeric data that is intended as parameter data in the final eight characters of fixed length records or in-stream data records.

## Parameter string requirements

A PARMDD parameter string can be formed by concatenating multiple data sets or files, subject to the constraints of the BSAM like data set concatenation rules.

The parameter string that is passed to the job step program is formed by a simple concatenation of each input record, up to a maximum of 32760 bytes. Blank records and trailing blanks on each record are ignored during the concatenation process. Input in excess of 32760 bytes (after any symbolic substitution, sequence number removal, and trailing blank removal) results in an error message written to the job log and the job being terminated. An input record can contain blank characters that are to become part of the parameter string, but you must end any sequence of blank characters on an input record with a non-blank character, or the blank characters will be ignored.

After concatenation, the parameter string that is passed to the job step program is examined for double ampersand character (&&) sequences. Double ampersands are

converted to single ampersands in the same way that double ampersands are converted to single ampersands by PARM= processing.

When the PARMDD= parameter references an in-stream (SYSIN) data set, the DD statement can use the SYMBOLS= parameter, and the data can contain symbols if the symbol name is exported (See "SYMBOLS parameter" on page 280.) (Note that substitution logging can be requested using the SYMBOLS parameter, but will be ignored.) In the following example, the parameter string that is presented to the program MYPGM is SBJ.DASD.LOAD:

```
//       EXPORT SYMLIST=SYMB1
//    SET SYMB1=DASD
//STEP1   EXEC PGM=MYPGM,PARMDD=MYPARMS
//MYPARMS DD   *,SYMBOLS=JCLONLY,DLM=$$
SBJ.&SYMB1..LOAD
```

If the job runs in a job class with the SYSSYM setting enabled, it can also use system symbols within the SYSIN data set.

Parameter strings that contain ampersand (&) characters are examined for symbol names. If there is no valid symbol name after the ampersand (&) character, the string is left unchanged, with the following exception: parameter strings that contain double ampersand characters (&&) within the string are converted to single ampersand characters, as they are done for the PARM= parameter string.

## Examples of the PARMDD parameter

***Example 1***

```
//STEP1   EXEC PGM=IEBCOPY,PARMDD=PARMIN
//PARMIN  DD   *,DLM=/*
LINECOUNT=75
/*
```

In the example, the PARMDD keyword specifies a DD name of PARMIN, which is then coded on a DD statement that specifies a SYSIN (or in-stream) data set. The DD DATA usage would be similar.

***Example 2***

```
//STEP1   EXEC PGM=MYPGM,PARMDD=MYPARMS
//MYPARMS DD   DSN=SYS1.PARMLIB(MYPGMPRM)
```

In the example, the PARMDD keyword specifies a DD name of MYPARMS, which is then coded on a DD statement that specifies a data set (in this case, a partitioned data set member) that contains the program's parameter information.

***Example 3***

```
//STEP1   EXEC PGM=MYPGM,PARMDD=MYPARMS
//MYPARMS DD   PATH=/SYSTEM/tmp/unixparm.txt
```

In the example, the PARMDD keyword specifies a DD name of MYPARMS, which is then coded on a DD statement that specifies a UNIX System Service file that contains the program's parameter information.

# PERFORM parameter

***Parameter type***

Keyword, optional

*Purpose*

**Important**

Beginning with z/OS V1R3, WLM compatibility mode is no longer available. Accordingly, the information that pertains specifically to WLM compatibility mode is no longer valid. It has been included for reference purposes, and for use on backlevel systems.

Use the PERFORM parameter in WLM compatibility mode to specify the performance group for the job step. The installation-defined performance groups determine the rate at which associated steps have access to the processor, storage, and channels.

In WLM goal mode, any PERFORM parameter on an EXEC statement for a job or a started procedure is ignored. However, for a TSO session, a PERFORM parameter specified on the EXEC statement of the TSO logon procedure, or entered on the TSO logon panel, can be used for classification of the session to a service class or report class. For details on how to use workload management classification rules to map a PERFORM value to a service class or report class, see *z/OS MVS Planning: Workload Management*.

## Syntax

```
PERFORM[.procstepname]=n
```

## Subparameter definition

**n** The n is a number from 1 through 999.

In WLM compatibility mode, n identifies a performance group that has been defined by your installation. The specified performance group should be appropriate for your step type according to your installation's rules.

## Defaults

In WLM compatibility mode, if no PERFORM parameter is specified or if the specified PERFORM number fails validity checks, the system uses an installation default specified at initialization. If the installation did not specify a default, the system uses a built-in default:

| Default | Use |
|---|---|
| 1 | For non-TSO/E job steps |
| 2 | For TSO/E sessions |

See *z/OS MVS Initialization and Tuning Guide* for details.

## Overrides

A JOB statement PERFORM parameter applies to all steps of the job and overrides any EXEC statement PERFORM parameters.

Code EXEC statement PERFORM parameters when each job step is to execute in a different performance group. The system uses an EXEC PERFORM parameter only when no PERFORM parameter is on the JOB statement and only during the job step.

## On an EXEC statement that calls a procedure

If an EXEC statement calls a cataloged or in-stream procedure, the PERFORM parameter overrides the PERFORM parameter on or is added to:

- The EXEC statement named in the procstepname qualifier. The parameter applies only to the named procedure step. The EXEC statement can have as many PERFORM.procstepname parameters as the procedure has steps; each PERFORM parameter must specify a unique procstepname.
- All EXEC statements in the procedure if procstepname is not coded. Then the parameter applies to all steps in the called procedure.

## Example of the PERFORM parameter

```
//STEPA  EXEC PGM=ADDER,PERFORM=60
```

This job step will be run in performance group 60 if it passes validity checks. The installation must have defined the significance of this performance group.

# PGM parameter

***Parameter type***

Positional, optional

***Purpose***

Use the PGM parameter to name the program that the system is to execute. The specified program must be a member of a partitioned data set (PDS) or partitioned data set extended (PDSE) used as a system library, a private library, or a temporary library.

## Syntax

```
PGM= {program-name                  }
     {*.stepname.ddname             }
     {*.stepname.procstepname.ddname}
     {JCLTEST                       }
     {JSTTEST                       }
```

The EXEC statement parameter field must begin with a PGM parameter or a PROC parameter. These two parameters must not appear on the same EXEC statement.

## Subparameter definition

**program-name**

Specifies the member name or alias of the program to be executed. The program-name is 1 through 8 alphanumeric or national ($, #, @) characters; the first character must be alphabetic or national.

Use this form of the parameter when the program resides in a system library, such as SYS1.LINKLIB, or in a private library specified in the job by a JOBLIB DD statement or in the step by a STEPLIB DD statement.

***.stepname.ddname**
Refers to a DD statement that defines, as a member of a partitioned data set (PDS) or a partitioned data set extended (PDSE), the program to be executed. Stepname identifies the EXEC statement of the earlier job step that contains the DD statement with ddname in its name field.

Use this form of the parameter when a previous job step creates a temporary library to store a program until it is required.

When referring to a DD statement, the system does not honor requests for special program properties as defined in the program properties table (PPT). (See *z/OS MVS Initialization and Tuning Reference*.)

***.stepname.procstepname.ddname**
Refers to a DD statement that defines, as a member of a partitioned data set (PDS) or a partitioned data set extended (PDSE), the program to be executed. The DD statement is in a cataloged or in-stream procedure that is called by an earlier job step. Stepname identifies the EXEC statement of the calling job step; procstepname identifies the EXEC statement of the procedure step that contains the DD statement with ddname in its name field.

Use this form of the parameter when a previous job step calls a procedure that creates a temporary library to store a program until it is required.

When referring to a DD statement, the system does not honor requests for special program properties as defined in the program properties table (PPT). (See *z/OS MVS Initialization and Tuning Reference*.)

**JCLTEST (JES3 only)**
**JSTTEST (JES3 only)**
Requests that the system scan the step's job control statements for syntax errors without executing the job or allocating devices. JCLTEST or JSTTEST provides for a step the same function as provided by the JOB statement TYPRUN=SCAN parameter for a job. See *z/OS JES3 Commands* for details.

**Note:** JCLTEST and JSTTEST are supported only in JES3 systems.

## Examples of the PGM parameter

***Example 1***

```
//JOB8   JOB  ,BOB,MSGLEVEL=(2,0)
//JOBLIB DD   DSNAME=DEPT12.LIB4,DISP=(OLD,PASS)
//STEP1  EXEC PGM=USCAN
```

These statements indicate that the system is to search the private library DEPT12.LIB4 for the member named USCAN, read the member into storage, and execute the member.

***Example 2***

```
//PROCESS JOB  ,MARY,MSGCLASS=A
//CREATE  EXEC PGM=IEWL
//SYSLMOD DD   DSNAME=&&PARTDS(PROG),UNIT=3390,DISP=(MOD,PASS),
//             SPACE=(1024,(50,20,1))
//GO      EXEC PGM=*.CREATE.SYSLMOD
```

The EXEC statement named GO contains a backward reference to DD statement SYSLMOD, which defines a library created in the step named CREATE. Program PROG is a member of the partitioned data set &&PARTDS, which is a temporary data set. Step GO executes program PROG. The data set &&PARTDS is deleted at the end of the job.

*Example 3*

```
//JOBC   JOB  ,JOHN,MSGCLASS=H
//STEP2  EXEC PGM=UPDT
//DDA    DD   DSNAME=SYS1.LINKLIB(P40),DISP=OLD
//STEP3  EXEC PGM=*.STEP2.DDA
```

The EXEC statement named STEP3 contains a backward reference to DD statement DDA, which defines system library SYS1.LINKLIB. Program P40 is a member of SYS1.LINKLIB; STEP3 executes program P40.

# PROC and procedure name parameters

***Parameter type***

Positional, optional

***Purpose***

Use the PROC parameter to specify that the system is to call and execute a cataloged or in-stream procedure.

## Syntax

```
{PROC=procedure-name}
{procedure-name     }
```

- The EXEC statement parameter field must begin with a PGM parameter or a PROC parameter. These two parameters must not appear on the same EXEC statement.
- You can omit PROC= and code only the procedure-name.

## Subparameter definition

**procedure-name**

Identifies the procedure to be called and executed:

- The member name or alias of a cataloged procedure.
- The name on the PROC statement that begins an in-stream procedure. The in-stream procedure must appear earlier in this job.

The procedure-name is 1 through 8 alphanumeric or national ($, #, @) characters; the first character must be alphabetic or national.

## Effect of PROC parameter on other parameters and following statements

Because this EXEC statement calls a cataloged or in-stream procedure, the other parameters on the statement are added to or override corresponding parameters on the EXEC statements in the called procedure. See the descriptions of the other parameters for details of their effects.

Any DD statements following this EXEC statement are added to the procedure, or override or nullify corresponding DD statements in the procedure.

## Examples of the PROC parameter

*Example 1*

```
//SP3  EXEC PROC=PAYWKRS
```

This statement calls the cataloged or in-stream procedure named PAYWKRS.

***Example 2***

```
//BK  EXEC OPERATE
```

This statement calls the cataloged or in-stream procedure named OPERATE.

## RD parameter

***Parameter Type***

Keyword, optional

***Purpose***

Use the RD (restart definition) parameter to:

- Specify that the system is to allow the operator the option of performing automatic step or checkpoint restart if a job step abends with a restartable abend code. (See the SCHEDxx parmlib member description in *z/OS MVS Initialization and Tuning Guide* for information about restartable abends.)
- Allow JES to perform automatic step restart after a system failure even if the journal option is not specified in the JES initialization parameters or JES control statements.
- Suppress, partially or totally, the action of the assembler language CHKPT macro instruction or the DD statement CHKPT parameter.

The system can perform automatic restart only if all of the following are true:

- The JOB or EXEC statement contains RD=R or RD=RNC.
- The step to be restarted abended with a restartable abend code.
- The operator authorizes a restart.

The system can perform automatic step restart for a job running during a system failure as long as the job has a job journal.

A job journal is a sequential data set that contains job-related control blocks needed for restart. If you use the automatic restart manager (ARM) to restart a job, you do not need to save the journal because ARM does not use the job journal when restarting jobs.

For JES2, specify a job journal by one of the following:

- JOURNAL=YES on the CLASS statement in the JES2 initialization parameters.
- RD=R or RD=RNC on either the JOB statement or any one EXEC statement in the job.

For JES3, specify a job journal in one of the following:

- JOURNAL=YES on the CLASS statement in the JES3 initialization parameters.
- RD=R or RD=RNC on either the JOB statement or any one EXEC statement in the job.
- JOURNAL=YES on a JES3 //*MAIN statement in the job.

***References***

For detailed information on deferred checkpoint restart, see *z/OS DFSMSdfp Checkpoint/Restart*.

***Considerations for an APPC scheduling environment***

The RD parameter has no function in an APPC scheduling environment. If you code RD, the system will check it for syntax and ignore it.

## Syntax

```
RD[.procstepname]=  {R  }
                    {RNC}
                    {NR }
                    {NC }
```

## Subparameter definition

**R (Restart, Checkpoints Allowed)**
Indicates that the operator can perform automatic step restart if the job step fails.

RD=R does not suppress checkpoint restarts:

- If the processing program executed in a job step does not include a CHKPT macro instruction, RD=R allows the system to restart execution at the beginning of the abnormally terminated step.
- If the program includes a CHKPT macro instruction, RD=R allows the system to restart execution at the beginning of the step, if the step abnormally terminates before the CHKPT macro instruction is executed.
- If the step abnormally terminates after the CHKPT macro instruction is executed, only checkpoint restart can occur. If you cancel the affects of the CHKPT macro instruction before the system performs a checkpoint restart, the request for automatic step restart is again in effect.

**RNC (Restart, No Checkpoints)**
Indicates that the operator can perform automatic step restart if the job step fails.

RD=RNC suppresses automatic and deferred checkpoint restarts. It suppresses:

- Any CHKPT macro instruction in the processing program: That is, the operator cannot perform an automatic checkpoint restart, and the system is not to perform a deferred checkpoint restart if the job is resubmitted.
- The DD statement CHKPT parameter.
- The checkpoint at end-of-volume (EOV) facility.

**NR (No Automatic Restart, Checkpoints Allowed)**
Indicates that the operator **cannot** perform automatic step restart if the job fails.

RD=NR suppresses automatic checkpoint restart but permits deferred checkpoint restarts. It permits:

- A CHKPT macro instruction to establish a checkpoint.
- The job to be resubmitted for restart at the checkpoint. On the JOB statement when resubmitting the job, specify the checkpoint in the RESTART parameter.

If you code RD=NR and the system fails, RD=NR does not prevent the job from restarting.

**NC (No Automatic Restart, No Checkpoints)**
Indicates that the operator **cannot** perform automatic step restart if the job step fails.

RD=NC suppresses automatic and deferred checkpoint restarts. It suppresses:

- Any CHKPT macro instruction in the processing program.
- The DD statement CHKPT parameter.
- The checkpoint at EOV facility.

## Defaults

If you do not code the RD parameter, the system uses the installation default from the job's job class specified at initialization.

## Overrides

- A JOB statement RD parameter applies to all steps of the job and overrides any EXEC statement RD parameters.

  When no RD parameter is on the JOB statement, the system uses an EXEC statement RD parameter, but only during the job step. Code EXEC statement RD parameters when you want to specify different restart types for each job step.
- A request by a CHKPT macro instruction for an automatic checkpoint restart overrides a request by a JOB or EXEC statement RD=R parameter for automatic step restart.

## Relationship to other control statements

Code RD=NC or RD=RNC to suppress the action of the DD statement CHKPT parameter.

## On an EXEC statement that calls a procedure

If an EXEC statement calls a cataloged or in-stream procedure, the RD parameter is added to or overrides the RD parameter on:

- The EXEC statement named in the procstepname qualifier. The information applies only to the named procedure step. The EXEC statement can have as many RD.procstepname parameters as the procedure has steps; each RD parameter must specify a unique procstepname.
- All EXEC statements in the procedure if procstepname is not coded. Then the parameter applies to all steps in the called procedure.

## Examples of the RD parameter

***Example 1***

```
//STEP1  EXEC PGM=GIIM,RD=R
```

RD=R specifies that the operator can perform automatic step restart if the job step fails.

***Example 2***

```
//NEST  EXEC PGM=T18,RD=RNC
```

RD=RNC specifies that, if the step fails, the operator can perform automatic step restart. RD=RNC suppresses automatic and deferred checkpoint restarts.

*Example 3*

```
//CARD  EXEC PGM=WTE,RD=NR
```

RD=NR specifies that the operator cannot perform automatic step restart or automatic checkpoint restart. However, a CHKPT macro instruction can establish checkpoints to be used later for a deferred restart.

*Example 4*

```
//STP4  EXEC PROC=BILLING,RD.PAID=NC,RD.BILL=NR
```

This statement calls a cataloged or in-stream procedure BILLING. The statement specifies different restart requests for each of the procedure steps: PAID and BILL.

# REGION parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the REGION parameter to specify the amount of central or virtual storage that the step requires.

The amount of storage that you request must be enough to accommodate the following:

- Storage for all programs in the step to execute.
- All additional storage that the programs in the step request with GETMAIN, STORAGE, and CPOOL macro instructions.
- Enough unallocated storage for task initialization and termination. Task initialization and termination can issue GETMAIN macro instructions for storage in the user's address space.

Two installation exits, IEFUSI and IEALIMIT, can also affect the size of the user address space assigned to the job step.

***References***

For more information on address space size, see "Resource Control of Address Space" in *z/OS MVS JCL User's Guide*. For more information on region size with checkpoint/restart jobs, see *z/OS DFSMSdfp Checkpoint/Restart*.

## Syntax

```
REGION[.procstepname]=  {valueK}
                        {valueM}
```

## Subparameter definition

**valueK**
Specifies the required storage in kilobytes (1 kilobyte = 1024 bytes). The value is 1 through 7 decimal numbers, from 1 through 2096128. Code a multiple of 4. For example, code REGION=68K. If the value you code is not a multiple of 4, the system rounds it up to the next multiple of 4.

**valueM**
Specifies the required storage in megabytes (1 megabyte = 1024 kilobytes). The value is 1 through 4 decimal numbers, from 1 through 2047. For example, REGION=3M.

**value=0M or 0K**
A value equal to 0K or 0M gives the step all the storage available below and above 16 megabytes. The resulting size of the region below and above 16 megabytes depends on system options and what system software is installed. When REGION=0K/0M is specified, the MEMLIMIT is set to NOLIMIT.

**Note:** This might cause storage problems. See the Considerations When Using the REGION parameter section for more information.

## Defaults

If no REGION parameter is specified, the system uses an installation default specified at JES initialization.

If your installation does not change the IBM-supplied default limits in the IEALIMIT or IEFUSI exit routine modules, or by using the SMFLIM rules, then specifying various values for the region size has the following results:

- A value equal to 0K or 0M - gives the job step all the storage available below and above 16 megabytes. The resulting size of the region below and above 16 megabytes depends on system options and what system software is installed. When REGION=0K/0M is specified, the MEMLIMIT is set to NOLIMIT.

  **Note:** This might cause storage problems. See the Considerations When Using the REGION parameter information for more information.
- A value greater than 0K or 0M and less than or equal to 16,384K or 16M - establishes the size of the private area below 16 megabytes. If the region size specified is not available below 16 megabytes, the job step abnormally ends with an ABEND822. The extended region size is the default value of 32 megabytes.
- A value greater than 16,384K or 16M and less than or equal to 32,768K or 32M - gives the job step all the storage available below 16 megabytes. The resulting size of the region below 16 megabytes depends on system options and what system software is installed. The extended region size is the default value of 32 megabytes.
- A value greater than 32,768K or 32M and less than or equal to 2,096,128K or 2047M - gives the job step all the storage available below 16 megabytes. The resulting size of the region below 16 megabytes depends on system options and what system software is installed. The extended region size is the specified value. If the region size specified is not available above 16 megabytes, the job step receives whatever storage is available above 16 megabytes, up to the requested amount, and the resulting size of the region above 16 megabytes depends on system options and what system software is installed.

## Overrides

A JOB statement REGION parameter applies to all steps of the job and overrides any EXEC statement REGION parameters.

When no REGION parameter is on the JOB statement, the system uses an EXEC statement REGION parameter, but only during the job step. Code EXEC statement REGION parameters when you want to specify a different region size for each job step.

## Relationship to the EXEC ADDRSPC parameter

*When ADDRSPC=REAL:* Code a REGION parameter to specify how much central storage (also called real storage) the step needs.

*When ADDRSPC=VIRT or ADDRSPC is Omitted:* Code a REGION parameter to specify how much virtual storage the step needs.

## On an EXEC statement that calls a procedure

If an EXEC statement calls a cataloged or in-stream procedure, the REGION parameter is added to or overrides the REGION parameter on:

- The EXEC statement named in the procstepname qualifier. The information applies only to the named procedure step. The EXEC statement can have as many REGION.procstepname parameters as the procedure has steps; each REGION parameter must specify a unique procstepname.
- All EXEC statements in the procedure if procstepname is not coded. Then the parameter applies to all steps in the called procedure.

## Relationship to the MEMLIMIT parameter

A specification of REGION=0K/0M will result in a MEMLIMIT value being set to NOLIMIT, when a MEMLIMIT value has not been specified on either the JOB or EXEC statements, and IEFUSI has not been used to set the MEMLIMIT.

## Relationship to the REGIONX parameter

REGION and REGIONX are mutually exclusive keywords within the same job. If REGION is specified on the JOB or any EXEC statement in a job, REGIONX must not also be specified on the JOB or any EXEC statement within the same job.

## Considerations when using the REGION parameter

Specifying a REGION size that gives the job all of the available storage, such as 0K or any value greater than 16,384K, can cause storage problems if the IBM- or installation-supplied routine IEALIMIT, or IEFUSI, or SMFLIMrules is not used to establish a limiting value.

## Examples of the REGION parameter

**Example 1**:

```
//MKBOYLE EXEC PROC=A,ADDRSPC=REAL,REGION=40K
```

The system assigns 40K bytes of central (real) storage to this job step.

**Example 2**:

```
//STP6  EXEC PGM=CONT,REGION=120K
```

The system assigns a region of 120K bytes. When the ADDRSPC parameter is not specified, the system defaults to ADDRSPC=VIRT.

---

# REGIONX parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the REGIONX parameter to specify the amount of central or virtual storage that the step requires below and above the 16 MB line.

The amount of storage that you request must be enough to accommodate the following:

- Storage for all programs in the step to run.
- All additional storage that the programs in the step request with GETMAIN, STORAGE, and CPOOL macro instructions.
- Enough unallocated storage for task initialization and termination. Task initialization and termination can issue GETMAIN macro instructions for storage in the user's address space.

## Syntax

```
REGIONX[.procstepname]=  {value1}
                         {([value1][,value2])}
```

## Subparameter definition

**value1**
Specifies the amount of memory to be assigned below the 16 MB line.

**value2**
Specifies the amount of memory to be assigned above the 16 MB line, but below 2 GB (that is, “below the bar”.)

Values for REGIONX are defined with *nnnnnM* for megabytes or *nnnnnG* for gigabytes.

The installation might reduce the amount of memory to be assigned through the MEMLIMIT keywords in the SMFPRM*xx* parmlib member, and/or by way of the IEFUSI installation exit or keywords in the SMFLIM*xx* parmlib member.

## Defaults

If no REGIONX value is specified on the EXEC statement, but REGIONX is specified on the JOB statement, the JOB REGIONX specification is used for each step in the job. Otherwise, system defined storage settings are used.

If, however, a REGIONX keyword is specified but with null values, the system uses the following rules for each value:

- First value: The system uses the JOB REGIONX first value, the REGION value from the JOBCLASS (if less than 16M) or 0M, in that order. When 0M is used, this implies that all below the line private storage is available to the program.
- Second value: The system uses the JOB REGIONX second value, the REGION value from the JOBCLASS (if greater than 16M) or 128M, in that order. If the program needs access to all available above the line private storage, a value of 0M is to be explicitly coded for the second value.

The installation might reduce these numbers through the MEMLIMIT keyword in the SMFPRM*xx* parmlib member, by way of the IEFUSI installation exit or by keywords in the SMFLIM*xx* parmlib member.

## Overrides

Unlike the REGION keyword, a JOB statement REGIONX parameter is used as a default for any step of the job that does NOT have a REGIONX keyword on the EXEC statement. You can set the region default and only use REGIONX as an override on the EXEC statements that require different region values.

### JCL procedure overrides:

Because REGIONX is mutually exclusive with REGION,

REGIONX can replace REGION, and

REGION can replace REGIONX

during EXEC statement procedure override processing.

This might result in mutually exclusive conflicts with a REGION= or REGIONX= specification on the JOB statement.

Examine the JCL output listing for uses of REGION and REGIONX.

### START command invocation

For a started job, a REGION= or REGIONX= specification on the START command can replace a REGION= or REGIONX= specification on the JOB statement.

For a started procedure, a REGION= or REGIONX= specification on the START command can replace a REGION= specification on the JOB statement that is generated internally by START command processing.

This might result in mutually exclusive conflicts with a REGION= or REGIONX= specification on the EXEC statement of the procedure.

Examine the JCL output listing for uses of REGION and REGIONX with the started job or started procedure.

## Relationship to the EXEC ADDRSPC parameter

***When ADDRSPC=REAL:*** Code a REGIONX parameter to specify how much central storage (also called real storage) the step needs.

***When ADDRSPC=VIRT or ADDRSPC is Omitted:*** Code a REGIONX parameter to specify how much virtual storage the step needs.

## On an EXEC statement that calls a procedure

The REGIONX keyword is mutually exclusive with the ADDRSPC keyword. If you need to use the ADDRSPC keyword, use REGION instead.

## Relationship to the MEMLIMIT parameter

A specification of REGIONX=(0M,0M) results in a MEMLIMIT value being set to NOLIMIT, when a MEMLIMIT value has not been specified on either the JOB or EXEC statements, and IEFUSI and SMFLIM have not been used to set the MEMLIMIT.

## Relationship to the REGION parameter

REGION and REGIONX are mutually exclusive keywords within the same job. If REGION is specified on the JOB or any EXEC statement in a job, REGIONX must not also be specified on the JOB or any EXEC statement within the same job.

## Examples of the REGIONX parameter

```
//REGNX001  JOB MSGLEVEL=1
//STEP001  EXEC PGM=ZTT,REGIONX=(512K,1G)
//ZTTOUT   DD SYSOUT=*,DCB=(LRECL=133,BLKSIZE=133,RECFM=FBA)
//ZTTIN    DD *
/*
```

To request 1 gigabyte above the line and 500 KB below the line for REGIONX values, specify:

```
REGIONX=(500 K,1G)
```

# RLSTMOUT parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the RLSTMOUT parameter to specify the maximum time in seconds that a VSAM RLS or DFSMStvs request is to wait for a required lock before the request is assumed to be in deadlock and ended with VSAM return code 8 and reason code 22(X'16'). Specify RLSTMOUT as a value in seconds in the range of 0 to 9999. A value of 0 means that the VSAM RLS or DFSMStvs request has no timeout value; the request waits as long as necessary to obtain the required lock.

The value does not apply to promote requests. A promote request is a request to change shared access to exclusive access without releasing the lock

VSAM RLS detects deadlocks within VSAM and DFSMStvs. It cannot detect deadlocks across other resource managers, and uses the timeout value to determine when such deadlocks might have occurred.

In addition to specifying RLSTMOUT on the JCL step level, you can also specify a global timeout value in the IGDSMSxx member of SYS1.PARMLIB, or on the RPL passed for each VSAM request. For a particular VSAM RLS or DFSMStvs request, the value used for time out is in the following order:

1. The value specified in the RPL, if any
2. The value specified in JCL at the step level, if any
3. The value specified in the IGDSMS*xx* member of SYS1.PARMLIB, if any

CICS specifies the timeout value in the VSAM RPL. The value used is specified in the CICS System Initialization Table or the transaction definition.

## Syntax

```
//[stepname]  EXEC
positional-parm[,RLSTMOUT={nnn|0}]
```

## Defaults

If you do not code the RLSTMOUT parameter, the value defaults to the RPL value or to the value specified in PARMLIB.

## Examples of the RLSTMOUT parameter

*Example 1*

```
//STEP04  EXEC PGM=VALKYRIE,RLSTMOUT=0
```

RLSTMOUT specifies that the VSAM RLS or DFSMStvs request has no timeout value. The request waits as long as necessary to obtain the required lock.

# TIME parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the TIME parameter to specify the maximum amount of time that a job step may use the processor or to find out through messages how much processor time a step used.

You can use the TIME parameter on an EXEC statement to increase or decrease the amount of processor time available to a job step over the default value.

A step that exceeds its allotted time abnormally terminates and causes the job to terminate, unless an installation exit routine extends the time for the job. The exit routine IEFUTL is established through System Management Facilities (SMF).

***References***

See "TIME parameter" on page 459 (the TIME parameter on the JOB statement) or *z/OS MVS Installation Exits*.

## Syntax

```
TIME[.procstepname]=   {([minutes][,seconds])}
                       {1440                 }
                       {NOLIMIT              }
                       {MAXIMUM              }
                       {0                    }
```

You can omit the parentheses if you code only 1440, 0, or the processor time in minutes.

## Subparameter definition

**minutes**
Specifies the maximum number of minutes the step can use the processor. Minutes must be a number from 0 through 357912 (248.55 days).

**seconds**
Specifies the maximum number of seconds that the step can use the processor, in addition to any minutes that are specified. Seconds must be a number from 0 through 59.

**1440** *or* **NOLIMIT**
Indicates that the step can use the processor for an unlimited amount of time. ("1440" literally means "24 hours.")

Also code TIME=1440 or TIME=NOLIMIT to specify that the system is to allow this step to remain in a continuous wait state for more than the installation time limit, which is established through SMF. "Continuous wait time" is defined as time spent waiting while the application program is in control. For example, the time required to recall a data set from HSM Migration Levels 1 or 2 and/or the time required to mount a tape is counted towards the job's continuous wait time if the allocation of the data set was dynamic (that is, issued while the program was running) while the time required for those activities will not be counted towards the job's continuous wait time if the allocation was static (that is, for a DD statement).

**MAXIMUM**
Indicates that the step can use the processor for the maximum amount of time. Coding TIME=MAXIMUM allows the step to run for 357912 minutes.

**0** Indicates that the step is to use the time remaining from the previous step. If the step exceeds the remaining time available, the step abnormally terminates.

## Defaults

Each job step has a time limit. If you do not specify a TIME parameter on the JOB statement, the time limit for any job step is:

- The value you specify for the TIME parameter on its EXEC statement, or
- The default time limit (that is, the JES default job step time limit), if you do not specify a TIME parameter on its EXEC statement.

## Overrides

If you specify either MAXIMUM or a value in minutes or seconds other than 1440 for the JOB statement TIME parameter, the system can reduce the processor time available to a job step. In those two cases, the system sets the time limit for the step to the smaller of the two following values:

- The job time remaining after all previous job steps have completed.
- The time limit that was specified or the default time limit.

See “Defaults” for an explanation of default time limits.

## On an EXEC statement that calls a procedure

If an EXEC statement calls a cataloged or in-stream procedure, the TIME parameter is added to or overrides the TIME parameter on:

- The EXEC statement named in the procstepname qualifier. The information applies only to the named procedure step. The EXEC statement can have as many TIME.procstepname parameters as the procedure has steps; each TIME parameter must specify a unique procstepname.

If procstepname is not coded, the TIME parameter applies to the entire procedure and nullifies any TIME parameters on EXEC statements in the procedure. For example, suppose you specify TIME=5 on an EXEC statement that calls a procedure. The first step in the procedure is allowed 5 minutes, the second step is allowed 5 minutes minus the time used by the first step, the third step is allowed 5 minutes minus the time used by the first and second steps, and so forth, regardless of any TIME parameter values on EXEC statements in the procedure.

TIME=1440 and TIME=NOLIMIT also nullify any TIME parameters on EXEC statements in the procedure. Specifying TIME=1440 or TIME=NOLIMIT on the calling EXEC statement allows the procedure to have unlimited processor time.

## Examples of the TIME parameter

For examples of TIME coded on both the JOB and EXEC statements, see "Examples of the TIME parameter on JOB and EXEC statements" on page 461.

***Example 1***

```
//STEP1  EXEC PGM=GRYS,TIME=(12,10)
```

This statement specifies that the maximum amount of time the step can use the processor is 12 minutes, 10 seconds.

***Example 2***

```
//FOUR  EXEC PGM=JPLUS,TIME=(,30)
```

This statement specifies that the maximum amount of time the step can use the processor is 30 seconds.

***Example 3***

```
//INT  EXEC PGM=CALC,TIME=5
```

This statement specifies that the maximum amount of time the step can use the processor is 5 minutes.

***Example 4***

```
//LONG  EXEC PGM=INVANL,TIME=NOLIMIT
```

This statement specifies that the step can have unlimited use of the processor. Therefore, the step can use the processor and can remain in a wait state for an unspecified period of time, if not restricted by the JOB statement TIME parameter.

***Example 5***

```
//STP4  EXEC PROC=BILLING,TIME.PAID=(45,30),TIME.BILL=(112,59)
```

This statement calls cataloged or in-stream procedure BILLING. The statement specifies different time limits for each of the procedure steps: PAID and BILL.

***Example 6***

```
//STP6  EXEC  PGM=TIMECARD,TIME=MAXIMUM
```

This statement specifies that the step can use the processor for 357912 minutes, if not restricted by the JOB statement TIME parameter.

***Example 7***

```
//TEST1 JOB MSGLEVEL=(1,1)
//STEP1 EXEC PGM=USES40,TIME=(,50)
//STEP2 EXEC PGM=USESREST,TIME=0
```

STEP1 can use the processor for 50 seconds. If STEP1 actually uses the processor for only 40 seconds, STEP2 can use the processor for 10 seconds, because that is the time remaining from the previous step.

***Example 8***

```
//TEST1 JOB MSGLEVEL=(1,1),TIME=(,50)
//STEP1 EXEC PGM=USES15,TIME=(,25)
//STEP2 EXEC PGM=USES30,TIME=(,40)
//STEP3 EXEC PGM=USESREST,TIME=0
```

STEP1 can use the processor for 25 seconds. If STEP1 actually uses the processor for only 15 seconds, the time limit for STEP2 is the smaller of the following values:

- The job time remaining (35 seconds)
- The time limit specified on the EXEC statement for STEP2 (40 seconds).

In this case, the job time remaining is the smaller value, so STEP2 can use the processor for 35 seconds. If STEP2, then, actually uses the processor for only 30 seconds, STEP3 can use the processor for 5 seconds, because that is the time remaining from the previous step.

***Example 9***

```
//TEST2 JOB MSGLEVEL=(1,1),TIME=8,CLASS=5
//STEP1 EXEC PGM=USES4
//STEP2 EXEC PGM=USESREST
```

Assume that the default time limit for class 5 is 5 minutes. The time limit for STEP1 is 5 minutes (the default). If STEP1 actually uses the processor for 4 minutes, the time limit for STEP2 is the smaller of the following values:

- The job time remaining (4 minutes)
- The default time limit (5 minutes).

In this case, the job time remaining is the smaller value, so STEP2 can use the processor for 4 minutes.

# Chapter 17. EXPORT statement

***Purpose:*** Use the EXPORT statement to make specific JCL symbols available to the job step program. Exported JCL symbols can be accessed during the job execution phase using the JCL Symbol Service (IEFSJSYM) or the JES Symbol Service (IAZSYMBL). Symbols must be set to a value subsequent to the EXPORT statement for the symbol value to be exported.

**Note:** Do not use the EXPORT statement to override the symbol names on the SYMLIST on an EXPORT statement that is inside a procedure.

***References:*** IEFSJSYM is documented in *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG*. IAZSYMBL is documented in *z/OS JES Application Programming*.

## Description

The following **EXPORT** statement syntax is required:

### Syntax

| //[*label*] EXPORT [*parameter*,...]...*comments* |
|---|
| The EXPORT statement consists of the characters // in columns 1 and 2 followed by a label, operation (EXPORT), parameters, and optional comments. |

### Label field

The Label field is optional, but can be included for readability. When coded, the following rules apply:

- The *label* must begin in column three.
- The *label* must be 1–8 alphanumeric and national ($, #, @) characters.
- The first character must be alphabetic or national ($, #, @).
- The *label* must be followed by at least one blank space.

### Operation field

The operation field consists of the characters EXPORT and must be preceded and followed by at least one blank space.

### Parameter field

The EXPORT statement, as coded in the JCL, can only contain the SYMLIST parameter. However, other versions of the EXPORT statement will be generated by the system and will appear in the job log. The parameters that are seen on the generated version of the EXPORT statement are not allowed to be coded in the job stream JCL.

© Copyright IBM Corp. 1988, 2015

*Table 23. SYMLIST keyword parameter on the EXPORT statement.* The table describes the SYMLIST keyword parameter on the EXPORT statement.

| KEYWORD PARAMETERS | VALUES | PURPOSE |
|---|---|---|
| SYMLIST=(*symbolic parameter*, *symbolic parameter*...) | Up to 128 comma-separated symbol names. Enclosing parentheses or apostrophes characters are required when specifying multiple symbol names. | Names the symbols to be exported. |
| SYMLIST=* | The asterisk character (*), which specifies to export all symbol names. | Specifies to export all symbols. |

## Comments field

The optional comments field must follow the list of symbol names or the asterisk character and at least one intervening blank space.

## Location in the JCL

An EXPORT statement can be located anywhere in the JCL after the JOB statement.

# SYMLIST parameter

***Parameter type:*** Keyword on **EXPORT** statement, required

***Purpose:*** Use the **SYMLIST** parameter to list the JCL symbolic parameters to be exported, and therefore, made available to the job step program. Symbolic parameters can be set to values in the JCL with the SET statement or through PROC symbolic parameter processing. Exported symbolic parameters must be set to a value at a point in time in the job stream after the EXPORT command, and prior to the execution of the job step program. Symbolic parameters can be set to values in the JCL with the SET statement or through PROC symbolic parameter processing. Exported symbolic parameter values can also be passed into in-stream (sysin) data; see "Using symbols in JES in-stream data" on page 54 for details.

Exported symbolic parameters must be set to a value in the job stream after the **EXPORT** statement. Exported symbol values are resolved to the last value set prior to or within the current job step. JCL Converter processing generates **EXPORT EXPSET** statements to manage how exported JCL symbol values are resolved. These statements appear in the job log. Reviewing the placement of **EXPORT EXPSET** statements in the job log might be helpful in understanding exported symbol resolution for a given job.

## Syntax

```
SYMLIST=(symbolic parameter,symbolic parameter...)

or
SYMLIST=*
```

**Single subparameter**: You can omit the parentheses if you are exporting only one symbol.

**Length**: The entire symbol string must not exceed 142 characters. The total character count:

- Includes any commas, which are considered part of the information.
- Excludes any enclosing parentheses or apostrophes, which are not considered part of the information.

**Multiple subparameters**: When exporting more than one symbol, you must separate the symbols by commas and enclose the information within parentheses or apostrophes. For example, SYMLIST=(SYM86,SYM87) or SYMLIST='SYM86,SYM87'. Duplicate SYMLIST parameters are accepted but ignored.

For details on coding a symbol name, refer to "Defining and nullifying JCL symbols" on page 39.

## Subparameter definition

**Symbolic parameters**
Identifies one or more symbols to export.

## Examples

1. In this example, the symbol parameters COUNTY, TOWN, and STATE are set by the SET statement. The EXPORT statement indicates that the symbolic parameters COUNTY and STATE are to be made available to the MYPROG program that is executed in STEP1:

```
//MYEXP  EXPORT SYMLIST=(COUNTY,STATE)
//STEP1 SET COUNTY=DUTCHESS,TOWN=FISHKILL,STATE=NY
//STEP1 EXEC PGM=MYPROG
```

2. Exported symbolic parameters are resolved to the most recent value to which they are set. In the following example, MYPROG1 in STEP1 receives an exported value of SYMVAL1 for SYM1. Subsequently, the program MYPROG1 in STEP2 receives an exported value of NEWSYMVAL for SYM1. In STEP3, the exported value for SYM3 is null, because its value was not set before MYPROG1 executing in STEP3. In STEP4 and STEP5, MYPROG1 receives the exported value of SYMVAL3 for SYM3.

   The value of SYMVAL2, for symbol SYM2, is made available to all of the job steps following the export statement labeled MYEXPR1. The value of SYMVAL1, for symbol SYM1, is made available to STEP1. The updated value of NEWSYMVAL, for symbol SYM1, is made available to STEP2 and the remaining job steps:

```
//MYEXPR1 EXPORT SYMLIST=(SYM1,SYM2)
//        SET    SYM1=SYMVAL1,SYM2=SYMVAL2
//STEP1   EXEC   PGM=MYPROG1
//STEP2   EXEC   PGM=MYPROG1
//        SET    SYM1=NEWSYMVAL
//MYEXPR2 EXPORT SYMLIST=SYM3
//STEP3   EXEC   PGM=MYPROG1
//STEP4   EXEC   PGM=MYPROG1
//        SET    SYM3=SYMVAL3
//STEP5   EXEC   PGM=MYPROG1
```

3. This example shows exported symbol resolution when symbols are used inside procedures:

```
_//         EXPORT SYMLIST=*
_//PROC2    PROC WHAT=SPOCK
_//         SET WHO=LEONARD
_//P2STEP1  EXEC PGM=IEBGENER
_//SYSIN    DD DUMMY
```

```
//SYSPRINT DD SYSOUT=*
//SYSUT2   DD SYSOUT=*
//SYSUT1   DD *,SYMBOLS=(JCLONLY)
WHO = &WHO
WHAT = &WHAT
//PROC2    PEND
//PROC1    PROC
//P1STEP1  EXEC PGM=IEBGENER
//SYSIN    DD DUMMY
//SYSPRINT DD SYSOUT=*
//SYSUT2   DD SYSOUT=*
//SYSUT1   DD *,SYMBOLS=(JCLONLY)
WHO = &WHO
//P1STEP2  EXEC PROC2,WHAT=ROCK
//         SET WHAT=SCISSORS
//         SET WHO=SHELDON
//PROC1    PEND
//         SET WHO=HOWARD
//JSTEP1   EXEC PROC1
//         SET WHO=PENNY
//         SET WHAT=PAPER
//JSTEP2   EXEC PROC1
//JSTEP3   EXEC PROC1
//         SET WHO=AMY
//JSTEP4   EXEC PROC2,WHAT=LIZARD
//JSTEP5   EXEC PROC2
```

Exported symbolics for this job resolve for each step as follows:

```
Step Name  Proc Step   WHO        WHAT
---------  ---------   ---        ----
JSTEP1     P1STEP1     HOWARD
P1STEP2    P2STEP1     PENNY      PAPER
JSTEP2     P1STEP1     PENNY
P1STEP2    P2STEP1     SHELDON    SCISSORS
JSTEP3     P1STEP1     PENNY
P1STEP2    P2STEP1     AMY        SCISSORS
JSTEP4     P2STEP1     LEONARD    LIZARD
JSTEP5     P2STEP1     LEONARD    SPOCK
```

# Chapter 18. IF/THEN/ELSE/ENDIF statement construct

This topic describes the IF/THEN, ELSE, and ENDIF statements, collectively called the IF/THEN/ELSE/ENDIF statement construct.

***Purpose:*** Use the IF/THEN/ELSE/ENDIF statement construct to conditionally execute job steps within a job.

The IF statement is always followed by a relational-expression and a THEN clause. Optionally, an ELSE clause can follow the THEN clause. An ENDIF statement always follows the ELSE clause, if present, or the THEN clause.

- The THEN clause specifies the job steps that the system processes when the evaluation of the relational-expression for the IF statement is a true condition. The system evaluates the relational-expression at execution time.
- The ELSE clause specifies the job steps that the system processes when the evaluation of the relational-expression for the IF statement is a false condition.
- The ENDIF statement indicates the end of the IF/THEN/ELSE/ENDIF statement construct, and must be coded for each construct.

You can nest IF/THEN/ELSE/ENDIF statement constructs up to a maximum of 15 levels. The steps that execute in a THEN clause and an ELSE clause can be another IF/THEN/ELSE/ENDIF statement construct.

## Description

### Syntax

```
//[name] IF  [(]relational-expression[)] THEN   [comments]
    .
    .    action when relational-expression is true
    .
//[name] ELSE   [comments]
    .
    .    action when relational-expression is false
    .
//[name] ENDIF   [comments]
```

The IF statement consists of the characters // in columns 1 and 2 and the five fields: name, operation (IF), the relational-expression, the characters THEN, and comments. The relational-expression can be enclosed in parentheses.

The ELSE statement consists of the characters // in columns 1 and 2 and the three fields: name, operation (ELSE), and comments.

The ENDIF statement consists of the characters // in columns 1 and 2 and the three fields: name, operation (ENDIF), and comments.

### Name field

A name is optional on IF/THEN, ELSE, and ENDIF statements. If used, code it as follows:

- The name should be unique within the job.
- The name must begin in column 3.

© Copyright IBM Corp. 1988, 2015

- The name is 1 through 8 alphanumeric or national ($, #, @) characters.
- The first character must be alphabetic or national ($, #, @).
- The name must be followed by at least one blank.
- The name may be preceded by up to 8 alphanumeric or national characters, and then separated by a period. Coding the name in this way should not be confused with specifying an override, as can be done when coding DD statements.

If a name is not coded, column 3 must be blank.

## Operation field

The operation field consists of the characters IF, ELSE, or ENDIF and must be preceded and followed by at least one blank. It can begin in any column.

## Relational-expression field

The relational-expression field follows the IF operation field after at least one intervening blank and is followed by at least one blank before the characters THEN. For example, to test that a return code is greater than 4, code:

```
//   IF  RC > 4  THEN
```

You can enclose the relational-expression in parentheses. For example:

```
//   IF  (RC > 4)  THEN
```

A relational-expression indicates the condition that the system evaluates. The result of the evaluation of the relational-expression always depends on two factors: the operation specified, and the values of the operands or expressions that are compared at execution time. The result of evaluating a relational-expression is either true or false.

If you specify a stepname as part of a relational-expression, the system first determines whether the step executed. If the step did not execute, the evaluation of the relational-expression is false.

***Continuing a relational expression***

You can continue relational-expressions on the next JCL statement. Break the relational-expression where a blank is valid on the current statement, and continue the expression beginning in column 4 through 16 of the next statement. Do not put comments on the statement that you are continuing. You can code comments after you have completed the statement. For example:

```
//TESTCON  IF (RC = 8 | RC = 10 | RC = 12 |
//            RC = 14)  THEN         COMMENTS OK HERE
            .
            .
```

A relational-expression consists of:

- Comparison operators
- Logical operators
- NOT (¬) operators
- Relational-expression keywords
- Numeric values

## Priorities of operators

The operators that you can use in a relational-expression and their processing priority are shown in Figure 1.

The system evaluates operators in the order indicated. Code operators with the same priority in the order in which you want the system to evaluate them.

You can specify either the alphabetic characters or the special characters for an operator. For example, GT and > have the same meaning. (RC GT 10) and (RC > 10) are the same.

```
                                                Order of
Operator        Operation                       Evaluation
--------        ---------                       ----------

NOT operator:

NOT or  ¬       NOT                             first

Comparison operators:

GT  or  >       Greater than                    second
LT  or  <       Less than                       second
NG  or  ¬>      Not greater than                second
NL  or  ¬<      Not less than                   second
EQ  or  =       Equal to                        second
NE  or  ¬=      Not equal to                    second
GE  or  >=      Greater than or equal to        second
LE  or  <=      Less than or equal to           second

Logical operators:

AND or  &       AND                             third
OR  or  |       OR                              third
```

*Figure 1. Operators on IF/THEN/ELSE/ENDIF Statement Construct*

## Comparison operators

Use comparison operators in a relational-expression to compare a keyword with a numeric value. The comparison results in a true or false condition.

For example, to test for a return code of 8, code:

```
//TESTA  IF  (RC = 8)  THEN
```

In the example, if a return code is 8, the expression is true; otherwise, the expression is false.

Blanks are not required to precede and follow special character comparison operators (such as > or ¬=). However, it is good practice to use blanks so your code is easier to read. Blanks are required to precede and follow alphabetic comparison operators (such as GT or EQ). Precede and follow the special character & with at least one blank so that it is not confused with symbolic parameters.

## Logical operators

Use the & (AND) and | (OR) logical operators in a complex relational-expression to indicate that the Boolean result of two or more relational-expressions is to be evaluated.

You must precede and follow the & (AND) and | (OR) operators with at least one blank.

The & (AND) operator indicates that all of the specified expressions must be true. For example, to test that a return code is both greater than 8 and less than 24 (in the range 9 through 23), code:

```
//TESTAND  IF  (RC > 8 & RC < 24)  THEN
```

The | (OR) operator specifies that only one of the expressions need be true. For example, to test that a return code is either equal to 8, equal to 10, or greater than 24, code:

```
//TESTOR  IF  (RC = 8 | RC = 10 | RC > 24)  THEN
```

## NOT operator

Use the ¬ (NOT) operator to reverse the testing of relational-expressions.

For example, the statements TESTNOTA and TESTNOTB make the same test. The relational expression is true when the return code is between 0 and 8:

```
//TESTNOTA  IF  ¬(RC > 8)  THEN
//TESTNOTB  IF  (RC <= 8)  THEN
```

The statements TESTNOTC and TESTNOTD make the same test; the relational expression is true when the return code is 0, 1, 2, 3, 4, 5, 6, 7, 9, or 10.

```
//TESTNOTC  IF  ¬(RC = 8 | RC > 10)  THEN
//TESTNOTD  IF  (RC ¬= 8 & RC <= 10)  THEN
```

Note that the use of the ¬ operator reverses both the logical and comparison operators.

You do not need to code a blank between the ¬ operator and the expression it is reversing.

## Relational-expression keywords

The following keywords are the only keywords supported by IBM and recommended for use in relational-expressions. Any other keywords, even if accepted by the system, are not intended or supported keywords.

**Keyword**
**Use**

**RC** indicates a return code

**ABEND**
indicates an abend condition occurred

**¬ABEND**
indicates no abend condition occurred

**ABENDCC**
indicates a system or user completion code

**RUN** indicates that the specified step started execution

**¬RUN** indicates that the specified step did not start execution

Descriptions of the keywords follow:

**RC**

Indicates that the relational-expression tests a return code. Evaluate a return code by coding RC, a comparison operator, and a numeric value. For example, the expression (RC = 8) tests for a return code equal to 8, and (RC >= 10) tests for a return code greater than or equal to 10.

The return code must be within the range of 0 - 4095.

If you omit stepname, RC refers to the highest job step return code that occurred during job processing prior to the time of evaluation. This applies only to steps that execute. Any step that did not start execution, is cancelled, or abnormally ends is not evaluated.

**Note:** At the start of execution, RC is initially set to zero.

**stepname.RC**
Indicates that the relational-expression tests a return code for a specific step (stepname) of the job.

**stepname.procstepname.RC**
Indicates that the relational-expression tests a return code for a specific step (stepname) and procedure step (procstepname) of the job.

**ABEND**
**ABEND=TRUE**
Indicates that the relational-expression tests for an abend condition that occurred during processing of the job prior to the time of evaluation. The statement IF ABEND THEN tests true when an abend occurred on any previous job step.

If stepname is omitted, ABEND and ABEND=TRUE refer to all previous steps.

Certain types of abnormal termination by the system prevent the execution of the THEN or ELSE clauses of an IF/THEN/ELSE/ENDIF statement construct, regardless of any tests for abnormal termination conditions. See "Errors that prevent execution, regardless of if statement tests" on page 387 for further information.

**stepname.ABEND**
**stepname.ABEND=TRUE**
Indicates that the relational-expression tests for an abend that occurred on a specific step (stepname) of the job.

**stepname.procstepname.ABEND**
**stepname.procstepname.ABEND=TRUE**
Indicates that the relational-expression tests for an abend that occurred on a specific step (stepname) and procedure step (procstepname) of the job.

**¬ABEND**
**ABEND=FALSE**
Indicates that the relational-expression tests that an abend condition did not occur during the processing of the job prior to the time of evaluation. The statement IF ¬ABEND THEN tests true when no abend occurred on any previous job step.

If stepname is omitted, ¬ABEND and ABEND=FALSE refer to all previous steps.

Certain types of abnormal termination by the system prevent the execution of the THEN or ELSE clauses of an IF/THEN/ELSE/ENDIF statement construct, regardless of any tests for abnormal termination conditions. See "Errors that prevent execution, regardless of if statement tests" on page 387 for further information.

**¬stepname.ABEND**
**stepname.ABEND=FALSE**
Indicates that the relational-expression tests that no abend occurred on a specific step (stepname) of the job.

**¬stepname.procstepname.ABEND**
**stepname.procstepname.ABEND=FALSE**
Indicates that the relational-expression tests that no abend occurred on a specific step (stepname) and procedure step (procstepname) of the job.

**ABENDCC=Sxxx**
**ABENDCC=Uxxxx**
Indicates that the relational-expression tests for a system abend completion code (Sxxx) or user-defined abend completion code (Uxxxx). Specify S with a hexadecimal value (3 characters) for system abend codes, and U with a decimal value (4 digits) for user abend codes. For example, ABENDCC=S0C4 tests for system abend code 0C4, and ABENDCC=U0100 tests for user abend code 0100.

If stepname is omitted, ABEND=Sxxx and ABENDCC=Uxxxx refer to the most recent abend code that occurred during the execution of the job prior to the time of evaluation.

Certain types of abnormal termination by the system prevent the execution of the THEN or ELSE clauses of an IF/THEN/ELSE/ENDIF statement construct, regardless of any tests for abnormal termination completion codes. See "Errors that prevent execution, regardless of if statement tests" on page 387 for further information.

**stepname.ABENDCC=Sxxx**
**stepname.ABENDCC=Uxxxx**
Indicates that the relational-expression tests the abend code for a specific step (stepname) of the job.

**stepname.procstepname.ABENDCC=Sxxx**
**stepname.procstepname.ABENDCC=Uxxxx**
Indicates that the relational-expression tests the abend code for a specific step (stepname) and procedure step (procstepname) of the job.

**stepname.RUN**
**stepname.RUN=TRUE**
Indicates that the relational expression tests that a specific job step (stepname) started execution.

**stepname.procstepname.RUN**
**stepname.procstepname.RUN=TRUE**
Indicates that the relational expression tests that a specific job step (stepname) and procedure step (procstepname) started execution.

**¬stepname.RUN**
**stepname.RUN=FALSE**
Indicates that the relational expression tests that a specific job step (stepname) did not start execution.

**¬stepname.procstepname.RUN**
**stepname.procstepname.RUN=FALSE**
Indicates that the relational expression tests that a specific job step (stepname) and procedure step (procstepname) did not start execution.

## Specification of step names in relational expression keywords

If you specify stepname.keyword, where keyword is any of the relational expression keywords, stepname must identify a step containing the EXEC PGM keyword rather than one that invokes a procedure. If you specify stepname.procstepname.keyword, procstepname must identify a step containing the PGM keyword. In this case, stepname identifies the EXEC statement that invokes the procedure in which procstepname appears. Note that if stepnames are

not unique within the job, such as when the same procedure is executed multiple times, results might be unpredictable; but in most cases, references to non-unique stepnames will resolve to the first occurrence of that stepname.

When you specify a step name as part of a relational expression keyword, the system tests whether the specified step started executing. If the step started executing, the system performs the test indicated by the relational expression. If the step did not start executing, the system evaluates that part of the expression as false.

You must always specify a step name when using the RUN relational-expression keywords to determine if a step or procedure step executed. For more information about step names in relational expression keywords, see *z/OS MVS JCL User's Guide*.

### Use of parentheses with relational expressions

The system evaluates relational-expressions that are enclosed within parentheses prior to expressions found outside of parentheses. Therefore, you can control the way in which complex relational-expressions are evaluated.

For example, code the following to test that a return code is 0, 1, 2, or 3:

```
//TESTPAR  IF  (RC LT 4 & (RC LT 12 | RC = 16))  THEN
```

By keeping the same expressions but changing the position of the parentheses, you can test that a return code is 0, 1, 2, 3 or 16:

```
//TESTPAR1  IF  ((RC LT 4 & RC LT 12) | RC = 16)  THEN
```

## Comments field

The comments field follows THEN, ELSE, and ENDIF after at least one intervening blank.

## Location in the JCL

An IF/THEN/ELSE/ENDIF statement construct can appear anywhere in the job. However, an IF statement specified before the first EXEC statement in a job is not evaluated before the first step executes. If the IF statement applies to later steps in the job, the statement will be evaluated when the system decides whether to process the later steps.

## Relationship to other parameters

When you specify both an IF/THEN/ELSE/ENDIF statement construct and a COND parameter for an EXEC statement, the system executes the job step represented by the EXEC statement only when both the IF/THEN/ELSE/ENDIF statement construct and the COND parameter evaluate to execute.

### Defaults

By default, job steps within the IF/THEN/ELSE/ENDIF statement construct do not execute when

- An abend occurred, and
- the IF/THEN/ELSE/ENDIF structure containing the job steps does not specify the ABEND, ABENDCC, or ¬ABEND keyword. If any of these keywords is specified (with or without stepname or procstepname), the job steps do execute despite the abend.
- The step's COND parameter, if any, does not specify an abend condition (COND=EVEN or COND=ONLY).

## THEN and ELSE clauses

A THEN clause consists of the JCL statements between the IF/THEN statement and, if specified, its matching ELSE statement; otherwise, its matching ENDIF statement. If you do not specify any statements, it is a null THEN clause.

An ELSE clause consists of the JCL statements between the ELSE statement and its matching ENDIF statement. If you do not specify any statements, it is a null ELSE clause.

In an IF/THEN/ELSE/ENDIF statement construct, the THEN clause or the ELSE clause must contain at least one EXEC statement to identify a job step.

The system executes the following statements conditionally, in either the THEN clause or the ELSE clause of an IF/THEN/ELSE/ENDIF statement construct. Execution of the statement depends on the evaluation of the relational-expression at execution time:

- Nested IF/THEN/ELSE/ENDIF statement constructs
- EXEC statements
- DD (including DD * and DD DATA) statements
- STEPLIB DD statements
- SYSABEND, SYSMDUMP, and SYSUDUMP DD statements
- SYSCHK (step level) and SYSCKEOV DD statements
- SYSIN DD statements
- OUTPUT JCL statements
- CNTL and ENDCNTL statements

Do not place the following statements in a THEN or ELSE clause:

- JOB statement
- JCLLIB statement
- JOBLIB statement
- SYSCHK (job level) statement
- XMIT JCL statement

The system processes the following statements regardless of the logic of the IF/THEN/ELSE/ENDIF statement construct. They can be placed in a THEN or ELSE clause, but they are not executed conditionally.

- PROC and PEND statements
- JES2 and JES3 statements and commands
- JCL command statements
- Comment (//*) statements
- INCLUDE statements
- Delimiter (/*) statements
- Null statements
- SET statements

## Considerations when using the IF/THEN/ELSE/ENDIF construct

Be aware of the following considerations when using the IF/THEN/ELSE/ENDIF statement construct:

- The IF/THEN/ELSE/ENDIF statement construct does not conditionally control the processing of JCL; rather, it conditionally controls the execution of job steps.
- The result of processing an IF/THEN/ELSE/ENDIF statement construct, once determined, remains unchanged regardless of the outcome from running any remaining steps in a job. The system does not reexamine the original condition at any later job step termination, either normal or abnormal. See Example 9.
- The system allocates all DD statements defined to a step if the execution time evaluation of the relational-expression determines that a step is to be executed.
- All data sets defined on DD statements in the job must be available at the time the job is selected for execution.
- You can nest IF/THEN/ELSE/ENDIF statement constructs up to a maximum of 15 levels.
- You can specify symbolic parameters on IF/THEN/ELSE/ENDIF statements provided that they resolve to one of the supported relational-expression keywords. Any other symbolic parameters, even if accepted by the system, are not intended or supported. Refer to "Relational-expression keywords" on page 382.
- An IF statement specified before the first EXEC statement in a job is not evaluated before the first step executes. If the IF statement applies to later steps in the job, the statement will be evaluated when the system decides whether to process the later steps.
- When you specify an IF statement before the first EXEC statement in a job and the job contains a JOBLIB DD statement, the maximum limit for the number of steps in the job is 254 steps.

There are additional considerations related to errors that prevent execution of the THEN or ELSE clause, no matter what is specified on the IF statement, and there are special considerations related to restarted jobs.

## Errors that prevent execution, regardless of if statement tests

Certain error conditions prevent the system from executing the THEN or ELSE clauses of an IF/THEN/ELSE/ENDIF statement construct. When such an error condition occurs, the system does not execute the THEN or ELSE clause, regardless of any tests on the IF statement. These conditions are as follows:

***Abnormal termination by the system:*** After certain types of abnormal termination by the system, remaining job steps are not executed, regardless of any tests for abnormal termination conditions. The completion codes associated with these types of abnormal termination are:

**122** Operator canceled job

**222** Operator or TSO/E user canceled job

You might encounter other system completion codes for which the THEN or ELSE clause is not executed, regardless of any tests for abnormal termination conditions. See *z/OS MVS System Codes* for further information about specific system completion codes.

***When job time expires:*** The system abnormally terminates processing if a step has exceeded the time limit for the job. The specification of the IF/THEN/ELSE/ENDIF construct has no effect on this type of abnormal termination.

***When a referenced data set is not complete:*** When a job step that contains the IF/THEN/ELSE/ENDIF statement construct references a data set that was to be created or cataloged in a preceding step, the data set

- Will not exist if the step creating it was bypassed, or
- May be incomplete if the step creating it abnormally terminated.

As a result, the system may be unable to execute the step.

***When the program does not have control:*** For the system to act on the IF/THEN/ELSE/ENDIF statement construct, the step must abnormally terminate while the program has control. If a step abnormally terminates during scheduling, (due to failures such as JCL errors or the inability to allocate space), the system bypasses the remaining steps. The steps specified by the IF/THEN/ELSE/ENDIF statement construct do not execute.

### Considerations for restarted jobs

There are four types of restarts:

- Automatic step restart
- Automatic checkpoint restart
- Deferred step restart
- Deferred checkpoint restart

Only the automatic restarts retain the information (step completion codes) necessary to perform valid evaluations of any relational expressions based on prior steps.

If you plan to use either type of deferred restart, you should keep certain points in mind when coding the JCL for the job. Planning ahead in this manner can help prevent the need to update the JCL when the job is submitted for restart. The points to consider are the following:

- Relational expressions on IF/THEN statements that refer to a step preceding the restarted step are evaluated as false.
- Relational expressions on IF/THEN statements on steps following the restarted step can still refer to these following steps, but should also check to see whether the referenced steps actually ran during this invocation. The default value for relational expressions on IF/THEN statements is false, which, unlike COND, will cause the system to skip steps. Adding a ¬STEP.RUN condition is recommended. See “Example 7” on page 391 for an example of a statement construct with a deferred checkpoint restart.

## Examples of IF/THEN/ELSE/ENDIF statement constructs

### Example 1

The following example shows the use of the alphabetic characters rather than special characters for comparison operators.

```
//IFBAD     IF  (ABEND | STEP1.RC > 8) THEN
    or
//IFBAD     IF  (ABEND OR STEP1.RC GT 8) THEN
    .
    .
//IFTEST2   IF  (RC > 4 & RC < 8) THEN
    or
//IFTEST2   IF  (RC GT 4 AND RC LT 8) THEN
```

## Example 2

The following example shows a simple IF/THEN/ELSE/ENDIF statement construct without an ELSE statement.

```
//JOBA      JOB   ...
//STEP1     EXEC  PGM=RTN
              .
              .
//IFBAD     IF  (ABEND | STEP1.RC > 8) THEN
//TRUE      EXEC  PROC=ERROR
//IFBADEND  ENDIF
//NEXTSTEP  EXEC  PROC=CONTINUE
```

The IF statement named IFBAD invokes procedure ERROR if either an abend has occurred on a previous step of the job, or STEP1 has returned a return code that is greater than 8. Otherwise, step TRUE is bypassed and the system processes step NEXTSTEP.

## Example 3

The following example shows a simple IF/THEN/ELSE/ENDIF statement construct with a null ELSE clause.

```
//JOBB      JOB   ...
//STEP1     EXEC  PGM=RTN
              .
              .
//IFBAD     IF  (ABEND | STEP1.RC > 8) THEN
//TRUE      EXEC  PROC=ERROR
//          ELSE
//IFBADEND  ENDIF
//NEXTSTEP  EXEC  PROC=CONTINUE
```

The IF statement named IFBAD invokes procedure ERROR if either an abend has occurred on a previous step of the job, or STEP1 has returned a return code that is greater than 8. Otherwise, the system bypasses step TRUE, and the null ELSE clause passes to NEXTSTEP.

## Example 4

The following example shows a simple IF/THEN/ELSE/ENDIF statement construct with an ELSE clause.

```
//JOBC      JOB    ...
//STEP0     EXEC   PGM=RTN1
              .
              .
//IFTEST2   IF  (RC > 4 & RC < 8) THEN
//*                *** WARNING CONDITION REPORTING GROUP ***
//STEP1     EXEC   PGM=IEFBR14
//REPORT    EXEC   PROC=REPTRTN
//*                *** WARNING CONDITION REPORTING GROUP END ***
//          ELSE
//ERRORSTP  EXEC   PROC=ERRORTN
//ENDTEST2  ENDIF
//NEXTSTEP  EXEC   PROC=CONTINUE
```

Processing for this IF/THEN/ELSE/ENDIF statement construct is:

1. If the relational-expression for the IF/THEN statement construct named IFTEST2 is true (the highest step return code for the job is greater than 4 and less than 8 at the point when this statement is being processed), the system processes the THEN clause. The system executes program IEFBR14 and procedure REPTRTN on EXEC statements STEP1 and REPORT.

2. Otherwise, the relational-expression for IFTEST2 is false and the system processes the ELSE clause (procedure ERRORTN on EXEC statement ERRORSTP).
3. Processing then continues with procedure CONTINUE on step NEXTSTEP.

## Example 5

The following example shows nested IF/THEN/ELSE/ENDIF statement constructs with ELSE clauses. The nested statements are indented so that they are easier to read.

```
//JOBD      JOB   ...
//PROC1     PROC
//PSTEPONE  EXEC  PGM=...
//PSTEP11   EXEC  PGM=...
//PSTEP12   EXEC  PGM=...
//          PEND
//PROC2     PROC
//PSTEPTWO  EXEC  PGM=...
//          PEND
//EXP1      EXEC  PROC=PROC1
//EXP2      EXEC  PROC=PROC2
//IFTEST3   IF  (RC > 12) THEN
//STEP1BAD    IF  (EXP1.PSTEP11.RC > 12 OR EXP1.PSTEP12.RC > 12) THEN
//STEP1ERR    EXEC  PGM=ERRTN,PARM=(EXP1)
//            ELSE
//STEP2ERR    EXEC  PGM=ERRTN,PARM=(EXP2)
//END1BAD     ENDIF
//          ELSE
//NOPROB    EXEC  PROC=RUNOK
//ENDTEST3  ENDIF
//NEXTSTEP  EXEC  ...
```

Processing for the IF/THEN/ELSE/ENDIF construct named IFTEST3 is:

1. If the relational-expression for IFTEST3 is true (the highest step return code for the job is greater than 12 at the point where this statement is being processed), the system processes the THEN clause of IFTEST3. It evaluates the relational-expression of the IF/THEN/ELSE/ENDIF construct named STEP1BAD.
2. If the STEP1BAD relational-expression is true (the return code is greater than 12 for either of the two steps in procedure PROC1, which is invoked by step EXP1), the system processes the THEN clause of STEP1BAD. Step STEP1ERR invokes program ERRTN, passing EXP1 as a parameter.
3. If the STEP1BAD relational-expression is not true, the system processes the ELSE clause for STEP1BAD. Step STEP2ERR invokes program ERRTN, passing EXP2 as a parameter.
4. However, if the relational-expression for IFTEST3 is false, the system processes the ELSE clause. Step NOPROB invokes procedure RUNOK.
5. Processing then continues with step NEXTSTEP.

## Example 6

The following example shows two IF/THEN/ELSE/ENDIF statement constructs, one of which is nested in the ELSE clause of the other. The nested statements are indented so that they are easier to read.

```
//JOBE      JOB   ...
//PROC1     PROC
//PSTEPONE  EXEC  PGM=...
//          PEND
//PROC2     PROC
//PSTEPTWO  EXEC  PGM=...
//          PEND
```

```
//EXP1      EXEC  PROC=PROC1
//EXP2      EXEC  PROC=PROC2
//IFTEST4   IF  (EXP1.PSTEPONE.RC > 4) THEN
//STEP1ERR  EXEC  PGM=PROG1
//          ELSE
//IFTEST5     IF  (EXP2.PSTEPTWO.ABENDCC=U0012) THEN
//STEP2ERR    EXEC  PGM=PROG2
//            ELSE
//NOERR       EXEC  PGM=PROG3
//ENDTEST5    ENDIF
//ENDTEST4  ENDIF
//NEXTSTEP  EXEC  ...
```

Processing for the IF/THEN/ELSE/ENDIF construct named IFTEST4 is:

1. If the relational-expression for IFTEST4 is true (the return code is greater than 4 for PSTEPONE in procedure PROC1, which is invoked by step EXP1), the system processes the THEN clause of IFTEST4. EXEC statement STEP1ERR invokes program PROG1. The system then passes control to ENDIF statement ENDTEST4, and processing continues with step NEXTSTEP.
2. However, if the relational-expression for IFTEST4 is false (the return code is 4 or less for PSTEPONE in procedure PROC1, which is invoked by step EXP1), the system processes the ELSE clause of IFTEST4. It evaluates the IF/THEN/ELSE/ENDIF statement construct IFTEST5.

   Processing for the IF/THEN/ELSE/ENDIF construct named IFTEST5 is:

   a. If the relational-expression for IFTEST5 is true (the user-defined abend completion code is 0012 from PSTEPTWO in procedure PROC2, which is invoked by step EXP2), the system processes the THEN clause of IFTEST5. EXEC statement STEP2ERR invokes program PROG2. The system then passes control to ENDIF statement ENDTEST5, and then ENDTEST4. Processing continues with EXEC statement NEXTSTEP.
   b. However, if the relational-expression for IFTEST5 is false (that is, the user-defined abend completion code is not 0012 from PSTEPTWO in procedure PROC2, which is invoked by step EXP2), the system processes the ELSE clause of IFTEST5. EXEC statement NOERR invokes program PROG3. Processing then continues with step NEXTSTEP.

## Example 7

The following example shows an IF/THEN/ELSE/ENDIF statement construct with a deferred checkpoint restart.

```
//DEFER1 JOB RESTART=(STEP2,CHECK004)
//STEP1  EXEC PGM=IEFBR14
//IF1    IF STEP1.RC=0 | ¬STEP1.RUN THEN
//STEP2  EXEC PGM=DEBIT1
//STEP3  EXEC PGM=CREDIT1
//STEP4  EXEC PGM=SUMMARY1
//       ELSE
//STEP5  EXEC PGM=DEBIT2
//STEP6  EXEC PGM=CREDIT2
//STEP7  EXEC PGM=SUMMARY2
//       ENDIF
```

Processing for the IF/THEN/ELSE/ENDIF construct named IF1 is as follows:

1. The conditions on statement IF1 will be checked before executing STEP2.
2. STEP1.RC=0 tests false because STEP1 did not execute and cannot be correctly evaluated.
3. ¬STEP1.RUN tests true; therefore, STEP2, STEP3, and STEP4 will execute and STEP5, STEP6, and STEP7 will not execute.

**Note:** Without the ¬STEP.RUN condition, STEP2, STEP3, and STEP4 would not execute and STEP5, STEP6, and STEP7 would execute.

## Example 8

The following example shows an IF/THEN/ELSE/ENDIF statement construct with a deferred step restart.

```
//DEFER2 JOB RESTART=(STEP3)
//STEP1  EXEC PGM=IEFBR14
//IF1    IF STEP1.RC=0 | ¬STEP1.RUN THEN
//STEP2  EXEC PGM=DEBIT1
//STEP3  EXEC PGM=CREDIT1
//STEP4  EXEC PGM=SUMMARY1
//       ELSE
//STEP5  EXEC PGM=DEBIT2
//STEP6  EXEC PGM=CREDIT2
//STEP7  EXEC PGM=SUMMARY2
//       ENDIF
```

Processing for the IF/THEN/ELSE/ENDIF construct named IF1 is:

1. The conditions on statement IF1 will be checked before executing STEP3.
2. STEP1.RC=0 tests false because STEP1 did not execute and cannot be correctly evaluated.
3. ¬STEP1.RUN tests true; therefore, STEP3 and STEP4 will execute and STEP5, STEP6, and STEP7 will not execute.

**Note:** Without the ¬STEP1.RUN condition, STEP3 and STEP4 would not run, and STEP5, STEP6, and STEP7 would run.

## Example 9

The following example specifies that if STEP1 does not abend, the system is to run STEP2 and STEP3. Otherwise it is to run STEP4.

```
//JOBF     JOB ...
//STEP1    EXEC  PGM=...
//IFTEST6  IF    ¬ABEND THEN
//STEP2    EXEC  PGM=...
//STEP3    EXEC  PGM=...
//         ELSE
//STEP4    EXEC  PGM=...
//         ENDIF
```

The determination of which steps to run is made when the IF/THEN/ELSE/ENDIF statement construct is processed immediately after STEP1 executes. This determination is not subject to change based on the results of running steps after STEP1.

Thus, if STEP1 does not abend, even if STEP2 does, STEP3 (and not STEP4) still runs. If, however, STEP1 does abend, STEP4 is the next step to run, as prescribed by the ELSE clause.

# Chapter 19. INCLUDE statement

***Purpose:*** Use the INCLUDE statement to:

- Identify the name of the member of a partitioned data set (PDS) or partitioned data set extended (PDSE) that contains a set of JCL statements (such as DD and OUTPUT JCL statements) called an INCLUDE group.
- Imbed the INCLUDE group in the JCL stream at the position of the INCLUDE statement.

The INCLUDE group replaces the INCLUDE statement, and the system processes the imbedded JCL statements as part of the JCL stream. The JCL statements, which are subject to all JCL processing rules, must be complete statements; that is, you cannot use an imbedded statement to continue the statement that precedes INCLUDE.

## Description

### Syntax

```
//[name]  INCLUDE  MEMBER=name   [comments]
```

The INCLUDE statement consists of the characters // in columns 1 and 2 and four fields: name, operation (INCLUDE), keyword parameter (MEMBER), and comments.

### Name field

A name is optional on an INCLUDE statement. If used, code it as follows:

- The name should be unique within the job.
- The name must begin in column 3.
- The name is 1 through 8 alphanumeric or national ($, #, @) characters.
- The first character must be alphabetic or national ($, #, @).
- The name must be followed by at least one blank.
- The name may be preceded by up to 8 alphanumeric or national characters, and then separated by a period. Coding the name in this way should not be confused with specifying an override, as can be done when coding DD statements.

If you do not code a name, column 3 must be blank.

### Operation field

The operation field consists of the characters INCLUDE and must be preceded and followed by at least one blank. It can begin in any column.

### Parameter field

The INCLUDE statement contains one keyword parameter:

© Copyright IBM Corp. 1988, 2015

**MEMBER=name**
Specifies the name of a member of a PDS or partitioned data set extended (PDSE) that contains the set of JCL statements (called an INCLUDE group) to be imbedded in the JCL stream.

The PDS or PDSE must be one of the following:

- A system procedure library (such as SYS1.PROCLIB), or
- An installation-defined procedure library, or
- A private library that you must specify on a JCLLIB statement appearing earlier in the job.

## Comments field

The comments field follows the parameter field after at least one intervening blank.

## Location in the JCL

An INCLUDE statement:

- Can appear anywhere in the job after the JOB statement, with one exception: if there is a JCLLIB statement, the INCLUDE statement must follow the JCLLIB statement.
- Must follow a complete JCL statement.
- Can appear within an INCLUDE group. INCLUDE groups can contain INCLUDE statements and can be nested up to a maximum of 15 levels of nesting.
- Cannot appear in a CNTL/ENDCNTL group, which contains program control statements delimited by the CNTL and ENDCNTL statements.

## Considerations for using INCLUDE groups

System and private libraries can contain both procedures and INCLUDE groups. The order in which the system searches system and private libraries for INCLUDE groups is the same as the search order used for procedures (see "Using a procedure" on page 28).

INCLUDE groups **cannot** contain the following JCL statements:

- JOB statements
- PROC and PEND statements
- JCLLIB statements
- JES2 and JES3 statements and commands

Do not define procedures in an INCLUDE group. However, you can put EXEC statements that invoke procedures in an INCLUDE group.

You can use INCLUDE statements to imbed INCLUDE groups that contain DD and OUTPUT JCL statements, which allows you to use the same data set definitions for various jobs.

When the INCLUDE statement and the INCLUDE group contain symbolic parameters, the system substitutes the values that are current at the time the symbolic parameter is encountered. Values assigned to symbolic parameters in an INCLUDE group (such as with the SET statement) are valid for use on subsequent JCL statements.

## Examples of the INCLUDE statement:

The following examples show INCLUDE statement usage:

1. The following INCLUDE group is defined in member SYSOUT2 of private library CAMPBELL.SYSOUT.JCL.

```
//*  THIS INCLUDE GROUP IS CATALOGED AS...
//*  CAMPBELL.SYSOUT.JCL(SYSOUT2)
//SYSOUT2  DD      SYSOUT=A
//OUT1     OUTPUT  DEST=POK,COPIES=3
//OUT2     OUTPUT  DEST=KINGSTON,COPIES=30
//OUT3     OUTPUT  DEST=MCL,COPIES=10
//*  END OF INCLUDE GROUP...
//*  CAMPBELL.SYSOUT.JCL(SYSOUT2)
```

   The system executes the following program:

```
//TESTJOB  JOB  ...
//LIBSRCH  JCLLIB  ORDER=CAMPBELL.SYSOUT.JCL
//STEP1    EXEC    PGM=OUTRTN
//OUTPUT1  INCLUDE MEMBER=SYSOUT2
//STEP2    EXEC    PGM=IEFBR14
```

   The JCLLIB statement specifies that the system is to search private library CAMPBELL.SYSOUT.JCL for the INCLUDE group SYSOUT2 before it searches any system libraries.

   After the system processes the INCLUDE statement, the JCL stream appears as:

```
//TESTJOB  JOB  ...
//LIBSRCH  JCLLIB  ORDER=CAMPBELL.SYSOUT.JCL
//STEP1    EXEC    PGM=OUTRTN
//*  THIS INCLUDE GROUP IS CATALOGED AS...
//*  CAMPBELL.SYSOUT.JCL(SYSOUT2)
//SYSOUT2  DD      SYSOUT=A
//OUT1     OUTPUT  DEST=POK,COPIES=3
//OUT2     OUTPUT  DEST=KINGSTON,COPIES=30
//OUT3     OUTPUT  DEST=MCL,COPIES=10
//*  END OF INCLUDE GROUP...
//*  CAMPBELL.SYSOUT.JCL(SYSOUT2)
//STEP2    EXEC    PGM=IEFBR14
```

   The system imbeds the INCLUDE group in the JCL stream (replacing the INCLUDE statement), and processes the included JCL statements with the JCL stream.

2. The following example shows the use of the SET statement to assign values to symbolic parameters in an INCLUDE group.

```
//*  THIS INCLUDE GROUP IS CATALOGED AS...
//*  LAMAN.SYSOUT.JCL(SYSOUT2)
//SYSOUT2  DD      SYSOUT=A
//OUT1     OUTPUT  DEST=POK,COPIES=3
//OUT2     OUTPUT  DEST=&AA,COPIES=&NC
//OUT3     OUTPUT  DEST=&BB,COPIES=10
//*  END OF INCLUDE GROUP...
//*  LAMAN.SYSOUT.JCL(SYSOUT2)
```

   The following program is executed.

```
//JOBA     JOB  ...
//LIBS     JCLLIB  ORDER=LAMAN.SYSOUT.JCL
//SET1     SET     AA=KINGSTON,BB=STL,NC=10
//STEP1    EXEC    PGM=OUTRTN
//OUTPUT1  INCLUDE MEMBER=SYSOUT2
//STEP2    EXEC    PGM=IEFBR14
```

   The SET statement, which is easy to change for different jobs, assigns values to the symbolic parameters in INCLUDE group SYSOUT2.

   After the system processes the INCLUDE statement, it executes the JCL stream as:

```
//JOBA     JOB  ...
//LIBS     JCLLIB  ORDER=LAMAN.SYSOUT.JCL
//STEP1    EXEC    PGM=OUTRTN
//*  THIS INCLUDE GROUP IS CATALOGED AS...
//*  LAMAN.SYSOUT.JCL(SYSOUT2)
//SYSOUT2  DD      SYSOUT=A
//OUT1     OUTPUT  DEST=POK,COPIES=3
//OUT2     OUTPUT  DEST=KINGSTON,COPIES=10
//OUT3     OUTPUT  DEST=STL,COPIES=10
//*  END OF INCLUDE GROUP...
//*  LAMAN.SYSOUT.JCL(SYSOUT2)
//STEP2    EXEC    PGM=IEFBR14
```

The system imbeds the INCLUDE group in the JCL stream (replacing the INCLUDE statement), and assigns the values to the symbolic parameters in the INCLUDE group.

3. The following example shows a JES instream data set within an INCLUDE statement:

```
//INCLUDE   MEMBER=HELLO
//STEPA     EXEC  PGM=IEBGENER
//SYSIN     DD    DUMMY
//SYSPRINT  DD    SYSOUT=A
//SYSUT2    DD    SYSOUT=A
//SYSUT1    DD    DATA
HELLO WORLD
/*
```

# Chapter 20. JCLLIB statement

***Purpose:*** Use the JCLLIB statement to:

- Identify the names of the private libraries that the system uses for the job. The system searches the libraries for:
  - Procedures named on any EXEC statements
  - Groups of JCL statements (called INCLUDE groups) named on any INCLUDE statements.
- Identify the names of the system procedure libraries and installation-defined procedure libraries that the system uses for the job.
- Identify the order in which the libraries are to be searched. The system searches the libraries in the order in which you specify them on the JCLLIB statement, prior to searching any unspecified default system procedure libraries.

The JCLLIB statement allows you to code and use procedures and INCLUDE groups in a private library without the need to use system procedure libraries.

You can code only one JCLLIB statement per job.

***Considerations for an APPC scheduling environment:*** In an APPC environment, see the information about scheduler JCL for TP profiles in *z/OS MVS Planning: APPC/MVS Management*.

***Considerations for JES3:*** In a JES3 environment, the system on which the job is submitted and/or converted must have access to any libraries named on the JCLLIB statement.

## Description

### Syntax

```
//[name]  JCLLIB  [keyword-parameter]...  [comments]
```

The JCLLIB statement consists of the characters // in columns 1 and 2 and four fields: name, operation (JCLLIB), keyword parameter and comments.

### Name field

A name is optional on a JCLLIB statement. If used, code it as follows:

- The name must begin in column 3.
- The name is 1 through 8 alphanumeric or national ($, #, @) characters.
- The first character must be alphabetic or national ($, #, @).
- The name must be followed by at least one blank.
- The name may be preceded by up to 8 aphanumeric or national characters, and then separated by a period. Coding the name in this way should not be confused with specifying an override, as can be done when coding DD statements.

© Copyright IBM Corp. 1988, 2015

If a name is not coded, column 3 must be blank.

## Operation field

The operation field consists of the characters JCLLIB and must be preceded and followed by at least one blank. It can begin in any column.

## Parameter field

The JCLLIB statement includes the following parameters:

**ORDER=(***library***[,***library***...])**
Specifies the names of the libraries to be searched. The maximum number of libraries that may be specified is 15. You can specify private libraries, system procedure libraries, and installation-defined procedure libraries. The system searches the libraries in the order in which you specify them, before it searches any unspecified default system procedure libraries. The ORDER parameter can be specified only once.

Do not specify a library that is a temporary data set (&&dsname), partitioned data set if a member name is included, or relative generation number for a GDG.

**Note:** GDGs are not supported.

If only one library is listed in the search order, the parentheses are optional. For example:

```
//MYLIB  JCLLIB  ORDER=MY.PROC1
```

Library names can be enclosed in apostrophes, for example:

```
//MYLIB  JCLLIB  ORDER=('MY.PROC1','MY.PROC2')
```

You can continue the list of libraries to the following statement by breaking the statement after a comma in the list, and continuing the list on the next statement, beginning in any column from 4 to 16. For example:

```
//MYLIB  JCLLIB  ORDER=(MY.PROC1,MY.PROC2,
//           MY.PROC3)
```

You can continue a parameter enclosed in quotation marks by breaking the parameter in column 71 and continuing the parameter in column 16 of the next statement.

```
                                                                column 71
                                                                      |
//MYLIB  JCLLIB  ORDER=('MY.PROC1','MY.PROC2','MY.PROC3','MY.PROC4','MY
//             .PROC5')
               |
             column 16
```

**PROCLIB=***ddname*

For JES2:

Requests a JES2 procedure library by its DDname, as defined in the JES2 procedure used to initialize JES2 in the JES2 environment.

JES2 procedure libraries are defined by:

- DD statements in the JES2 procedure used to initialize JES2
- PROCLIB definitions in the JES2 initialization file (HASPPARM)
- PROCLIB definitions added by the $ADD PROCLIB command

Typically, JES2 procedure library DDnames in the JES2 procedure are in the format PROC*nn*, where *nn* is either 00 or 1 or 2 decimal numbers 1-99.

However, you can use any valid DDname as long as the name matches the DDname in the JES2 procedure or is specified in PROCLIB definitions. The system retrieves called cataloged procedures from the requested JES2 procedure library.

If you omit the PROCLIB parameter, or the DDname cannot be found in the procedure that started JES2, or the DDNAME cannot be found in any PROCLIB definitions, JES2 uses the procedure library specified on the PROC=*nn* parameter for one of the following JES2 initialization statements:

**JOBCLASS(v)**
for each job class.

**JOBCLASS(STC)**
for all started tasks.

**JOBCLASS(TSU)**
for all time-sharing tasks.

If the PROC=*nn* parameter is not defined on the appropriate initialization statement, or if it is not valid, JES2 uses the default library, PROC00. See *z/OS JES2 Initialization and Tuning Guide*for information about creating the JES2 cataloged procedure and *z/OS JES2 Initialization and Tuning Reference* for information about defining JES2 initialization statements.

For JES3:

Specifies the ddname for the procedure library that the system is to search for cataloged procedures called by EXEC statements in the job. The procedure libraries are defined by DD statements in the JES3 procedure used to start JES3 and/or DYNALLOC statements in the JES3 initialization deck (JES3INxx).

JES3 procedure libraries are defined by:

- IATPLB*xx*
- PROC*xx*

Where *xx* is two alphanumeric/national characters.

If the procedure library requested is not found, JES3 flushes the job before execution.

If you omit the PROCLIB parameter and no //*MAIN statement with a PROC= parameter is present, the default depends on the source of the job. If the job is submitted as a batch job, the default is IATPLBST. If the job is submitted from an internal reader, the default can be another procedure library as specified on the STANDARDS initialization statement (the INTPROC, STCPROC, or TSOPROC parameters).

See *z/OS JES3 Initialization and Tuning Guide* for information about creating the JES3 cataloged procedure and *z/OS JES3 Initialization and Tuning Reference* for information about JES3 initialization statements.

## Comments field

The comments field follows the parameter field after at least one intervening blank.

## Location in the JCL

A JCLLIB statement:

- Must appear after the JOB statement and before the first EXEC statement in the job.
- Must appear before any INCLUDE statement.

- Must not appear within an INCLUDE group.

## Considerations for using the JCLLIB statement

You can specify only one JCLLIB statement in a job.

The system and private libraries that you specify on the JCLLIB statement can contain both procedures and INCLUDE groups.

The private libraries that you specify on the JCLLIB statement must comply with the following rules:

- The private library must be cataloged.
- The private library must be accessible to the job. The library must be permanently resident and online.
- The JCLLIB data set cannot be a password-protected data set.
- The job must have read access to any system or private libraries specified on JCLLIB.
- The private library must have the same data set attributes as a system library, which are:
  - Logical record length of 80 bytes (LRECL=80)
  - Fixed length records (RECFM=F or RECFM=FB). If the JCLLIB data set is a PDSE, the record format can only be RECFM=FB.
  - When multiple libraries are specified on the JCLLIB statement, these libraries will be concatenated.

## Examples of the JCLLIB statement

**Note:** For each example, assume that the system default procedure library includes SYS1.PROCLIB only. If you do not specify the JCLLIB statement, then the system searches only SYS1.PROCLIB. ("Using a procedure" on page 28 describes how the system determines the default procedure library.)

***Example 1:***

```
//MYJOB1   JOB     ...
//MYLIBS1  JCLLIB  ORDER=CAMPBEL.PROCS.JCL
//S1       EXEC    PROC=MYPROC1
              .
              .
```

The system searches the libraries for procedure MYPROC1 in the following order:

1. CAMPBEL.PROCS.JCL
2. SYS1.PROCLIB

***Example 2:***

```
//MYJOB2   JOB     ...
//MYLIBS2  JCLLIB  ORDER=(CAMPBEL.PROCS.JCL,PUCHKOF.PROCS.JCL,
//            YUILL.PROCS.JCL,GARY.PROCS.JCL)
//S2       EXEC    PROC=MYPROC2
              .
              .
//INC2     INCLUDE  MEMBER=MYINC2
              .
              .
```

The system searches the libraries for procedure MYPROC2 and INCLUDE group MYINC2 in the following order:

1. CAMPBEL.PROCS.JCL
2. PUCHKOF.PROCS.JCL
3. YUILL.PROCS.JCL
4. GARY.PROCS.JCL
5. SYS1.PROCLIB

***Example 3:*** You can specify a system procedure library.

```
//MYJOB3   JOB     ...
//MYLIBS3  JCLLIB  ORDER=(SYS1.PROCLIB,CAMPBEL.PROCS.JCL)
//S3       EXEC    PROC=MYPROC3
            .
            .
```

The system searches the libraries for procedure MYPROC3 in the following order:

1. SYS1.PROCLIB
2. CAMPBEL.PROCS.JCL
3. SYS1.PROCLIB (the system default procedure library is searched again)

# Chapter 21. JOB statement

***Purpose:*** Use the JOB statement to mark the beginning of a job and to tell the system how to process the job. Also, when jobs are stacked in the input stream, the JOB statement marks the end of the preceding job.

**Note:** The JOB statement can be specified in source JCL for started tasks. For more information, refer to Chapter 7, "Started tasks," on page 59.

The parameters you can specify for job processing are arranged alphabetically in the following sections.

***References:*** For information about the JES initialization parameters that provide installation defaults, see *z/OS JES2 Initialization and Tuning Reference* and *z/OS JES3 Initialization and Tuning Reference*.

## Description

### Syntax

```
//jobname JOB  positional-parameters[,keyword-parameter]...  [comments]

//jobname JOB
```

The JOB statement consists of the characters // in columns 1 and 2 and four fields: name, operation (JOB), parameter, and comments. Do not code comments if the parameter field is blank.

A JOB statement is required for each job.

### Name field

Code a jobname on every JOB statement, as follows:

- Each jobname must be unique.
- The jobname must begin in column 3.
- The jobname is 1 through 8 alphanumeric or national ($, #, @) characters. If your system uses ANSI tapes, the jobname must contain only alphanumeric characters; it must not contain national ($, #, @) characters.
- The first character must be alphabetic or national ($, #, @).
- The jobname must be followed by at least one blank.
- For the job types TSO logon and batch processing, the jobname must be unique, otherwise:
  - For TSO logon, duplicate jobnames fail. For example, if IBMUSER is logged on, another attempt to logon as IBMUSER will fail.
  - For batch processing, duplicate jobnames are delayed. For example, if job BATCH01 is executing, then another job named BATCH01 will be delayed until the original job has completed. However, in JES2 it depends on the parameter DUPL_JOB=DELAY or NODELAY on JOBCLASS.

© Copyright IBM Corp. 1988, 2015

## Operation field

The operation field consists of the characters JOB and must be preceded and followed by at least one blank. It can begin in any column.

## Parameter field

A JOB statement has two kinds of parameters: positional and keyword. All parameters are optional, however, your installation may require the accounting information parameter and the programmer's name parameter.

**Note:** The following parameters are not supported on the JOB statement for a started task:

- CLASS *
- GROUP
- PASSWORD
- RD *
- RESTART
- SCHENV
- SYSTEM
- SYSAFF
- SECLABEL
- TYPRUN
- USER

An asterisk indicates that the parameter is ignored. The other parameters listed result in a JCL error and job failure.

If JES detects an error in any parameter on the JOB statement, the error causes a JCL error and a job failure; the system flushes all subsequent JCL statements, including any SYSOUT-specific DD statements directing output to any other class or destination.

***Positional Parameters:*** A JOB statement can contain two positional parameters. They must precede all keyword parameters. You must code the accounting parameter first, followed by the programmer's name parameter.

| POSITIONAL PARAMETERS | VALUES | PURPOSE |
|---|---|---|
| `([account-number]`<br>`[,accounting-information]...)`<br><br>See section "Accounting information parameter" on page 409 | account-number<br><br>,accounting-information: up to 143 characters | Specifies an account number and other accounting information, formatted as required by the installation. This parameter may be required by the installation. |
| `programmer's-name`<br><br>See section "Programmer's name parameter" on page 440 | programmer's-name: 1 - 20 characters | Identifies the owner of the job. This parameter may be required by the installation. |

***Keyword Parameters:*** A JOB statement can contain the following keyword parameters. You can code any of the keyword parameters in any order in the parameter field after the positional parameters.

*Table 24. JOB statement keyword parameters*

| KEYWORD PARAMETERS | VALUES | PURPOSE |
|---|---|---|
| ADDRSPC= {VIRT}<br>{REAL}<br><br>See section "ADDRSPC parameter" on page 412 | VIRT: virtual (pageable) storage<br><br>REAL: central (nonpageable) storage | Indicates the type of storage required for the job. |
| BYTES={nnnnnn }<br>{([nnnnnn][,CANCEL]) }<br>{([nnnnnn][,DUMP]) }<br>{([nnnnnn][,WARNING])}<br><br>See section "BYTES parameter" on page 413 | nnnnnn: 0 - 999999 | Indicates the maximum amount of output to be printed for the job's sysout data sets, in thousands of bytes, and the action the system is to take if the maximum is exceeded. |
| CARDS={nnnnnnnn }<br>{([nnnnnnnn][,CANCEL]) }<br>{([nnnnnnnn][,DUMP]) }<br>{([nnnnnnnn][,WARNING])}<br><br>See section "CARDS parameter" on page 415 | nnnnnnnn: 0 - 99999999 | Indicates the maximum amount of output, in cards, to be punched for the job's sysout data sets, and the action the system is to take if the maximum is exceeded. |
| CCSID=nnnnn | nnnnn: 1 - 65535 | Specifies the coded character set identifier indicating the character code conversion performed on reads from and writes to tapes accessed in ISO/ANSI Version 4 format. |
| CLASS=jobclass<br><br>See section "CLASS parameter" on page 418 | jobclass: 1-8 characters: A - Z, 0 - 9, and some special characters: See "Subparameter definition" on page 419. | In a non-APPC scheduling environment, assigns the job to a job class. |
| COND=((code,operator)[,(code,operator)]...)<br><br>See section "COND parameter" on page 419 | code: 0 - 4095<br><br>operator: GT Code from<br>GE chart on<br>EQ section<br>Table 25 on page 421<br>LT<br>LE<br>NE | Specifies the return code tests used to determine whether a job continues processing or be terminated. |
| GROUP=group-name<br><br>See section "GROUP parameter" on page 423 | group-name: 1 - 8 alphanumeric or national characters ($, #, @) | In a non-APPC scheduling environment, identifies a group to which a RACF-defined user is to be connected. |
| JESLOG= {SPIN}<br>{NOSPIN}<br>{SUPPRESS}<br><br>See section "JESLOG parameter" on page 424 | SPIN: JESLOG is spin-eligible. There is an optional second operand that specifies the time or the time interval.<br><br>NOSPIN: JESLOG is not spun.<br><br>SUPPRESS: JESLOG is suppressed. | Has meaning when the subsystem is a version of JES2 or JES3 that supports this function. It specifies whether the JESLOG data set should be spin-eligible and if it should be automatically spun at a particular time or time interval. |

*Table 24. JOB statement keyword parameters (continued)*

| KEYWORD PARAMETERS | VALUES | PURPOSE |
|---|---|---|
| JOBRC= {MAXRC}<br>{LASTRC}<br>{(STEP,stepname[.procstepname]}<br><br>See section "JOBRC parameter" on page 426 | MAXRC: The job completion code is set to the highest return code of any step in the job, or if the completion of the job fails because of an ABEND, the job completion code is set to the last ABEND code; this is the default parameter.<br><br>LASTRC: The job completion code is set to the return code or ABEND code of the last step that is executed in the job.<br><br>(STEP,stepname[.procstepname]): The job completion code is set to the return code or ABEND code of the step that is indicated by the ***stepname*.[*.procstepname*]** parameter. If this step does not exist, a JCL error is generated. If this step does not run, the processing is the same as if MAXRC is specified.<br><br>*stepname*: the name of the job step.<br><br>*procstepname*]: the name of the job step within the procedure. | Use the JOBRC parameter to control how the job completion code (presented by JES2 or JES3) is set. By default (when JOBRC is not specified), the job completion code is set to the highest return code of any step, or if the job's execution fails because of an ABEND, the job completion code is set to the last ABEND code; however, this parameter can be used to request that the job completion code be set to the return code of the last executed step or a particular step that more accurately reflects the success or failure of the job. |
| LINES={nnnnnn }<br>{([nnnnnn][,CANCEL]) }<br>{([nnnnnn][,DUMP]) }<br>{([nnnnnn][,WARNING])}<br><br>See section "LINES parameter" on page 427 | nnnnnn: 0 - 999999 | Indicates the maximum amount of output to be printed for the job's sysout data sets, in thousands of lines, and the action the system is to take if the maximum is exceeded. |
| MEMLIMIT={nnnnnM}<br>{nnnnnG}<br>{nnnnnT}<br>{nnnnnP}<br>{NOLIMIT}<br><br>See section "MEMLIMIT parameter" on page 429 | nnnnn: 0 - 99999 | Specifies the limit on the total number of usable virtual pages above the bar for a single address space. |
| MSGCLASS=class<br><br>See section "MSGCLASS parameter" on page 430 | class: A - Z, 0 - 9 | In a non-APPC scheduling environment, assigns the job log to an output class. |
| MSGLEVEL=([statements][,messages])<br><br>See section "MSGLEVEL parameter" on page 432 | statements:<br>**0** Only JOB statement<br>**1** All JCL and procedure statements<br>**2** Only JCL statements<br><br>messages:<br>**0** Only JCL messages<br>**1** JCL, JES, and operator messages | Indicates the job control information to be printed in the job log. |
| NOTIFY={nodename.userid}<br>{userid }<br><br>See section "NOTIFY parameter" on page 433 | nodename: 1 - 8 alphanumeric or national characters ($, #, @)<br><br>userid: 1 - 8 alphanumeric or national characters ($, #, @), 1 - 7 alphanumeric or national characters ($, #, @) when userid specified without nodename | In a non-APPC scheduling environment, requests that the system send a message to a userid when this background job completes. |

*Table 24. JOB statement keyword parameters (continued)*

| KEYWORD PARAMETERS | VALUES | PURPOSE |
|---|---|---|
| PAGES={nnnnnnnn }<br>{([nnnnnnnn][,CANCEL]) }<br>{([nnnnnnnn][,DUMP]) }<br>{([nnnnnnnn][,WARNING])}<br><br>See section "PAGES parameter" on page 435 | nnnnnnnn: 0 - 99999999 | Indicates the maximum amount of output, in pages, to print for the job's sysout data sets, and the action the system is to take if the maximum is exceeded. |
| PASSWORD=(password[,new-password])<br><br>See section "PASSWORD parameter" on page 437 | password or new-password: 1 - 8 alphanumeric or national characters ($, #, @) | In a non-APPC scheduling environment, identifies the current RACF password or specifies a new RACF password. |
| PERFORM=n<br><br>See section "PERFORM parameter" on page 438 | n: 1 - 999 | In WLM compatibility mode (not available on z/OS V1R3 or later systems), specifies the job's performance group.<br><br>In WLM goal mode, can be used for classification of the job to a service class or report class. |
| PRTY=priority<br><br>See section "PRTY parameter" on page 442 | priority (JES2): 0 - 15 priority (JES3): 0 - 14 | In a non-APPC scheduling environment, JES2: Assigns the job's queue selection priority. JES3: Assigns the job's initiation or selection priority in its job class. |
| RD= {R }<br>{RNC}<br>{NR }<br>{NC }<br><br>See section "RD parameter" on page 443 | R: restart, checkpoints allowed<br>RNC: restart, no checkpoints<br>NR: no restart, checkpoints allowed<br>NC: no restart, no checkpoints | In a non-APPC scheduling environment, indicates whether the operator should perform automatic step restart, if the job fails, and controls whether checkpoints are written for CHKPT macros or DD statement CHKPT parameters. |
| REGION= {valueK}<br>{valueM}<br><br>See section "REGION parameter" on page 446 | valueK: 1 - 7 digits<br>from 1 - 2096128<br>valueM: 1 - 4 digits<br>from 1 - 2047 | Specifies the amount of space in kilobytes or megabytes required by the job. |
| RESTART= ({* } [,checkid] )<br>{stepname }<br>{stepname.procstepname}<br><br>See section "RESTART parameter" on page 450 | *: at first step<br>stepname: at named step<br>procstepname: step is in named procedure<br>checkid: at checkpoint in first or named step | In a non-APPC scheduling environment, specifies restart of a job at the beginning of a step or from a checkpoint within a step. |
| SECLABEL=seclabel-name<br><br>See section "SECLABEL parameter" on page 453 | seclabel-name: 1 - 8 alphanumeric or $, #, @, characters | In a non-APPC scheduling environment, identifies the security label of the job to RACF. |
| SCHENV=schenv-name<br><br>See section "SCHENV parameter" on page 454 | schenv-name: 1 - 16 alphanumeric or $, #, @, _ characters | In a non-APPC scheduling environment, identifies the name of the WLM scheduling environment associated with this job. |

*Table 24. JOB statement keyword parameters (continued)*

| KEYWORD PARAMETERS | VALUES | PURPOSE |
|---|---|---|
| SYSAFF={MemberName}<br>{(MemberName,MemberName,...,MemberName)}<br>{(-MemberName,MemberName,...,MemberName)}<br>{-MemberName}<br>{(MemberName,...,IND)}<br>{(-MemberName,...,IND)}<br>{ANY}<br>{(ANY,IND)} | Specifies 1-33 JES2 member and JES3 system names. Each is a 1-4 character name. A value of * (asterisk character) indicates the submitting system. A value of IND indicates that the job must run on a member in independent mode. | Specifies 1-33 JES2 members and JES3 systems that are eligible to process the job. |
| SYSTEM={SystemName}<br>{(SystemName,SystemName,...,SystemName)}<br>{(-SystemName,SystemName, ...,SystemName)}<br>{-SystemName}<br>{ANY}<br>{JGLOBAL}<br>{JLOCAL}}<br>{JLOCAL} | Specifies 1-32 system names. Each is a 1-8 character valid system name. A value of * (asterisk character) indicates the submitting system. | Specifies 1-32 systems that are eligible to process the job. |
| TIME= {([minutes][,seconds])}<br>{1440 }<br>{NOLIMIT }<br>{MAXIMUM }<br><br>See section "TIME parameter" on page 459 | minutes: 1 - 357912<br><br>seconds: 1 - 59<br><br>1440: Specifies that the job can use the processor for an unlimited amount of time.<br><br>NOLIMIT: Specifies that the job can use the processor for an unlimited amount of time.<br><br>MAXIMUM: Specifies that the job can use the processor for the maximum amount of time, 357912 minutes. | Specifies the maximum time the job is to use the processor and requests messages giving the time used. |
| TYPRUN= {COPY }<br>{HOLD }<br>{JCLHOLD}<br>{SCAN }<br><br>See section "TYPRUN parameter" on page 462 | COPY: copies job stream to sysout data set (JES2 only)<br>HOLD: holds job<br>JCLHOLD: holds job before JCL processing (JES2 only)<br>SCAN: scans JCL for syntax errors | In a non-APPC scheduling environment, requests special job processing. |
| USER=userid<br><br>See section "USER parameter" on page 465 | userid: 1 - 7 alphanumeric or $, #, @ characters | In a non-APPC scheduling environment, identifies the job's owner to RACF, SRM, and other system components. |

## Comments field

The comments field follows the parameter field after at least one intervening blank space. If you do not code any parameters on a JOB statement, do not code any comments. In a Started Job case, the comment can be parsed out of the job statement as part of JCL merge processing between the JCL that was created by the START command and any input JOB statement that was supplied (IEFJOBS or IEFPDSI DD).

## Location in the JCL

A JOB statement must be the first statement in each job. JOB statements never appear in cataloged or in-stream procedures.

## Examples of JOB statements

```
//ALPHA   JOB   843,LINLEE,CLASS=F,MSGCLASS=A,MSGLEVEL=(1,1)
//LOS     JOB   ,'J M BUSKIRK',TIME=(4,30),MSGCLASS=H,MSGLEVEL=(2,0)
//MART    JOB   1863,RESTART=STEP4 THIS IS THE THIRD JOB STATEMENT.
```

```
//TRY8    JOB
//RACF1   JOB   'D83,123',USER=RAC01,GROUP=A27,PASSWORD=XYY
//RUN1    JOB   'D8306P,D83,B1062J12,S=C','JUDY PERLMAN',MSGCLASS=R,
//      MSGLEVEL=(1,1),CLASS=3,NOTIFY=D83JCS1,
//      COND=(8,LT)
```

## Accounting information parameter

***Parameter type***

Positional, required (according to installation procedures)

***Purpose***

Use the accounting information parameter to enter an account number and any other accounting information that your installation requires.

***References***

For more information on how to add accounting routines, see *z/OS MVS System Management Facilities (SMF)*.

### Syntax

```
([account-number][,accounting-information]...)
```

**Location**: Code the accounting information parameter first in the parameter field.

**Omission**: If you omit the accounting information parameter but you are coding a programmer's name parameter, code a comma to indicate the omitted parameter. If you omit both positional parameters, do not code any commas before the first keyword parameter.

**Length**: The entire accounting information parameter must not exceed 143 characters:

- Including any commas, which are considered part of the information.
- Excluding any enclosing parentheses, which are not considered part of the information.

**Multiple subparameters**: When the accounting information parameter consists of more than one subparameter, separate the subparameters by commas and enclose the parameter in parentheses or apostrophes. For example, (5438,GROUP6) or '5438,GROUP6'. If you use apostrophes, all information inside the apostrophes is considered one field.

**Special characters**: When a subparameter contains special characters, other than hyphens, enclose it in apostrophes and the entire parameter in parentheses or enclose all of the parameter in apostrophes. For example, (12A75,'DEPT/D58',706) or '12A75,DEPT/D58,706'. Code each apostrophe or ampersand that is part of the accounting information as two consecutive apostrophes or ampersands. For example, code DEPT'D58 as (12A75,'DEPT''D58',706) or '12A75,DEPT''D58,706'. Code 34&251 as '34&&251'.

**Continuation onto another statement**: Enclose the accounting information parameter in parentheses. End each statement with a comma after a complete subparameter. For example:

```
//JOB1  JOB  (12A75,'DEPT/D58',
//    706)
```

## Subparameter definition

**account-number**
Specifies an accounting number, as defined by the installation.

**accounting-information**
Specifies more information, as defined by the installation. For example, your department and room numbers.

## Relationship to other control statements

If you are to provide accounting information for an individual step within a job, code an ACCT parameter on the EXEC statement for that step.

## JES2 accounting information format

Except for the first subparameter, the JES2 accounting information shown in the syntax can, alternatively, appear on the JES2 /*JOBPARM statement. If you code the accounting information parameter in the JES2 format, JES2 can interpret and use it.

***References:*** For a discussion of the JES2 scan of the accounting information parameter, see *z/OS JES2 Initialization and Tuning Guide*.

## Syntax

```
(pano,room,time,lines,cards,forms,copies,log,linect)
```

Code a comma in place of each omitted subparameter when other subparameters follow.

***Subparameter definition:***

**pano**
Specifies the programmer's accounting number. pano is 1 through 4 alphanumeric characters.

**room**
Specifies the programmer's room number. room is 1 through 4 alphanumeric characters.

**time**
Specifies the estimated execution time in minutes. time is 1 through 4 decimal numbers. For example, code **30** for 30 minutes. If you omit a time subparameter and a TIME parameter on the JES2 /*JOBPARM statement, JES2 uses an installation default specified at initialization. If job execution exceeds the time, JES2 sends a message to the operator.

**lines**
Specifies the estimated line count, in thousands of lines, from this job's sysout data sets. lines is 1 through 4 decimal numbers. For example, code **5** for 5000 lines. If you omit lines, JES2 uses an installation default specified at initialization.

**cards**
Specifies the estimated number of cards JES2 is to punch from this job's sysout data sets. cards is 1 through 4 decimal numbers. If you omit cards, JES2 uses an installation default specified at initialization.

**forms**
Specifies the forms that JES2 is to use for printing this job's sysout data sets. forms is 1 through 4 alphanumeric characters. For example, code **5** for 5-part forms. If you omit forms, JES2 uses an installation default specified at initialization.

**copies**
Specifies the number of times JES2 is to print and/or punch this job's sysout data sets. copies is 1 through 3 decimal numbers not exceeding an installation-specified limit. The maximum is 255. For example, code **2** for two copies. If you omit copies, JES2 assumes one copy.

The copies subparameter is ignored and only one copy is produced if the output class for the job log, as specified in the JOB MSGCLASS parameter, or the output class of any of the job's system output data sets is a held class.

**log**
Specifies whether or not JES2 is to print the job log. Code **N** to request no job log. If you code any other character or omit this subparameter, JES2 prints the job log. If your installation specified NOLOG for this job's class during JES2 initialization, JES2 will not print a job log.

**linect**
Specifies the number of lines JES2 is to print per page for this job's sysout data sets. linect is 1 through 3 decimal numbers. When you send a data set across a network, linect cannot exceed 254. When you print the data set locally, linect cannot exceed 255. If you omit linect, JES2 uses an installation default specified at initialization. If you code a zero, JES2 does not eject to a new page when the number of lines exceeds the installation default.

***Invalid subparameters:*** Your installation can initialize JES2 to do one of the following if the accounting information contains subparameters that are invalid to JES2:

- Ignore the invalid subparameters.
- Terminate the job. In this case, JES2 requires the first two subparameters: **pano** and **room**.

***Overrides:*** A parameter on any of the following statements overrides an equivalent accounting information subparameter on the JOB statement:

- JOB statement
- JES2 /*JOBPARM statement
- JES2 /*OUTPUT statement
- OUTPUT JCL statement
- DD statement

## Examples of the accounting information parameter

***Example 1***

```
//JOB43  JOB  D548-8686
```

***Example 2***

```
//JOB44  JOB  (D548-8686,'12/8/85',PGMBIN)
```

Because this statement contains an account-number plus additional accounting-information, parentheses are required.

*Example 3*

```
//JOB45  JOB  (CFH1,2G14,15,,,,2)
```

This statement shows a JES2 accounting information parameter: programmer's accounting number, CFH1; room number, 2G14; estimated job time, 15 minutes; and copies, 2. Parentheses are required. Standard values are assumed for the other JES2 subparameters.

# ADDRSPC parameter

*Parameter type*

Keyword, optional

*Purpose*

Use the ADDRSPC parameter to indicate to the system that the job requires virtual storage (which is pageable) or central storage (also called real storage, which is nonpageable).

## Syntax

```
ADDRSPC= {VIRT}
         {REAL}
```

## Subparameter definition

**VIRT**
Requests virtual storage. The system **can** page the job.

**REAL**
Requests central storage (also called real storage). The system **cannot** page the job and must place each step of the job in central storage.

## Defaults

If no ADDRSPC parameter is specified, the default is VIRT.

## Overrides

The JOB statement ADDRSPC parameter applies to all steps of the job and overrides any EXEC statement ADDRSPC parameters.

Code EXEC statement ADDRSPC parameters when each job step requires different types of storage. The system uses an EXEC statement ADDRSPC parameter only when no ADDRSPC parameter is on the JOB statement and only during the job step.

## Relationship to the JOB REGION parameter

***When ADDRSPC=REAL:*** Code a REGION parameter to specify how much central storage (also called real storage) the job needs. If you omit the REGION parameter, the system uses an installation default specified at JES initialization.

***When ADDRSPC=VIRT or ADDRSPC is Omitted:*** Code a REGION parameter to specify how much virtual storage the job needs. If you omit the REGION parameter, the system uses an installation default specified at JES initialization.

## Examples of the ADDRSPC parameter

***Example 1***

```
//PEH  JOB  ,BAKER,ADDRSPC=VIRT
```

The ADDRSPC parameter requests virtual (pageable) storage. The space available to the job is the installation-specified default.

***Example 2***

```
//DEB  JOB  ,ERIC,ADDRSPC=REAL,REGION=100K
```

The ADDRSPC parameter requests central (nonpageable) storage. The REGION parameter specifies 100K of storage for the job.

# BYTES parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Use the BYTES parameter to:

- Indicate the maximum amount of output, in thousands of bytes, to be printed for this job's sysout data sets
- Specify the action that the system is to take if the maximum is exceeded. You can indicate that the job is to be cancelled with or without a dump, or that the job is to continue and the system is to notify the operator that the maximum was exceeded.

## Syntax

```
BYTES={nnnnnn              }
      {([nnnnnn][,CANCEL]) }
      {([nnnnnn][,DUMP])   }
      {([nnnnnn][,WARNING])}
```

## Subparameter definition

**nnnnnn**

Indicates the maximum amount of output to be printed for this job, in thousands of bytes. An nnnnnn value of 500 represents 500,000 bytes. The value for nnnnnn is 0 through 999999.

In a JES2 system, a value of 0 for nnnnnn will produce an amount of output that is based on the record blocking factor. When the system recognizes that the 0 value has been exceeded, one of the following will get control:

- The CANCEL, DUMP, or WARNING option (if coded)
- The installation exit.

In a JES3 system, a value of 0 for nnnnnn will cause JES3 to use the system default defined at initialization.

**CANCEL**
Indicates that the system is to cancel the job without dumping storage when the output for the job exceeds the maximum.

**DUMP**
Indicates that the system is to cancel the job when the output for the job exceeds the maximum, and requests a storage dump.

**WARNING**
Indicates that the job is to continue, and the system is to send a message to the operator, when the output for the job exceeds the maximum. The system issues subsequent warning messages at an interval defined by the installation.

## Defaults

If you do not code the BYTES parameter, the system uses the installation-defined default value.

If you do not code nnnnnn, the system uses an installation-defined limit.

If you do not code CANCEL, DUMP, or WARNING, the system uses the installation-defined default option.

## Overrides

Specifying BYTES on the JOB statement overrides BYTES on the JES2 /*JOBPARM statement, the JES3 //*MAIN statement, and the installation-defined default.

## Relationship to other parameters

In addition to BYTES, the following parameters also limit the amount of output for a job:

- CARDS
- LINES
- PAGES

If the job's output exceeds the limits defined by any of these parameters, the system might cancel the job. When coding BYTES, determine whether the values coded on these related parameters are sufficient to produce the output you require.

## Relationship to other control statements

The OUTLIM parameter of the DD statement controls the number of logical records in the sysout data set defined by that DD statement. If the sysout limit defined on the BYTES parameter is exceeded before the limit defined on OUTLIM, the system will take the action defined on BYTES. If the sysout limit defined on the OUTLIM parameter is exceeded before the limit defined on BYTES, the system exits to the sysout limit exit routine.

## Examples of the BYTES parameter

***Example 1***

```
//JOB1   JOB  (123456),'R F B',BYTES=(500,CANCEL)
```

In this example, the job JOB1 will be cancelled when its output exceeds 500 thousand bytes. The system will not produce a storage dump.

***Example 2***

```
//JOB2   JOB  (123456),'R F B',BYTES=40
```

In this example, when the output for JOB2 exceeds 40 thousand bytes, the installation default determines whether the job is

- Cancelled, and a dump is requested
- Cancelled, and no dump is requested
- Allowed to continue, with a warning message issued to the operator.

# CARDS parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Use the CARDS parameter to:

- Indicate the maximum amount of output, in cards, to be punched for this job's sysout data sets
- Specify the action that the system is to take if the maximum is exceeded. You can indicate that the job is to be cancelled with or without a dump, or that the job is to continue and the system is to notify the operator that the maximum was exceeded.

## Syntax

```
CARDS={nnnnnnnn                }
      {([nnnnnnnn][,CANCEL]) }
      {([nnnnnnnn][,DUMP])   }
      {([nnnnnnnn][,WARNING])}
```

## Subparameter definition

**nnnnnnnn**
Indicates the maximum number of sysout output cards to be punched for this job. For JES2 systems, nnnnnnnn is a value from 0 to 99999999. For JES3 systems, nnnnnnnn is a value from 0 through 6500000. If you specify a value greater than 6500000 in a JES3 system, it will be treated as 6500000.

In a JES2 system, a value of 0 for nnnnnnnn will produce an amount of output that is based on the record blocking factor. When the system recognizes that the 0 value has been exceeded, one of the following will get control:

- The CANCEL, DUMP, or WARNING option (if coded)
- The installation exit.

In a JES3 system, a value of 0 for nnnnnnnn will produce no output.

**CANCEL**
Indicates that the system is to cancel the job without dumping storage when the output for the job exceeds the maximum.

**DUMP**
Indicates that the system is to cancel the job when the output for the job exceeds the maximum, and requests a storage dump.

**WARNING**
Indicates that the job is to continue, and the system is to send a message to the operator, when the output for the job exceeds the maximum. The system issues subsequent warning messages at an interval defined by the installation.

## Defaults

If you do not code the CARDS parameter, the system uses the installation-defined default value.

If you do not code nnnnnnnn, the system uses an installation-defined limit.

If you do not code CANCEL, DUMP, or WARNING, the system uses the installation-defined default option.

## Overrides

Specifying CARDS on the JOB statement overrides CARDS on the JES2 /*JOBPARM statement, the JES3 //*MAIN statement, the JES2 accounting subparameter for cards on the JOB statement, and the installation-defined default.

## Relationship to other parameters

In addition to CARDS, the following JOB statement parameters also limit the amount of output for a job.

- BYTES
- LINES
- PAGES

If the job's output exceeds the limits defined by any of these parameters, the system might cancel the job. When coding CARDS, determine whether the values coded on these related parameters are sufficient to produce the output you require.

## Relationship to other control statements

The OUTLIM parameter of the DD statement controls the number of logical records in the sysout data set defined by that DD statement. If the sysout limit defined on the CARDS parameter is exceeded before the limit defined on OUTLIM, the system will take the action defined on CARDS. If the sysout limit defined on the OUTLIM parameter is exceeded before the limit defined on CARDS, the system exits to the sysout limit exit routine.

## Examples of the CARDS parameter

***Example 1***

```
//JOB1   JOB  (123456),'R F B',CARDS=(500,CANCEL)
```

In this example, the job JOB1 will be cancelled when its output exceeds 500 cards. The system will not produce a storage dump.

***Example 2***

```
//JOB2   JOB  (123456),'R F B',CARDS=4000
```

In this example, when the output for JOB2 exceeds 4000 cards of output, the installation default determines whether the job is

- Cancelled, and a dump is requested
- Cancelled, and no dump is requested
- Allowed to continue, with a warning message issued to the operator.

## CCSID parameter

***Parameter type***

Keyword, optional

***Purpose***

You can request the access method to convert data between the coded character set identifier (CCSID) specified on the JOB or EXEC statement and the CCSID specified on the DD statement. Data conversion is supported on access to ISO/ANSI Version 4 tapes using access methods BSAM or QSAM, but not using EXCP.

ISO/ANSI tapes are identified by the LABEL=(,AL) or LABEL=(,AUL) keyword. The CCSID parameter does not apply to ISO/ANSI Version 1 or ISO/ANSI/FIPS Version 3 tapes or to tapes with labels other than AL or AUL. See *z/OS DFSMS Using Data Sets*for selecting ISO/ANSI Version 4 tapes. It also contains a list of supported CCSIDs.

The CCSID value of 65535 has a special meaning: it suppresses conversion.

When CCSID is not specified at the JOB, EXEC, or DD levels, data passed to BSAM and QSAM is converted to 7-bit ASCII when writing to ISO/ANSI tapes. This might result in data loss on conversion. On READ operations the CCSID (if recorded) on the tape header label is used for conversion.

The CCSID is recorded in the tape header label if conversion is not defaulted.

### Syntax

```
CCSID= nnnnn
```

### Subparameter definition

**nnnnn**
The CCSID as a decimal number from 1 through 65535.

### Default

500.

### Overrides

The CCSID parameter specified on the JOB statement can be overridden by specifying the CCSID parameter on the EXEC statement.

### Relationship to other parameters

Do not code the following parameters with the CCSID parameter:

| | | |
|---|---|---|
| * | DDNAME | QNAME |
| BURST | DYNAM | SYSOUT |
| CHARS | FCB | TERM |
| COPIES | FLASH | UCS |
| DATA | MODIFY | |

### Examples of the CCSID parameter

For examples of the CCSID parameter, see "CCSID parameter" on page 119.

## CLASS parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the CLASS parameter to assign the job to a class. The class you should request depends on the characteristics of the job and your installation's rules for assigning classes.

**Note:** The CLASS parameter is ignored for a started task in a JES2 environment. For a started task in a JES3 environment all class related attributes and functions are ignored except device fencing, SPOOL partitioning, and track group allocation. Refer to the *z/OS JES3 Initialization and Tuning Guide* for more information about class attributes and functions.

In a JES2 system, the assigned job class can affect whether or how a job is executed. A job class can be defined during JES2 initialization as:

- Held. The system holds any job assigned to this class until the operator releases it.
- To be copied only. The system copies the input stream for the job directly to a sysout data set and schedules the sysout data set for output processing. The system does not execute the job or allocate devices.
- To be scanned for job control statement syntax errors. The system does not execute the job or allocate devices.

In a JES2 system, there are a number of factors that determine the order in which a particular job is selected for execution. Therefore, you cannot be assured that job priority (based on the PRTY you assign a job), job class, or the order of job submission will guarantee that the jobs will execute in a particular order. If you need to submit jobs in a specific order, contact your JES2 system programmer for advice based on how your system honors such requests. (*z/OS JES2 Initialization and Tuning Guide* provides JES2 system programmer procedures concerning job queuing and how to control job execution sequence.)

***Considerations for an APPC scheduling environment***

The CLASS parameter has no function in an APPC scheduling environment. If you code CLASS, the system will check it for syntax and ignore it.

### Syntax

```
CLASS=jobclass
```

- The CLASS parameter cannot have a null value.
- The first character of the 1-8 character job class name must be a member of the 36-character set A-Z and 0-9. Following the first character, alphabetic, national and numeric characters are supported.

## Subparameter definition

**jobclass**
Identifies the class for the job. The jobclass value is 1-8 characters. The jobclass must be a valid class specified at JES initialization. The first character of the 1-8 character job class name must be a member of the 36-character set A-Z and 0-9. Following the first character, alphabetic, national and numeric characters are supported.

## Defaults

If you do not specify the CLASS keyword, JES uses the installation default specified at initialization, as follows:

- In a JES2 system, the default is based on the source of the job: The system makes the job's class the same as the installation-specified default class for the particular card reader, work station, or time-sharing user that submitted the job.
- In a JES3 system, the default is an installation-defined standard default class.

## Overrides

A JES3 //*MAIN statement CLASS parameter overrides a JOB statement CLASS parameter.

## Relationship to other control statements

In JES3 systems, you can also code a CLASS parameter on a JES3 //*MAIN statement.

## Example of the CLASS parameter

```
//SETUP  JOB  1249,SMITH,CLASS=M
```

This statement assigns the job to class M.

---

# COND parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the COND parameter to specify the return code tests the system uses to determine whether a job will continue processing. Before and after each job step is executed, the system performs the COND parameter tests against the return codes from completed job steps. If none of these tests is satisfied, the system executes the job step; if any test is satisfied, the system bypasses all remaining job steps and terminates the job.

The tests are made against return codes from the current execution of the job. A step bypassed because of an EXEC statement COND parameter does not produce a return code.

Bypassing a step because of a return code test is not the same as abnormally terminating the step. The system abnormally terminates a step following an error so serious that it prevents successful execution. In contrast, bypassing of a step is merely its omission.

**Note:** In both JES2 and JES3 systems, a JOB COND parameter determines if steps are executed or bypassed. However, JES3 processes all jobs as though each step will execute; therefore, JES3 allocates devices for steps that are bypassed.

Depending on the program invoked, a test showing that a return code from a step is zero is not sufficient to verify that the step did not fail. The system can fail a step (or job) even if the return code is zero. For example, this could happen as a result of specifying CATLG_ERR FAILJOB(YES) and incurring a "post execution error." To determine if a step failed due to a "post execution error", the SMF type 30, sub-type 4 record for the job step can be examined. In this record, bit SMF30SYE in the two-byte SMF30STI field will be on if the job failed due to a "post execution error."

## Syntax

```
COND=(code,operator)
COND=((code,operator)[,(code,operator)]...)
```

- One return code test is: (code,operator)
- You can omit the outer parentheses if you code only one return code test.
- Specify up to eight return code tests for a job.
- The COND parameter cannot have a null value.

## Subparameter definition

**code**
Specifies a number that the system compares to the return code from each job step. *code* is a decimal number from 0 through 4095.

**Note:** Specifying a decimal number greater than 4095 could result in invalid return code testing or invalid return codes in messages.

**operator**
Specifies the type of comparison to be made to the return code. If the specified test is true, the system bypasses all remaining job steps. Use the chart on this page to select the correct operator. Operators and their meanings are:

| Operator | Meaning |
|---|---|
| GT | Greater than |
| GE | Greater than or equal to |
| EQ | Equal to |
| LT | Less than |
| LE | Less than or equal to |
| NE | Not equal to |

## Overrides

If you code the COND parameter on the JOB statement and on one or more of the job's EXEC statements, and if a return code test on the JOB statement is satisfied, the job terminates. In this case, the system ignores any EXEC statement COND parameters.

If the tests on the JOB statement are not satisfied, the system then performs the return code tests on the EXEC statement. If an EXEC return code test is satisfied, the step is bypassed.

## Summary of COND parameters

*Table 25. Continuation or Termination of the Job Based on the COND Parameter*

| Test in the COND parameter | Return Code (RC) from the just completed step | |
|---|---|---|
| | **Continue job** | **Terminate job** |
| COND=(code,GT) | RC >= code | RC < code |
| COND=(code,GE) | RC > code | RC <= code |
| COND=(code,EQ) | RC ¬= code | RC = code |
| COND=(code,LT) | RC <= code | RC > code |
| COND=(code,LE) | RC < code | RC >= code |
| COND=(code,NE) | RC = code | RC ¬= code |

## Examples of the COND parameter

***Example 1***

```
//TYPE  JOB  (611,402),BOURNE,COND=(7,LT)
```

The COND parameter specifies that if 7 is less than the return code, the system terminates the job. Any return code less than or equal to 7 allows the job to continue.

***Example 2***

```
//TEST  JOB  501,BAXTER,COND=((20,GE),(30,LT))
```

The COND parameter specifies that if 20 is greater than or equal to the return code or if 30 is less than the return code, the system terminates the job. Any code of 21 through 30 allows the job to continue.

# DSENQSHR parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Indicates how the system will treat changes in data set disposition between job steps.

When a step includes a DD with OLD, NEW or MOD on the DISP (disposition) keyword, and a later step requests the same data set as SHR, this Data Set Enqueue of Share parameter controls whether the system can change the serialization on the data set to shared control. This then allows other jobs to also share that data set.

**Note:** Only changes to SHR are honored. In addition, if the data set is requested as OLD or MOD again later in the job, exclusive control will remain until only SHR requests remain. This ensures that all updates to the data set are complete before allowing other jobs to request the data.

## Syntax

```
JOB 'programmer info',DSENQSHR={DISALLOW|USEJC|ALLOW}
```

## Subparameter definition

**DISALLOW**
The system is not allowed to change the serialization on the data set to shared control.

**USEJC**
The system may change the serialization on the data set to shared control when the DSENQSHR parameter value for the JES jobclass is AUTO. When the DSENQSHR JES jobclass parameter value is ALLOW or DISALLOW, the system is not allowed to change the serialization of the data set.

**ALLOW**
The system may change the serialization on the data set to shared control when the DSENQSHR parameter value for the JES jobclass is AUTO or ALLOW. When the DSENQSHR JES jobclass parameter value is DISALLOW, the system is not allowed to change the serialization of the data set.

## Defaults

USEJC

## Overrides

A similar parameter for the JES JOBCLASS. If the JOBCLASS includes a DSENQSHR parameter set to DISALLOW, the job specification will be ignored. A job class with DSENQSHR set to AUTO or ALLOW must be used to exploit this function.

**Note:** The DSENQSHR jobclass attribute is JES2-only. When using a JES3 environment, the DSENQSHR function is never active. Additionally, when GRS is in RING mode, the DSENQSHR function is disabled.

*Table 26. JOBCLASS attribute for DSENQSHR*

| LANGUAGE | JOBCLASS attribute for DSENQSHR | | | |
|---|---|---|---|---|
| | | AUTO | ALLOW | DISALLOW |
| JCL | ALLOW | yes | yes | no |
| | USEJC | yes | no | no |
| | DISALLOW | no | no | no |

When yes is indicated, the system is allowed to change the data set serialization to shared control and other jobs may share that data set with this job.

## Relationship to other control statements

This keyword is related to the DISP parameter on DD statements within the job.

## Examples of the DSENQSHR parameter

***Example 1***

```
//JOB1 JOB DSENQSHR=ALLOW
```

In this example, the JOB statement specifies that for any data set allocated for this job, the serialization may be changed to shared control from exclusive control. Whether the function is enabled depends on the DSENQSHR JES jobclass attribute value.

***Example 2***

```
//JOB2 JOB DSENQSHR=DISALLOW
```

In this example, the JOB statement specifies that for any data set allocated for this job, the serialization may not be changed to shared control from exclusive control.

***Example***

```
//JOB3 JOB DSENQSHR=USEJC
```

In this example, the JOB statement specifies that for any data set allocated for this job, the serialization may be changed to shared control from exclusive control. Whether the function is enabled depends on the DSENQSHR JES jobclass attribute value.

## GROUP parameter

***Parameter type***

Keyword, optional

**Note:** Do not specify this parameter for a started task; if GROUP is specified, the job will fail.

***Purpose***

Use the GROUP parameter to specify a RACF-defined group to which a RACF-defined user is to be connected. RACF places each RACF-defined user in a default group; the GROUP parameter is needed only to specify a group other than a user's default group.

If the installation contains the feature for propagation of the user and group identification, the USER and PASSWORD parameters are required, and the GROUP parameter is optional on JOB statements only for the following:

- Batch jobs submitted through an input stream, such as a card reader, if:
  - the job requires access to RACF-protected resources, or
  - the installation requires that all jobs have RACF identification.
- Jobs submitted by one RACF-defined user for another user. In this case, the JOB statement must specify the other user's userid and may need a password. The group id is optional.
- Jobs that execute at another network node that uses RACF protection.

Otherwise, the USER, PASSWORD, and GROUP parameters can be omitted from JOB statements. RACF uses the userid, password, and default group id of the submitting TSO/E user or job.

***References***

For more information on RACF-protected facilities, see the *z/OS Security Server RACF Security Administrator's Guide*.

*Considerations for an APPC scheduling environment*

The GROUP parameter has no function in an APPC scheduling environment. If you code GROUP, the system will check it for syntax and ignore it.

## Syntax

```
GROUP=group-name
```

## Subparameter definition

**group-name**
Identifies the group with which the system is to associate the user. group-name is 1 through 8 alphanumeric or national ($, #, @) characters. The first character must be alphabetic or national ($, #, @).

## Defaults

If you do not code the GROUP parameter, but do code the USER and PASSWORD parameters, the system assigns the RACF default group name associated with the specified userid. However, the default group name is not passed to JES and thus is not available to JES installation exits.

## Example of the GROUP parameter

```
//TEST  JOB  'D83,123456',GROUP=MYGROUP,USER=MYNAME,PASSWORD=ABC
```

This statement requests that the system connect RACF-defined user MYNAME to the group named MYGROUP for the duration of the job.

# JESLOG parameter

*Parameter type*

Keyword, optional

*Purpose*

Use the JESLOG parameter to indicate whether the JESLOG (JESMSGLG and JESYSMSG) data sets should be spin-eligible. You can also indicate if they should be automatically spun at a particular time or on an interval based on time or lines entered.

## Syntax

```
JESLOG= {SPIN,'hh:mm'}
        {SPIN,'+hh:mm'}
        {SPIN,nnn}
        {SPIN,nnnK}
        {SPIN,nnnM}
        {NOSPIN}
        {SUPPRESS}
```

## Subparameter definition

**SPIN**
JESLOG is spin-eligible after job execution begins. The initial JESLOG data sets,

which contain messages before job execution, are not spin-eligible. At the start of job execution, a new set of JESLOG data sets are created which are spin-eligible.

If you specify SPIN without the optional second operand, the JESLOG data sets can be spun at any time during job execution when a JES-specific operator command is issued. The optional second operand is as follows:

**JESLOG=(SPIN,'hh:mm')**
JESLOG will be spun at time 'hh:mm' each 24 hour period. *hh* is hours and has a range of 00 through 23. *mm* is minutes and has a range of 00 through 59.

**Note:**

1. The time must be specified within apostrophes.
2. JESLOG will be spun when the next message is written to the data set after the specified time.

**JESLOG=(SPIN,'+hh:mm')**
JESLOG will be spun every 'hh:mm' time interval. *hh* is hours and has a range of 00 through 23. *mm* is minutes and has a range of 00 through 59. The minimum interval which can be specified is 10 minutes. *hh* must be specified even if zero. For example, JESLOG=(SPIN,'+00:20') specifies that JESLOG be spun at 20 minute intervals.

**Note:**

1. The time interval must be specified within apostrophes.
2. JESLOG will be spun when the next message is written to the data set after the specified time interval has passed.

**JESLOG=(SPIN,nnn)**
**JESLOG=(SPIN,nnnK)**
**JESLOG=(SPIN,nnnM)**
JESLOG will be spun when either data set has *n* lines. A minimum of 500 lines must be specified. *K* is thousands and *M* is millions.

**NOSPIN**
JESLOG will not be spun.

**SUPPRESS**
JESLOG will be suppressed.

## Defaults

If no JESLOG parameter is specified, the default is NOSPIN.

There is no default for the optional second operand. If you specify SPIN with or without the second parameter, the JESLOG data sets can be spun at any time during job execution when a JES-specific operator command is issued. If you specify SPIN with the second parameter, both JESLOG data sets are spun when either data set meets the SPIN criterion.

## Examples of the JESLOG parameter

***Example 1***

```
//PEH  JOB  ,BAKER,JESLOG=(SPIN,'+08:00')
```

The JESLOG parameter requests that JESLOG be spun every 8 hours.

*Example 2*

```
//DEB  JOB  ,ERIC,JESLOG=(SPIN,090K)
```

The JESLOG parameter requests that JESLOG be spun every 90,000 lines.

# JOBRC parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the JOBRC parameter to control how the job completion code (presented by JES2) is set.

## Syntax

```
JOBRC= {MAXRC}
       {LASTRC}
       {(STEP,stepname[.procstepname]}
```

## Subparameter definition

**MAXRC**

The job completion code is set to the highest return code of any step in the job, or if the completion of the job fails because of an ABEND, the job completion code is set to the last ABEND code. This is the default.

**LASTRC**

The job completion code is set to the return code or ABEND code of the last step that is executed in the job.

**(STEP,stepname[.procstepname])**

The job completion code is set to the return code or ABEND code of the step that is indicated by the **stepname.[.procstepname]** parameter. If there are duplicate stepnames, than the last matching step is used. If this step does not exist, a JCL error is generated. If this step does not execute, the processing is the same as if MAXRC is specified. .

## Defaults

If the whole parameter is not specified, then the JOBCLASS setting in jesparm is used (either it's MACRC(=default) or LASTRC). If only the _subparameter_ is not specified, MAXRC is used.

## Overrides

None

## Relationship to other control statements

None.

## Examples of the JOBRC parameter

*Example 1*

```
JOBRC=LASTRC
```

This specification indicates to use the return code of the last executed step as the completion code for the job.

*Example 2*

```
JOBRC=(STEP,C.HLASM)
```

Use the return code for the C step in the HLASM procstepname as the completion code for the job.

# LINES parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Use the LINES parameter to:

- Indicate the maximum amount of output, in thousands of lines, to be printed for this job's sysout data sets
- Specify the action that the system is to take if the maximum is exceeded. You can indicate that the job is to be cancelled with or without a dump, or that the job is to continue and the system is to notify the operator that the maximum was exceeded.

## Syntax

```
LINES={nnnnnn               }
      {([nnnnnn][,CANCEL]) }
      {([nnnnnn][,DUMP])   }
      {([nnnnnn][,WARNING])}
```

## Subparameter definition

**nnnnnn**
Indicates the maximum amount of output to be printed for this job, in thousands of lines. An nnnnnn value of 500 represents 500,000 lines. The value for nnnnnn is 0 through 999999.

In a JES2 system, a value of 0 for nnnnnn will produce an amount of output that is based on the record blocking factor. When the system recognizes that the 0 value has been exceeded, one of the following will get control:

- The CANCEL, DUMP, or WARNING option (if coded)
- The installation exit.

In a JES3 system, a value of 0 for nnnnnn produces no output.

**CANCEL**
Indicates that the system is to cancel the job without dumping storage when the output for the job exceeds the maximum.

**DUMP**
Indicates that the system is to cancel the job when the output for the job exceeds the maximum, and requests a storage dump.

**WARNING**
Indicates that the job is to continue, and the system is to send a message to the operator, when the output for the job exceeds the maximum. The system issues subsequent warning messages at an interval defined by the installation.

## Defaults

If you do not code the LINES parameter, the system uses the installation-defined default value.

If you do not code nnnnnn, the system uses an installation-defined limit.

If you do not code CANCEL, DUMP, or WARNING, the system uses the installation-defined default option.

## Overrides

Specifying LINES on the JOB statement overrides LINES on the JES2 /*JOBPARM statement, the JES3 //*MAIN statement, the JES2 accounting subparameter for lines on the JOB statement, and the installation-defined default.

## Relationship to other parameters

In addition to LINES, the following JOB statement parameters also limit the amount of output for a job:

- BYTES
- CARDS
- PAGES

If the job's output exceeds the limits defined by any of these parameters, the system might cancel the job. When coding LINES, determine whether the values coded on these related parameters are sufficient to produce the output you require.

## Relationship to other control statements

The OUTLIM parameter of the DD statement controls the number of logical records in the sysout data set defined by that DD statement. If the sysout limit defined on the LINES parameter is exceeded before the limit defined on OUTLIM, the system will take the action defined on LINES. If the sysout limit defined on the OUTLIM parameter is exceeded before the limit defined on LINES, the system exits to the sysout limit exit routine.

## Examples of the LINES parameter

***Example 1***

```
//JOB1   JOB  (123456),'R F B',LINES=(500,CANCEL)
```

In this example, the job JOB1 will be cancelled when its output exceeds 500 thousand lines. The system will not produce a storage dump.

***Example 2***

```
//JOB2   JOB  (123456),'R F B',LINES=40
```

In this example, when the output for JOB2 exceeds 40 thousand lines, the installation default determines whether the job is

- Cancelled, and a dump is requested
- Cancelled, and no dump is requested

- Allowed to continue, with a warning message issued to the operator.

## MEMLIMIT parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Use the MEMLIMIT parameter to specify the limit on the total number of usable virtual pages above the bar for a single address space.

### Syntax

```
MEMLIMIT={nnnnnM}
         {nnnnnG}
         {nnnnnT}
         {nnnnnP}
         {NOLIMIT}
```

### Subparameter definition

**nnnnnM**
**nnnnnG**
**nnnnnT**
**nnnnnP**
Specifies a value to be used as the limit on the total size of usable virtual storage above the bar in a single address space. The value may be expressed in megabytes (M), gigabytes (G), terabytes (T), or petabytes (P). nnnnn may be a value from 0 to 99999, with a maximum value of 16384P.

**NOLIMIT**
Specifies that there is no limit on the virtual pages to be used above the bar.

**Note:** Unlike the REGION parameter, MEMLIMIT=0M (or equivalent in G, T, or P) means that the step can not use virtual storage above the bar.

### Defaults

If no MEMLIMIT parameter is specified, the default is the value defined to SMF, except when REGION=0K/0M is specified, in which case the default is NOLIMIT.

### Overrides

Specifying MEMLIMIT on the JOB statement overrides MEMLIMIT on the EXEC statement.

If MEMLIMIT is not specified, SMF provides a default value. The IEFUSI installation exit can override any JCL- or SMF-supplied value.

### Relationship to the REGION parameter

A specification of REGION=0K/0M will result in a MEMLIMIT value being set to NOLIMIT, when a MEMLIMIT value has not been specified on either the JOB or EXEC statements, and IEFUSI has not been used to set the MEMLIMIT.

## Considerations when using the MEMLIMIT parameter

Specifying a REGION size that gives the job all the available storage, such as 0K or any value greater than 16,384K, can cause storage problems if the IBM- or installation-supplied routine IEALIMIT or IEFUSI is not used to establish a limiting value.

## Examples of the MEMLIMIT parameter

```
//TEST  JOB  'D83,123456',MEMLIMIT=10000M
```

This statement specifies that the job is limited to the use of 10000 megabytes of usable virtual pages above the bar.

# MSGCLASS parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the MSGCLASS parameter to assign the job log to an output class. The job log is a record of job-related information for the programmer. Depending on the JOB statement MSGLEVEL parameter, the job log can consist of:

- Only the JOB statement.
- All job control statements.
- In-stream and cataloged procedure statements.
- Job control statement messages.
- JES and operator messages about the job.

**Note:** In a JES3 environment, a job can complete processing before all of its messages have been written to the job log. When this occurs, the job's output is incomplete. For this reason, do not use the contents of the job log as an automation or as a programming interface.

***Considerations for an APPC scheduling environment***

The MSGCLASS parameter has no function in an APPC scheduling environment. If you code MSGCLASS, the system will check it for syntax and ignore it.

## Syntax

```
MSGCLASS=class
```

## Subparameter definition

**`class`**
Identifies the output class for the job log. The class is one character, A through Z or 0 through 9, and must be a valid output class specified at JES initialization.

**NJE Note:** If you specify an output class that is a held class in an NJE environment, the system does not hold the data set until it reaches its ultimate destination node.

## Defaults

The default is based on the source of the job: The system places the job log in the same output class as the installation-specified default class for the particular card reader, work station, or time-sharing user that submitted the job. The installation default is specified at JES initialization.

## Significance of output classes

To print the job log and any output data sets on the same output listing, code one of the following:

- The same output class in the DD SYSOUT parameter as in the JOB MSGCLASS parameter.
- DD SYSOUT=* to default to the JOB MSGCLASS output class.
- DD SYSOUT=(,) to default to one of the following:
  1. The CLASS parameter in an explicitly or implicitly referenced OUTPUT JCL statement. In this case, the OUTPUT JCL CLASS parameter should specify the same output class as the JOB MSGCLASS parameter.
  2. The JOB MSGCLASS output class, if no OUTPUT JCL statement is referenced or if the referenced OUTPUT JCL statement contains either CLASS= or CLASS=*.

## Examples of the MSGCLASS parameter

***Example 1***

```
//EXMP1 JOB  ,GEORGE,MSGCLASS=F
```

In this example, the JOB statement specifies output class F for the job log.

***Example 2***

```
//EXMP2 JOB  ,MENTLE,MSGLEVEL=(2,0)
```

This JOB statement does not specify an output class. In this case, the output class defaults to the installation default output class for the device from which the job was submitted.

***Example 3***

```
//A1403  JOB  ,BLACK,MSGCLASS=L
//STEP1  EXEC PGM=PRINT
//OUTDD1 DD   SYSOUT=L
```

In this example, the JOB statement and sysout DD statement OUTDD1 both specify the same output class. Consequently, the job log and data set OUTDD1 are written on the same output listing.

***Example 4***

```
//B209   JOB   ,WHITE,MSGCLASS=M
//STEPA  EXEC  PGM=PRINT
//OUTDDX DD    SYSOUT=*
```

In this example, the JOB statement specifies that the system route the job log to output class M. The system also routes sysout data set OUTDDX to class M because SYSOUT=* is specified.

# MSGLEVEL parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the MSGLEVEL parameter to control the listing of the JCL output for the job. You can request that the system print the following:

- The JOB statement and all comments and JECL statements up to the first EXEC statement.
- All job control statements in the input stream, that is, all JCL statements and JES2 or JES3 statements.
- In-stream and cataloged procedure statements for any procedure a job step calls.
- Messages about job control statements.
- JES and operator messages about the job's processing: allocation of devices and volumes, execution and termination of job steps and the job, and disposition of data sets.

***Considerations for an APPC scheduling environment***

For information about using the MSGLEVEL parameter in a TP message log definition, see *z/OS MVS Planning: APPC/MVS Management*.

## Syntax

```
MSGLEVEL=([statements][,messages])
```

You can omit the parentheses if you code only the first subparameter.

## Subparameter definition

The JCL output for a batch job or any piece of work handled by JES2 or JES3 is a collection of three data sets. These three data sets (in the order they appear in the output) are:

- JES JOB LOG (JESMSGLG)
- STATEMENT IMAGES (JESJCL)
- SYSTEM MESSAGES (JESYSMSG)

**`statements`**
Indicates which job control statements the system is to print in the statement images portion of the JCL output. This subparameter is one of the following numbers:

**0** The system prints the JOB statement and all comments and JECL statements up to the first EXEC statement.

**1** The system prints all JCL statements, JES2 or JES3 control statements, the procedure statements, and IEF653I messages, which give the values assigned to symbolic parameters in the procedure statements.

**2** The system prints only JCL statements and JES2 or JES3 control statements.

**messages**
Indicates which messages the system is to print in the system messages portion of the JCL output. This subparameter is one of the following numbers:

**0** The system prints only JCL messages. It prints JES and operator messages only if the job abnormally terminates, and prints SMS messages only if SMS fails the job.

**1** The system prints JCL, JES, operator, and SMS messages.

## Defaults

If you do not code the MSGLEVEL parameter, JES uses an installation default specified at initialization.

## Examples of the MSGLEVEL parameter

***Example 1***

```
//EXMP3 JOB  ,GEORGE,MSGLEVEL=(2,1)
```

In this example, the JOB statement requests that the system print JCL statements, JCL messages, JES and operator messages, and SMS messages.

***Example 2***

```
//EXMP4 JOB  ,MENTLE,MSGLEVEL=0
```

In this example, the JOB statement requests that the system print the JOB statement and any comments and JECL statements up to the first EXEC statement; and, that JES is to use the installation default for messages.

***Example 3***

```
//EXMP5 JOB  ,MIKE,MSGLEVEL=(,0)
```

In this example, the JOB statement requests that JES use the installation default for printing JCL statements and the system is not to print JES and operator messages unless the job abnormally terminates. SMS messages are printed only if SMS fails the job.

# NOTIFY parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the NOTIFY parameter to request that the system send a message to a user when this background job completes processing.

***Considerations for an APPC scheduling environment***

The NOTIFY parameter has no function in an APPC scheduling environment. If you code NOTIFY, avoid possible syntax and runtime errors by reading the information about scheduler JCL for TP profiles in *z/OS MVS Planning: APPC/MVS Management*.

## Syntax

```
The NOTIFY parameter for both JES2 and JES3 is the following:

NOTIFY={nodename.userid}
       {userid         }
```

## Subparameter definition for JES2 systems

**nodename.userid**
Identifies a node and a TSO/E or VM userid at that node. The nodename is a symbolic name defined by the installation during initialization; nodename is 1 through 8 alphanumeric or national ($, #, @) characters. The first character of nodename must be alphabetic or national ($, #, @). The userid must be defined at the node. It is 1 through 8 alphanumeric or national ($, #, @) characters; the first character must be alphabetic or national ($, #, @).

**userid**
Identifies the user that the system is to notify. The userid is 1 through 7 alphanumeric or national ($, #, @) characters. The first character must be an alphabetic or national ($, #, @) character. When you specify only a userid, JES2 assumes that the userid is at the origin node.

The userid may also be a valid remote ID in the form Rnnnn or a destid for a remote. If the userid is specified as R1-R9999, JES2 will assume the notify message is intended for a remote and not a userid. If the remote is defined to the system or is less than the highest defined remote for your system, the notify message is queued to the remote. If the remote value is greater than the highest defined remote but less than the maximum allowed remote, the notify message is discarded. If the Rxxxx value specified is greater than R9999, JES2 will consider that as a TSO/E userid and not a remote id.

## Subparameter definition for JES3 systems

**userid**
Identifies the user that the system is to notify. The userid is 1 through 7 alphanumeric characters and must be a valid TSO/E userid. JES3 assumes that the userid is at the node where the job runs.

## Receiving notification of job completion

***In a JES2 system:*** If you are logged on to the member of the JES2 multi-access spool from which you submitted the job, the system immediately notifies you when the job completes. If a TSO user is not currently logged on, notify messages are issued locally (processing as with BRODCAST=YES) instead of being routed to the member that submitted the job (processing as with BRODCAST=NO). You can no longer use BRODCAST= to specify which member issues TSO notify messages when a user is not logged on. BRODCAST= specifications continue to have no effect on message processing when the TSO user is logged on.

***In a JES3 system:*** If you are logged on, the system immediately notifies you at the system you are logged onto when the job completes. If you are not logged on, the system saves the message until you log on to the system from which you originally ran the job.

If you want to receive notification at a system of your choice, specify the system you want to be notified at on the ACMAIN parameter.

If a job is submitted by another job, the ACMAIN parameter specified for the first job is propagated to the second job.

If a //*ROUTE or XMIT JCL statement follows the JOB statement, you may not be notified when the transmitted job completes.

### Examples of the NOTIFY parameter

```
//SIGN  JOB  ,TKLOMP,NOTIFY=VMNODE.VMUSERID
```

When the job SIGN completes processing, the system sends a message to user VMUSERID on node VMNODE.

```
//SIGN  JOB  ,TKLOMP,NOTIFY=MVSUSER
```

When the job SIGN completes processing, the system sends a message to user MVSUSER on the job's origin node.

## PAGES parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Use the PAGES parameter to

- Indicate the maximum amount of output, in pages, to be printed for this job's sysout data sets
- Specify the action that the system is to take if the maximum is exceeded. You can indicate that the job is to be cancelled with or without a dump, or that the job is to continue and the system is to notify the operator that the maximum was exceeded.

### Syntax

```
PAGES={nnnnnnnn               }
      {([nnnnnnnn][,CANCEL]) }
      {([nnnnnnnn][,DUMP])   }
      {([nnnnnnnn][,WARNING])}
```

### Subparameter definition

**nnnnnnnn**

Indicates the maximum amount of output, in pages, to be printed for this job. The value for nnnnnnnn is 0 through 99999999.

In a JES2 system, a value of 0 for nnnnnnnn will produce an amount of output that is based on the record blocking factor. When the system recognizes that the 0 value has been exceeded, one of the following will get control:

- The CANCEL, DUMP, or WARNING option (if coded)
- The installation exit.

In a JES3 system, a value of 0 for nnnnnnnn will produce no output.

**CANCEL**
Indicates that the system is to cancel the job without dumping storage when the output for the job exceeds the maximum.

**DUMP**
Indicates that the system is to cancel the job when the output for the job exceeds the maximum, and requests a storage dump.

**WARNING**
Indicates that the job is to continue, and the system is to send a message to the operator, when the output for the job exceeds the maximum. The system issues subsequent warning messages at an interval defined by the installation.

## Defaults

If you do not code the PAGES parameter, the system uses the installation-defined default value.

If you do not code nnnnnnnn, the system uses an installation-defined limit.

If you do not code CANCEL, DUMP, or WARNING, the system uses the installation-defined default option.

## Overrides

Specifying PAGES on the JOB statement overrides PAGES on the JES2 /*JOBPARM statement, the JES3 //*MAIN statement, and the installation-defined default.

## Relationship to other parameters

In addition to PAGES, the following JOB statement parameters also limit the amount of output for a job:

- BYTES
- CARDS
- LINES

If the job's output exceeds the limits defined by any of these parameters, the system might cancel the job. When coding PAGES, determine whether the values coded on these related parameters are sufficient to produce the output you require.

## Relationship to other control statements

The OUTLIM parameter of the DD statement controls the number of logical records in the sysout data set defined by that DD statement. If the sysout limit defined on the PAGES parameter is exceeded before the limit defined on OUTLIM, the system will take the action defined on PAGES. If the sysout limit defined on the OUTLIM parameter is exceeded before the limit defined on PAGES, the system exits to the sysout limit exit routine.

## Examples of the PAGES parameter

***Example 1***

```
//JOB1   JOB  (123456),'R F B',PAGES=(500,CANCEL)
```

In this example, the job JOB1 will be cancelled when its output exceeds 500 pages.

***Example 2***

```
//JOB2   JOB  (123456),'R F B',PAGES=40
```

In this example, when the output for JOB2 exceeds 40 pages, the installation default determines whether the job is

- Cancelled, and a dump is requested
- Cancelled, and no dump is requested
- Allowed to continue, with a warning message issued to the operator.

## PASSWORD parameter

***Parameter type***

Keyword, optional

**Note:** Do not specify this parameter for a started task; if PASSWORD is specified, the job will fail.

***Purpose***

Use the PASSWORD parameter to identify a current RACF password or specify a new RACF password. You can specify a new password at any time and must specify a new password when your current one expires.

If the installation contains the installation exit routine used to verify the password, a new password specified in the PASSWORD parameter takes effect when the job is read in. The new password takes effect even if the job is held for execution later and may take effect even if the job fails because of JCL errors. When changing the password, other jobs that use the new or old password may fail, depending on when their passwords are verified.

If the installation contains the feature for propagation of the user and group identification, the USER and the PASSWORD parameters are required, and the GROUP parameter is optional on JOB statements only for the following:

- Batch jobs submitted through an input stream, such as a card reader, (1) if the job requires access to RACF-protected resources or (2) if the installation requires that all jobs have RACF identification.
- Jobs submitted by one RACF-defined user for another user. In this case, the JOB statement must specify the other user's userid and may need a password. The group id is optional.
- Jobs that execute at another network node that uses RACF protection.

Otherwise, the USER, PASSWORD, and GROUP parameters can be omitted from JOB statements. RACF uses the userid, password, and default group id of the submitting TSO/E user or job.

***References***

For more information on using RACF-protected facilities, see the *z/OS Security Server RACF Security Administrator's Guide*.

***Considerations for an APPC scheduling environment***

The PASSWORD parameter has no function in an APPC scheduling environment. If you code PASSWORD, the system will check it for syntax and ignore it.

## Syntax

```
PASSWORD=(password[,new-password])
```

You can omit the parentheses if you code only the first subparameter.

## Subparameter definition

**password**
Specifies the user's current RACF password. The password is 1 through 8 alphanumeric or national ($, #, @) characters.

**Note:** The system suppresses the value you code for password from the JESJCL and JESJCLIN data sets.

**new-password**
Specifies the user's new RACF password. The new-password is 1 through 8 alphanumeric or national ($, #, @) characters. The installation's security administrator can impose additional restrictions on passwords; follow your installation's rules.

**Note:** The system suppresses the value you code for new-password from the JESJCL and JESJCLIN data sets.

## Relationship to other parameters

If the installation does **not** contain the user and group identification propagation feature:

- Code a PASSWORD parameter when coding a USER or GROUP parameter on a JOB statement.
- Code a USER parameter when coding a PASSWORD parameter.

## Examples of the PASSWORD parameter

***Example 1***

```
//TEST1  JOB  'D83,123456',PASSWORD=ABCDE,USER=MYNAME
```

This JOB statement identifies ABCDE as the current password for the RACF user.

***Example 2***

```
//TEST2  JOB 'D83,123456',PASSWORD=(BCH,A12),USER=RAC1,GROUP=GRP1
```

This JOB statement requests that the system change the RACF password from BCH to A12.

# PERFORM parameter

***Parameter type***

Keyword, optional

***Purpose***

**Important**

Beginning with z/OS V1R3, WLM compatibility mode is no longer available. Accordingly, the information that pertains specifically to WLM compatibility mode is no longer valid. It has been left here for reference purposes, and for use on backlevel systems.

Use the PERFORM parameter in WLM compatibility mode to specify the performance group for the job. The installation-defined performance groups determine the rate at which associated jobs have access to the processor, storage, and channels.

In WLM goal mode, the PERFORM parameter on the JOB statement can be used to classify jobs and started procedures to a service class and/or report class. This classification method is provided to reduce the need to modify existing JCL when migrating to goal mode. Note that PERFORM on the EXEC statement is ignored in goal mode for jobs and started procedures.

For details on how to use the WLM application for managing a service definition and service policies, see *z/OS MVS Planning: Workload Management*.

## Syntax

```
PERFORM=n
```

## Subparameter definition

**n** In WLM compatibility mode, requests a performance group. The n is a number from 1 through 999 and must identify a performance group that has been defined by your installation. The specified performance group should be appropriate for your job type according to your installation's rules.

In WLM goal mode, n can be used to classify the job or started task to a service class and/or report class.

## Defaults

In compatibility mode, if no PERFORM parameter is specified or if the specified PERFORM number fails validity checks, the system uses an installation default specified at initialization. If the installation did not specify a default, the system uses a built-in default:

| Default | Use |
|---|---|
| 1 | For non-TSO/E job steps |
| 2 | For TSO/E sessions |

See *z/OS MVS Initialization and Tuning Guide* for details.

## Overrides

A JOB statement PERFORM parameter applies to all steps of the job and overrides any EXEC statement PERFORM parameters.

Code EXEC statement PERFORM parameters when each job step executes in a different performance group. The system uses an EXEC statement PERFORM parameter only when no PERFORM parameter is on the JOB statement and only during the job step.

## Examples of the PERFORM parameter

***Example 1: PERFORM in compatibility mode***

```
//STEP1  JOB  ,MARLA,CLASS=D,PERFORM=25
```

In this example, CLASS=D determines the class in which the system will execute the job. Once in the system, the job will run in performance group 25. The installation must have defined the significance of this performance group.

***Example 2: PERFORM in goal mode***

```
//STEP1  JOB  ,KIRTS,PERFORM=26
```

In this example, the job will be associated with service class PBATCH because the PERFORM value is specified as 26, and the PERFORM value of 26 is defined to workload management as being associated with the service class named PBATCH. To associate the PERFORM value with a service class, you need to define a classification rule in the workload management service definition. The following panel from the WLM application shows a rule for subsystem type JES that assigns any job with a PERFORM value of 26 to service class PBATCH.

```
                Modify Rules for the Subsystem Type        Row 1 to 1 of
Command ===> ____________________________________________ SCROLL ===> PA

Subsystem Type . : JES         Fold qualifier names?   Y  (Y or N)
Description  . . . batch

Action codes:  A=After    C=Copy         M=Move     I=Insert rule
               B=Before   D=Delete row   R=Repeat   IS=Insert Sub-rule
          -------Qualifier-------------             -------Class--------
Action    Type       Name     Start                 Service     Report
                                          DEFAULTS: TBATCH      ________
 ____  1  PF         26       ___                   PBATCH      ________
```

## Programmer's name parameter

***Parameter type***

Positional, required (according to installation procedures)

***Purpose***

Use the programmer's name parameter to identify the person or group responsible for a job.

## Syntax

```
programmer's-name
```

**Location**: Place the programmer's name parameter immediately after the accounting information parameter and before all keyword parameters.

**Omission**: Do not code a comma to indicate the absence of the programmer's name parameter. For example:

```
//JOBA  JOB  'D58/706',MSGCLASS=A
```

**Special characters**: Enclose the programmer's name in apostrophes (single quotation marks) when:

- The name contains special characters (including blanks), other than hyphens, leading periods, or embedded periods. For example:

  ```
  //JOBB JOB  ,S-M-TU
  //JOBC JOB  ,.ABC
  //JOBD JOB  ,P.F.M
  //JOBE JOB  ,'BUILD/PAUL'
  //JOBF JOB  ,'MAE BIRDSALL'
  ```

  If the character string contains a blank but with no enclosing single quotation marks, the blank is taken to indicate the end of the JCL statement.
- The last character of the name is a period. For example:

  ```
  //JOBG JOB  ,'A.B.C.'
  ```
- Code each apostrophe that is part of the name as two consecutive apostrophes. For example, code O'DONNELL as 'O''DONNELL'.

## Parameter definition

**programmer's-name**
Identifies the job's owner. The name must not exceed 20 characters, including all special characters.

## Examples of the programmer's name parameter

*Example 1*

```
//APP  JOB  ,G.M.HILL
```

This JOB statement specifies a programmer's name with no accounting information. The leading comma may be optional; check with your installation.

*Example 2*

```
//DELTA   JOB  'T.O''NEILL'
```

The programmer's name contains special characters. The installation requires no accounting information. The imbedded apostrophe is coded as two consecutive apostrophes; the entire name must be enclosed in apostrophes.

*Example 3*

```
//#308  JOB  (846349,GROUP12),MATTHEW
```

This JOB statement specifies an account number, additional accounting information, and a programmer's name.

*Example 4*

```
//JOBA   JOB  'DEPT. 15E'
```

This installation requires the department number in the programmer's name parameter.

# PRTY parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the PRTY parameter to assign a selection priority to your job. Within a JES2 job class or a JES3 job class group, the system selects jobs for execution in order by priority. A job with a higher priority is selected for execution sooner; jobs with the same priority are selected on a first-in first-out basis.

**Note:** Depending on the JES2 initialization options in use at your installation, JES2 may ignore the PRTY parameter.

In a JES2 system, there are a number of factors that determine the order in which a particular job is selected for execution. Therefore, you cannot be assured that job priority (based on the PRTY you assign a job), job class, or the order of job submission will guarantee that the jobs will execute in a particular order. If you need to submit jobs in a specific order, contact your JES2 system programmer for advice based on how your system honors such requests. (*z/OS JES2 Initialization and Tuning Guide* provides JES2 system programmer procedures concerning job queuing and how to control job execution sequence.)

***References***

For more information about priority, see *z/OS JES2 Initialization and Tuning Guide*.

***Considerations for an APPC scheduling environment***

The PRTY parameter has no function in an APPC scheduling environment. If you code PRTY, the system will check it for syntax and ignore it.

## Syntax

```
PRTY=priority
```

## Subparameter definition

**priority**
Requests a priority for the job. The priority is a number from 0 through 15 for JES2 and from 0 through 14 for JES3. The highest priority is 15 or 14.

Follow your installation's rules in coding a priority.

## Defaults

JES2 determines the job priority from the following, in override order:

1. A JES2 /*PRIORITY statement.
2. A PRTY parameter on the JOB statement.
3. A value calculated from the accounting information on a JES2 /*JOBPARM statement or the JOB statement.
4. An installation default specified at JES2 initialization.

JES3 determines the job priority from the following, in override order:

1. A PRTY parameter on the JOB statement. If the specified priority is invalid, JES3 issues an error message.
2. An installation default specified at JES3 initialization.

### Example of the PRTY parameter

```
//JOBA  JOB  1,'JIM WEBSTER',PRTY=12
```

This job has a priority of 12.

## RD parameter

***Parameter type***

Keyword, optional

**Note:** This parameter is ignored for a started task.

***Purpose***

Use the RD (restart definition) parameter to:

- Specify that the system is to allow the operator the option of performing automatic step or checkpoint restart if a job step abends with a restartable abend code. (See the SCHEDxx parmlib member description in *z/OS MVS Initialization and Tuning Guide* for information about restartable abends.)
- Allow JES to perform automatic step restart after a system failure even if the journal option is not specified in the JES initialization parameters or JES control statements.
- Suppress, partially or totally, the action of the assembler language CHKPT macro instruction or the DD statement CHKPT parameter.

The system can perform automatic restart only if all of the following are true:

- The JOB or EXEC statement contains RD=R or RD=RNC.
- The step to be restarted abended with a restartable abend code.
- The operator authorizes a restart.

The system can perform automatic step restart for a job running during a system failure as long as the job has a job journal. A job journal is a sequential data set that contains job-related control blocks needed for restart.

If you use checkpoint restart or restart a job step, you need to save the journal or the system cannot automatically restart the job if it fails or if there is a system restart. If you use the automatic restart manager (ARM) to restart a job, you do not need to save the journal because ARM does not use the job journal when restarting jobs.

For JES2, specify a job journal by one of the following:

- JOURNAL=YES on the CLASS statement in the JES2 initialization parameters.
- RD=R or RD=RNC on either the JOB statement or any one EXEC statement in the job.

For JES3, specify a job journal by one of the following:

- JOURNAL=YES on the CLASS statement in the JES3 initialization parameters.

- RD=R or RD=RNC on either the JOB statement or any one EXEC statement in the job.
- JOURNAL=YES on a JES3 //*MAIN statement in the job.

***References***

For detailed information on deferred checkpoint restart, see *z/OS DFSMSdfp Checkpoint/Restart*.

***Considerations for an APPC scheduling environment***

The RD parameter has no function in an APPC scheduling environment. If you code RD, the system will check it for syntax and ignore it.

## Syntax

```
RD= {R  }
    {RNC}
    {NR }
    {NC }
```

- The RD parameter cannot have a null value.

## Subparameter definition

**R (Restart, Checkpoints Allowed)**
Indicates that the operator can perform automatic step restart if the job fails.

RD=R does not suppress checkpoint restarts:

- If the processing program executed in a job step does not include a CHKPT macro instruction, RD=R allows the system to restart execution at the beginning of the abnormally terminated step.
- If the program includes a CHKPT macro instruction, RD=R allows the system to restart execution at the beginning of the step, if the step abnormally terminates before the CHKPT macro instruction is executed.
- If the step abnormally terminates after the CHKPT macro instruction is executed, only checkpoint restart can occur. If you cancel the affects of the CHKPT macro instruction before the system performs a checkpoint restart, the request for automatic step restart is again in effect.

**RNC (Restart, No Checkpoints)**
Indicates that the operator can perform automatic step restart if the job fails.

RD=RNC suppresses automatic and deferred checkpoint restarts. It suppresses:

- Any CHKPT macro instruction in the processing program: That is, the operator cannot perform an automatic checkpoint restart, and the system is not to perform a deferred checkpoint restart if the job is resubmitted.
- The DD statement CHKPT parameter.
- The checkpoint at end-of-volume (EOV) facility.

**NR (No Automatic Restart, Checkpoints Allowed)**
Indicates that the operator **cannot** perform automatic step restart if the job fails.

RD=NR suppresses automatic checkpoint restart but permits deferred checkpoint restarts. It permits:

- A CHKPT macro instruction to establish a checkpoint.
- The job to be resubmitted for restart at the checkpoint. On the JOB statement when resubmitting the job, specify the checkpoint in the RESTART parameter.

If the system fails, RD=NR does not prevent the job from restarting.

**NC (No Automatic Restart, No Checkpoints)**
Indicates that the operator **cannot** perform automatic step restart if the job fails.

RD=NC suppresses automatic and deferred checkpoint restarts. It suppresses:

- Any CHKPT macro instruction in the processing program.
- The DD statement CHKPT parameter.
- The checkpoint at EOV facility.

## Defaults

If you do not code the RD parameter, the system uses the installation default from the job's job class specified at initialization.

## Overrides

A JOB statement RD parameter applies to all steps of the job and overrides any EXEC statement RD parameters.

Code EXEC statement RD parameters when each job step requires different restart types. The system uses an EXEC statement RD parameter only when no RD parameter is on the JOB statement and only during the job step.

## Relationship to other control statements

RD=NC or RD=RNC suppresses the action of the DD statement CHKPT parameter.

## Examples of the RD parameter

***Example 1***

```
//JILL  JOB  333,TOM,RD=R
```

RD=R specifies that the operator can perform automatic step restart if the job fails.

***Example 2***

```
//TRY56  JOB  333,DICK,RD=RNC
```

RD=RNC specifies that, if the job fails, the operator can perform automatic step restart beginning with the step that abnormally terminates. RD=RNC suppresses automatic and deferred checkpoint restarts.

***Example 3***

```
//PASS  JOB  (721,994),HARRY,RD=NR
```

RD=NR specifies that the operator cannot perform automatic step restart or automatic checkpoint restart. However, a CHKPT macro instruction can establish checkpoints to be used later for a deferred restart.

# REGION parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the REGION parameter to specify the amount of central or virtual storage that the job requires. The system applies the value that you code on REGION to each step of the job.

The amount of storage requested must include the following:

- Storage for all programs to be executed.
- All additional storage the programs request with GETMAIN macro instructions during execution.
- Enough unallocated storage for task initialization and termination. Task initialization and termination can issue GETMAIN macro instructions for storage in the user's address space.

Two installation exits, IEFUSI and IEALIMIT, can also affect the size of the user address space assigned to the job step.

***References***

For more information on address space size, see *z/OS MVS Initialization and Tuning Guide*, and "Resource Control of Address Space" in *z/OS MVS JCL User's Guide*. For more information on region size with checkpoint/restart jobs, see *z/OS DFSMSdfp Checkpoint/Restart*.

## Syntax

```
REGION=  {valueK}
         {valueM}
```

## Subparameter definition

**valueK**
Specifies the required storage in kilobytes (1 kilobyte = 1024 bytes). The value is 1 through 7 decimal numbers, from 1 through 2096128. Code a multiple of 4. For example, code REGION=68K. If the value you code is not a multiple of 4, the system will round it up to the next multiple of 4.

**valueM**
Specifies the required storage in megabytes (1 megabyte = 1024 kilobytes). The value is 1 through 4 decimal numbers, from 1 through 2047. For example, REGION=3M.

**value=0M or 0K**
A value equal to 0K or 0M gives the step all the storage available below the 2 GB bar. This includes below and above 16 megabytes. The resulting size of the region below and above 16 megabytes depends on system options and what system software is installed. When REGION=0K/0M is specified, the MEMLIMIT value is set to NOLIMIT.

**Note:** This may cause storage problems. See the Considerations When Using the REGION parameter section for more information.

If your installation does not change the IBM-supplied default limits in the IEALIMIT or IEFUSI exit routine modules, then specifying various values for the region size has the following results:

- A value equal to 0K or 0M — gives the job all the storage available below and above 16 megabytes. The resulting size of the region below and above 16 megabytes is installation-dependent. When REGION=0K/0M is specified, the MEMLIMIT is set to NOLIMIT.

  **Note:** This may cause storage problems. See the Considerations When Using the REGION parameter section for more information.
- A value greater than 0K or 0M and less than or equal to 16,384K or 16M — establishes the size of the private area below 16 megabytes. If the region size specified is not available below 16 megabytes, the job step abnormally ends with an ABEND822. The extended region size is the default value of 32 megabytes.
- A value greater than 16,384K or 16M and less than or equal to 32,768K or 32M — gives the job all the storage available below 16 megabytes. The resulting size of the region below 16 megabytes is installation-dependent. The extended region size is the default value of 32 megabytes.
- A value greater than 32,768K or 32M and less than or equal to 2,096,128K or 2047M — gives the job all the storage available below 16 megabytes. The resulting size of the region below 16 megabytes is installation-dependent. The extended region size is the specified value. If the region size specified is not available above 16 megabytes, the job step receives any storage available above 16 megabytes, up to the requested amount, and the resulting size of the region above 16 megabytes is installation-dependent.

## Defaults

If no REGION parameter is specified, the system uses the REGION parameter specified on each EXEC statement. If no EXEC statement REGION parameter is specified, the system uses a job step installation default specified at JES initialization.

## Overrides

A JOB statement REGION parameter applies to all steps of the job and overrides any EXEC statement REGION parameters.

Code EXEC statement REGION parameters when each job step requires a different region size. The system uses an EXEC statement REGION parameter only when no REGION parameter is on the JOB statement and only during the job step.

## Relationship to the JOB ADDRSPC parameter

***When ADDRSPC=REAL:*** Code a REGION parameter to specify how much central storage (also called real storage) the job needs. If you omit the REGION parameter, the system uses the default.

***When ADDRSPC=VIRT or ADDRSPC is Omitted:*** Code a REGION parameter to specify how much virtual storage the job needs. If you omit the REGION parameter, the system uses the default.

## Relationship to the MEMLIMIT parameter

A specification of REGION=0K/0M will result in a MEMLIMIT value being set to NOLIMIT, when a MEMLIMIT value has not been specified on either the JOB or EXEC statements, and IEFUSI has not been used to set the MEMLIMIT.

## Relationship to the REGIONX parameter

A specification of REGIONX parameter specifies the amount of central or virtual storage that the step requires below and above the 16 MB line.

## Considerations when using the REGION parameter

Specifying a REGION size that gives the job all the available storage below the 2 GB bar, such as 0K or any value greater than 16,384K, can cause storage problems if the IBM- or installation-supplied routine IEALIMIT or IEFUSI is not used to establish a limiting value.

## Examples of the REGION parameter

***Example 1***

```
//ACCT1  JOB  A23,SMITH,REGION=100K,ADDRSPC=REAL
```

This JOB statement indicates that the job requires 100K of central storage.

***Example 2***

```
//ACCT4  JOB  175,FRED,REGION=250K
```

This JOB statement indicates that the job requires 250K of virtual storage. When the ADDRSPC parameter is omitted, the system defaults to ADDRSPC=VIRT.

# REGIONX parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the REGIONX parameter to specify the amount of central or virtual storage that the step requires below and above the 16 MB line.

The amount of storage that you request must include the following:

- Storage for all programs in the step to run.
- All additional storage that the programs in the step request with GETMAIN, STORAGE, and CPOOL macro instructions.
- Enough unallocated storage for task initialization and termination. Task initialization and termination can issue GETMAIN macro instructions for storage in the user's address space.

## Syntax

```
REGIONX=  {value1}
          {([value1][,value2])}
```

## Subparameter definition

**value1**
Specifies the amount of memory to be assigned below the 16 MB line.

**value2**
Specifies the amount of memory to be assigned above the 16 MB line, but below 2 GB (that is, "below the bar".)

Values for REGIONX are defined with *nnnnnM* for megabytes, *nnnnnG* for gigabytes, or *nnnnnK* for kilobytes.

The installation might reduce the amount of memory to be assigned through the REGIONABOVE or REGIONBELOW keywords in the SMFPRM*xx* parmlib member, by way of the IEFUSI installation exit or keywords in the SMFLIM*xx* parmlib member.

## Defaults

- If no REGIONX keyword is specified on the JOB statement, but the REGION keyword is, the system uses the REGION value.
- If no REGIONX or REGION value is specified on the JOB statement, normal REGION keyword processing is used.

If, however, a REGIONX keyword is specified but with null values, the system uses the following rules for each value:

- First value: The system uses the REGION value from the JOBCLASS (if less than 16M) or 0M, in that order. When 0M is used, this implies that all below the line private storage is available to the program.
- Second value: The system uses the REGION value from the JOBCLASS (if greater than 16M) or 128M, in that order. If the program needs access to all available above the line private storage, a value of 0M is to be explicitly coded for the second value.

The installation might reduce these numbers through the MEMLIMIT keyword in the SMFPRM*xx* parmlib member, by way of the IEFUSI installation exit or by keywords in the SMFLIM*xx* parmlib member.

## Overrides

Unlike the REGION keyword, a JOB statement REGIONX parameter is used as a default for any step of the job that does NOT have a REGIONX keyword on the EXEC statement. You can set the region default and only use REGIONX as an override on the EXEC statements that require different region values.

### JCL procedure overrides:

Because REGIONX is mutually exclusive with REGION,

REGIONX can replace REGION, and

REGION can replace REGIONX

during JOB statement procedure override processing.

This might result in mutually exclusive conflicts with a REGION= or REGIONX= specification on the JOB statement.

Examine the JCL output listing for uses of REGION and REGIONX.

### START command invocation

For a started job, a REGION= or REGIONX= specification on the START command can replace a REGION= or REGIONX= specification on the JOB statement.

For a started procedure, a REGION= or REGIONX= specification on the START command can replace a REGION= specification on the JOB statement that is generated internally by START command processing.

This might result in mutually exclusive conflicts with a REGION= or REGIONX= specification on the JOB statement of the procedure.

Examine the JCL output listing for uses of REGION and REGIONX with the started job or started procedure.

## Relationship to the JOB ADDRSPC parameter

***When ADDRSPC=REAL:*** Code a REGIONX parameter to specify how much central storage (also called real storage) the job needs.

***When ADDRSPC=VIRT or ADDRSPC is Omitted:*** Code a REGIONX parameter to specify how much virtual storage the job needs.

## On a JOB statement that calls a procedure

The REGIONX keyword is mutually exclusive with the ADDRSPC keyword. If you need to use the ADDRSPC keyword, use REGION instead.

## Relationship to the MEMLIMIT parameter

A specification of REGIONX=(0M.0M) results in a MEMLIMIT value being set to NOLIMIT, when a MEMLIMIT value has not been specified on either the JOB or EXEC statements, and IEFUSI and SMFLIM have not been used to set the MEMLIMIT.

## Examples of the REGIONX parameter

```
//REGNX001  JOB MSGLEVEL=1,REGIONX=(4M,256M)
//STEP001  EXEC PGM=ZTT
//ZTTOUT   DD SYSOUT=*,DCB=(LRECL=133,BLKSIZE=133,RECFM=FBA)
//ZTTIN    DD *
/*
```

For example, to request 1 gigabyte above the line and 500 KB below the line for REGIONX values, specify:

```
REGIONX=(500 K,1G)
```

# RESTART parameter

***Parameter type***

Keyword, optional

**Note:** Do not specify this parameter for a started task; if RESTART is specified, the job will fail.

***Purpose***

Use the RESTART parameter to indicate the step, procedure step, or checkpoint at which the system is to restart a job. You can specify that the system perform either of two restarts:

- **Deferred step restart**, which is a restart at the beginning of a job step.
- **Deferred checkpoint restart**, which is a restart from a checkpoint taken during step execution by a CHKPT macro instruction.

*References*

For detailed information on the deferred checkpoint restart, see *z/OS DFSMSdfp Checkpoint/Restart*.

*Considerations for an APPC scheduling environment*

The RESTART parameter has no function in an APPC scheduling environment. If you code RESTART, the system will check it for syntax and ignore it.

## Syntax

```
RESTART=  ({*                    } [,checkid] )
          ({stepname             }            )
          ({stepname.procstepname}            )
```

- You can omit the outer parentheses if you code only the first subparameter.
- The RESTART parameter cannot have a null value.

## Subparameter definition

***** Indicates that the system is to restart execution (1) at the beginning of or within the first job step or (2), if the first job step calls a cataloged or in-stream procedure, at the beginning of or within the first procedure step.

**stepname**
Indicates that the system is to restart execution at the beginning of or within a job step. If stepname refers to an EXEC statement that invokes a procedure, the step name of the step within the procedure must also be specified.

**stepname.procstepname**
Indicates that the system is to restart execution at the beginning of or within a step of a cataloged procedure. Stepname identifies the EXEC statement of the job step that calls the procedure; procstepname identifies the EXEC statement of the procedure step. The step identified by procstepname must contain the PGM keyword rather than invoke a procedure.

**checkid**
Specifies the name of the checkpoint at which the system is to restart execution. This checkpoint must be in the job step specified in the first subparameter.

Omit checkid to request restart at the beginning of the specified job step.

When the name contains special characters, enclose it in apostrophes. Code each apostrophe that is part of the name as two consecutive apostrophes. For example, code CHPT'1 as 'CHPT''1'.

## Relationship to other control statements

When the system is to restart execution in a job step, place a SYSCHK DD statement immediately following the JOB statement. The SYSCHK DD statement defines the data set on which the system entered the checkpoint for the step being restarted.

When preparing for a deferred checkpoint, code the DISP abnormal termination disposition subparameter in the step's DD statements as follows:

- KEEP, to keep all data sets that the restart step is to use.
- CATLG, to catalog all data sets that you are passing from steps preceding the restart step to steps following the restart step.

**In JES2 systems**, you can also use the RESTART parameter on the /*JOBPARM control statement.

**In JES3 systems**, you must also code the FAILURE parameter on the //*MAIN control statement.

## Cautions when coding the RESTART parameter

Before resubmitting a job:

- Check all backward references to steps before the restart step. Eliminate all backward references in EXEC statement PGM parameters and DD statement VOLUME=REF parameters.
- Review all EXEC statement COND parameters. If any of the COND parameters reference a step before the restart step, be aware that the system ignores the return code tests for those steps. See "Considerations when using the COND parameter" on page 346 for more information.
- Note that the stepname and procstepname specified to identify the restart step must be unique within the job. Otherwise, the system will not be able to determine the correct restart step. Results will be unpredictable.
- Review all IF/THEN/ELSE/ENDIF structures. If a relational expression references a step that is bypassed by the RESTART keyword, the system evaluates that part of the expression as false.

## Generation data sets in restarted jobs

In the restart step or following steps, do not use the original relative generation numbers to refer to generation data sets that were created and cataloged before the restart step. Instead, refer to a generation data set by its present relative generation number.

For example, if the last generation data set created and cataloged was assigned a generation number of +2, refer to it as 0 in the restart step and following steps. If generation data set +1 was also created and cataloged, refer to it as -1.

If generation data sets created in the restart step were kept instead of cataloged, that is, DISP=(NEW,CATLG,KEEP) was coded, then refer to them by the same relative generation numbers used to create them.

## Examples of the RESTART parameter

*Example 1*

```
//LINES  JOB  '1/17/95',RESTART=COUNT
```

This JOB statement indicates that the system is to restart execution at the beginning of the job step named COUNT.

***Example 2***

```
//@LOC5  JOB  '4/11/96',RESTART=(PROCESS,CHKPT3)
//SYSCHK DD   DSNAME=CHK,UNIT=3390,DISP=OLD
```

The JOB statement indicates that the system is to restart execution at checkpoint CHKPT3 in job step PROCESS. The SYSCHK DD statement must follow the JOB statement; it defines the data set on which the system wrote checkpoint CHKPT3.

***Example 3***

```
//WORK  JOB  ,PORTER,RESTART=(*,CKPT2)
//SYSCHK DD  DSNAME=CHKPT,UNIT=3390,DISP=OLD
```

The JOB statement indicates that the system is to restart execution at checkpoint CKPT2 in the first job step. The SYSCHK DD statement defines the data set on which the system wrote checkpoint CKPT2.

***Example 4***

```
//CLIP5  JOB  ,COLLINS,RESTART=(PAY.WEEKLY,CHECK8)
//SYSCHK DD   DSNAME=CHKPT,UNIT=3350,DISP=OLD
```

The JOB statement indicates that the system is to restart execution at checkpoint CHECK8 in procedure step WEEKLY. PAY is the name field on the EXEC statement that calls the cataloged procedure that contains procedure step WEEKLY. The SYSCHK DD statement defines the data set on which the system wrote checkpoint CHECK8.

## SECLABEL parameter

***Parameter type***

Keyword, optional

**Note:** Do not specify this parameter for a started task; if SECLABEL is specified, the job will fail.

***Purpose***

Use the SECLABEL parameter to specify the security level at which the job is to execute when submitted to the system. The security label represents a security level and categories as defined to RACF. You must have sufficient authority, granted by the security administrator at your installation, to run the job with the security label you specify.

***References***

For more information about security labels, see the *z/OS Security Server RACF Security Administrator's Guide*.

***Considerations for an APPC scheduling environment***

The SECLABEL parameter has no function in an APPC scheduling environment. If you code SECLABEL, the system will check it for syntax and ignore it.

## Syntax

```
SECLABEL=seclabel-name
```

## Subparameter definition

**seclabel-name**
Specifies the name of a security label defined by the security administrator at your installation. The seclabel-name is one through eight alphanumeric or national ($, #, @) characters. The first character must be alphabetic, $, #, or @.

## Defaults

If you do not specify the SECLABEL parameter, the system uses the default security label in your RACF profile.

## Relationship to other parameters

Use the SECLABEL parameter on the JOB statement with the DPAGELBL and SYSAREA parameters on an OUTPUT JCL statement, as instructed by your security administrator.

You may code SECLABEL with any other JOB statement parameters.

## Example of the SECLABEL parameter

```
//JOBA  JOB  1,'JIM WOOSTER',SECLABEL=CONF
```

In this example, JOBA executes at a security level defined for security label CONF.

# SCHENV parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the SCHENV parameter to specify the name of the Workload Manager (WLM) scheduling environment to associate with this job. A scheduling environment is a list of resources and their required settings. By associating a scheduling environment name with a job, you ensure that the job will be scheduled for execution only on a system that satisfies those resource state requirements. Note, however, that the job will go through JCL conversion prior to being held. If the JCL of the job refers to a subsystem (DD SUBSYS=), then TYPRUN=JCLHOLD is the only way to ensure that the required subsystem is actually up and functioning at JCL conversion-time.

***Reference***

For more information about WLM scheduling environments, see *z/OS MVS Planning: Workload Management*.

**Note:** Do not specify the SCHENV parameter for a started task; the job will fail.

***Considerations for an APPC scheduling environment***

The SCHENV parameter has no function in an APPC scheduling environment. If you code SCHENV, the system will check it for syntax and ignore it.

***Considerations for a JES2 environment***

You can provide a SCHENV default in a JES2 environment via a JOBCLASS(c) specification.

## Syntax

```
SCHENV=schenv-name
```

## Subparameter definition

**schenv-name**
Specifies the name of a WLM scheduling environment to be associated with this job. The schenv-name is 1 through 16 alphanumeric, national ($, #, @) characters, or the underscore (_). If you include an underscore character in the schenv-name, you must imbed the underscore and enclose the name in apostrophes (single quotation marks). For example, 'PLEX_D01' is valid, but 'PLEX_' and PLEX_D01 are not.

## Defaults

If you do not specify the SCHENV parameter, the job will not be associated with any WLM scheduling environment.

## Relationship to other control statements

You can use scheduling environments with the following parameters:

- The SYSAFF parameter or SYSTEM parameter on the JOB control statement.
- The SYSAFF parameter on the /*JOBPARM control statement for JES2.
- The SYSTEM parameter on the //*MAIN control statement for JES3.

For example, you can restrict a job to either SYS1 or SYS2 based on the scheduling environment associated with that work, and then use the SYSAFF or SYSTEM parameter to further restrict that work only to system SYS1.

## Example of the SCHENV parameter

```
//JOBA  JOB  1,'STEVE HAMILTON',SCHENV=DB2LATE
```

In this example, JOBA is associated with the DB2LATE scheduling environment.

# SYSAFF parameter

***Parameter type***

Keyword, optional

***Purpose***

Indicates the JES2 members and JES3 systems that are eligible to process the job (representing system affinity). Up to 33 names can be specified on the SYSAFF parameter, limited by the number of JES2 members and JES3 systems that can exist in a JESplex.

***Considerations for a JES3 environment***

The following parameters must be consistent with the SYSTEM or SYSAFF parameter, or JES3 will terminate the job:

- For the CLASS parameter on the JOB or //*MAIN statement, the requested processor must be assigned to execute jobs in the specified class.
- All devices specified on DD statement UNIT parameters must be available to the requested processor.
- The TYPE parameter on the //*MAIN statement must specify the system running on the requested processor.
- Dynamic support programs requested on //*PROCESS statements must be able to be executed on the requested processor.
- If any DD statement UNIT parameter in the job specifies a device-number, either a SYSTEM or SYSAFF parameter must be coded or the JES3 //*MAIN statement must contain a SYSTEM parameter.

## Syntax

```
SYSAFF={MemberName}
              {(MemberName,MemberName, ...,MemberName)}
              {(-MemberName,MemberName, ...,MemberName)}
              {-MemberName}
              {(MemberName,...,IND)}
              {(-MemberName,...,IND)}
              {ANY}
              {(ANY,IND)}
```

## Subparameter definition

**MemberName**
Specifies up to 33 1-4 character valid JES2 member names and 8-character JES3 system names. A of value of * (asterisk character) indicates the system that submitted the job. A - (minus character) preceding a member or system name indicates the JES2 member or JES3 system is not eligible for processing the job. A - (minus character) preceding the *first* member or system name in a list indicates that none of members or systems in the list are eligible for processing the job.

**ANY**
Indicates any system that satisfies the job's requirements.

**IND**
Specifies, after any other SYSAFF parameters, for JES2 to use system scheduling in independent mode. If IND is specified, the subparameters must be enclosed in parentheses characters. The IND value must be included with at least one JES2 member name or a JCL error will result. JES3 ignores the IND value if it is not a valid JES3 system name.

## Defaults

For JES2, the default system(s) for a job are set via SYSAFF values that are associated with the input device. For JES3, the default system is the processor that is defined for the job's class.

## Relationship to other control statements

The following JOB parameters cannot be specified with the SYSAFF parameter:

- SYSTEM

## Examples of the SYSAFF parameter

In the following example, the systems represented by member names SY1 and SY2 are eligible for processing the job:

```
SYSAFF=(SY1,SY2)
```

In the following example, the systems represented by member names SY1 and SY2 and the system that submitted the job are eligible for processing the job:

```
SYSTEM=(SY1,*,SY2)
```

In the following example, the systems represented by member names SY1 and SY2 and the system that submitted the job are not eligible for processing the job:

```
SYSTEM=(-SY1,*,SY2)
```

In the following example, any JES2 member or JES3 system in the JESplex that satisfies the job's requirements are eligible to process the job:

```
SYSAFF=ANY
```

In the following example, the systems represented by member names SY1 and SY2 and the system that submitted the job are eligible for processing the job. The IND value indicates that JES2 will use system scheduling in independent mode. JES3 ignores the IND value when it is not a valid JES3 system name. If IND is a valid JES3 system name, then the JES3 system that is represented by member name IND will also be eligible for processing the job.

```
SYSAFF=(SY1,*,SY2,IND)
```

# SYSTEM parameter

***Parameter type***

Keyword, optional

***Purpose***

Indicates the systems that are eligible to process the job. The parameter indicates the system affinity represented by a system name. Up to 32 system names can be coded on the SYSTEM parameter, limited by the number of JES systems that can exist in a JESplex. A minus character (-) preceding the first system name in a list indicates that none of the systems listed are eligible for processing the job. For JES2, the default systems for a job are set via SYSAFF values that are associated with the input device. For JES3, the default system is the processor used for the job's class.

***Considerations for a JES3 environment***

The following parameters must be consistent with the SYSTEM or SYSAFF parameter, or JES3 will terminate the job:

- For the CLASS parameter on the JOB or //*MAIN statement, the requested processor must be assigned to execute jobs in the specified class.
- All devices specified on DD statement UNIT parameters must be available to the requested processor.
- The TYPE parameter on the //*MAIN statement must specify the system running on the requested processor.
- Dynamic support programs requested on //*PROCESS statements must be able to be executed on the requested processor.
- If any DD statement UNIT parameter in the job specifies a device-number, either a SYSTEM or SYSAFF parameter must be coded or the JES3 //*MAIN statement must contain a SYSTEM parameter.

## Syntax

```
SYSTEM={SystemName}
       {(SystemName,SystemName, ...,SystemName)}
       {(-SystemName,SystemName, ...,SystemName)}
       {-SystemName}
       {ANY}}
       {JGLOBAL}
       {JLOCAL}
```

## Subparameter definition

**SystemName**
Specifies up to 32 1-8 character system names. A value of * (asterisk character) indicates the system that submitted the job. A minus character (-) preceding a system name indicates the system is not eligible for processing the job. A minus character (-) preceding the first system name in a list indicates that none of the systems listed are eligible for processing the job.

**ANY**
Indicates any system that satisfies the job requirements.

**JGLOBAL**
Indicates that the job is to run on the JES3 global processor only.

**JLOCAL**
Indicates that the job is to run on a JES3 local processor only.

## Relationship to other control statements

The following JOB parameters cannot be specified with the SYSTEM parameter:

- SYSAFF

## Examples of the SYSTEM parameter

In the following example, systems SYSTEM01 and SYSTEM02 are eligible to process the job:

```
SYSTEM=(SYSTEM01,SYSTEM02)
```

In the following example, systems SYSPROD, SYSTEST, and the system that submitted the job are eligible to process the job:

```
SYSTEM=(SYSPROD,*,SYSTEST)
```

In the following example, systems SYSTEST, SYSPROD and the system that submitted the job are not eligible to process the job:

```
SYSTEM=(-SYSPROD,*,SYSTEST)
```

In the following example, any system in the JESplex that satisfies the job's requirements are eligible to process the job:

```
SYSTEM=ANY
```

## TIME parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the TIME parameter to specify the maximum amount of time that a job may use the processor or to find out through messages how much processor time a job used.

The system terminates a job that exceeds the specified time limit unless an installation exit routine at exit IEFUTL extends the time. Exit routine IEFUTL is established through System Management Facilities (SMF).

You can use the TIME parameter on a JOB statement to *decrease* the amount of processor time available to a job or job step below the default value. You cannot use the TIME parameter on a JOB statement to *increase* the amount of time available to a job step over the default value. To increase the allowable time over the default value, use the TIME parameter on the EXEC statement.

For releases prior to MVS/ESA SP Version 4 Release 3.0, the amount of time that a job step receives might be slightly more or less than the requested processor time. The exact amount of processor time is based on certain system events.

As of MVS/ESA SP Version 4 Release 3.0, the job step receives at least the requested amount of CPU time. Based on system events, additional CPU time might be provided.

***Reference***

See *z/OS MVS Installation Exits*.

### Syntax

```
TIME= {([minutes][,seconds])}
      {1440                 }
      {NOLIMIT              }
      {MAXIMUM              }
```

You can omit the parentheses if you code only 1440 or the processor time in minutes.

### Subparameter definition

**minutes**
Specifies the maximum number of minutes a job may use the processor. Minutes must be a number from 0 through 357912 (248.55 days).

Do not code TIME=0 on a JOB statement. The results are unpredictable.

**seconds**
Specifies the maximum number of seconds that a job may use the processor, in addition to any minutes that you specify. Seconds must be a number from 0 through 59.

**1440** *or* **NOLIMIT**
Indicates that the job can use the processor for an unlimited amount of time. ("1440" literally means "24 hours.")

Also code TIME=1440 or TIME=NOLIMIT to specify that the system is to allow any of the job's steps to remain in a continuous wait state for more than the installation time limit, which is established through SMF. "Continuous wait time" is defined as time spent waiting while the application program is in control. For example, the time required to recall a data set from HSM Migration Levels 1 or 2 and/or the time required to mount a tape is counted towards the job's continuous wait time if the allocation of the data set was dynamic (that is, issued while the program was running) while the time required for those activities will not be counted towards the job's continuous wait time if the allocation was static (that is, for a DD statement).

**MAXIMUM**
Indicates that the job can use the processor for the maximum amount of time. Coding TIME=MAXIMUM allows a job to run for 357912 minutes.

## Defaults

Every job *step* has a time limit. If you do not specify a TIME parameter on the JOB statement, the time limit for each job step is:

- The value you specify for the TIME parameter on its EXEC statement, or
- The default time limit (that is, the JES default job step time limit), if you do not specify a TIME parameter on its EXEC statement.

If you specify a value other than TIME=NOLIMIT or TIME=1440, SMF uses its current job wait time limit.

## Overrides

For a JOB statement TIME parameter of TIME=NOLIMIT or TIME=1440, the system nullifies any TIME parameters on EXEC statements as well as the default TIME values. All steps within the job will have unlimited processor time.

For a JOB statement TIME parameter other than TIME=NOLIMIT or TIME=1440, the system sets the time limit for each step to one of the following:

- The step time limit specified on the EXEC statement TIME parameter or the job time remaining after execution of previous job steps, whichever is smaller.
- If no EXEC TIME parameter was specified: the default time limit, or the job time remaining after execution of previous steps, whichever is smaller.

## Examples of the TIME parameter

**Note:** The following examples assume the default time limit (set by the installation) to be greater than the TIME=parameter specified in each example.

***Example 1***

```
//STD1  JOB  ACCT271,TIME=(12,10)
```

This statement specifies that the maximum amount of time the job can use the processor is 12 minutes, 10 seconds.

***Example 2***

```
//TYPE41 JOB  ,GORDON,TIME=(,30)
```

This statement specifies that the maximum amount of time the job can use the processor is 30 seconds.

***Example 3***

```
//FORMS  JOB  ,MORRILL,TIME=5
```

This statement specifies that the maximum amount of time the job can use the processor is 5 minutes.

***Example 4***

```
//RAINCK JOB  374231,MORRISON,TIME=NOLIMIT
```

This statement specifies an unlimited amount of time for job execution; the job can use the processor and remain in wait state for an unspecified period of time. The system will issue messages telling how much processor time the job used.

## Examples of the TIME parameter on JOB and EXEC statements

**Note:** The following examples assume the default time limit (set by the installation) to be greater than the TIME=parameter specified in each example.

***Example 1***

```
//FIRST JOB     ,SMITH,TIME=2
//STEP1 EXEC    PGM=READER,TIME=1
         .
         .
         .
//STEP2 EXEC    PGM=WRITER,TIME=1
         .
```

In this example, the job is allowed 2 minutes for execution and each step is allowed 1 minute. If either step continues executing beyond 1 minute, the entire job abnormally terminates beginning with that step.

***Example 2***

```
//SECOND JOB     ,JONES,TIME=3
//STEP1  EXEC    PGM=ADDER,TIME=2
          .
          .
          .
//STEP2  EXEC    PGM=PRINT,TIME=2
          .
```

In this example, the job is allowed 3 minutes for execution, and each step is allowed 2 minutes. If either step continues executing beyond 2 minutes, the entire job abnormally terminates beginning with that step. If STEP1 executes for 1.74 minutes and STEP2 tries to execute beyond 1.26 minutes, the job abnormally terminates because of the 3-minute limit specified on the JOB statement.

# TYPRUN parameter

*Parameter type*

Keyword, optional

**Note:** Do not specify this parameter for a started task; if TYPRUN is specified, the job will fail.

*Purpose*

Use the TYPRUN parameter to request special job processing. The TYPRUN parameter can tell the system to:

- In a JES2 system, copy the input job stream directly to a sysout data set and schedule it for output processing.
- In a JES2 or JES3 system, place a job on hold until a special event occurs. When the event occurs, the operator, following your directions, must release the job from its hold to allow the system to select the job for processing. Use the JES2 /*MESSAGE statement or the JES3 //*OPERATOR statement to notify the operator to release the job.
- In a JES2 or JES3 system, scan a job's JCL for syntax errors.

*Considerations for an APPC scheduling environment*

The TYPRUN parameter has no function in an APPC scheduling environment. If you code TYPRUN, the system will check it for syntax and ignore it.

## Syntax

```
TYPRUN= {COPY   }
        {HOLD   }
        {JCLHOLD}
        {SCAN   }
```

**Note:** The TYPRUN parameter can have a null value only in JES2 systems.

## Subparameter definition

**COPY (JES2 only)**
Requests that JES2 copy the input job stream, as submitted, directly to a sysout data set and schedule the sysout data set for output processing. The system does not schedule the job for execution. The class of this sysout data set is the same as the message class of the job and is controlled by the JOB MSGCLASS parameter.

**Note:** COPY is supported only in JES2 systems.

**HOLD**
Requests that the system hold the job before execution until the operator releases it. The operator should release the job when a particular event occurs. If an error occurs during input service processing, JES does not hold the job.

**JCLHOLD (JES2 only)**
Requests that JES2 hold the job before completing JCL processing. JES2 holds the job until the operator releases it. However, if there are error messages

pending for the job, which surface as a result of completing JCL processing, the job is not held. The job completes JCL processing and then is placed in the OUTPUT queue.

**Note:** JCLHOLD is supported only in JES2 systems.

**SCAN**
Requests that the system scan this job's JCL for syntax errors, without executing the job or allocating devices. This parameter asks the system to check for:

- Spelling of parameter keywords and some subparameter keywords that is not valid.
- Characters that are not valid.
- Unbalanced parentheses.
- Misplaced positional parameters on some statements.
- In a JES3 system only, parameter value errors or excessive parameters.
- Invalid syntax on JCL statements in cataloged procedures invoked by any scanned EXEC statements.

The system does not check for misplaced statements, for invalid syntax in JCL subparameters, or for parameters and/or subparameters that are inappropriate together.

In a JES3 system, the system does not scan the JCL on the submitting system when a //*ROUTE or XMIT JCL statement follows the JOB statement.

TYPRUN=SCAN checks the JCL only through the converter, not the interpreter. The difference is that the converter basically checks all expressions to the LEFT of an equal sign plus SOME expressions to the right of an equal sign (and issues messages that start with IEFC), while the interpreter checks all expressions to the RIGHT of an equal sign (and issues messages that start with IEF). For example, a data set name containing a qualifier that exceeds eight characters, such as

```
DSN=L9755TB.JCL.TEST19970103
```

would NOT be flagged by TYPRUN=SCAN but would be caught by the interpreter.

## Relationship to other control statements

In a JES3 system, code PGM=JCLTEST or PGM=JSTTEST on the EXEC statement to scan a job step's JCL. JCLTEST or JSTTEST provides for a step the same function as provided by TYPRUN=SCAN for a job.

## Example of the TYPRUN parameter

```
//UPDATE JOB   ,HUBBARD
//STEP1  EXEC  PGM=LIBUTIL
          .
          .
          .
//LIST   JOB   ,HUBBARD,TYPRUN=HOLD
//STEPA  EXEC  PGM=LIBLIST
          .
          .
          .
```

Jobs UPDATE and LIST are submitted for execution in the same input stream. UPDATE executes a program that adds and deletes members of a library; LIST

executes a program that lists the members of that library. For an up-to-date listing of the library, LIST must execute after UPDATE. To force this execution order, code TYPRUN=HOLD on JOB statement LIST.

If a MONITOR JOBNAMES command is executed from the input stream or by the operator, the system notifies the console operator when UPDATE completes. The operator can then release LIST, allowing the system to select LIST for execution.

## UJOBCORR parameter

***Parameter type***

Keyword, optional

***Purpose***

Specifies the user portion of the job correlator that will be associated with the current job. The job correlator (JOBCORR parameter) is a 64-byte token that uniquely identifies a job to JES. The JOBCORR value is composed of a 32-byte system portion, which ensures a unique value, and a 32-byte user portion which helps identify the job to the system. The UJOBCORR parameter specifies this 32-byte user portion of the job correlator.

The UJOBCORR value can be overridden when the job is submitted by using the JES SYS_CORR_USRDATA symbol. Both the UJOBCORR and SYS_CORR_USRDATA values can be overridden by JES2 installation exits 2 and 52 for JOB JCL statement scan, and by exits 20 and 50 for end of job input. For information on modifying the user portion of the job correlator using JES2 installation exits, see *z/OS JES2 Installation Exits* For SYS_CORR_USRDATA symbol information, see *z/OS JES Application Programming*.

In JES3 environments, this UJOBCORR parameter is accepted but ignored.

***Networking considerations***

A value that is passed over NJE will override the value that is passed on this parameter—an NJE value comes from the JES symbol service or from installation exits and passes over NJE.

***Additional information***

The job correlator is used to identify the job in multiple interfaces, including:

- JES operator commands
- ENF messaging
- Subsystem interfaces such as extended status and SAPI
- SMF records.

## Syntax

```
UJOBCORR={user-correlator}
```

## Subparameter definition

**user-correlator**
The user portion of the job correlator, 1-32 characters in length. This value must start with an alphabetic or national character, which can be followed by alphanumeric, national, and underscore ('_') characters. If the underscore character is used, then the entire value must be enclosed within single quotation marks (' ').

## Examples of the UJOBCORR parameter

In the following example, the user portion of the job correlator is set to JMAN_COMPILE:

```
//TEST JOB 333,STEVE,UJOBCORR='JMAN_COMPILE'
```

Subsequently, this value will be combined with the system portion of the correlator to form a job correlator similar to the following example:

```
J0000025NODE1...C910E4EC.......:JMAN_COMPILE
|<-system portion------------>||<-user portion--------------->|
```

# USER parameter

***Parameter type***

Keyword, optional

**Note:** Do not specify this parameter for a started task; if USER is specified, the job will fail.

***Purpose***

Code the USER parameter to identify to the system the person submitting the job. The userid is used by RACF, the system resources manager (SRM), and other system components.

If the installation contains the feature for propagation of the user and group identification, the USER and PASSWORD parameters are required, and the GROUP parameter is optional on JOB statements only for the following:

- Batch jobs submitted through an input stream, such as a card reader, (1) if the job requires access to RACF-protected resources or (2) if the installation requires that all jobs have RACF identification.
- Jobs submitted by one RACF-defined user for another user. In this case, the JOB statement must specify the other user's userid and may need a password. The group id is optional.
- Jobs that execute at another network node that uses RACF protection.

Otherwise, the USER, PASSWORD, and GROUP parameters can be omitted from JOB statements. RACF uses the userid, password, and default group id of the submitting TSO/E user or job.

***References***

For more information on RACF-protected facilities, see the *z/OS Security Server RACF Security Administrator's Guide*.

***Considerations for an APPC scheduling environment***

The USER parameter has no function in an APPC scheduling environment. If you code USER, the system will check it for syntax and ignore it.

## Syntax

```
USER=[(]userid[)]
```

## Subparameter definition

**userid**
Identifies a user to the system. The userid consists of 1 through 8 alphanumeric or national ($, #, @) characters; the first character must be alphabetic or national ($, #, @).

## Defaults

When not required by the installation and if the JOB statement or the submitting TSO/E user does not supply identification information, RACF assigns a default userid and group id, unless the job enters the system via a JES internal reader. In this case, the user and default group identification of the submitting TSO/E user or job is used.

## Relationship to other parameters

If the JOB statement contains a GROUP or PASSWORD parameter, the statement must also contain a USER parameter.

## Example of the USER parameter

```
//TEST  JOB  'D83,123456',USER=MYNAME,PASSWORD=ABCD
```

This statement identifies the user submitting this job as MYNAME.

# Chapter 22. Null Statement

Use the null statement to mark the end of a job.

## Description

### Syntax

```
//
```

- The null statement consists of the characters // in columns 1 and 2.
- Columns 3 through 72 **must** be blank.

### Location in the JCL

Place a null statement (1) at the end of a job's control statements and data and (2) at the end of an input stream.

The system can also recognize the end of a job when it reads the next JOB statement or when the input stream contains no more records.

A null statement that does not end an input stream should be immediately followed by a JOB statement. The system ignores statements between a null statement and the next valid JOB statement.

Note: JES2 ignores a NULL statement when it is included in a job's JCL statements. JES2 processes JES2 control statements following a NULL statement as part of the job (until the next JOB statement or EOF). Optionally, JES2 can stop processing JECL statements after the NULL statement. See the description of the INPUTDEF JES2 initialization statement in *z/OS JES2 Initialization and Tuning Reference*.

If a null statement follows a control statement that is being continued, the system treats the null statement as a blank comment field and assumes that the control statement contains no other parameters.

### Example of the null statement

```
//MYJOB JOB   ,'C BROWN'
//STEP1 EXEC  PROC=FIELD
//STEP2 EXEC  PGM=XTRA
//DD1   DD    UNIT=3400-5
//DD2   DD    *
        .
        .
        data
        .
/*
//
```

The null statement indicates the end of job MYJOB.

© Copyright IBM Corp. 1988, 2015

# Chapter 23. OUTPUT JCL statement

***Purpose:*** Use the OUTPUT JCL statement to specify processing options for a system output (sysout) data set. These processing options are used only when the OUTPUT JCL statement is explicitly or implicitly referenced by a sysout DD statement. JES combines the options from this OUTPUT JCL statement with the options from the referencing DD statement.

OUTPUT JCL statements are useful in processing the output of one sysout data set in several ways. For example, a sysout data set can be sent to a distant site for printing, as shown in statement OUT1, while it is also printed locally, as shown in statement OUT2:

```
//OUT1  OUTPUT  DEST=STLNODE.WMSMITH
//OUT2  OUTPUT  CONTROL=DOUBLE
//DS    DD      SYSOUT=C,OUTPUT=(*.OUT1,*.OUT2)
```

The parameters you can specify for sysout data set processing are arranged alphabetically in the following sections.

***References:*** For information about the JES initialization parameters that provide installation defaults, see *z/OS JES2 Initialization and Tuning Reference* and *z/OS JES3 Initialization and Tuning Reference*. For examples of OUTPUT statement processing on the JES3 hold queue and writer queue, see *z/OS JES3 Initialization and Tuning Guide*.

## Description

### Syntax

```
//name  OUTPUT parameter[,parameter]...  [comments]
```

The OUTPUT JCL statement consists of the characters // in columns 1 and 2 and four fields: name, operation (OUTPUT), parameter, and comments.

### Name field

Code a name in the name field of every OUTPUT JCL statement, as follows:

- Each job-level OUTPUT JCL name must be unique within a job.
- Each step-level OUTPUT JCL name must be unique within the same job step.
- The name must begin in column 3.
- The name is 1 through 8 alphanumeric or national ($, #, @) characters.
- The first character must be alphabetic or national ($, #, @).
- The name must be followed by at least one blank.

### Operation field

The operation field consists of the characters OUTPUT and must be preceded and followed by at least one blank. It can begin in any column.

© Copyright IBM Corp. 1988, 2015

## Parameter field

The OUTPUT JCL statement contains only keyword parameters. All parameters are optional; however, do not leave the parameter field blank. You can code any of the keyword parameters in any order in the parameter field.

*Table 27. Keyword parameters*

| KEYWORD PARAMETERS | VALUES | PURPOSE |
|---|---|---|
| `ADDRESS={('delivery address'`<br>`[,'delivery address']...)}`<br>`{delivery-address}`<br><br>See section "ADDRESS parameter" on page 480 | delivery address: 1 - 4 delivery-address subparameters; a delivery-address is 1 - 60 valid EBCDIC text values | Specifies an address to be printed on output separator pages. |
| `AFPPARMS=datasetname`<br><br>See section "AFPPARMS parameter" on page 482 | datasetname must be a sequential data set.<br><br>datasetname must be cataloged.<br><br>See the DSNAME parameter on the DD statement for additional syntax rules. | Use the AFPPARMS keyword to reference the data set name which specifies the parameter file that contains the parameters and values for the AFP print distributor feature of PSF. The parameters specified in this parameter file augment parameters specified on the output JCL statement. |
| `AFPSTATS= {YES}`<br>`{Y  }`<br>`{NO }`<br>`{N  }`<br><br>See section "AFPSTATS parameter" on page 483 | YES or Y: Requests that PSF produce an AFPSTATS report for the printing of this sysout data set.<br><br>NO or N: Specifies that PSF should not produce an AFPSTATS report for the printing of this sysout data set. | Specifies to Print Services Facility (PSF) that an AFP Statistics About the Printfile (AFPSTATS) report is to be generated while printing this sysout data set. |
| `BUILDING= {'building identification'}`<br>`{building-identification  }`<br><br>See section "BUILDING parameter" on page 484 | building identification: 1 - 60 valid EBCDIC text values | Specifies a building location to be printed on output separator pages. |
| `BURST= {YES}`<br>`{Y  }`<br>`{NO }`<br>`{N  }`<br><br>See section "BURST parameter" on page 485 | YES or Y: burster-trimmer-stacker<br>NO or N: continuous forms stacker | Directs output to a stacker on a 3800 Printing Subsystem. |
| `CHARS= {font-name  }`<br>`{(font-name[,font-name]...)}`<br>`{STD  }`<br>`{DUMP  }`<br>`{(DUMP[,font-name]...)  }`<br><br>See section "CHARS parameter" on page 486 | 1 - 4 font-name subparameters:<br>1 - 4 alphanumeric or national ($, #, @) characters<br>STD: character-arrangement table (JES3 only)<br>DUMP: 204-character print lines on 3800 dump | Names character-arrangement tables for printing on an AFP printer. Can request a high-density dump on a SYSABEND or SYSUDUMP DD statement. |
| `CKPTLINE=nnnnn`<br><br>See section "CKPTLINE parameter" on page 488 | nnnnn: 0 - 32,767 | Specifies the maximum lines in a logical page. (JES3 supports this parameter only when PSF prints the sysout data set on an AFP printer.) |
| `CKPTPAGE=nnnnn`<br><br>See section "CKPTPAGE parameter" on page 488 | nnnnn: 1 - 32,767 | Specifies the number of logical pages to be printed or transmitted before JES takes a checkpoint. (JES3 supports this parameter only when PSF prints the sysout data set on an AFP printer.) |

*Table 27. Keyword parameters (continued)*

| KEYWORD PARAMETERS | VALUES | PURPOSE |
|---|---|---|
| `CKPTSEC=nnnnn`<br><br>See section "CKPTSEC parameter" on page 489 | nnnnn: 1 - 32,767 | Specifies how many seconds of printing are to elapse between each checkpoint of this sysout data set. (JES3 supports this parameter only when PSF prints the sysout data set on an AFP printer.) |
| `CLASS= {class}`<br>`{*    }`<br><br>See section "CLASS parameter" on page 490 | class: A - Z, 0 - 9 *: same output class as MSGCLASS parameter on JOB statement | Assigns the sysout data set to an output class. |
| `COLORMAP=resource`<br><br>See section "COLORMAP parameter" on page 492 | resource: 1 - 8 alphanumeric or national ($, #, @) characters | Specifies the AFP resource (object) for the data set that contains color translation information. |
| `COMPACT=compaction-font-name`<br><br>See section "COMPACT parameter" on page 492 | compaction-font-name: 1 - 8 alphanumeric characters | Specifies a compaction table for sending this sysout data set to a SNA remote terminal. |
| `COMSETUP=resource`<br><br>See section "COMSETUP parameter" on page 493 | resource: 1 - 8 alphanumeric or national ($, #, @) characters | Specifies the name of a macrofile setup resource that contains the SETUP information. |
| `CONTROL= {PROGRAM}`<br>`{SINGLE }`<br>`{DOUBLE }`<br>`{TRIPLE }`<br><br>See section "CONTROL parameter" on page 494 | PROGRAM: each logical record begins with a carriage control character<br><br>SINGLE: single spacing<br>DOUBLE: double spacing<br>TRIPLE: triple spacing | Specifies that the data set records begin with carriage control characters or specifies line spacing. |
| `COPIES= {nnn                              }`<br>`{(,(group-value[,group-value]...))}`<br><br>See section "COPIES parameter" on page 495 | nnn (JES2): 1 - 255<br>nnn (JES3): 0 - 255<br>1 - 8 group-values (JES2): 1 - 255<br>1 - 8 group values (JES3): 1 - 254 | Specifies number of copies printed. For an AFP printer, can instead specify number of copies of each page printed before the next page is printed. |
| `COPYCNT= {xxx}`<br><br>See section "COPYCNT parameter" on page 497 | xxx: 0 - 2147483647 | Specifies number of copies printed where the limit is 2G in size. |
| `DATACK= {BLOCK  }`<br>`{UNBLOCK}`<br>`{BLKCHAR}`<br>`{BLKPOS }`<br><br>See section "DATACK parameter" on page 498 | BLOCK: indicates errors are not reported<br>UNBLOCK: indicates errors are reported<br>BLKCHAR: indicates print errors are blocked<br>BLKPOS: indicates data errors are blocked | Indicates whether or not print-positioning errors and invalid character data-check errors are to be blocked or not blocked. |
| `DDNAME= {`*`ddname`*`}`<br>`{`*`procstepname.ddname`*`}`<br>`{`*`stepname.ddname`*`}`<br>`{`*`stepname.procstepname.ddname`*`}`<br><br>See section "DDNAME parameter" on page 499 | *ddname*: DD to apply OUTPUT specifications to.<br><br>*procstepname.ddname*: DD to apply OUTPUT specifications to, with preceding PROCSTEP name.<br><br>*stepname.ddname*: DD to apply OUTPUT specifications to, with preceding STEP name.<br><br>*stepname.procstepname.ddname*: DD to apply OUTPUT specifications to, with preceding STEP and PROCSTEP names. | Specifies the DDs to apply the specifications on the OUTPUT statement. |

*Table 27. Keyword parameters (continued)*

| KEYWORD PARAMETERS | VALUES | PURPOSE |
|---|---|---|
| DEFAULT= {YES}<br>{Y }<br>{NO }<br>{N }<br><br>See section "DEFAULT parameter" on page 500 | YES or Y: this statement can be implicitly referenced by sysout DD statements<br><br>NO or N: this statement cannot be implicitly referenced by sysout DD statements. | Specifies that this is a default OUTPUT JCL statement. |
| DEPT= {'department identification'}<br>{department-identification }<br><br>See section "DEPT parameter" on page 502 | department identification: 1 - 60 valid EBCDIC text values | Specifies a department identification to be printed on output separator pages. |
| DEST=destination<br><br>destination (JES2):<br>LOCAL<br>'IP:ipaddr'<br>name<br>Nnnnnn<br>NnRmmmmm to NnnnnnRm<br>(node,remote)<br>nodename.userid<br>'nodename.IP:ipaddr'<br>Rnnnnn or RMnnnnn or RMTnnnnn<br>Unnnnn<br><br>destination (JES3):<br>ANYLOCAL<br>'IP:ipaddr'<br>device-name<br>group-name<br>nodename<br>'nodename.IP:ipaddr'<br>nodename.remote<br><br>See section "DEST parameter" on page 503 | LOCAL: local device<br>ipaddr identifies a TCP/IP routing designation.<br>name: named local or remote device<br>Nnnnnn: node (1 - 32,767)<br>NnRmm: node (1 - 32,767) and remote work station (1 - 32,767); 6 digits<br>maximum for n and m combined<br>nodename.userid: node (1 - 8 alphanumeric characters) and userid (1 - 8 alphanumeric characters)<br>Rnnnnn or RMnnnnn or RMTnnnnn: remote terminal (1 - 32,767)<br>Unnnnn: local terminal (1 - 32,767)<br>ANYLOCAL: any local device<br>device-name: local device (1 - 8 alphanumeric or national ($, #, @) characters)<br>group-name: 1 or more local devices or remote stations (1 - 8 alphanumeric or national ($, #, @) characters)<br>nodename: node (1 - 8 alpha-numeric or national ($, #, @) characters)<br>remote: remote workstation (1 - 8 alphanumeric or national ($, #, @) characters) | Sends a sysout data set to the specified destination. |
| DPAGELBL= {YES}<br>{Y }<br>{NO }<br>{N }<br><br>See section "DPAGELBL parameter" on page 507 | YES or Y: requests that the system print a security label on each page.<br><br>NO or N: requests that the system not print a security label on each page. | Indicates whether the system should print a security label on each page of output. |
| DUPLEX= {NO }<br>{N }<br>{NORMAL}<br>{TUMBLE}<br><br>See section "DUPLEX parameter" on page 508 | X'80' for NO<br>X'40' for NORMAL<br>X'20' for TUMBLE | Specifies whether the job prints on one or both sides of the paper. Overrides comparable FORMDEF. |
| FCB= {fcb-name}<br>{STD }<br><br>See section "FCB parameter" on page 509 | fcb-name: 1 - 4 alphanumeric or national ($, #, @)<br><br>STD: standard FCB (JES3 only) | Specifies FCB image, carriage control tape for 1403 Printer, or data-protection image for 3525 Card Punch. |

*Table 27. Keyword parameters (continued)*

| KEYWORD PARAMETERS | VALUES | PURPOSE |
|---|---|---|
| `FLASH= {overlay-name }`<br>`{(overlay-name[,count])}`<br>`{(,count) }`<br>`{NONE }`<br>`{STD }`<br><br>See section "FLASH parameter" on page 511 | overlay-name: forms overlay frame (1 - 4 alphanumeric or national ($, #, @) characters)<br><br>count: copies with overlay (0 - 255)<br><br>NONE: suppresses flashing<br><br>STD: standard forms flash overlay (JES3 only) | For printing on a 3800 Printing Subsystem, indicates that the data set is to be printed with forms overlay and can specify how many copies are to be flashed. |
| `FORMDEF=membername`<br><br>See section "FORMDEF parameter" on page 513 | membername: 1 - 6 alphanumeric or national ($, #, @) characters | Names a library member that PSF uses in printing the sysout data set on an AFP printer. |
| `FORMLEN=nn[.mmm]{IN\|CM}`<br><br>See section "FORMLEN parameter" on page 514 | nn= 0-99<br>mmm= 0-999<br>IN= inches<br>CM= centimeters | Allows PSF users to set the length of pages for print without reconfiguring the printer. |
| `FORMS= {form-name}`<br>`{STD }`<br><br>See section "FORMS parameter" on page 515 | form-name: 1 - 8 alphanumeric or national ($, #, @) characters<br><br>STD: standard form (JES3 only) | Identifies forms on which the sysout data set is to be printed or punched. |
| `FSSDATA=value`<br><br>See section "FSSDATA parameter" on page 515 | values: 1-127 EBCDIC characters | Defined by a functional subsystem. Refer to that subsystem's documentation for the intent and use of this keyword. |
| `GROUPID=output-group`<br><br>See section "GROUPID parameter" on page 518 | output-group: 1 - 8 alphanumeric characters | Specifies that this sysout data set belongs to a user-named output group. (JES2 only) |
| `INDEX=nn`<br><br>See section "INDEX parameter" on page 519 | nn: 1 - 31 | Specifies how many print positions the left margin is to be indented for a sysout data set printed on a 3211 Printer with the indexing feature. (JES2 only) |
| `INTRAY=nnn`<br><br>See section "INTRAY parameter" on page 520 | nnn: 1 - 255 | Specifies the printer input tray from which to take paper for the print job. Overrides comparable FORMDEF specification. |
| `JESDS= {ALL}`<br>`{JCL}`<br>`{LOG}`<br>`{MSG}`<br><br>See section "JESDS parameter" on page 521 | ALL: all of the job's JCL, LOG, and MSG data sets<br><br>JCL: all JCL processing data sets<br>LOG: job's hard-copy log<br>MSG: job's system messages | Requests that the indicated data sets for the job be processed according to the parameters on this OUTPUT JCL statement. |
| `LINDEX=nn`<br><br>See section "LINDEX parameter" on page 523 | nn: 1 - 31 | Specifies how many print positions to move the right margin in from the full page width for a sysout data set printed on a 3211 Printer with the indexing feature. (JES2 only) |
| `LINECT=nnn`<br><br>See section "LINECT parameter" on page 524 | nnn: 0 - 255 | Specifies the maximum lines JES2 is to print on each page. (JES2 only) |
| `MAILBCC= {('bcc address'['bcc address']...)}`<br>`{bcc-address }`<br><br>See section "MAILBCC parameter" on page 524 | bcc address: 1-60 valid EBCDIC text values<br><br>bcc address: 1 - 32 addresses allowed | Specifies the e-mail addresses of the recipients on the blind copy list. |

*Table 27. Keyword parameters (continued)*

| KEYWORD PARAMETERS | VALUES | PURPOSE |
|---|---|---|
| `MAILCC= {('cc address'[,'cc address']...)}`<br>`{cc-address }`<br><br>See section "MAILCC parameter" on page 525 | cc address: 1-60 valid EBCDIC text values<br><br>cc address: 1 - 32 addresses allowed | Specifies the e-mail addresses of the recipients on the copy list. |
| `MAILFILE= {'file id'}`<br>`{file-id }`<br><br>See section "MAILFILE parameter" on page 526 | file id: 1- 60 valid EBCDIC text values | Specifies the file name of the attachment to an e-mail. |
| `MAILFROM= {'from address'}`<br>`{from-address }`<br><br>See section "MAILFROM parameter" on page 527 | from address: 1- 60 valid EBCDIC text values | Specifies the descriptive name or identifier of the sender of an e-mail. |
| `MAILTO= {('to address'[,'to address']...)}`<br>`{to-address }`<br><br>See section "MAILTO parameter" on page 528 | to address: 1- 60 valid EBCDIC text values<br><br>to address: 1 - 32 addresses allowed | Specifies the e-mail addresses of the e-mail recipients. |
| `MERGE= {YES}`<br>`{Y }`<br>`{NO }`<br>`{N }`<br><br>See section "MERGE parameter" on page 529 | YES or Y: OUTPUT JCL parameters will be the job's default<br><br>NO or N: OUTPUT JCL parameters will not be the job's default | Indicates whether or not the parameters specified on the OUTPUT JCL statement will be the default OUTPUT parameters for the job. |
| `MODIFY= {module-name }`<br>`{([module-name][,trc])}`<br><br>See section "MODIFY parameter" on page 529 | module-name: 1 - 4 alphanumeric or national ($, #, @) characters trc: font-name in CHARS parameter (0 for first, 1 for second, 2 for third, and 3 for fourth font-name) | Specifies a copy-modification module in SYS1.IMAGELIB to be used by JES to print the data set on a 3800 Printing Subsystem. |
| `NAME= {'preferred name'}`<br>`{preferred-name }`<br><br>See section "NAME parameter" on page 531 | preferred name: 1 - 60 valid EBCDIC text values | Specifies the preferred name to be printed on output separator pages. |
| `NOTIFY= {[node.]userid }`<br>`{([node.]userid1,...[node.]userid4)}`<br><br>See section "NOTIFY parameter" on page 532 | {node.}userid: node and userid to receive print complete message. | Specifies the node and userid to receive a print complete message when the sysout data set is printed. |
| `OFFSETXB=mmmm[.nnn]{IN }`<br>`{CM }`<br>`{MM }`<br>`{PELS }`<br>`{POINTS}`<br><br>See section "OFFSETXB parameter" on page 533 | mmmm: 0 - 9999<br>nnn: 0 - 999<br>IN: inches<br>CM: centimeters<br>MM: millimeters | Specifies the offset in the X direction from the page origin (or partition origin for N_UP) for the back side of each output page. |
| `OFFSETXF=mmmm[.nnn]{IN }`<br>`{CM }`<br>`{MM }`<br>`{PELS }`<br>`{POINTS}`<br><br>See section "OFFSETXF parameter" on page 534 | mmmm: 0 - 9999<br>nnn: 0 - 999<br>IN: inches<br>CM: centimeters<br>MM: millimeters | Specifies the offset in the X direction from the page origin (or partition origin for N_UP) for the front side of each output page. |
| `OFFSETYB=mmmm[.nnn]{IN }`<br>`{CM }`<br>`{MM }`<br>`{PELS }`<br>`{POINTS}`<br><br>See section "OFFSETYB parameter" on page 534 | mmmm: 0 - 9999<br>nnn: 0 - 999<br>IN: inches<br>CM: centimeters<br>MM: millimeters | Specifies the offset in the Y direction from the page origin (or partition origin for N_UP) for the back side of each output page. |
| `OFFSETYF=mmmm[.nnn]{IN }`<br>`{CM }`<br>`{MM }`<br>`{PELS }`<br>`{POINTS}`<br><br>See section "OFFSETYF parameter" on page 534 | mmmm: 0 - 9999<br>nnn: 0 - 999<br>IN: inches<br>CM: centimeters<br>MM: millimeters | Specifies the offset in the Y direction from the page origin (or partition origin for N_UP) for the front side of each output page. |

*Table 27. Keyword parameters (continued)*

| KEYWORD PARAMETERS | VALUES | PURPOSE |
|---|---|---|
| `OUTBIN=nnnnn`<br><br>See section "INDEX parameter" on page 519 | nnnnn: 1 - 65535 | Specifies the ID of the printer output bin where the data set is to be sent. |
| `OUTDISP=[normal-output-disposition, abnormal-output-disposition]`<br><br>See section "OUTDISP parameter" on page 535 | normal output disposition: WRITE, HOLD, KEEP, LEAVE, or PURGE.<br><br>abnormal output disposition: WRITE, HOLD, KEEP, LEAVE, or PURGE. | Specifies the disposition of the sysout process instance for normal or, in a non-APPC scheduling environment, abnormal termination of the job step. |
| `OVERLAYB=name`<br><br>See section "OVERLAYB parameter" on page 537 | name: 1 - 8 alphanumeric or national ($, #, @) characters | Specifies placing the named medium overlay on the back side of each printed sheet. |
| `OVERLAYF=name`<br><br>See section "OVERLAYF parameter" on page 538 | name: 1 - 8 alphanumeric or national ($, #, @) characters | Specifies placing the named medium overlay on the front side of each printed sheet. |
| `OVFL= {ON }`<br>`{OFF}`<br><br>See section "OVFL parameter" on page 538 | ON: JES3 should check for forms overflow on an output printer.<br><br>OFF: JES3 should not check for forms overflow on an output printer. | Specifies whether or not JES3 should check for forms overflow on an output printer. (JES3 only) |
| `PAGEDEF=membername`<br><br>See section "PAGEDEF parameter" on page 539 | membername: 1 - 6 alphanumeric or national ($, #, @) characters | Names a library member that PSF uses in printing the sysout data set on an AFP printer. |
| `PIMSG= {(YES[,msg-count])}`<br>`{(NO[,msg-count]) }`<br><br>See section "PIMSG parameter" on page 540 | YES: print messages from a functional subsystem<br><br>NO: not print messages from a functional subsystem<br><br>msg-count: number of errors to cause printing to be terminated (0-999) | Indicates that messages from a functional subsystem should or should not be printed in the listing following the sysout data set. |
| `PORTNO=nnnnn`<br><br>See section "PORTNO parameter" on page 542 | nnnnn: 1 - 65535 | Specifies the TCP port number at which Infoprint Server connects to the printer rather than connecting to LPD on the printer. Specify either PORTNO or PRTQUEUE, but not both. PRTQUEUE indicates the queue used when connecting to LPD on the printer. |
| `PRMODE= {LINE }`<br>`{PAGE }`<br>`{process-mode}`<br><br>See section "PRMODE parameter" on page 542 | LINE: send data set to line-mode printer<br><br>PAGE: send data set to page-mode printer<br><br>process-mode: installation-defined mode (1 - 8 alphanumeric characters) | Identifies the process mode required to print the sysout data set. |
| `PRMODE= {LINE }`<br>`{PAGE }`<br>`{process-mode}`<br><br>See section "PRMODE parameter" on page 542 | LINE: send data set to line-mode printer<br><br>PAGE: send data set to page-mode printer<br><br>process-mode: installation-defined mode (1 - 8 alphanumeric characters) | Identifies the process mode required to print the sysout data set. |

*Table 27. Keyword parameters (continued)*

| KEYWORD PARAMETERS | VALUES | PURPOSE |
|---|---|---|
| `PRTATTRS= {'attributename=value attributename=value ...'}`<br><br>See section "PRTATTRS parameter" on page 543 | The minimum length is one character.<br><br>The maximum length is 127 characters.<br><br>Enclose the parameter in apostrophes because attribute names contain lower case letters.<br><br>All EBCDIC text characters are valid. | Use the PRTATTRS keyword to specify one or more job attributes for Infoprint Server. See *z/OS Infoprint Server User's Guide*. |
| `PRTERROR= {DEFAULT}`<br>`{QUIT }`<br>`{HOLD }`<br><br>See section "PRTERROR parameter" on page 544 | DEFAULT= Specifies that PSF will take the standard action if a terminating error occurs during printing. This is the default.<br><br>QUIT= Specifies that PSF will release the data set complete even if a terminating error occurs during printing.<br><br>HOLD= Specifies that if a terminating error occurs during printing, the data set will remain on the JES SPOOL until the system operator releases it. | On the OUTPUT statement or dynamic output descriptor, indicates the disposition of the SYSOUT data set to use if a terminating error occurs during printing of the SYSOUT data with the PSF functional subsystem. |
| `PRTOPTNS= {options data set entry name }`<br>`{'options data set entry name'}`<br><br>See section "PRTOPTNS parameter" on page 545 | print options: 1-16 valid EBCDIC characters. | Identifies the print options data. |
| `PRTQUEUE= {print queue name }`<br>`{'print queue name'}`<br><br>See section "PRTQUEUE parameter" on page 546 | print queue: 1-127 valid EBCDIC characters. | Identifies the target print queue name. |
| `PRTY=nnn`<br><br>See section "PRTY parameter" on page 546 | nnn: 0 - 255 (0 is lowest, 255 is highest) | Specifies initial priority at which the sysout data set enters the output queue. |
| `REPLYTO= {'reply address' }`<br>`{reply-address }`<br><br>See section "REPLYTO parameter" on page 547 | reply address: 1- 60 valid EBCDIC text values | Specifies the e-mail address to which recipients can respond. |
| `RESFMT= {P240}`<br>`{P300}`<br><br>See section "RESFMT parameter" on page 548 | P240: specifies 240 pels per inch resolution.<br><br>P300: specifies 300 pels per inch resolution. | Specifies the resolution used to format the print data set. |
| `RETAINS= {'<hhhh>:<mm>:<ss>'}`<br>`{FOREVER }`<br><br>See section "RETAINS *and* RETAINF parameters" on page 548 | retain time: 1-10 numeric characters or FOREVER. | The successful transmission retain time specification. |
| `RETAINF= {'<hhhh>:<mm>:<ss>'}`<br>`{FOREVER }`<br><br>See section "RETAINS *and* RETAINF parameters" on page 548 | retain time: 1-10 numeric characters or FOREVER. | The failed transmission retain time specification. |
| `RETRYL= {nnn}`<br><br>See section "RETRYL *and* RETRYT parameters" on page 550 | nnnnn: one to five numeric characters. | The maximum number of retries. |
| `RETRYT= {'<hh>:<mm>:<ss>'}`<br><br>See section "RETRYL *and* RETRYT parameters" on page 550 | retry time: 1-10 numeric characters. | Wait time between transmission retries. |
| `ROOM= {'room identification'}`<br>`{room-identification }`<br><br>See section "ROOM parameter" on page 551 | room identification: 1-60 valid EBCDIC text values | Specifies a room identification to be printed on output separator pages. |

*Table 27. Keyword parameters (continued)*

| KEYWORD PARAMETERS | VALUES | PURPOSE |
|---|---|---|
| SYSAREA= {YES}<br>{Y }<br>{NO }<br>{N }<br><br>See section "SYSAREA parameter" on page 552 | YES or Y: requests that the system reserve a system area.<br><br>NO or N: requests that the system not reserve a system area. | Indicates whether the system should reserve a system area on each page of output. |
| THRESHLD=limit<br><br>See section "THRESHLD parameter" on page 553 | limit: 1 - 99999999 | Specifies the maximum size for a sysout data set. Use it to obtain simultaneous printing of large data sets or many data sets from one job. (JES3 only) |
| TITLE= {'description of output'}<br>{description-of-output }<br><br>See section "TITLE parameter" on page 554 | description of output: 1 - 60 valid EBCDIC characters | Identifies a report title to be printed on separator pages. |
| TRC= {YES}<br>{Y }<br>{NO }<br>{N }<br><br>See section "TRC parameter" on page 555 | YES or Y: data set contains TRC codes<br><br>NO or N: data set does not contain TRC codes | Specifies whether or not the sysout data set's records contain table reference codes (TRC) as the second character. |
| UCS=character-set-code<br><br>See section "UCS parameter" on page 556 | character-set-code: 1 - 4 alphanumeric or national ($, #, @) characters | Specifies universal character set, print train, or character-arrangement table for an AFP printer. |
| USERDATA=value<br>(value[,value]...)<br><br>See section "USERDATA parameter" on page 558 | From 1 to 16 values; each value may be from 1 to 60 EBCDIC characters. | Defined by the installation. Refer to your installation's definition on the intent and use of this keyword. If your installation does not define any use for this keyword, the information will be syntax checked, stored as part of the output descriptor's information, and then ignored. |
| USERLIB={data-set-name }<br>{(data-set-name1,data-set-name2,<br>...data-set-name8)}<br><br>See section "USERLIB parameter" on page 561 | data-set-name: 1 - 8 library data set names containing AFP resources | Identifies libraries containing AFP resources for PSF to use when processing sysout data sets. |
| USERPATH={path }<br>{(path1,path2,<br>...path8)} | pathname: 1 - 8 z/OS UNIX file paths. For more information on specifying paths, see "PATH parameter" on page 237. | Specifies a private path for TrueType/OpenType fonts to PSF for the print application owner. |
| WRITER=name<br><br>See section "WRITER parameter" on page 564 | name: 1 - 8 alphanumeric characters | Names an external writer to process the sysout data set rather than JES. |

***Default OUTPUT JCL statement:*** An OUTPUT JCL statement that contains a DEFAULT=YES parameter is called a default OUTPUT JCL statement.

## Using enclosing apostrophes in OUTPUT parameters

Several of the parameters (such as ADDRESS, BUILDING, MAILBCC, and USERDATA) on the OUTPUT JCL statement have variables that can be specified with or without apostrophes. The rules governing the use of apostrophes are as follows:

**Valid characters within enclosing apostrophes**:

- A variable enclosed in apostrophes can contain any EBCDIC text character.
- Enclose a value that contains a blank in apostrophes.
- To code an apostrophe in a variable, code 2 apostrophes, and enclose the entire delivery address in single apostrophes. For example, you might code the ADDRESS parameter as follows:

```
//OUTDS   OUTPUT  ADDRESS='O''DARBY AVE'
```

- Each value may optionally be enclosed in apostrophes.

**Valid characters without enclosing apostrophes**: When a variable is not enclosed in apostrophes, the following characters are valid:

- Alphanumeric and national (@, $, #) characters
- Period (.) and asterisk (*); however, an asterisk followed by a period indicates a referral and is **not** allowed as the start (first and second characters) of the value.
- Ampersand (&). An ampersand that refers to a symbolic is substituted. Two consecutive ampersands are not substituted, but they will result in a single ampersand as part of the value.
- Plus sign (+)
- Hyphen (-)
- Slash (/)

**Null Subparameters:** You may code a null subparameter to cause a blank line to appear in the delivery address. Code a comma to indicate the omitted subparameter.

**Symbolic Parameters**: Do not enclose symbolic parameters within apostrophes. Symbolic parameters enclosed in apostrophes are not resolved for this keyword.

## Comments field

The comments field follows the parameter field after at least one intervening blank.

## Location in the JCL

***References by sysout DD statements:*** An OUTPUT JCL statement can be referenced by a sysout DD statement in two ways:

- **Explicitly.** The sysout DD statement contains an OUTPUT parameter that specifies the name of the OUTPUT JCL statement. You must place the OUTPUT JCL statement in the input stream before any sysout DD statement that refers to it.
- **Implicitly.** The sysout DD statement does not contain an OUTPUT parameter. Implicit references are to default OUTPUT JCL statements. The sysout DD statement implicitly references all step-level default OUTPUT JCL statements in the same step.

**Note:** If the sysout DD statement does not contain an OUTPUT parameter and the job or step does not contain a default OUTPUT JCL statement, processing of the sysout data set is controlled only by the DD statement, a JES2 /*OUTPUT statement or a JES3 //*FORMAT statement, and appropriate installation defaults.

***Job-level OUTPUT JCL statements:*** This statement appears after the JOB statement and before the first EXEC statement. It cannot be used for a started procedure.

***Step-level OUTPUT JCL statements:*** This statement appears in a step, that is, anywhere after the first EXEC statement in a job, except within a concatenated DD statement.

***Location of default OUTPUT JCL statements:*** Where you place default OUTPUT JCL statements determines to which statements a sysout DD statement implicitly refers. A sysout DD statement implicitly references all job-level default OUTPUT JCL statements when the step containing the DD statement does not contain any step-level default OUTPUT JCL statements.

You can place more than one job- or step-level default OUTPUT JCL statement in a job or step.

***OUTPUT JCL statement with JESDS parameter:*** Place an OUTPUT JCL statement with a JESDS parameter after the JOB statement and before the first EXEC statement.

***OUTPUT JCL statements in cataloged or in-stream procedures:*** OUTPUT JCL statements can appear in procedure steps. The referencing DD statement can appear later in the procedure, in the calling job step, or in a later step in the job.

An OUTPUT JCL statement must not be placed before the first EXEC statement in a procedure; for this reason, procedures cannot contain job-level OUTPUT JCL statements or OUTPUT JCL statements with JESDS parameters.

A procedure DD statement can refer to an OUTPUT JCL statement in an earlier job step or to a job-level OUTPUT JCL statement. However, a procedure DD statement cannot refer to an OUTPUT JCL statement in the calling step.

*Table 28. Job- and Step-Level OUTPUT JCL Statements in the JCL*

| Job/Step | Statement | Description |
|---|---|---|
| *Job in Input Stream* | `//jobname JOB ...`<br>`//name OUTPUT ...` | Job-level OUTPUT JCL statement |
| *Step 1* | `//STEP1 EXEC PGM=X`<br>`//name OUTPUT ...`<br>`//DD1 DD ...`<br>`//DD2 DD ...`<br>`//DD3 DD ...` | Step-level OUTPUT JCL statement for STEP1 |
| *Step 2* | `//STEP2 EXEC PROC=A`<br>`//name OUTPUT ...`<br>`//DD1 DD ...`<br>`//DD2 DD ...`<br>`//DD3 DD ...` | Step-level OUTPUT JCL statement for STEP2 |
| *Procedure A in SYS1.PROCLIB*<br>*Procedure Step 1* | `// PROC ...`<br>`//PSTEP1 EXEC PGM=G`<br>`//name OUTPUT ...`<br>`//DD4 DD ...`<br>`//DD5 DD ...`<br>`//DD6 DD ...` | Step-level OUTPUT JCL statement for PSTEP1 |
| *Procedure Step 2* | `//PSTEP2 EXEC PGM=H`<br>`//name OUTPUT ...`<br>`//DD7 DD ...`<br>`//DD8 DD ...`<br>`//DD9 DD ...` | Step-level OUTPUT JCL statement for PSTEP2 |

## Overrides

- Parameters on a sysout DD statement override corresponding parameters on an OUTPUT JCL statement.
- In a JES2 system, either HOLD=YES or HOLD=NO on the DD statement overrides the sysout data set disposition specified on the OUTDISP parameter of the OUTPUT JCL statement.

- In a JES3 system, HOLD=YES on the DD statement is applied along with the sysout data set disposition specified on the OUTDISP parameter of the OUTPUT JCL statement. HOLD=NO on the DD statement overrides the sysout data set disposition specified on the OUTDISP parameter of the OUTPUT JCL statement.

### Relationship to sysout DD statement

Do not refer to an OUTPUT JCL statement in a sysout DD statement that defines a JES internal reader. Such a DD statement contains an INTRDR subparameter in the SYSOUT parameter.

DDNAME= keyword is ignored when the OUTPUT= keyword is coded on the DD statement for a sysout DD.

### Relationship to the JES2 /*OUTPUT statement

JES2 ignores a JES2 /*OUTPUT statement when either of the following appears in the same job or step:

- A default OUTPUT JCL statement implicitly referenced by the sysout DD statement.
- An OUTPUT JCL statement explicitly referenced by the OUTPUT parameter of the sysout DD statement.

In this case, JES2 uses the third positional subparameter of the DD SYSOUT parameter as a form name, and not as a reference to a JES2 /*OUTPUT statement.

### Relationship to the JES3 //*FORMAT statement

- When a sysout DD statement implicitly or explicitly references an OUTPUT JCL statement, JES3 ignores any default JES3 //*FORMAT statements in the job. A default //*FORMAT statement contains a DDNAME=, parameter.
- When a JES3 //*FORMAT statement contains a DDNAME parameter that explicitly references a sysout DD statement, JES3 ignores any default OUTPUT JCL statements in the job.
- JES3 uses the processing options from both a JES3 //*FORMAT statement and an OUTPUT JCL statement in a job when (1) the //*FORMAT statement DDNAME parameter names a sysout DD statement and (2) the sysout DD statement's OUTPUT parameter names an OUTPUT JCL statement. Two separate sets of output are created from the data set defined by the sysout DD statement:
  - One processed according to the options on the JES3 //*FORMAT statement combined with the sysout DD statement.
  - One processed according to the options on the OUTPUT JCL statement combined with the sysout DD statement.

For more information on the use of the OUTPUT JCL statement with JES3, see *z/OS JES3 Initialization and Tuning Guide*.

## ADDRESS parameter

Keyword, optional

***Purpose***

Use the ADDRESS parameter to print an address on the separator pages of an output data set. An installation can use the address to assist in sysout distribution.

## Syntax

```
ADDRESS= {('delivery address'[,'delivery address']...)}
         {delivery-address                            }
```

See "Using enclosing apostrophes in OUTPUT parameters" on page 477 for the delivery address.

## Subparameter definition

**delivery address**
Specifies the delivery address for the output data set. You can code up to 4 delivery addresses. Each delivery address can be 1 - 60 EBCDIC text characters. See "Character sets" on page 21 for a description of EBCDIC text characters.

## Defaults

- **In an APPC scheduling environment:** In both JES2 and JES3 systems, if you do not code ADDRESS, the system uses the value defined in the transaction program (TP) user's RACF profile when:
  - The user submitting the TP profile has a RACF profile defined for him, and
  - The transaction program profile includes TAILOR_SYSOUT(YES).
- **In a non-APPC scheduling environment:** There is no default for the ADDRESS parameter on the OUTPUT JCL statement.

## Overrides

- **In an APPC scheduling environment:** In both JES2 and JES3 systems, the ADDRESS parameter on the OUTPUT JCL statement overrides the address in the RACF profile.
- **In a non-APPC scheduling environment:** In both JES2 and JES3 systems, there are no override considerations for ADDRESS.

## Examples of the ADDRESS parameter

***Example 1***

```
//OUTDS2   OUTPUT  ADDRESS=('J. Plant','1234 Main Street',
//           'POUGHKEEPSIE, NY','zipcd')
```

In this example, the address

J. Plant
1234 Main Street
POUGHKEEPSIE, NY
zipcd

is printed on the separator pages of each output data set that references OUTDS2. You may code a name in the address field when the name associated with an address is not the name you want to associate with the output (coded on the OUTPUT NAME statement.) The name appears in the address field on the separator pages.

***Example 2***

```
//OUTDS3   OUTPUT  ADDRESS=(,'57 FAIR LANE','OMAHA,NE',12121)
```

In this example, the following address is printed on the separator pages of each output data set that references OUTDS3.

```
57 FAIR LANE
OMAHA,NE
12121
```

The first line reserved for the address on the separator page will be blank. Note that 12121 does not require enclosing apostrophes, because it contains only characters that are valid without them.

# AFPPARMS parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the AFPPARMS keyword to reference the data set name which specifies the parameter file that contains the parameters and values for the AFP print distributor feature of PSF. The parameters specified in this parameter file augment parameters specified on the output JCL statement.

***References:*** For more information, see *PSF for z/OS: User's Guide* and *PSF for z/OS: Customization*.

## Syntax

```
AFPPARMS=datasetname
```

## Parameter definition

*datasetname*
Specifies the data set name that AFP print distributor uses to locate the parameter file.

## Defaults

No Default.

## Overrides

None.

## Relationship to other control statements

None.

## Example of the AFPPARM keyword

```
//OUTPUT1   OUTPUT AFPPARMS='JOHNDOE.MY.PARM.FILE'
//SOMEDD    DD    SYSOUT=*,OUTPUT=(*.OUTPUT1)
```

# AFPSTATS parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the AFPSTATS keyword on the OUTPUT statement to indicate to the Print Services Facility (PSF) that an AFP Statistics (AFPSTATS) report is to be generated while printing this sysout data set. The AFPSTATS report can provide sysout data set processing detail for:

- Determining in which resource libraries PSF found particular resources.
- Diagnosing some resource selection problems.
- Obtaining statistical data about the printing of a sysout data set. These statistics may contain some inaccuracies caused by error recovery and repositioning within the sysout data set that make them unsuitable for accounting purposes.
- Diagnosing some sysout data set printing performance situations.

***References:*** For more information, see *PSF for z/OS: User's Guide* and *PSF for z/OS: Customization*.

## Syntax

```
AFPSTATS= {YES}
          {Y  }
          {NO }
          {N  }
```

## Parameter definition

**YES**
Requests that PSF produce an AFPSTATS report for the printing of this sysout data set. This parameter may also be coded as Y.

**NO**
Specifies that PSF should not produce an AFPSTATS report for the printing of this sysout data set. This parameter may also be coded as N.

## Defaults

If you do not code an AFPSTATS keyword, PSF will assume the value NO.

## Overrides

Specification of AFPSTATS=YES will be ignored on systems with a PSF older than PSF for OS/390 3.3.0.

The specification for the AFPSTATS keyword is exposed by PSF on the PSF Installation Exit. The Installation Exit can override the user's OUTPUT statement specification. When the Installation Exit overrides the OUTPUT statement specification of the AFPSTATS keyword, the user will receive a message in the sysout data set messages printed at the end of the sysout data set.

## Relationship to other control statements

Coding the AFPSTATS=YES keyword on the OUTPUT statement will not generate an AFPSTATS report unless the AFPSTATS DD statement in the PSF startup procedure has been coded. The AFPSTATS DD statement identifies the data set where PSF will place the AFPSTATS report. If you code the AFPSTATS keyword, but the PSF start-up procedure does not have a valid AFPSTATS DD statement, the sysout data set will be processed without PSF producing an AFPSTATS report.

## Example of the AFPSTATS keyword

```
//OUT1   OUTPUT AFPSTATS=YES
//PRINT  DD     SYSOUT=*,OUTPUT=*.OUT1
```

In this example, PSF will produce an AFPSTATS report for the sysout data set defined by the PRINT DD statement.

# BUILDING parameter

***Parameter Type***

Keyword, optional

***Purpose***

Use the BUILDING parameter to print a building identification on the separator pages of an output data set. An installation can use the building location to assist in sysout distribution.

## Syntax

```
BUILDING= {'building identification'}
          {building-identification  }
```

See “Using enclosing apostrophes in OUTPUT parameters” on page 477 for the BUILDING parameter.

## Subparameter definition

**building identification**
Specifies the building location associated with the output data set. The value for building location is 1 - 60 EBCDIC text characters. See “Character sets” on page 21 for a description of EBCDIC text characters.

## Defaults

- **In an APPC scheduling environment:** In both JES2 and JES3 systems, if you do not code BUILDING, the system uses the value defined in the transaction program (TP) user's RACF profile when:
  - The user submitting the TP profile has a RACF profile defined for him, and
  - The transaction program profile includes TAILOR_SYSOUT(YES).
- **In a non-APPC scheduling environment:** There is no default for the BUILDING parameter on the OUTPUT JCL statement.

## Overrides

- **In an APPC scheduling environment:** In both JES2 and JES3 systems, the BUILDING parameter on the OUTPUT JCL statement overrides the building in the RACF profile.
- **In a non-APPC scheduling environment:** In both JES2 and JES3 systems, there are no override considerations for BUILDING.

## Example of the BUILDING parameter

```
//OUTDS3   OUTPUT  BUILDING='920'
```

In this example, 920 will be printed on the line reserved for BUILDING on the separator pages of any output data set that references OUTDS3.

# BURST parameter

*Parameter type*

Keyword, optional

*Purpose*

Use the BURST parameter to specify that the output for the sysout data set printed on a continuous-forms AFP printer is to go to:

- The burster-trimmer-stacker, to be burst into separate sheets.
- The continuous forms stacker, to be left in continuous fanfold.

If the specified stacker is different from the last stacker used, or if a stacker was not previously requested, JES issues a message to the operator to thread the paper into the required stacker.

**Note:** BURST applies only for a data set printed on a 3800 or 3900 equipped with a burster-trimmer-stacker.

## Syntax

```
BURST= {YES}
       {Y  }
       {NO }
       {N  }
```

## Subparameter definition

**YES**
Requests that the printed output is to be burst into separate sheets. This subparameter can also be coded as Y.

**NO**
Requests that the printed output is to be in a continuous fanfold. This subparameter can also be coded as N.

## Defaults

If you do not code a BURST parameter and the sysout data set is printed on a 3800 or 3900 that has a burster-trimmer-stacker, JES uses an installation default specified at initialization.

## Overrides

A BURST parameter on the sysout DD statement overrides the OUTPUT JCL BURST parameter.

## Example of the BURST parameter

```
//OUTDS1 OUTPUT BURST=YES
```

In this example, the output from the 3800 will be burst into separate sheets.

# CHARS parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the CHARS parameter to specify the name of one or more coded fonts for printing this sysout data set on an AFP printer.

**Note:**

- CHARS applies only for a data set that is either printed on an AFP printer or processed by Infoprint Server.
- STD applies only on a JES3 system.

***References***

For more information on coded font names, see *IBM AFP Fonts: Font Summary for AFP Font Collection*.

## Syntax

```
CHARS= {font-name                    }
       {(font-name[,font-name]...)}
       {STD                           }
       {DUMP                          }
       {(DUMP[,font-name]...)       }
```

- You can omit the parentheses if you code only one font-name.
- Null positions in the CHARS parameter are invalid. For example, you **cannot** code CHARS=(,font-name) or CHARS=(font-name,,font-name).

## Subparameter definition

**font-name**
Names a coded font or character-arrangement table. Each font-name is 1 through 4 alphanumeric or national ($, #, @) characters. Code one to four names.

**STD**
Specifies the standard character-arrangement table. JES3 uses the standard table specified at initialization.

**Note:** STD is supported only on JES3 systems.

**DUMP**
Requests a high-density dump of 204-character print lines from a 3800. If more than one font-name is coded, DUMP must be first.

**Note:** DUMP is valid only on the OUTPUT JCL statement referenced in a SYSABEND or SYSUDUMP DD statement that specifies a sysout data set for the dump.

## Defaults

If you do not code the OUTPUT JCL CHARS parameter, JES uses the following, in order:

1. The DD CHARS parameter.
2. The DD UCS parameter value, if coded.
3. The OUTPUT JCL UCS parameter value, if coded.

If no character-arrangement table is specified on the DD or OUTPUT JCL statements, JES uses an installation default specified at initialization.

## Overrides

A CHARS parameter on the sysout DD statement overrides the OUTPUT JCL CHARS parameter.

For a data set scheduled to the Print Services Facility (PSF), PSF uses the following parameters, in override order, to select the font list:

1. Font list in the library member specified by an OUTPUT JCL PAGEDEF parameter.
2. DD CHARS parameter.
3. OUTPUT JCL CHARS parameter.
4. DD UCS parameter.
5. OUTPUT JCL UCS parameter.
6. JES installation default for the device.
7. Font list on the PAGEDEF parameter in the PSF-cataloged procedure.

See “PAGEDEF parameter” on page 539 for more information.

## Requesting a high-density dump

You can request a high-density dump on the 3800 through two parameters on the DD statement for the dump data set or on an OUTPUT JCL statement referenced by the dump DD statement:

- FCB=STD3. This parameter produces dump output at 8 lines per inch.
- CHARS=DUMP. This parameter produces 204-character print lines.

You can code one or both of these parameters. You can place both on the same statement or one on each statement.

## Example of the CHARS parameter

```
//OUTDS2 OUTPUT CHARS=(GT12,GB12,GI12)
```

In this example, the output from the AFP printer will be printed in three upper and lower case fonts: GT12, Gothic 12-pitch; GB12, Gothic Bold 12-pitch; and GI12, Gothic Italic 12-pitch.

# CKPTLINE parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the CKPTLINE parameter to specify the maximum number of lines in a logical page. JES uses this value, with the CKPTPAGE parameter, to determine when to take checkpoints while printing the sysout data set or transmitting the systems network architecture (SNA) data set.

**Note:** A JES3 system supports this parameter only when PSF prints the sysout data set on an AFP printer.

## Syntax

```
CKPTLINE=nnnnn
```

## Subparameter definition

**nnnnn**
Specifies the maximum number of lines in a logical page. nnnnn is a number from 0 through 32,767.

## Defaults

If you do not code the CKPTLINE parameter, JES2 uses an installation default specified at initialization. JES3 provides no installation default.

## Example of the CKPTLINE parameter

```
//OUTDS3 OUTPUT CKPTLINE=4000,CKPTPAGE=5
```

In this example, the sysout data set will be checkpointed after every 5 logical pages. Each logical page contains 4000 lines.

# CKPTPAGE parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the CKPTPAGE parameter to specify the number of logical pages:

- To print before JES takes a checkpoint.
- To transmit as a single systems network architecture (SNA) chain to an SNA work station before JES takes a checkpoint.

The CKPTLINE parameter specifies the number of lines in these logical pages.

**Note:** A JES3 system supports this parameter only when PSF prints the sysout data set on an AFP printer.

## Syntax

```
CKPTPAGE=nnnnn
```

## Subparameter definition

**nnnnn**
Specifies the number of logical pages to print or transmit before the next sysout data set checkpoint is taken. nnnnn is a number from 1 through 32,767.

## Defaults

If you do not code the CKPTPAGE parameter, JES2 uses an installation default specified at initialization; the default may also indicate whether checkpoints are to be based on page count or time. JES3 provides no installation default.

## Relationship to other parameters

If you code both the CKPTPAGE and CKPTSEC parameters:

- JES2 uses the value on the CKPTSEC parameter, provided the installation did not specify at initialization that checkpoints are to be based only on page count or time.
- JES3 uses the value on the CKPTPAGE parameter.

## Example of the CKPTPAGE parameter

```
//OUTDS4 OUTPUT CKPTPAGE=128,CKPTLINE=58
```

In this example, the sysout data set will be checkpointed after every 128 logical pages. Each logical page contains 58 lines.

# CKPTSEC parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the CKPTSEC parameter to specify how many seconds are to elapse between checkpoints of the sysout data set that JES is printing.

**Note:** A JES3 system supports this parameter only when PSF prints the sysout data set on an AFP printer.

## Syntax

```
CKPTSEC=nnnnn
```

## Subparameter definition

**nnnnn**
Specifies the number of seconds that is to elapse between checkpoints. nnnnn is a number from 1 through 32,767.

## Defaults

If you do not code the CKPTSEC parameter, JES2 uses an installation default specified at initialization; the default may also indicate whether checkpoints are to be based on page count or time. JES3 provides no installation default.

## Relationship to other parameters

If you code both the CKPTPAGE and CKPTSEC parameters:

- JES2 uses the value on the CKPTSEC parameter, provided the installation did not specify at initialization that checkpoints are to be based only on page count or time.
- JES3 uses the value on the CKPTPAGE parameter.

## Example of the CKPTSEC parameter

```
//OUTDS5 OUTPUT CKPTSEC=120
```

In this example, the sysout data set will be checkpointed after every 120 seconds, or 2 minutes.

# CLASS parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the CLASS parameter to assign the sysout data set to an output class.

**Note:** If a sysout data set has the same class as the JOB statement MSGCLASS parameter, the job log appears on the same output listing as the sysout data set.

## Syntax

```
CLASS= {class}
       {*    }
```

## Subparameter definition

**class**
Identifies the output class for the data set. The class is one character: A through Z or 0 through 9, which you may optionally include in quotation marks. The attributes of each output class are defined during JES initialization; specify the class with the desired attributes.

***** Requests the output class in the MSGCLASS parameter on the JOB statement.

**null value**
Requests the output class in the MSGCLASS parameter on the JOB statement.

## Overrides

The class subparameter of the DD statement SYSOUT parameter overrides the OUTPUT JCL CLASS parameter. On the DD statement, you must code a null class in order to use the OUTPUT JCL CLASS parameter; for example:

```
//OUTDS DD SYSOUT=(,),OUTPUT=*.OUT1
```

## Held Classes in a JES2 system

An installation option at JES2 initialization determines if both the class for the sysout data set and the class for the job's messages must be held in order for a sysout data set to be held.

A sysout data set is held in the following cases:

- The sysout DD statement contains HOLD=YES.
- The sysout DD statement does not contain a HOLD parameter or contains HOLD=NO but requests a class that the installation defined as held and defined as:
  - Not requiring the message class to be a held class in order for the sysout data set to be held. The JOB statement MSGCLASS parameter can specify any class.
  - Requiring the message class to be a held class in order for the sysout data set to be held. The JOB MSGCLASS parameter must also specify a held class.
- The OUTPUT JCL statement specifies OUTDISP=HOLD.

A sysout data set is not held in the following cases:

- The sysout DD statement does not contain a HOLD parameter or contains HOLD=NO and requests:
  - A class that the installation defined as not held.
  - A class that the installation defined as held and defined as requiring the message class to be a held class in order for the sysout data set to be held. The JOB MSGCLASS parameter must specify a class that is not held.

Contact the installation to find out if holding the sysout class depends on a held MSGCLASS class.

## Held Classes in a JES3 system

If CLASS specifies a class-name that is defined to JES3 as a held class for the output service hold queue (Q=HOLD), all of the new output characteristics might not be included in the data set on the writer queue when (1) the data set is moved from the hold queue to the output service writer queue (Q=WTR), (2) the data set includes an OUTPUT JCL statement, and (3) the NQ= or NCL= keyword is used.

For more information, see *z/OS JES3 Initialization and Tuning Guide*.

## Significance of output classes

To print this sysout data set and the messages from your job on the same output listing, code one of the following:

- The same output class in the DD SYSOUT parameter as in the JOB MSGCLASS parameter.
- DD SYSOUT=* to default to the JOB MSGCLASS output class.
- DD SYSOUT=(,) to default to one of the following:
  1. The CLASS parameter in an explicitly or implicitly referenced OUTPUT JCL statement. In this case, the OUTPUT JCL CLASS parameter should specify the same output class as the JOB MSGCLASS parameter.
  2. The JOB MSGCLASS output class, if no OUTPUT JCL statement is referenced or if the referenced OUTPUT JCL statement contains either CLASS= or CLASS=*.

## Examples of the CLASS parameter

*Example 1*

```
//OUTDS6 OUTPUT CLASS=D
//OUT1   DD     SYSOUT=(,),OUTPUT=*.OUTDS6
```

In this example, JES processes the sysout data set defined in DD statement OUT1 in output class D.

*Example 2*

```
//PRINTALL JOB    ACCT123,MAEBIRD,MSGCLASS=H
//STEP1    EXEC   PGM=PRINTER
//OUTDS7   OUTPUT CLASS=*
//OUTPTR   DD     SYSOUT=(,),OUTPUT=*.OUTDS7
```

In this example, JES processes the sysout data set defined in DD statement OUTPTR in output class H, as specified in the JOB statement MSGCLASS parameter. The same result could be obtained by the following:

```
//PRINTALL JOB    ACCT123,MAEBIRD,MSGCLASS=H
//STEP1    EXEC   PGM=PRINTER
//OUTPTR   DD     SYSOUT=H
```

# COLORMAP parameter

***Parameter Type:*** Keyword, optional

***Purpose:*** Use COLORMAP to specify the AFP Resource (object) for the data set that contains color translation information. For more information see *PSF for z/OS: User's Guide*.

## Syntax

```
COLORMAP=resource
```

## Subparameter definition

**resource**
Specifies the name of an AFP resource, where the resource name is 1 through 8 alphanumeric or national ($, #, @) characters and the first must be alphabetic or national.

## Example of the COLORMAP parameter

```
//OUTCOLOR OUTPUT COLORMAP=M1SETUP
```

In this example, M1SETUP is the name of the AFP resource.

# COMPACT parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the COMPACT parameter to specify a compaction table for JES to use when sending the sysout data set, which is a systems network architecture (SNA) data set, to a SNA remote terminal.

## Syntax

```
COMPACT=compaction-font-name
```

## Subparameter definition

**compaction-font-name**
Specifies a compaction table by a symbolic name. The name is 1 through 8 alphanumeric characters. The symbolic name must be defined by the installation during JES initialization.

## Defaults

If you do not code the COMPACT parameter, compaction is suppressed for the data set.

## Overrides

This parameter overrides any compaction table value defined at the SNA remote terminal.

## Example of the COMPACT parameter

```
//OUTDS8 OUTPUT DEST=N555R222,COMPACT=TBL77
```

In this example, the sysout data set will be sent to remote terminal 222 at node 555; JES will use compaction table TBL77.

# COMSETUP parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the COMSETUP parameter to specify the name of a microfile setup resource that contains setup information.

***References***

For more information, see *PSF for z/OS: User's Guide*.

## Syntax

```
COMSETUP=resource
```

## Subparameter definition

**resource**
Specifies the name of a macrofile setup resource, where the resource name is 1 through 8 alphanumeric or national ($, #, @) characters. (The first must be alphabetic or national.)

## Example of the COMSETUP parameter

```
//RPTDS OUTPUT COMSETUP=H1SETUP
```

In this example, H1SETUP is the name of a microfilm setup resource.

# CONTROL parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the CONTROL parameter to specify either that each logical record starts with a carriage control character or that the output is to be printed with single, double, or triple spacing.

## Syntax

```
CONTROL= {PROGRAM}
         {SINGLE }
         {DOUBLE }
         {TRIPLE }
```

## Subparameter definition

**PROGRAM**
Indicates that each logical record in the data set begins with a carriage control character, which must be identified in the DD statement, DCB macro, or data set label. You might identify these carriage control characters in the DD statement through the DCB subparameter in the RECFM of the data set as being A (ASA) or M (machine). The carriage control characters are given in *z/OS DFSMS Using Data Sets*.

**SINGLE**
Indicates forced single spacing.

**DOUBLE**
Indicates forced double spacing.

**TRIPLE**
Indicates forced triple spacing.

## Defaults

In a JES3 system, if you do not code the CONTROL parameter, JES3 uses an installation default specified at initialization.

In a JES2 system, if you do not code the CONTROL parameter, JES2 uses the settings from the device statement in the JES2 initialization deck or the operator command to modify the spacing for that device. This modify device command can set single, double, or triple spacing.

### Example of the CONTROL parameter

```
//OUTDS9 OUTPUT CONTROL=PROGRAM
```

In this example, the sysout data set is printed using the first character of each logical record for carriage control.

## COPIES parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the COPIES parameter to specify how many copies of the sysout data set to print. The printed output is in page sequence for each copy.

For printing on an AFP printer, this parameter can instead specify how many copies of each page are to be printed before the next page is printed.

### Syntax

```
COPIES= {nnn                                }
        {(nnn,(group-value[,group-value]...))}
        {(,(group-value[,group-value]...))   }
```

- You can omit the parentheses if you code only COPIES=nnn.
- The following are **not** valid:
  - A null group-value, for example, COPIES=(5,(,)) or COPIES=(5,)
  - A zero group-value, for example, COPIES=(5,(1,0,4))
  - A null within a list of group-values, for example, COPIES=(5,(1,,4))

### Subparameter definition

**nnn**

A number (1 through 255 in a JES2 system, 0 through 255 in a JES3 system) that specifies how many copies of the sysout data set to print. Each copy will be in page sequence order.

For a data set printed on an AFP printer, JES ignores nnn if any group values are specified.

**group-value**

Specifies how many copies of each page are to be printed before the next page is printed. Each group-value is 1 through 3 decimal numbers from 1 through 255 in a JES2 system and from 1 through 254 in a JES3 system. You can code a maximum of eight group-values. The total copies of each page equals the sum of the group-values.

**Note:**

- This subparameter is valid only for output processed by PSF.
- For output printed on an AFP printer, this subparameter overrides an nnn subparameter, if coded.

## Defaults

**For JES2**, on the DD, OUTPUT JCL, or /*OUTPUT statement: if you do not code a COPIES parameter, code it incorrectly, or code COPIES=0, the system uses a default of 1, which is the default for the DD COPIES parameter.

**For JES3**, on the DD, OUTPUT JCL, or //*FORMAT statement: if you do not code a COPIES parameter, code it incorrectly, or code COPIES=0 on the DD statement, the system uses a default of 1, which is the default for the DD COPIES parameter.

## Overrides

A COPIES parameter on the sysout DD statement overrides the OUTPUT JCL COPIES parameter.

If the OUTPUT JCL statement contains a FORMDEF parameter, which specifies a library member, the COPYGROUP parameter on a FORMDEF statement in that member overrides any group-value subparameters on the OUTPUT JCL COPIES parameter or the sysout DD COPIES parameter. For more information, see "FORMDEF parameter" on page 513.

## Relationship to other parameters

If the OUTPUT JCL or the sysout DD statement contains a FLASH parameter, JES prints with the forms overlay the number of copies specified in one of the following:

- COPIES=nnn, if the FLASH count is larger than nnn. For example, if COPIES=10 and FLASH=(LTHD,12) JES prints 10 copies, all with the forms overlay.
- The sum of the group-values specified in the COPIES parameter, if the FLASH count is larger than the sum. For example, if COPIES=(,(2,3,4) and FLASH=(LTHD,12) JES prints nine copies in groups, all with the forms overlay.
- The count subparameter in the FLASH parameter, if the FLASH count is smaller than nnn or the sum from the COPIES parameter. For example, if COPIES=10 and FLASH=(LTHD,7) JES prints seven copies with the forms overlay and three copies without.

## Relationship to other control statements

**For JES2**, if you request copies of the entire job on the JES2 /*JOBPARM COPIES parameter and also copies of the data set on the DD COPIES or OUTPUT JCL COPIES parameter, JES2 prints the number of copies equal to the **product** of the two requests.

## Examples of the COPIES parameter

***Example 1***

```
//RPTDS OUTPUT COPIES=4,FORMS=WKREPORT
```

This example asks JES to print four copies of the weekly report on forms named WKREPORT.

***Example 2***

```
//EXPLD OUTPUT COPIES=(,(3)),FORMS=ACCT
```

This example asks JES to print the first page three times, then the second page three times, the third page three times, etc., on forms named ACCT.

***Example 3***

```
//QUEST OUTPUT COPIES=(,(8,25,18,80)),FORMS=ANS
```

This example asks JES to print each page eight times before printing the next page, then 25 times before the next, then 18 times before the next, and finally 80 times before the next. The forms are named ANS.

***Example 4***

```
//EXMP  OUTPUT COPIES=(5,(3,2))
```

This example asks JES to do one of the following:

- If the data set is printed on other than an AFP printer, to print five copies.
- If it is printed on an AFP printer, to print each page three times before printing the next page and then to print each page twice before printing the next page.

# COPYCNT parameter

***Parameter type***

Keyword, optional

***Purpose***

The COPYCNT=xxx keyword supercedes the COPIES=xxx keyword on the output statement. Use it to define the number of copies of output. Where COPIES is limited to 255 maximum size, COPYCNT can be 0 - 2147483647 in size.

**Note:** This is only supported for PSF 4.4.0 and above.

## Syntax

```
COPYCNT= {xxx}
```

## Subparameter definition

None.

## Defaults

None.

## Overrides

COPYCNT supercedes COPIES. If both are coded, COPYCNT takes precedence.

## Relationship to other parameters

None.

## Relationship to other control statements

None.

## Examples of the COPYCNT parameter

*Example 1*

```
COPYCNT=3500
```

This example provides a larger range of values for the number of copies of output to be produced.

# DATACK parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the DATACK parameter to indicate whether or not print-positioning and invalid-character data-check errors are to be blocked or unblocked for printers accessed through the Print Services Facility (PSF) functional subsystem.

A print-positioning error occurs when the designated position of any kind of printable information is beyond the limits of either the physical page, or the overlay or logical page of which it is part.

An invalid-character data-check error occurs when the hexadecimal representation of a text character has no mapping in the code page to a member of the font raster patterns.

If an error type is unblocked, the printer reports the error at the end of the page in which it occurs, and PSF processes the error and generates an error message. (See the PIMSG parameter for more information on the printing of error messages.)

If an error type is blocked, the printer does not report the error to PSF. Printing continues but data may be lost on the output.

***References***

For more information on data-check errors and their processing through PSF, see *PSF for z/OS: Customization* or *PSF for z/OS: User's Guide*.

## Syntax

```
DATACK=  {BLOCK  }
         {UNBLOCK}
         {BLKCHAR}
         {BLKPOS }
```

## Subparameter definition

**BLOCK**
Indicates that print-positioning errors and invalid-character errors are not reported to PSF.

**UNBLOCK**
Indicates that print-positioning errors and invalid-character errors are reported to PSF.

**BLKCHAR**
Indicates that invalid-character errors are blocked, and not reported to PSF. Print-positioning errors are reported normally.

**BLKPOS**
Indicates that print-positioning errors are blocked, and not reported to PSF. Invalid-character errors are reported normally.

## Defaults

If you do not code the DATACK parameter, the DATACK specification from the PSF PRINTDEV statement is used. If not specified in the PRINTDEV statement, the default is BLOCK. For information about the PRINTDEV statement, see *PSF for z/OS: Customization*.

## Relationship to other parameters

If DATACK is specified as UNBLOCK, BLKCHAR, or BLKPOS, and an unblocked error occurs, the printer reports the error to PSF which processes the error. The coding of the PIMSG parameter then determines whether or not printing of the data set continues after the page in error, and if error messages are printed at the end of the data set.

## Example of the DATACK parameter

```
//OUTDS1 OUTPUT DATACK=BLKCHAR,PIMSG=(YES,0)
```

In this example, when a print-position error occurs, it is reported to the user via a printed error message. If an invalid-character error occurs, it is not reported. In either case, the printing of the data set continues, and all functional subsystem messages are printed.

# DDNAME parameter

Keyword, optional

***Purpose***

Specifies the DD statements to apply the specifications on the OUTPUT statement to.

## Syntax

```
DDNAME={ddname                          }
       {procstepname.ddname             }
       {stepname.ddname                 }
       {stepname.procstepname.ddname    }
```

## Subparameter definition

*ddname*
Indicates all DD statements with the name, *ddname*, in this job.

*stepname.ddname*
Indicates all DD statements with the name, *ddname*, in step, *stepname*, in this job.

*procstepname.stepname.ddname*
Indicates all DD statements with the name, *ddname*, in procedure step, *procstepname*, of a procedure that is called by a step, *stepname*, in this job.

*procstepname.ddname*
Indicates all DD statements with the name, *ddname*, in procedure step, *procstepname*, in this job.

**Note:** If a DD statement matches more than one OUTPUT statement DDNAME=keyword, then the OUTPUT statements which have most qualifiers for the DDNAME=keyword will be used. Statements which have fewer qualifiers are ignored.

## Example of the DDNAME parameter

In the following example, the specifications on the OUTPUT statement are applied to the DD statement named DDNAME1, which resides in STEP1 of procstep PSTEP1.

```
DDNAME=STEP1.PSTEP1.DDNAME1
```

# DEFAULT parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the DEFAULT parameter to specify that this OUTPUT JCL statement can or cannot be implicitly referenced by a sysout DD statement. An OUTPUT JCL statement that contains a DEFAULT=YES parameter is called a default OUTPUT JCL statement.

## Syntax

```
DEFAULT= {YES}
         {Y  }
         {NO }
         {N  }
```

## Subparameter definition

**YES**
Indicates that this OUTPUT JCL statement can be implicitly referenced by sysout DD statements. This subparameter can also be coded as Y.

**NO**
Indicates that this OUTPUT JCL statement cannot be implicitly referenced by sysout DD statements. This subparameter can also be coded as N.

## Defaults

If you do not code DEFAULT=YES, the default is NO. In order to take effect, an OUTPUT JCL statement without DEFAULT=YES must be explicitly referenced in an OUTPUT parameter on a sysout DD statement.

## Location in the JCL

- A step-level OUTPUT JCL statement appears within a step, that is, anywhere after the first EXEC statement in a job.
- A job-level OUTPUT JCL statement appears after the JOB statement and before the first EXEC statement.
- You can place more than one job- or step-level default OUTPUT JCL statement in a job or step.
- You must place an OUTPUT JCL statement in the input stream **before** any sysout DD statement that explicitly or implicitly refers to it.

## References to default OUTPUT JCL statements

- A sysout DD statement makes an explicit reference in an OUTPUT parameter that specifies the name of an OUTPUT JCL statement.
- A sysout DD statement makes an implicit reference when it does not contain an OUTPUT parameter, and the job or step contains one or more default OUTPUT JCL statements.
- A sysout DD statement implicitly references all step-level default OUTPUT JCL statements in the same step.
- A sysout DD statement implicitly references all job-level default OUTPUT JCL statements when the step containing the DD statement does not contain any step-level default OUTPUT JCL statements.
- A sysout DD statement can explicitly reference a default OUTPUT JCL statement.

## Example of the DEFAULT parameter

```
//EXMP2    JOB     ACCT555,MAEBIRD,MSGCLASS=B
//OUTDAL   OUTPUT  DEFAULT=YES,DEST=DALLAS
//OUTPOK   OUTPUT  DEST=POK
//STEP1    EXEC    PGM=REPORT
//OUTHERE  OUTPUT  CLASS=D
//SYSIN    DD      *
            .
            .
            .
/*
//WKRPT    DD      UNIT=VIO,DISP=(,PASS)
//RPT1     DD      SYSOUT=(,),OUTPUT=*.OUTHERE
//RPT2     DD      SYSOUT=A
//STEP2    EXEC    PGM=SUMMARY
//OUTHQ    OUTPUT  DEFAULT=YES,DEST=HQ
//WKDATA   DD      UNIT=VIO,DISP=(OLD,DELETE),DSNAME=*.STEP1.WKRPT
//MONTH    DD      SYSOUT=(,),OUTPUT=*.STEP1.OUTHERE
//SUM      DD      SYSOUT=A
//FULRPT   DD      SYSOUT=A,OUTPUT=(*.OUTDAL,*.OUTPOK)
```

In this example, the JOB named EXMP2 contains two job-level OUTPUT JCL statements: OUTDAL and OUTPOK. OUTDAL is a default OUTPUT JCL statement because it contains DEFAULT=YES; OUTDAL can be implicitly referenced by a sysout DD statement. OUTPOK must be explicitly referenced in a sysout DD OUTPUT parameter for its processing options to be used. The purpose of both of these OUTPUT JCL statements is to specify a destination for a sysout data set.

STEP1 contains a step-level OUTPUT JCL statement: OUTHERE. The purpose of this statement is to specify that JES process the data set locally in output class D. OUTHERE can be used only if it is explicitly referenced.

STEP2 contains a step-level default OUTPUT JCL statement: OUTHQ. The purpose of this statement is to specify a destination for a sysout data set. OUTHQ can be implicitly referenced.

The references in this job are as follows:

- In STEP1 and STEP2, sysout DD statements RPT1 and MONTH explicitly reference OUTPUT JCL statement OUTHERE. These two sysout data sets are printed locally in the same output class.

  **Note:** You can explicitly reference an OUTPUT JCL statement in a preceding job step.
- In STEP1, DD statement RPT2 implicitly references OUTPUT JCL statement OUTDAL. This implicit reference occurs because all of the following are true:
  1. DD statement RPT2 contains a SYSOUT parameter but does not contain an OUTPUT parameter. Thus, this DD statement is making an implicit reference.
  2. STEP1 does not contain a default OUTPUT JCL statement, so the implicit reference must be to job-level default OUTPUT JCL statements.
  3. OUTDAL is the only job-level default OUTPUT JCL statement.
- In STEP2, DD statement SUM implicitly references OUTPUT JCL OUTHQ because all of the following are true:
  1. DD statement SUM contains a SYSOUT parameter but does not contain an OUTPUT parameter. Thus, this DD statement is making an implicit reference.
  2. STEP2 contains a default OUTPUT JCL statement: OUTHQ. Therefore, the implicit reference is to OUTHQ and cannot be to any job-level default OUTPUT JCL statements.
- In STEP2, DD statement FULRPT explicitly references OUTPUT JCL statements OUTDAL and OUTPOK.

# DEPT parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the DEPT parameter to print the department identification on the separator pages of output for a sysout data set. An installation can use the department identification to assist in sysout distribution.

## Syntax

```
DEPT= {'department identification'}
      {department-identification  }
```

See “Using enclosing apostrophes in OUTPUT parameters” on page 477 for the DEPT parameter.

## Subparameter definition

**department identification**
Specifies the department identification associated with the sysout. The value for department identification is 1 - 60 EBCDIC text characters. See “Character sets” on page 21 for a description of EBCDIC text characters.

## Defaults

- **In an APPC scheduling environment:** In both JES2 and JES3 systems, if you do not code DEPT, the system uses the value defined in the transaction program (TP) user's RACF profile when:
  - The user submitting the TP profile has a RACF profile defined for him, and
  - The transaction program profile includes TAILOR_SYSOUT(YES).
- **In a non-APPC scheduling environment:** There is no default for the DEPT parameter on the OUTPUT JCL statement.

## Overrides

- **In an APPC scheduling environment:** In both JES2 and JES3 systems, the DEPT parameter on the OUTPUT JCL statement overrides the department in the RACF profile.
- **In a non-APPC scheduling environment:** In both JES2 and JES3 systems, there are no override considerations for DEPT.

## Example of the DEPT parameter

```
//OUTDS4   OUTPUT  DEPT='PAYROLL'
```

In this example, PAYROLL will be printed on the line reserved for DEPT on separator pages of any sysout data set that references OUTDS4.

# DEST parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the DEST parameter to specify a destination for the sysout data set. The DEST parameter can send a sysout data set to a remote or local terminal, a node, a node and remote work station, a local device or group of devices, or a node and userid.

## Syntax

```
DEST=destination

The destination subparameter for JES2 is one of the following:

     LOCAL|ANYLOCAL
     'IP:ipaddr'
     name
     Nnnnnn
     NnRmmmmm
     NnnRmmmm
     NnnnRmmm
     NnnnnRmm
     NnnnnnRm
     (node.remote)
     nodename.userid
     'nodename.IP:ipaddr'
     Rnnnnn
     RMnnnnn
     RMTnnnnn
     Unnnnn
     userid

The destination subparameter for JES3 is one of the following:

     ANYLOCAL
     'IP:ipaddr'
     device-name
     group-name
     nodename
     'nodename.IP:ipaddr'
     nodename.remote
```

## Subparameter definition for JES2 systems

**LOCAL|ANYLOCAL**
Indicates any local device.

**'IP:ipaddr' | 'nodename.IP:ipaddr'**
Identifies a TCP/IP routing designation, where *ipaddr* can be any printable character string of from 1 to 124 characters. The entire parameter list is limited to 127 characters, and it must be enclosed in single quotation marks.

**name**
Identifies a destination by a symbolic name (for example, a local device, remote device, or a userid) which is defined by the installation during JES2 initialization. The name can be, for example, a local device, remote device, or a userid. The name is 1 through 8 alphanumeric or national ($, #, @) characters.

**Nnnnnn**
Identifies a node. nnnnn is 1 through 5 decimal numbers from 1 through 32,767. For example, N103.

**NnRmmmmm**
**NnnRmmmm**
**NnnnRmmm**
**NnnnnRmm**
**NnnnnnRm**
Identifies a node and a remote work station connected to the node. The node number, indicated in the format by n, is 1 through 5 decimal numbers from 1 through 32,767. The remote work station number, indicated in the format by m,

is 1 through 5 decimal numbers from 1 through 32,767. Do not code leading zeros in n or m. The maximum number of digits for n and m combined cannot exceed six.

**Note:** R0 is equivalent to LOCAL specified at node Nn.

**nodename.userid**
Identifies a destination node and a VM or a TSO/E userid, a remote workstation, or a symbolic name defined at the destination node. The nodename is a symbolic name defined at the node of execution. nodename is 1 through 8 alphanumeric or national ($, #, @) characters. userid is 1 through 8 alphanumeric or national ($, #, @) characters, and must be defined at the specified node.

**Rnnnnn**
**RMnnnnn**
**RMTnnnnn**
Identifies a remote workstation. nnnnn is 1 through 5 decimal numbers from 1 through 32,767. Note that with remote pooling, the installation may translate this route code to another route code.

If you send a job to execute at a remote node and the job has a ROUTE PRINT RMTnnnn statement, JES returns the output to RMTnnnn at the node of origin. For JES2 to print the output at RMTnnnn at the executing node, code DEST=NnnnRmmm on an OUTPUT JCL statement or sysout DD statement.

**Note:** R0 is equivalent to LOCAL.

**Unnnnn**
Identifies a local terminal with special routing. nnnnn is 1 through 5 decimal numbers from 1 through 32,767.

If you send a job to execute and the job has a ROUTE PRINT Unnnn statement, JES returns the output to Unnnn at the node of origin.

**userid**
Identifies a userid at the local node. Userid for TSO/E is 1 through 7 alphanumeric or national ($, #, @) characters. The userid can also be a destination name defined in a JES2 DESTID initialization statement.

**Note:** JES2 initialization statements determine whether or not the node name is required when coding a userid. See your system programmer for information about how routings will be interpreted by JES2.

## Subparameter definition for JES3 systems

**ANYLOCAL**
Indicates any local device.

**'IP:ipaddr' | 'nodename.IP:ipaddr'**
Identifies a TCP/IP routing designation, where *ipaddr* can be any printable character string of from 1 to 124 characters. The entire parameter list is limited to 127 characters, and it must be enclosed in single quotation marks.

**device-name**
Identifies a local device by a symbolic name defined by the installation during JES3 initialization. device-name is 1 through 8 alphanumeric or national ($, #, @) characters.

**group-name**
Identifies a group of local devices, an individual remote station, or a group of

remote stations by a symbolic name defined by the installation during JES3 initialization. group-name is 1 through 8 alphanumeric or national ($, #, @) characters.

**nodename**
Identifies a node by a symbolic name defined by the installation during JES3 initialization. The node is 1 through 8 alphanumeric or national ($, #, @) characters. If the node you specify is the same as the node you are working on, JES3 treats the output as though you had specified ANYLOCAL.

**nodename.remote**
Identifies a destination node and either a remote work station or VM userid at that node, as follows:

**nodename**
A symbolic name defined by the installation during JES3 initialization. The nodename is 1 through 8 alphanumeric or national ($, #, @) characters.

**remote**
A name for a remote work station. The name is 1 through 8 alphanumeric or national ($, #, @) characters and must be defined at the node. Enclose it in apostrophes when it contains special characters or begins with a number.

## Defaults

In a JES2 system, if you do not code a DEST parameter, JES directs the sysout data set to the default destination for the input device from which the job was submitted.

In a JES3 system, if you do not code a DEST parameter, the default destination is the submitting location. For jobs submitted through TSO/E and routed to NJE for execution, the default is the node from which the job was submitted, and the destination ANYLOCAL.

If a specified destination is invalid, the job fails.

**Note:** Most JCL syntax errors are detected and reported by JES or the functional subsystem that is processing the sysout data set, rather than when the system first reads in the JCL.

## Overrides

A DEST parameter on the sysout DD statement overrides the OUTPUT JCL DEST parameter.

## Relationship to other parameters

For JES3, you can code the DEST=nodename parameter with the OUTPUT JCL WRITER=name parameter; however, do not code DEST=nodename.userid with WRITER=name.

## Examples of the DEST parameter

***Example 1***

```
//REMOT1 OUTPUT DEST=R444
```

In this example, JES2 sends the sysout data set to remote terminal 444.

***Example 2***

```
//REMOT2 OUTPUT DEST=STAT444
```

In this example, JES sends the sysout data set to an individual remote station named by the installation STAT444.

***Example 3***

```
//REMOT3 OUTPUT DEST=KOKVMBB8.DP58HHHD
```

In this example, JES sends the sysout data set to VM userid DP58HHHD at node KOKVMBB8.

***Example 4***

```
//REMOT4 OUTPUT DEST='NEWYORK.IP:bldprt-2'
```

In this example JES2 sends the sysout data set to node NEWYORK, where a functional subsystem that can process IP-distributed data sets sends the data to the bldprt-2 host system.

***Example 5***

```
//REMOT5 OUTPUT DEST='IP:9.117.84.53'
```

In this example the functional subsystem sends the sysout data to the host machine at IP address 9.117.84.53.

## DPAGELBL parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the DPAGELBL (data page labelling) parameter to indicate whether the system should print the security label on each page of printed output. The security label represents a security level and categories as defined to RACF.

The security label that the system prints is determined by the SECLABEL parameter of the JOB statement. If you do not specify SECLABEL on the JOB statement, the security level at which the job is executing is printed.

***Reference***

For additional information on data page labelling, refer to *PSF for z/OS: Customization* and *PSF for z/OS: Security Guide*.

### Syntax

```
DPAGELBL=  {YES}
           {Y  }
           {NO }
           {N  }
```

## Subparameter definition

**YES**
Requests the system to print the security label on each page of printed output. You can also code this parameter as Y.

**NO**
Requests that the system print no security label on each page of printed output. You can also code this parameter as N.

## Defaults

If you do not code the DPAGELBL parameter, an installation default determines if a security label is printed.

## Relationship to other parameters

Use the DPAGELBL parameter with the SYSAREA parameter on the OUTPUT JCL statement and the SECLABEL parameter on the JOB statement as instructed by your security administrator.

You can code the DPAGELBL parameter with any other OUTPUT JCL statement parameters.

## Example of the DPAGELBL parameter

```
//JOBA   JOB   1,'JIM WOOSTER',SECLABEL=CONF
                .
                .
//VPRPT  OUTPUT  DPAGELBL=YES,FORMS=VP20
```

In this example, the security label CONF (specified on the SECLABEL parameter of the JOB statement) is printed on each page of printed output. The sysout data set is printed on forms named VP20.

# DUPLEX parameter

***Parameter type***

Keyword, optional

***Purpose***

Use DUPLEX to specify whether or not printing is to be done on both sides of the sheet. This overrides what is specified in the FORMDEF in use.

## Syntax

```
DUPLEX={NO    }
       {N     }
       {NORMAL}
       {TUMBLE}
```

## Subparameter definition

**NO or N**
Specifies to print on one side only.

**NORMAL**
Specifies that the physical page is rotated about the Y axis. For most page orientations (including the default orientation), the Y axis is the long edge of the sheet. This allows for binding on the long side of the sheet.

**TUMBLE**
Specifies that the physical page is rotated about the X axis. For most page orientations (including the default orientation), the X axis is the short edge of the sheet. This allows for binding on the short side of the sheet.

### Relationship to other keywords on this statement

The DUPLEX keyword overrides the duplex option from the forms definition, which may be specified by the FORMDEF keyword.

### Example of the DUPLEX parameter

```
//OUTDUP  OUTPUT DUPLEX=NO
```

In this example, the output is to be printed in simplex (printed on only one side of the paper).

## FCB parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the FCB parameter to specify:

- The forms control buffer (FCB) image JES is to use to guide printing of the sysout data set by a 1403 Printer, 3211 Printer, 3203 Printer Model 5, 3800 Printing Subsystem, 4245 Printer, or 4248 Printer, or by a printer supported by systems network architecture (SNA) remote job entry (RJE).
- The page definition member to be used if the data set is line-mode and is printed on a page-mode printer and you do not code PAGEDEF.
- The carriage control tape JES is to use to control printing of the sysout data set by a 1403 Printer or by a printer supported by SNA RJE.
- The data-protection image JES is to use to control output by a 3525 Card Punch.
- The name of a page definition to be used by PSF in formatting a print data set.

The FCB image specifies how many lines are to be printed per inch and the length of the form. JES loads the image into the printer's forms control buffer. The FCB image is stored in SYS1.IMAGELIB. IBM provides three standard FCB images:

- STD1, which specifies 6 lines per inch on an 8.5-inch-long form. (3211 and 3203-5 only)
- STD2, which specifies 6 lines per inch on an 11-inch-long form. (3211 and 3203-5 only)
- STD3, which specifies 8 lines per inch on an 11-inch form for a dump. (3800 only)

***References***

For more information on the forms control buffer, see *z/OS DFSMSdfp Advanced Services* or *PSF for z/OS: User's Guide*.

## Syntax

```
FCB= {fcb-name}
     {STD     }
```

- Code the fcb-name as STD1 or STD2 only to request the IBM-supplied images.
- Code the fcb-name as STD3 only for a high-density dump.

## Subparameter definition

**fcb-name**
Identifies the FCB image. The name is 1 through 4 alphanumeric or national ($, #, @) characters and is the last characters of a SYS1.IMAGELIB member name:

- FCB2xxxx member, for a 3211, a 3203 Model 5, or a printer supported by SNA.
- FCB3xxxx member, for a 3800.
- FCB4xxxx member, for a 4248.

Identifies a PAGEDEF member in the PSF libraries.

**STD**
Indicates the standard FCB. JES3 uses the standard FCB specified at JES3 initialization.

**Note:** STD is supported only on JES3 systems.

## Defaults

If you do not code the FCB parameter for a data set on an impact printer, the system checks the FCB image that was last loaded in the printer; if it is a default image, as indicated by its first byte, JES uses it. If it is not a default image, JES loads the FCB image that is the installation default specified at JES initialization.

The FCB parameter names a default page definition to be used if the data set is line-mode, is printed on a page-mode printer and PAGEDEF is not coded on the OUTPUT or DD statements.

## Overrides

An FCB parameter on the sysout DD statement overrides the OUTPUT JCL FCB parameter. If the data set is line-mode and is printed on a page-mode printer and you code PAGEDEF on the DD statement or OUTPUT statement, then PAGEDEF overrides FCB.

## Relationship to other parameters

The FCB parameter is mutually exclusive with the FRID subparameter of the DD statement DCB parameter.

## Requesting a high-density dump

You can request a high-density dump on the 3800 through two parameters on the DD statement for the dump data set or on an OUTPUT JCL statement referenced by the dump DD statement:

- FCB=STD3. This parameter produces dump output at 8 lines per inch.

- CHARS=DUMP. This parameter produces 204-character print lines.

You can code one or both of these parameters. You can place both on the same statement or one on each statement.

### Example of the FCB parameter

```
//OUTDS1 OUTPUT FCB=AA33
```

In this example, JES will print the sysout data set using the FCB image named AA33.

## FLASH parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the FLASH parameter to identify the forms overlay to be used in printing the sysout data set on a 3800 Printing Subsystem and, optionally, to specify the number of copies on which to print the forms overlay.

**Note:** FLASH applies only for a data set printed on a 3800.

***Reference***

For information on forms overlays, see the *Forms Design Reference Guide for the 3800*.

### Syntax

```
FLASH= {overlay-name          }
       {(overlay-name[,count])}
       {(,count)              }
       {NONE                  }
       {STD                   }
```

The count subparameter is optional. If you omit it, you can omit the parentheses.

### Subparameter definition

**overlay-name**
Identifies the forms overlay frame that the operator is to insert into the printer before printing begins. The name is 1 through 4 alphanumeric or national ($, #, @) characters.

**count**
Specifies the number, 0 through 255, of copies that JES is to flash with the overlay, beginning with the first copy printed. Code a count of 0 to flash **no** copies.

**NONE**
Suppresses flashing for this sysout data set.

If FLASH=NONE is on an OUTPUT JCL statement in a job to be executed at a remote node, JES3 sets the overlay-name to zero before sending the job to the node.

**STD**
Indicates the standard forms flash overlay. JES3 uses the standard forms overlay specified at JES3 initialization.

**Note:** STD is supported only on JES3 systems.

## Defaults

If you do not code a FLASH parameter and an installation default was not specified at JES2 or JES3 initialization, forms are not flashed.

If you specify an overlay-name without specifying a count, all copies are flashed. That is, the default for count is 255.

## Overrides

A FLASH parameter on the sysout DD statement overrides the OUTPUT JCL FLASH parameter.

## Relationship to other parameters

If the OUTPUT JCL or the sysout DD statement also contains a COPIES parameter, JES prints with the forms overlay the number of copies specified in one of the following:

- COPIES=nnn, if the FLASH count is larger than nnn. For example, if COPIES=10 and FLASH=(LTHD,12) JES prints 10 copies, all with the forms overlay.
- The sum of the group-values specified in the COPIES parameter, if the FLASH count is larger than the sum. For example, if COPIES=(,(2,3,4)) and FLASH=(LTHD,12) JES prints nine copies in groups, all with the forms overlay.
- The count subparameter in the FLASH parameter, if the FLASH count is smaller than nnn or the sum from the COPIES parameter. For example, if COPIES=10 and FLASH=(LTHD,7) JES prints seven copies with the forms overlay and three copies without.

## Verification of forms overlay frame

Before printing starts, the system requests the operator to load the specified forms overlay frame in the printer. A frame must be loaded but the system cannot verify that it is the correct frame.

## Printing without flashing

To print without flashing, specify one of the following:

- FLASH=NONE on the DD or OUTPUT JCL statement.
- Omit the FLASH parameter on all of the statements for the data set and on all JES initialization statements.
- FLASH=(,0) on the OUTPUT JCL statement.

## Example of the FLASH parameter

```
//OUTDS1 OUTPUT COPIES=16,FLASH=(LTHD,7)
```

In this example, JES issues a message to the operator requesting that the forms overlay frame named LTHD be inserted in the printer. Then JES prints the first seven copies of the sysout data set with the forms overlay and the last nine without.

# FORMDEF parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the FORMDEF parameter to identify a library member that contains statements to tell the Print Services Facility (PSF) how to print the sysout data set on a page-mode printer (such as the 3800 Printing Subsystem Model 3). The statements can specify the following:

- Overlay forms to be used during printing.
- Location on the page where overlays are to be placed.
- Suppressions that can be activated for specified page formats.

The member must be in the library named in the cataloged procedure that was used to initialize PSF, or in a library specified in the USERLIB parameter.

**Note:** FORMDEF applies only for data sets printed on a PSF-managed AFP printer.

***References***

For more information, see *PSF for z/OS: User's Guide*.

## Syntax

```
FORMDEF=membername
```

## Subparameter definition

**membername**
Specifies the name of a library member. membername is 1 through 6 alphanumeric or national ($, #, @) characters; the first two characters are pre-defined by the system.

## Overrides

The library member specified by the OUTPUT JCL FORMDEF parameter can contain:

- Statements that override the installation's FORMDEF defaults in the PSF-cataloged procedure.
- A FORMDEF statement with a COPYGROUP parameter. The COPYGROUP parameter overrides any group-value subparameters on the OUTPUT JCL COPIES parameter or the sysout DD COPIES parameter.

  **Note:** The FORMDEF statement in the library member does not override a sysout DD or OUTPUT JCL COPIES=nnn parameter.

### Example of the FORMDEF parameter

```
//PRINT3 OUTPUT FORMDEF=JJPRT
```

In this example, PSF is to print the sysout data set on an AFP printer according to the parameters in the library member JJPRT.

## FORMLEN parameter

***Parameter type:*** Keyword, optional

***Purpose:*** A PSF user can use the FORMLEN parameter to set the length of pages for print without reconfiguring the printer.

### Syntax

```
FORMLEN=nn[.mmm]{IN|CM}
```

### Subparameter definition

**nn** Required. A one or two digit number, which can be zero.

**.mmm**
Optional. A decimal point (period) followed by up to three digits.

**{IN|CM}**
Required. The unit the decimal digits represent. Code IN for inches or CM for centimeters.

### Relationship to other control statements

FORMLEN is coordinated with FORMDEF (which may also be specified on the OUTPUT or PSF PRINTDEV statements).

### Examples of the FORMLEN parameter

***Example 1***

```
//OUTFORML OUTPUT FORMLEN=12.345CM
```

In this example the PSF user has requested that a specification of paper length 12.345 centimeters be sent to the printer.

***Example 2***

```
//OUTFORML OUTPUT FORMLEN=2IN
```

In this example the PSF user has requested that a specification of 2-inch paper length be sent to the printer.

**Note:** The decimal point and fractional portion are optional.

***Example 3***

```
//OUTFORML OUTPUT FORMLEN=0.1IN
```

In this example the PSF user has requested that a specification of 0.1-inch paper length be sent to the printer.

**Note:** You must specify at least one digit to the left of the decimal point.

# FORMS parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the FORMS parameter to identify the forms on which the sysout data set is to be printed or punched.

## Syntax

```
FORMS= {form-name}
       {STD      }
```

## Subparameter definition

**form-name**
Identifies the print or punch forms. form-name is 1 through 8 alphanumeric or national ($, #, @) characters.

**STD**
Indicates that JES3 is to use the standard form specified at JES3 initialization.

**Note:** STD is supported only on JES3 systems.

## Defaults

If you do not code a form-name subparameter, JES uses an installation default specified at initialization.

## Overrides

The form-name subparameter of the SYSOUT parameter on the sysout DD statement overrides the OUTPUT JCL FORMS parameter. Note that the SYSOUT form-name subparameter can be only four characters maximum while both the OUTPUT JCL FORMS form-name and the JES initialization default form names can be eight characters maximum.

## Example of the FORMS parameter

```
//OUTDS1 OUTPUT FORMS=ACCT4010
```

In this example, the sysout data set will be printed on forms named ACCT4010.

# FSSDATA parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the FSSDATA parameter for the intended purpose of each functional subsystem that documents this parameter.

## Syntax

```
FSSDATA=value
```

## Subparameter definition

**value**
Required. A subsystem-defined parameter (maximum = 127) to pass from a spooling product to a despooler.

The following considerations apply when you supply the value a functional subsystem requires:

**Characters valid when enclosed in apostrophes**

- You may include any EBCDIC text characters in an FSSDATA parameter value if you enclose the value in apostrophes. See *Character Set* in topic 4.2 for a list and description of valid EBCDIC text characters.
- You must enclose the value in apostrophes if it contains a blank.
- The system preserves trailing blanks you include as part of a value you enclose in apostrophes. For example, if you specify

  ```
  FSSDATA='SUNDAY   '
  ```

  the parameter value for the FSSDATA keyword is eight (8) characters, and a functional subsystem may deem it to be different from

  ```
  FSSDATA='SUNDAY'
  ```

  (6 characters) or

  ```
  FSSDATA='SUNDAY  '
  ```

  (7 characters).
- To code an apostrophe as part of the value, code two apostrophes, as well as enclosing the entire value in single apostrophes. Example:

  ```
  //OUT1     OUTPUT FSSDATA='New Year''s Day'
  ```

**Characters not requiring enclosing apostrophes**
Apostrophes are optional when "value" consists only of:

- Uppercase alphanumeric characters
- National characters @, $, and #
- Period (.)
- Asterisk (*). However, an asterisk followed by a period indicates a referral; *. is NOT allowed as the first two characters of the value
- Ampersand (&). An ampersand referring to a symbolic is substituted. Two consecutive ampersands are not substituted; they will result in a single ampersand as part of the value
- Plus sign (+)
- Hyphen(-)
- Slash (/)

**Characters you may not enclose in aphstrophes**
Do not enclosed **symbolic parameters** within apostrophes. The system will not resolve them.

## Defaults

None.

## Overrides

None.

## Relationship to other keywords on this statement

None.

## Relationship to other system functions

None.

## Examples of the FSSDATA parameter

A functional subsystem defines its intended content for the FSSDATA parameter value. The following are examples of the allowable syntax for the FSSDATA parameter. Parentheses enclose the resulting values (or portions thereof) to help distinguish them.

***Example 1***

```
//OUTDS1   OUTPUT FSSDATA=FSSVALUE
```

In Example 1 the FSSDATA parameter contains a value (FSSVALUE) that does not require apostrophes around it. This is because the value contains no blanks and consists only of characters that are valid without apostrophes.

***Example 2***

```
//OUTDS2   OUTPUT FSSDATA='Subsystem data'
```

In Example 2 the FSSDATA parameter contains a single value (Subsystem data) which you must enclose in apostrophes because of the embedded blank.

***Example 3***

```
//OUTDS3   OUTPUT FSSDATA='AOPPT=CFF'
```

In Example 3 the FSSDATA parameter contains a value (AOPPT=CFF) within apostrophes. The parameter value consists of a string that a functional subsystem could use to identify a defined keyword (AOPPT) and its parametric value (CFF).

***Example 4***

```
//PROCC   PROC   PARM1=FSSDATA
//STEPC   EXEC   PGM=MYPGM
//OUTDS4 OUTPUT FSSDATA=&PARM1
```

In Example 4 the FSSDATA parameter contains a value whose first character is an ampersand (&). "Value" consists of a string that a functional subsystem could use to identify a symbolic parameter. The system takes the procedure default for the value from the PROC statement (FSSDATA).

***Example 5***

```
//PROCD   PROC   PARM1=FSSDATA
//STEPD   EXEC   PGM=MYPGM
//OUTDS5 OUTPUT FSSDATA='&PARM1'
```

In Example 5 the FSSDATA parameter contains a value enclosed within apostrophes, where the first character of the value is an ampersand (&). The value consists of a string that a functional subsystem could use to identify a symbolic

parameter. Because the subsystem-defined parameter is enclosed within apostrophes, the system does not resolve the &PARM1 symbolic; it leaves the parameter value unchanged (&PARM1).

***Example 6***

```
//OUTDS3   OUTPUT FSSDATA='printer=MyPrinter'
```

In Example 6, the FSSDATA parameter contains a value (printer=MyPrinter) within apostrophes. The parameter value consists of a string that Infoprint Server uses to identify a defined keyword (printer), and its value (MyPrinter). For more information about Infoprint Server and the printer keyword, see *z/OS Infoprint Server User's Guide*.

# GROUPID parameter

***Parameter type***

Keyword, optional, JES2 only

***Purpose***

Use the GROUPID parameter to specify that the sysout data set belongs to an output group. The data sets in an output group are processed together in the same location and time. Data sets to be grouped should have similar characteristics: the same output class, destination, process mode, and external writer name.

**Note:** GROUPID is supported only on JES2 systems.

## Syntax

```
GROUPID=output-group
```

## Subparameter definition

**output-group**
Specifies the name of an output group. The output-group is 1 through 8 alphanumeric characters and is selected by the programmer to define an output group for this job. The name is not installation-defined.

## Relationship to other control statements

If you code FREE=CLOSE on a sysout DD statement that references an OUTPUT JCL statement containing a GROUPID parameter, JES2 will not group the data sets into one output group. Instead, JES2 produces one copy of the sysout data set for each OUTPUT JCL statement that the DD statement references.

## Examples of the GROUPID parameter

***Example 1***

```
//EXMP5  JOB     ACCT1984,MAEBIRD,MSGCLASS=A
//OUTRPT OUTPUT  GROUPID=RPTGP,DEFAULT=YES,DEST=TDC
//STEP1  EXEC    PGM=RPTWRIT
//SYSIN  DD      *
         .
         .
```

```
        .
/*
//RPTDLY DD      SYSOUT=C
//RPTWK  DD      SYSOUT=C
```

In this example, the DD statements RPTDLY and RPTWK implicitly reference the default OUTPUT JCL statement OUTRPT. JES2 creates two output groups:

1. Group RPTGP is created because of the GROUPID parameter in the OUTPUT JCL statement. It contains the two reports from the sysout DD statements RPTDLY and RPTWK and is printed at the destination TDC. The programmer named this group RPTGP.
2. The other group is named by JES2. It contains the system-managed data set for the job's messages.

***Example 2***

```
//EXAMP    JOB     MSGCLASS=A
//JOBOUT   OUTPUT  GROUPID=SUMM,DEST=HQS,CHARS=GT10
//STEP1    EXEC    PGM=RWRITE
//OUT1     OUTPUT  FORMS=STD,CHARS=GS10,DEST=LOCAL
//RPT1     DD      SYSOUT=A,OUTPUT=(*.OUT1,*.JOBOUT)
//STEP2    EXEC    PGM=SWRITE
//OUT2     OUTPUT  FORMS=111,CHARS=GB10,DEST=LOCAL
//RPT2     DD      SYSOUT=B,OUTPUT=(*.OUT2,*.JOBOUT)
```

This job causes JES2 to produce five sets of output:

- 1.1.1, containing the system-managed data sets. This set is specified through the JOB statement MSGCLASS parameter.
- SUMM.1.1, containing a copy of the data set defined by DD statement RPT1. This set is specified through the second OUTPUT subparameter: *.JOBOUT. It is for output class A.
- SUMM.2.1, containing a copy of the data set defined by DD statement RPT2. This set is specified through the second OUTPUT subparameter: *.JOBOUT. Because it is for output class B, it is in a separate subgroup from the SUMM.1.1 subgroup.
- 4.1.1, containing a copy of the data set defined by DD statement RPT1. This set is specified through the first OUTPUT subparameter: *.OUT1.
- 5.1.1, containing a copy of the data set defined by DD statement RPT2. This set is specified through the first OUTPUT subparameter: *.OUT2.

## INDEX parameter

***Parameter type***

Keyword, optional, JES2 only

***Purpose***

Use the INDEX parameter to set the left margin for output on a 3211 Printer with the indexing feature. The width of the print line is reduced by the INDEX parameter value.

**Note:** INDEX is supported only on JES2 systems and only for output printed on a 3211 with the indexing feature. JES2 ignores the INDEX parameter if the printer is not a 3211 with the indexing feature.

## Syntax

```
INDEX=nn
```

## Subparameter definition

**nn**
Specifies how many print positions the left margin on the 3211 output is to be indented. nn is a decimal number from 1 through 31. n=1 indicates flush-left; n=2 through n=31 indent the print line by n-1 positions.

## Defaults

The default is 1, which indicates flush left. Thus, if you do not code an INDEX or LINDEX parameter, JES2 prints full-width lines.

## Relationship to other parameters

INDEX and LINDEX are mutually exclusive; if you code both, JES2 uses the last one encountered. Note that you cannot index both the left and right margins.

## Example of the INDEX parameter

```
//OUT17  OUTPUT INDEX=6
```

In this example, because the printed report is to be stapled, extra space is needed on the left. Assuming the data set is printed on a 3211 with the indexing feature, all lines are indented 5 print positions from the left page margin.

# INTRAY parameter

*Parameter type*

Keyword, optional

*Purpose*

Use INTRAY to specify the paper source. This overrides what is specified in the FORMDEF in use.

## Syntax

```
INTRAY=nnn
```

## Subparameter definition

**nnn**
Specifies the paper source, where nnn is a number from 1 to 255. To determine what value to specify, see the documentation for your printer.

## Relationship to other keywords on this statement

If OUTBIN is specified, the paper from the INTRAY must be compatible with the output bin.

## Example of the INTRAY parameter

```
//OUTTRAY  OUTPUT INTRAY=2
```

In this example, 2 is the paper source.

# JESDS parameter

*Parameter type*

Keyword, optional

*Purpose*

Use the JESDS parameter to process the job's system-managed data sets according to the parameters on this OUTPUT JCL statement. The system-managed data sets consist of:

- The job log, which is a record of job-related information for the programmer. Printing of the job log is controlled by two JOB statement parameters: the MSGLEVEL parameter controls what is printed and the MSGCLASS parameter controls the system output class.
- The job's hard-copy log, which is a record of all message traffic for the job to and from the operator console.
- System messages for the job.

**Note:** In a JES3 environment, a job can complete processing before all of its messages have been written to the job log. When this occurs, the job's output is incomplete. For this reason, do not use the contents of the job log as an automation or as a programming interface.

*References*

For more information on the job log, see *z/OS MVS System Commands*.

## Syntax

```
JESDS= {ALL}
       {JCL}
       {LOG}
       {MSG}
```

## Subparameter definition

**ALL**
Indicates that this OUTPUT JCL statement applies to all of the job's system-managed data sets.

**LOG**
Indicates that this OUTPUT JCL statement applies only to the JESMSGLG data set, which contains the JES and operator messages for this job.

**JCL**
Indicates that this OUTPUT JCL statement applies only to the JESJCL data set, which contains the JCL statements for this job.

**MSG**
Indicates that this OUTPUT JCL statement applies only to the JESYSMSG data set, which contains any JCL error messages and any system messages for this job.

## Overrides

The NOLOG parameter on a JES2 /*JOBPARM statement overrides the OUTPUT JCL JESDS=ALL parameter.

If an OUTPUT JCL statement contains both JESDS and CLASS parameters, the CLASS parameter will override the MSGCLASS parameter on the JOB statement for the specified JES data sets.

## Location in the JCL

Place an OUTPUT JCL statement containing JESDS before the first EXEC statement of the job. An OUTPUT JCL statement containing JESDS placed after an EXEC statement is a JCL error.

You can place more than one OUTPUT JCL statement containing JESDS before the first EXEC statement. JES creates a copy of the job's system data sets for each.

## Destination for the system data sets

If you want the job's system data sets processed at a particular destination, code a DEST parameter on the OUTPUT JCL statement containing JESDS. Otherwise, JES routes the system data sets to a local device.

## JES2 processing with JESDS

JES2 processes OUTPUT JCL statements for system-managed data sets (JESDS parameter) only if a job starts execution.

System-managed data sets are not processed for the following jobs because the jobs do not start execution:

- Jobs that specify a TYPRUN value on the JOB statement that prevents execution, such as COPY or SCAN.
- Jobs that do not execute because of a JCL error, an error in a JES2 control statement, or a system failure in JES2 input processing.

## JES3 processing with JESDS

System-managed data sets are not processed by JES3 for the following jobs because the jobs do not complete execution:

- Jobs that specify a TYPRUN value on the JOB statement that prevents execution, such as SCAN.
- Jobs that do not execute because of a JCL error.

## Example of the JESDS parameter

```
//EXMP    JOB     MSGCLASS=A
//OUT1    OUTPUT  JESDS=ALL
//OUT2    OUTPUT  JESDS=ALL,DEST=AUSTIN
          .
          .
          .
```

In this example, JES produces two copies of the system-managed data sets: one copy for OUTPUT JCL statement OUT1 and one copy for OUTPUT JCL statement OUT2. The copy for statement OUT2 is sent to AUSTIN.

## LINDEX parameter

***Parameter Type***

Keyword, optional, JES2 only

***Purpose***

Use the LINDEX parameter to set the right margin for output on a 3211 Printer with the indexing feature. The width of the print line is reduced by the LINDEX parameter value.

**Note:** LINDEX is supported only on JES2 systems and only for output printed on a 3211 with the indexing feature. JES2 ignores the LINDEX parameter if the printer is not a 3211 with the indexing feature.

### Syntax

```
LINDEX=nn
```

### Subparameter definition

**nn**
Specifies how many print positions the right margin on the 3211 output is to be moved in from the full page width. nn is a decimal number from 1 through 31. n=1 indicates flush-right; n=2 through n=31 move the right margin over by n-1 positions.

### Defaults

The default is 1, which indicates flush right. Thus, if you do not code an INDEX or LINDEX parameter, JES2 prints full-width lines.

### Relationship to other parameters

INDEX and LINDEX are mutually exclusive; if you code both, JES2 uses the last one encountered. Note that you cannot index both the left and right margins.

### Example of the LINDEX parameter

```
//OUT18  OUTPUT LINDEX=21
```

In this example, the author of the report wanted extra space on the right side of the paper for notes. Assuming the data set is printed on a 3211 with the indexing feature, all lines are ended 20 print positions from the right page margin.

# LINECT parameter

***Parameter type***

Keyword, optional, JES2 only

***Purpose***

Use the LINECT parameter to specify the maximum number of lines JES2 is to print on each output page.

**Note:** LINECT is supported only on JES2 systems.

## Syntax

```
LINECT=nnn
```

## Subparameter definition

**nnn**
Specifies the maximum number of lines JES2 is to print on each page. nnn is a number from 0 through 255.

Specify LINECT=0 to keep JES2 from starting a new page when the number of lines exceeds the JES2 initialization parameter.

## Defaults

If you do not code the LINECT parameter, JES2 obtains the value from one of the following sources, in order:

1. The linect field of the accounting information parameter on the JOB statement.
2. The installation default specified at JES2 initialization.

## Example of the LINECT parameter

```
//PRNTDS OUTPUT LINECT=45
```

In this example, JES2 will start a new page after every 45 lines.

# MAILBCC parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Use the MAILBCC parameter to specify one or more e-mail addresses of the blind copy (bcc) recipients of an e-mail. A bcc means that other recipients of the e-mail do not see the bcc recipient listed.

## Syntax

```
MAILBCC= {('bcc address'[,'bcc address']...)}
         {bcc-address                       }
```

See “Using enclosing apostrophes in OUTPUT parameters” on page 477 for the MAILBCC parameter.

## Subparameter definition

**bcc address**
Specifies the e-mail addresses of the blind copy (bcc) recipients of an e-mail. You can code up to 32 bcc-addresses. Each address can be 1 - 60 EBCDIC text characters. See "Character sets" on page 21 for a description of EBCDIC text characters.

## Defaults

There is no default for MAILBCC.

## Overrides

There are no override considerations for MAILBCC.

## Relationship to other system functions

This keyword can be used by Infoprint Server. For more information about this keyword when you use Infoprint Server, see *z/OS Infoprint Server User's Guide*.

## Examples of the MAILBCC parameter

***Example 1***

```
//OUTDS2   OUTPUT  MAILBCC=('robertanders@companya.net',
//           'suesmth1@companyb.net')
```

In this example, the system will send a blind copy to the following e-mail addresses:

- robertanders@companya.net
- suesmth1@companyb.net

***Example 2***

```
//OUTDS2   OUTPUT  MAILBCC=(ROBERT@XYZ.NET,
//           SUE@XYZ.NET)
```

In this example, the system will send a blind copy to the following e-mail addresses:

- robert@xyz.net
- sue@xyz.net

# MAILCC parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Use the MAILCC parameter to specify one or more e-mail addresses of the copy (cc) recipients of an e-mail. A cc means that other recipients of the e-mail can see the cc recipient listed.

## Syntax

```
MAILCC= {('cc address'['cc address']...)}
        {cc-address                     }
```

See "Using enclosing apostrophes in OUTPUT parameters" on page 477 for the MAILCC parameter.

## Subparameter definition

**cc address**
Specifies the e-mail addresses of the copy (cc) recipients of an e-mail. You can code up to 32 cc addresses. Each address can be 1 - 60 EBCDIC text characters. See "Character sets" on page 21 for a description of EBCDIC text characters.

## Defaults

There is no default for MAILCC.

## Overrides

There are no override considerations for MAILCC.

## Relationship to other system functions

This keyword can be used by Infoprint Server. For more information about this keyword when you use Infoprint Server, see *z/OS Infoprint Server User's Guide*.

## Examples of the MAILCC parameter

***Example 1***

```
//OUTDS2   OUTPUT  MAILCC=('robertanders@companya.net',
//           'suesmth1@companyb.net')
```

In this example, the system will send a copy to the following e-mail addresses:

- robertanders@companya.net
- suesmth1@companyb.net

***Example 2***

```
//OUTDS2   OUTPUT  MAILCC=(ROBERT@XYZ.NET,
//           SUE@XYZ.NET)
```

In this example, the system will send a copy to the following e-mail addresses:

- robert@xyz.net
- sue@xyz.net

# MAILFILE parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Use the MAILFILE parameter to specify the file name of the attachment to an e-mail.

## Syntax

```
MAILFILE= {'file id'}
          {file-id  }
```

See "Using enclosing apostrophes in OUTPUT parameters" on page 477 for the MAILFILE parameter.

## Subparameter definition

**file id**
Specifies the name of a file attached in an e-mail. The file id can be 1 - 60 EBCDIC text characters. See "Character sets" on page 21 for a description of EBCDIC text characters.

## Defaults

Infoprint Server uses the last qualifier of the data set name as the name of the e-mail attachment. You can specify the last qualifier in the DSNAME parameter of the DD statement.

## Overrides

If you do not specify the DSNAME parameter, Infoprint Server uses the job name.

## Relationship to other system functions

This keyword can be used by Infoprint Server. For more information about this keyword when you use Infoprint Server, see *z/OS Infoprint Server User's Guide*.

## Example of the MAILFILE parameter

```
//OUTDS2   OUTPUT  MAILFILE='third quarter growth chart'
```

In this example, the system will use the name `third quarter growth chart` for the attached file.

# MAILFROM parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Use the MAILFROM parameter to specify the descriptive name or other identifier of the sender of an e-mail.

## Syntax

```
MAILFROM = {'from address'}
           {from-address  }
```

See "Using enclosing apostrophes in OUTPUT parameters" on page 477 for the MAILFROM parameter.

## Subparameter definition

**from address**
Specifies descriptive name or other identifier of the sender of an e-mail. The from address can be 1 - 60 EBCDIC text characters. See "Character sets" on page 21 for a description of EBCDIC text characters.

## Defaults

There is no default for MAILFROM. However, Infoprint Server always includes *userid@domainname* to identify the sender. *userid* is the TSO user ID of the job submitter and *domainname* is the domain name where Infoprint Server is running. For example, if someone with a TSO user ID of JOHN sends an e-mail, and Infoprint Server is running on domain SYSTEM1, Infoprint Server will include JOHN@SYSTEM1.

## Overrides

There are no override considerations for MAILFROM.

## Relationship to other system functions

This keyword can be used by Infoprint Server. For more information about this keyword when you use Infoprint Server, see *z/OS Infoprint Server User's Guide*.

## Example of the MAILFROM parameter

```
//OUTDS2   OUTPUT  MAILFROM='John Q. Sender'
```

In this example, the system will identify John Q. Sender <JOHNS@COMPANY1.COM> as the sender of the e-mail. JOHNS is the job submitter's TSO user ID, and COMPANY1.COM is the domain name of the z/OS system.

# MAILTO parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Use the MAILTO parameter to specify one or more e-mail address of the recipients of an e-mail.

## Syntax

```
MAILTO= {('to address'[,'to address']...)}
        {to-address                      }
```

See "Using enclosing apostrophes in OUTPUT parameters" on page 477 for the MAILTO parameter.

## Subparameter definition

**to address**
Specifies the e-mail addresses of the recipients. You can code up to 32 addresses. Each address can be 1 - 60 EBCDIC text characters. See "Character sets" on page 21 for a description of EBCDIC text characters.

## Defaults

There is no default for MAILTO.

## Overrides

There are no override considerations for MAILTO.

## Relationship to other system functions

This keyword can be used by Infoprint Server. For more information about this keyword when you use Infoprint Server, see *z/OS Infoprint Server User's Guide*.

## Example of the MAILTO parameter

```
 //OUTDS2   OUTPUT  MAILTO=('gwashngtn1@companya.com',
//           'mwshngtn@companyb.org')
```

In this example, the system will send the output to the following e-mail addresses:

- gwashngtn1@companya.com
- mwshngtn@companyb.org

## MERGE parameter

Keyword, optional

***Purpose***

Specifies whether or not the parameters which are specified on the OUTPUT JCL statement will be merged with the output statement for the job. The default value is NO.

### Syntax

```
MERGE={YES}
      {Y}
      {NO}
      {N}
```

### Subparameter definition

**YES | Y**
Indicates that the parameters which are specified on the OUTPUT JCL statement will be the default OUTPUT parameters for the job.

**NO | N**
Indicates that the parameters which are specified on the OUTPUT JCL statement will not be the default OUTPUT statement parameters for the job.

### Defaults

The default value for MERGE is NO. This indicates that the parameters which are specified on the OUTPUT JCL statement will not be the default OUTPUT statement parameters for the job.

### Example of the MERGE parameter

In the following example, the parameters specified on the OUTPUT JCL statement will be the default OUTPUT parameters for the job:

```
MERGE=YES
```

## MODIFY parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the MODIFY parameter to specify a copy-modification module that tells JES how to print the sysout data set on a 3800 Printing Subsystem. The module can specify the following:

- Legends.
- Column headings.

- Where and on which copies to print the data.

The module is defined and stored in SYS1.IMAGELIB using the IEBIMAGE utility program.

**Note:** MODIFY applies only for the 3800 Printing Subsystem Model 1 and 2 and the 3800 Printing Subsystem Models 3, 6 and 8 in compatibility mode. For page-mode printers (such as the 3800 Model 3 or the Infoprint 4000), use the FORMDEF and PAGEDEF parameters to obtain the same functions.

*References*

For more information on the copy modification module and the IEBIMAGE utility program, see *z/OS DFSMSdfp Utilities*.

## Syntax

```
MODIFY= {module-name          }
        {([module-name][,trc])}
```

- You can omit the module-name, thereby obtaining the initialization default. For example, MODIFY=(,2).
- The trc subparameter is optional. If you omit it, you can omit the parentheses.

## Subparameter definition

**module-name**
Identifies a copy-modification module in SYS1.IMAGELIB. The module-name is 1 through 4 alphanumeric or national ($, #, @) characters.

**trc**
Identifies which character-arrangement table named in the CHARS parameter is to be used. This **table reference character** is 0 for the first font-name specified, 1 for the second, 2 for the third, or 3 for the fourth. The CHARS parameter used is on the following, in override order:

1. The DD statement.
2. This OUTPUT JCL statement.
3. A statement in the library member specified on the OUTPUT JCL PAGEDEF parameter.
4. A statement in the SYS1.IMAGELIB member obtained by default.
5. A JES3 initialization statement.

## Defaults

If you do not code module-name in the MODIFY parameter, JES3 uses an installation default specified at initialization. JES2 provides no installation default at initialization.

If you do not specify **trc**, the default is 0. If the trc value is greater than the number of font-names in the CHARS parameter, JES2 uses the first character-arrangement table named in the CHARS parameter and JES3 uses the last character-arrangement table named in the CHARS parameter.

## Overrides

A MODIFY parameter on the sysout DD statement overrides the OUTPUT JCL MODIFY parameter.

## Relationship to other parameters

The second character of each logical record can be a TRC code, so that each record can be printed in a different font. This way of specifying fonts is indicated by the OUTPUT JCL TRC parameter.

## Example of the MODIFY parameter

```
//OUTDS1 OUTPUT  CHARS=(GT12,GB12,GI12),MODIFY=(MODA,2)
```

In this example, JES loads the MODA module in SYS1.IMAGELIB into the 3800 and uses GI12, Gothic Italic 12-pitch font, which is the third table name specified in the CHARS parameter.

# NAME parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the NAME parameter to print a preferred name on the separator pages of the output for a sysout data set. The preferred name is the name associated with the output. An installation can use the preferred name to assist in sysout distribution.

## Syntax

```
NAME= {'preferred name'}
      {preferred-name  }
```

See “Using enclosing apostrophes in OUTPUT parameters” on page 477 for the NAME parameter.

## Subparameter definition

**preferred name**
Specifies the preferred name that is associated with the sysout. The preferred name is 1 - 60 EBCDIC text characters. See “Character sets” on page 21 for a description of EBCDIC text characters.

## Defaults

- **In an APPC scheduling environment:** In both JES2 and JES3 systems, if you do not code the NAME parameter on the OUTPUT JCL statement, the system uses the value defined in the transaction program (TP) user's RACF profile when:
  - The user submitting the TP profile has a RACF profile defined for him, and
  - The transaction program profile includes TAILOR_SYSOUT(YES).

  Otherwise, the system uses the value defined on the transaction initiator's job statement.

- **In a non-APPC scheduling environment:** In a JES2 system, if you do not code the NAME parameter on the OUTPUT JCL statement, the system uses the name defined on the job statement.

  In a JES3 system, there is no default for the NAME parameter on the OUTPUT JCL statement.

## Overrides

- **In an APPC scheduling environment:** In both JES2 and JES3 systems, the NAME parameter on the OUTPUT JCL statement overrides the name defined in the RACF profile. The name in the RACF profile overrides the name defined in the transaction initiator's JOB statement.
- **In a non-APPC scheduling environment:** In both JES2 and JES3 systems, the NAME parameter on the OUTPUT JCL statement overrides the name defined on the JOB statement.

## Example of the NAME parameter

```
//OUTDS7   OUTPUT  NAME='R. ROPER'
```

In this example, the name R. ROPER will be printed on the line reserved for NAME on separator pages of any sysout data set that references OUTDS7.

# NOTIFY parameter

***Parameter type***

Keyword, optional - Use this parameter only for printers managed by PSF or Infoprint Server.

***Purpose***

Use the NOTIFY parameter to have PSF issue a print completion message to up to four users. The message identifies the output that has completed printing, and indicates whether the printing was successful. This parameter is effective for PSF devices and any FSS products that support the NOTIFY keyword (such as Infoprint Server); it has no effect for JES-mode devices. The print completion message is issued:

- on a JES2 system: when printing for all the sysout data sets for an output group has completed. An output group consists of the sysout data sets printed between the output header page and the output trailer page of a job.
- on a JES3 system: when the sysout data sets for the same printer and the same job have been printed.

## Syntax

```
NOTIFY= [node.]userid
        ([node1.]userid1,[node2.]userid2,...[node4.]userid4)
```

- You can omit the parentheses if you code only one destination.
- For any destination, you can omit the node name.

## Subparameter definitions

**[node]userid**
Specifies the node name and userid of a recipient of the print completion message.

## Defaults

If you do not code the NOTIFY parameter, the system will not issue a print completion message. If you do not specify node, it will default to the node where the job was submitted.

## Examples of the NOTIFY parameter

*Example 1*

```
//OUT1    OUTPUT   NOTIFY=BLDVM2.RICH1
```

In this example, the system sends a print completion message to RICH1 at BLDVM2.

*Example 2*

```
//OUT1    OUTPUT   NOTIFY=(BLDVM2.RICH1,CARTER)
```

In this example, the system sends a print completion message to RICH1 at BLDVM2, and to the userid CARTER at the node where the job was submitted.

# OFFSETXB parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Use OFFSETXB to specify the offset in the X direction from the page origin (or partition origin for N_UP) for the back side of each page of output. This overrides what is specified in the FORMDEF in use. For more information on page offsets see the section "Page Position" in *PSF for z/OS: User's Guide*.

## Syntax

```
OFFSETXB=mmmm[.nnn]{IN    }
                   {CM    }
                   {MM    }
                   {PELS  }
                   {POINTS}
```

## Subparameter definition

**mmmm[.nnn]**
Specifies a *value*, which may be one (m), two (mm), three (mmm), or four (mmmm) digits (and which may be zero), and which optionally may be followed by a decimal point (a period) and up to three (nnn) digits.

**IN | CM | MM | PELS | POINTS**
A mandatory *unit* that follows the *value*. The *unit* can be inches (IN), centimeters (CM), millimeters (MM), pels, or points. If you specify the unit as pels or points you must specify the value as a whole number with no decimal point.

### Relationship to other keywords on this statement

The OFFSETXB parameter is used in conjunction with the OFFSETXF, OFFSETYB, and OFFSETYF parameter to define the page origin.

### Example of the OFFSETXB parameter

```
//OUTSET  OUTPUT OFFSETXB=10.5MM
```

In this example, the page is to be offset 10.5 millimeters in the X direction from the page origin on the back of each sheet.

## OFFSETXF parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Similar to OFFSETXB (with the same units, values, and restrictions), OFFSETXF is used to specify the offset in the X direction from the page origin (or partition origin for N_UP) for the front side of each page of output.

## OFFSETYB parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Similar to OFFSETXB (with the same units, values, and restrictions), OFFSETYB is used to specify the offset in the Y direction from the page origin (or partition origin for N_UP) for the back side of each page of output.

## OFFSETYF parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Similar to OFFSETXB (with the same units, values, and restrictions), OFFSETYF is used to specify the offset in the Y direction from the page origin (or partition origin for N_UP) for the front side of each page of output.

## OUTBIN parameter

***Parameter type:*** Keyword, optional

***Purpose:*** The OUTBIN keyword specifies the printer output bin identifier to be used for the sysout data set. See *PSF for z/OS: User's Guide* for more information on multiple media destinations and OUTBIN processing.

### Syntax

```
OUTBIN = nnnnn
```

### Subparameter definition

**nnnnn**
Species the ID of the printer output bin where the data set is to be sent. nnnnn is 1 through 5 decimal digits from 1 to 65535.

## Defaults

If the OUTBIN keyword is not specified, PSF (Print Services Facility) will stack the output in the printer default output bin. If OUTBIN specifies a value that is not one of the supported identifiers, PSF will stack the output in the printer default output bin and issue a message indicating that the requested bin is not available.

## Overrides

The OUTBIN value can be overridden via the JES3 *MODIFY command.

## Relationship to other system functions

JES3 printers use OUTBIN as a grouping attribute and will print header and trailer pages around each group of data sets with unique OUTBIN specifications.

## Example of the OUTBIN parameter

```
//OUT1 OUTPUT DATACK=UNBLOCK,OUTBIN=2,TRC=N0
```

In this example, the user has specified an output bin id of '2'.

# OUTDISP parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Use the OUTDISP parameter to indicate the disposition of a sysout data set. You can code different dispositions based on whether the job completes successfully.

***Considerations for an APPC scheduling environment:*** In an APPC scheduling environment, sysout data sets are treated as spin data sets. The system will process only the normal output disposition. If you code an abnormal output disposition, the system will check it for syntax and then ignore it.

If the automatic restart manager (ARM) restarts a job, JES discards all non-spin sysout data sets created during the previous execution. (You can avoid losing that output by adding SPIN=UNALLOC to the DD statement for the SYSOUT data set.)

## Syntax

```
{OUTDISP=(normal-output-disposition,abnormal-output-disposition)}

OUTDISP= ([WRITE] [,WRITE])
         ([HOLD ] [,HOLD ])
         ([KEEP ] [,KEEP ])
         ([LEAVE] [,LEAVE])
         ([PURGE] [,PURGE])
```

- If you code only the normal-output-disposition, you can omit the parentheses.
- If you code only the abnormal-output-disposition, enclose the disposition in parentheses and precede it with a comma. For example:

```
//OUTDS   OUTPUT  OUTDISP=(,PURGE)
```

## Subparameter definitions

**WRITE**
Indicates that the system is to print the sysout data set. After printing the data set, the system purges it.

Unless it is held by the system or operator, a sysout data set with the disposition WRITE will always print.

**HOLD**
Indicates that the system is to hold the sysout data set until the user or operator releases it. Releasing the sysout data set changes its disposition to WRITE.

If HOLD output is not released, the system holds it until the user or operator purges it.

**NJE Note:** In an NJE environment, the system does not hold the data set until it reaches its ultimate destination node.

**KEEP**
Indicates that the system is to print the sysout data set. After printing the data set, the system changes its disposition to LEAVE.

**LEAVE**
Indicates that after the user or operator releases the sysout data set, the disposition of the data set changes to KEEP.

If LEAVE output is not released, the system holds it until the user or operator purges it.

**PURGE**
Indicates that the system is to delete the sysout data set without printing it.

## Defaults

In a JES2 system, if you do not specify OUTDISP, the system uses the installation defaults for normal and abnormal disposition for the sysout class of the data set.

In a JES3 system, if you do not specify OUTDISP, then any normal and abnormal disposition defined for the sysout class of the data set is used. If OUTDISP is not specified or defined for the sysout class, then no output disposition is used.

If you do not specify an abnormal output disposition, the system uses the normal disposition that you specified.

If you specify an abnormal disposition but do not specify a normal disposition, the normal disposition defaults to WRITE.

## Overrides

In a JES2 system, the DD statement HOLD=YES or HOLD=NO parameter overrides the OUTDISP parameter.

In a JES3 system, the DD statement HOLD=NO parameter overrides the OUTDISP parameter.

## Relationship to other control statements

A data set defined by a sysout DD statement that contains a DSID parameter is always held. The system ignores the OUTDISP parameter on an OUTPUT JCL statement that is referenced by such a DD statement.

## Examples of the OUTDISP parameter

***Example 1***

```
//OUTDS6   OUTPUT  OUTDISP=(KEEP,PURGE)
```

When the job completes successfully, the disposition of the data set is KEEP. After the sysout is printed, the data set disposition changes to LEAVE, and the sysout data set is held until released by the user or operator.

If the job does not complete normally, the system purges the data set without any post-execution processing.

***Example 2***

```
//OUTNORM   OUTPUT  OUTDISP=(WRITE,PURGE),DEST=ROOM111
//OUTBAD    OUTPUT  OUTDISP=(PURGE,HOLD),NAME='D JONES'
//DD5       DD      SYSOUT=A,OUTPUT=(*.OUTNORM,*.OUTBAD)
```

If the job completes successfully, the output for DD DD5 is to be sent to the destination ROOM111. If the job does not complete successfully, the output is to be held for a programmer named D JONES. D JONES can view the output on the screen, and then purge it or release it to be printed if further diagnosis is required.

There are two OUTPUT statements, OUTNORM and OUTBAD. In any given case, however, only one of the OUTPUT statements actually produces output. For successful completion, the WRITE option on the OUTNORM statement specifies that the output should be printed and sent to ROOM111, and the PURGE option on OUTBAD specifies that no output is produced for the OUTBAD statement. For unsuccessful completion, the HOLD option on the OUTBAD statement specifies that the output should be held for D JONES, and the PURGE option on OUTNORM specifies that no output is produced for the OUTNORM statement.

***Example 3***

```
//SYSOUTK   OUTPUT  OUTDISP=(WRITE,HOLD)
//REPORT1   DD      SYSOUT=K,OUTPUT=*.SYSOUTK
```

The system processes the data set using OUTPUT statement SYSOUTK.

When the job completes successfully, the WRITE option specifies that the system should print and then purge the output.

When the job does not complete successfully, the HOLD option specifies that the system should hold the output.

# OVERLAYB parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Specifies to place the named medium overlay on the back side of each sheet to print.

## Syntax

```
OVERLAYB=ovlyname
```

### Subparameter definition

**ovlyname**
Specifies the medium overlay name, where the overlay name is 1 though 8 alphanumeric or national ($, #, @) characters and the first of those characters is alphabetic or national.

### Relationship to other keywords on this statement

The overlay specified is in addition to any overlays from other sources.

### Example of the OVERLAYB parameter

```
//OUTOVLY  OUTPUT OVERLAYB=MYOVLY
```

In this example, the overlay named MYOVLY will be included on the back side of each sheet for this data set.

## OVERLAYF parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Similar to OVERLAYB, with the same restrictions on the name, OVERLAYF specifies to place the named medium overlay on the front side of each sheet to print.

## OVFL parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Use the OVFL parameter to specify whether the printer program (JES3 output writer) should check for forms overflow (by sensing channel 12 as defined in the FCB that is used for printing the output).

**Note:** OVFL is supported only on JES3 systems. Neither JES2 nor Print Services Facility (PSF) supports OVFL.

### Syntax

```
OVFL = [ON|OFF]
```

### Subparameter definition

**ON** Indicates that the printer program should eject (skip to channel 1) whenever the end-of-forms indicator (channel 12) is sensed.

**OFF**
Indicates that forms overflow control is not to be used.

### Defaults

If you do not code the OVFL parameter, the default is ON.

## Example of the OVFL parameter

```
//WRTO   JOB    ACNO77,MAEBIRD,MSGCLASS=B
//DS23   OUTPUT DEFAULT=YES,FORMS=STD,OVFL=OFF
//STEP1  EXEC   PGM=DLYRPT
//DAILY  DD     SYSOUT=A
```

In this example, sysout DD statement DAILY implicitly references the default job-level OUTPUT JCL statement DS23. This OUTPUT JCL statement directs JES3 to print the daily report on standard forms. If no carriage control characters are used, the JES3 output writer will print the output as a continuous stream of data with no blank lines between pages.

# PAGEDEF parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the PAGEDEF parameter to identify a library member that contains statements to tell the Print Services Facility (PSF) how to print the sysout data set on a page-mode printer (such as the Infoprint 4000). The data set may be sysout or a data set that is allocated directly to a printer. The statements can specify the following:

- Logical page length and width.
- Fonts.
- Page segments.
- Multiple page types or formats.
- Lines within a page; for example:
  - Line origin.
  - Carriage controls.
  - Spacing.
- Multiple logical pages on a physical page.

The member must be in the library named in the cataloged procedure that was used to initialize PSF, or in a library specified in the USERLIB parameter.

**Note:** PAGEDEF applies only for data sets printed on a page-mode printer controlled by PSF.

***References***

For more information, see *PSF for z/OS: User's Guide*.

## Syntax

```
PAGEDEF=membername
```

## Subparameter definition

**membername**
Specifies the name of the library member. membername is 1 through 6 alphanumeric or national ($, #, @) characters; the first two characters are pre-defined by the system.

## Overrides

The statements in the library member specified by the OUTPUT JCL PAGEDEF parameter override the installation's PAGEDEF defaults in the PSF-cataloged procedure.

PSF uses the following parameters, in override order, to select the font list:

1. Font list in the library member specified by an OUTPUT JCL PAGEDEF parameter.
2. DD CHARS parameter.
3. OUTPUT JCL CHARS parameter.
4. DD UCS parameter.
5. OUTPUT JCL UCS parameter.
6. JES installation default for the device.
7. Font list on the PAGEDEF parameter in the PSF-cataloged procedure.

## Example of the PAGEDEF parameter

```
//OUTDS1 OUTPUT PRMODE=PAGE,PAGEDEF=SSPGE
```

In this example, PSF is to print the sysout data set on an AFP printer operating in page mode. The printing is to be done according to the parameters in the library member SSPGE.

# PIMSG parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Use the PIMSG parameter to indicate the handling of messages by Print Services Facility (PSF). PIMSG is used to specify whether all error messages are to be printed, and the number of errors sufficient to cause the printing process to be terminated and the data set to be purged.

When you code PIMSG=YES, the system prints all these messages at the end of the output data set.

When you code PIMSG=NO, no messages are printed unless there is an error that forces premature termination of the printing of the data set. If an error occurs, the system prints the set of messages (called a message group) associated with the error that caused the termination.

As errors are detected by PSF or reported to PSF by the printer, a count is kept of the associated message groups. When the count equals the number specified on the PIMSG parameter, PSF terminates the printing of the current data set. PSF interprets a count of zero as infinite, and does not terminate the printing of the data set on the basis of the number of errors detected.

**Note:** PIMSG can be specified only for data sets printed through PSF.

## Syntax

```
PIMSG=  {(YES[,msg-count])}
        {(NO[,msg-count]) }
```

- You can omit the parentheses if you do not specify msg-count.

## Subparameter definition

**YES**
Requests the system to print all messages generated by PSF. You can also code this subparameter as Y.

**NO**
Requests that the system print no error messages, unless printing of the data set is prematurely terminated. If a terminating error occurs, only the set of messages (called a message group) associated with the error that caused the termination is printed. You can also code this subparameter as N.

**msg-count**
Requests the system to cancel the printing of the current data set after the specified number of errors (as represented by the associated message groups) have been detected by PSF or reported to PSF by the printer. In this context, errors refers to data-stream errors, and errors resulting from any malfunction that would cause the printer to halt, such as a mechanism failure, or out-of-paper condition. However, these errors do not include those caused by operator intervention.

Valid values for msg-count are 0-999, where 0 is interpreted as infinite.

The types of errors that increment the message count are those that induce a message group to be printed at the end of the data set. However, even though the printing of the message groups is inhibited by PIMSG=NO, the associated error still increments the message count. (A message group consists of a primary message and variable number of informational messages that result from a single error.)

In the case that multiple transmissions have been specified for a single data set (user-specified multiple copies), the message count would apply on a per copy basis. If the specified number of errors are discovered during the printing of any copy, the subject copy is terminated, and the remaining copies are not printed.

## Defaults

If you do not code the PIMSG parameter, the PIMSG specification from the PSF PRINTDEV statement applies. If not specified in the PRINTDEV statement, the default is PIMSG=(YES,16). For information about the PRINTDEV statement, see *PSF for z/OS: Customization*.

## Examples of the PIMSG parameter

***Example 1***

```
//OUTDS2 OUTPUT DATACK=UNBLOCK,PIMSG=(YES,0)
```

In this example, regardless of how many message-generating errors are detected by PSF or reported to PSF by the printer, the printing of the current data set continues to completion or until a terminating error is encountered. All the messages are printed by the system.

*Example 2*

```
//OUTDS2 OUTPUT DATACK=UNBLOCK,PIMSG=(NO,5)
```

In this example, after five message-generating errors are detected by PSF or reported to PSF by the printer, the printing of the current data set is terminated. Only the last message group is printed by the system.

# PORTNO parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Use the PORTNO parameter to specify the TCP/IP port number at which the FSS (for example, Infoprint Server) connects to the printer.

## Syntax

```
PORTNO=nnnnn
```

## Subparameter definition

**nnnnn**
Specifies the TCP/IP port number, where nnnnn is a number from 1 through 65,535.

## Relationship to other system functions

The port number must match the port number configured at the printer.

## Example of the PORTNO parameter

```
//OUTPORT  OUTPUT PORTNO=5005
```

In this example, 5005 is the TCP/IP port number.

# PRMODE parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Use the PRMODE parameter to identify the process mode required to print the sysout data set. JES schedules the data set to a printer that can operate in the specified mode.

For a list of valid process modes, contact your system programmer.

## Syntax

```
PRMODE= {LINE        }
        {PAGE        }
        {process-mode}
```

## Subparameter definition

**LINE**
Indicates that the data set is to be scheduled to a line-mode printer.

**PAGE**
Indicates that the data set is to be scheduled to a page-mode printer.

**process-mode**
Specifies the required process mode. The process-mode is 1 through 8 alphanumeric characters.

For an NJE-transmitted data set, use PRMODE to request specific processing without having to obtain output classes for the node that processes the data set.

## Defaults

If you do not code the PRMODE parameter, JES schedules output processing as follows:

- If the sysout data set does not contain page-mode formatting controls, the process mode of line is given to the data set.
- If the sysout data set contains page-mode formatting controls, the process mode of page is given to the data set.

## Printing a line-mode data set using PSF

To print a line-mode data set using the Print Services Facility (PSF) and an AFP printer, code PRMODE=PAGE. PSF formats this line-mode data set using the installation's default values for PAGEDEF and FORMDEF defined in the PSF-cataloged procedure; if these defaults are unsatisfactory, code the PAGEDEF and FORMDEF parameters on the OUTPUT JCL statement.

## Example of the PRMODE parameter

```
//DS18   OUTPUT  PRMODE=LINE
```

In this example, JES schedules the sysout data set to a printer with a process mode of line.

# PRTATTRS parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Use the PRTATTRS keyword to specify one or more job attributes for Infoprint Server. The *z/OS Infoprint Server User's Guide* information supported job attributes and their syntax.

## Syntax

```
 PRTATTRS={'attributename=value attributename=value ...'}

-----------------------------------------------------------

 o   The minimum length is one character.

 o   The maximum length is 127 characters.

 o   Enclose the parameter in apostrophes because attribute
     names contain lower case letters.

 o   All EBCDIC text characters are valid.
```

## Parameter definition

**attributename=value**
Specifies an Infoprint Server job attribute. For more information on job attribute names and syntax for acceptable values, see *z/OS Infoprint Server User's Guide*.

## Defaults

No Default.

## Overrides

Specification of PRTATTRS might be ignored if the OUTPUT statement is associated with a SYSOUT data set that is not processed by the IP PrintWay™ extended mode component of Infoprint Server.

## Relationship to other keywords on this statement

None.

## Relationship to other control statements

None.

## Example of the PRTATTRS parameter

```
//OUTDS3  OUTPUT  PRTATTRS='document-codepage=ISO8859-1'
```

In this example, the IP PrintWay extended mode component of Infoprint Server uses the document-codepage specification to control code page translation for the SYSOUT data set associated with this OUTPUT statement.

# PRTERROR parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Specifies the disposition of the SYSOUT data set used if a terminating error occurs during printing through the PSF functional subsystem. (A terminating error is an error that the automated recovery of PSF cannot correct.) You can specify which of the following actions PSF is to take for a terminating error:

- Use the default error disposition (DEFAULT),
- Release the SYSOUT data set back to JES as complete (QUIT), or
- Hold for operator action (HOLD).

## Syntax

```
PRTERROR=(DEFAULT|QUIT|HOLD)
```

## Subparameter definition

**DEFAULT**
Specifies that PSF will take the standard (default) action if a terminating error occurs during printing. When operator action is expected to correct the error, PSF releases the SYSOUT data set for hold. Otherwise, it treats the SYSOUT data as complete. For JES2, processing of the data set proceeds according to the

OUTDISP keyword value that is associated with it. For JES3, the data set is deleted from the SPOOL. See the "Relationship to Other Control Statements" below.

**QUIT**
Specifies that PSF is to release the data set complete even if a terminating error occurs during printing. JES then disposes of the data set as if it completed printing successfully. For JES2, processing of the data set proceeds according to the OUTDISP keyword value that is associated with it. For JES3, the data set is deleted from the SPOOL. See "Relationship to Other Control Statements," below.

**HOLD**
Specifies that if a terminating error occurs during printing, the data set will remain on the JES SPOOL until the system operator releases it.

## Relationship to other control statements

For the JES2 subsystem, OUTDISP affects the processing when PRTERROR=DEFAULT or PRTERROR=QUIT is performed and PSF releases the data set as complete.

An installation can control (through the PSF PRINTDEV initialization statement) whether the system honors or ignores the specification of the PRTERROR keyword on the OUTPUT JCL statement or dynamic output descriptors.

**Note:** There are conditions under which PRTERROR has no effect. See *PSF for z/OS: Customization* for additional information.

## Examples of the PRTERROR parameter

***Example 1***

```
//OUTPRTER OUTPUT PRTERROR=HOLD
```

In this example, if a terminating error occurs during printing, the data set remains on the JES SPOOL until the system operator releases it.

***Example 2***

```
//OUTPRTER OUTPUT PRTERROR=QUIT
```

In this example, if a terminating error occurs during printing, PSF quits processing the data set and releases it as "complete," and JES applies processing appropriate for a completed data set.

# PRTOPTNS parameter

***Parameter type:*** Keyword, optional

***Purpose:*** PRTOPTNS defines a named entity that contains additional print options. JES does not use PRTOPTNS, but passes it to JES subsystems during data set selection.

## Syntax

```
PRTOPTNS=<options name>
```

### Subparameter definition

**<options data set entry name>**
Identifies the print options data. The name can be 1 to 16 characters long. If the name includes any special characters (for example, a dash), enclose the entire parameter in single quotation marks. You can also specify this keyword by using a dynamic output descriptor.

### Relationship to other system functions

This keyword can be used by Infoprint Server. For more information about this keyword when you use Infoprint Server, see *z/OS Infoprint Server User's Guide*.

### Example of the PRTOPTNS parameter

```
//OUTOPTNS OUTPUT PRTOPTNS='Boulder4019Optns'
```

In this example *Boulder4019Optns* is the name of the entity used to reference additional print options.

## PRTQUEUE parameter

***Parameter type:*** Keyword, optional

***Purpose:*** PRTQUEUE defines the name of the target print queue on a remote host system. JES does not use PRTQUEUE, but passes it to JES subsystems during data set selection.

### Syntax

```
PRTQUEUE=<print queue name>
```

### Subparameter definition

**<print queue name>**
Identifies the target print queue name. The name can be 1 to 127 characters long and may include any printable character. If the name includes any special character (for example, a dash or lower case letter), enclose the entire parameter in single quotation marks. You can also specify this keyword by using a dynamic output descriptor.

### Relationship to other system functions

This keyword can be used by Infoprint Server. For more information about this keyword when you use Infoprint Server, see *z/OS Infoprint Server User's Guide*.

### Example of the PRTQUEUE parameter

```
//OUTQUEUE OUTPUT PRTQUEUE='4019'
```

In this example *4019* is the name of the target print queue destination.

## PRTY parameter

***Parameter type:*** Keyword, optional

*Purpose:* Use the PRTY parameter to specify the priority at which the sysout data set enters the output queue. A data set with a higher priority is printed sooner.

## Syntax

```
PRTY=nnn
```

## Subparameter definition

**nnn**
Specifies the initial priority. nnn is a decimal number from 0 through 255; 0 is the lowest priority while 255 is the highest.

## Defaults

If you do not code the PRTY parameter, JES3 uses an installation default specified at initialization. JES2 uses a priority that is calculated for all output.

## Overrides

**In JES2 systems**, the installation can specify at JES2 initialization that JES2 is to ignore the OUTPUT JCL PRTY parameter.

**In JES3 systems**, the OUTPUT JCL PRTY parameter is ignored for JES3 networking.

## Example of the PRTY parameter

```
//PRESRPT OUTPUT PRTY=200,FORMS=TOPSEC
```

In this example, JES prints one copy of the president's report, PRESRPT, on forms named TOPSEC. Because a priority of 200 is specified, the report is probably printed immediately after entering the output queue.

# REPLYTO parameter

*Parameter type:* Keyword, optional

*Purpose:* Use the REPLYTO parameter to specify the e-mail address to which recipients of the e-mail can respond.

## Syntax

```
REPLYTO = {'reply address'}
          {reply-address  }
```

See "Using enclosing apostrophes in OUTPUT parameters" on page 477 for the REPLYTO parameter.

## Subparameter definition

**reply address**
Specifies the e-mail address to which recipients of the e-mail can respond. The reply address can be 1 - 60 EBCDIC text characters. See "Character sets" on page 21 for a description of EBCDIC text characters.

### Defaults

There is no default for REPLYTO.

### Overrides

There are no override considerations for REPLYTO.

### Relationship to other system functions

This keyword can be used by Infoprint Server. For more information about this keyword when you use Infoprint Server, see *z/OS Infoprint Server User's Guide*.

### Example of the REPLYTO parameter

```
//OUTDS2   OUTPUT  REPLYTO='jdoe@abc.net'
```

In this example, the system will identify `jdoe@abc.net` as the address to which the e-mail recipients can respond.

## RESFMT parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Use the RESFMT parameter to specify the resolution used to format the print data set. PSF will use this value to select the resource libraries for the resolution indicated. For more information see *PSF for z/OS: User's Guide*.

### Syntax

```
RESFMT = {P240 | P300}
```

### Subparameter definition

**P240**
Indicates that the data set was formatted with resources at 240 pels per inch.

**P300**
Indicates that the data set was formatted with resources at 300 pels per inch.

### Relationship to other control statements

If RESFMT is the highest priority specified resolution for a data set, then PSF will use the corresponding resource libraries as defined by the system programmer on the PRINTDEV statement. For information about the PRINTDEV statement, see *PSF for z/OS: Customization*.

### Example of the RESFMT parameter

```
//OUTRES  OUTPUT RESFMT=P240
```

In this example, the print data set was formatted for printing at 240 pels per inch.

## RETAINS *and* RETAINF parameters

***Parameter type:*** Keyword, optional

***Purpose:*** RETAINS specifies the amount of time to retain a successfully transmitted data set. RETAINF specifies the amount of time to retain a data set that failed to be transmitted. Each of these keywords consists of a numeric value indicating hours, minutes, and seconds.

These parameters apply only to data sets processed by a functional subsystem that can perform Internet Protocol (IP) transmission. JES does not use the RETAINS or RETAINF parameters, but passes them to the functional subsystem during data set selection.

Use RETAINS= when the functional subsystem has successfully transmitted the data set. Use RETAINF= when the functional subsystem employing the IP routing has not successfully transmitted the data set, despite performing all the indicated retries through any RETRY parameters specified. You have the option of manipulating the data set through the facilities provided by the functional subsystem before that subsystem releases the data set to JES. See the documentation for the particular subsystem for additional information.

## Syntax

```
RETAINS='<hhhh>:<mm>:<ss>' -or- RETAINS=FOREVER
RETAINF='<hhhh>:<mm>:<ss>' -or- RETAINF=FOREVER
```

## Subparameter definition

**<hhhh>:<mm>:<ss>**

Specifies the successful (RETAINS=) and failed (RETAINF=) time intervals to retain the data set.

One to ten characters, where <hhhh>, <mm>, and <ss> are numeric. This format requires that for either keyword you enclose the entire parameter in single quotation marks due to the colon as a special character.

You may specify *FOREVER* to request the system to retain the data set indefinitely.

You can also specify these keywords by using a dynamic output descriptor.

Only functional subsystems may use these keywords. See the documentation for the particular subsystem for additional information.

## Relationship to other control statements

The *RETAIN* keywords interact with the *RETRY* keywords in determining how long the functional subsystem is to hold on to the data set after either a successful or failed transmission of the data set before releasing it back to JES.

## Relationship to other system functions

The RETAINS and RETAINF keywords can be used by Infoprint Server to perform Internet Protocol (IP) transmission.

## Examples of the RETAIN keywords

***Example 1: RETAINS and RETAINF***

```
//OUTRETRY OUTPUT RETAINS='0001:00:00',RETAINF='0002:00:00'
```

In this example the functional subsystem will not release the data set to JES until one hour after a successful transmission. If the data set was not successfully

transmitted, the subsystem will not release the data set to JES until two hours after the last unsuccessful transmission attempt.

*Example 2: RETAINF Only*

```
//OUTRETRY OUTPUT RETAINF='0003:00:00'
```

In this example the functional subsystem will retain the data set for three hours following a failed transmission before releasing it to JES.

# RETRYL *and* RETRYT parameters

***Parameter type:*** Keyword, optional

***Purpose:*** Each of these keywords specifies a numeric value, as follows:

- *RETRYL=limit* defines the maximum number of attempts to transmit a data set before the *RETAIN* keyword options take effect.
- *RETRYT='retry time'* defines how much time to wait between each attempt to transmit the data set. It is formatted into hours, minutes, and seconds.

RETRYL and RETRYT apply only to data sets processed by a functional subsystem that can perform Internet Protocol (IP) transmission. JES does not use the RETRYL or RETRYT parameters, but passes them to the functional subsystem during data set selection. See the documentation for the particular subsystem for additional information.

## Syntax

```
RETRYL=nnnnn
RETRYT='<hhhh>:<mm>:<ss>'
```

## Subparameter definition

**<nnnnn>**
An integer from 0 to 32,767 (decimal) that specifies the maximum number of retries to attempt before the *RETAIN* keyword options are to take effect.

**<hhhh>:<mm>:<ss>**
One to ten characters, where <hhhh>, <mm>, and <ss> are numeric. This format requires that you enclose this entire parameter in single quotation marks due to the colon as a special character.

You can also specify these keywords by using a dynamic output descriptor.

## Relationship to other control statements

The RETRYL and RETRYT keywords interact with the RETAINS and RETAINF keywords to determine the number and frequency of retry attempts to transmit the data set before the values of RETAIN for successful or failed attempts, respectively, take effect.

## Relationship to other system functions

The RETRYL and RETRYT keywords can be used by Infoprint Server to perform Internet Protocol (IP) transmission.

## Examples of the RETRY keywords

*Example 1: RETRYT and RETRYL*

```
//OUTRETRY OUTPUT RETRYT='0001:00:00',RETRYL=5
```

In this example a retry is attempted every hour, for a maximum of five attempts.

*Example 2: RETRYT Only*

```
//OUTRETRY OUTPUT RETRYT='0000:05:00'
```

In this example a retry is attempted every five minutes.

# ROOM parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Use the ROOM parameter to print a room identification on the separator pages of the output for a sysout data set. An installation can use the room identification to assist in sysout distribution.

## Syntax

```
ROOM= {'room identification'}
      {room-identification  }
```

See “Using enclosing apostrophes in OUTPUT parameters” on page 477 for the ROOM parameter.

## Subparameter definition

**`room identification`**
Specifies the room identification to be associated with the sysout. The room identification is 1 - 60 EBCDIC text characters. See “Character sets” on page 21 for a description of EBCDIC text characters.

## Defaults

- **In an APPC scheduling environment:** In both JES2 and JES3 systems, if you do not code the ROOM parameter on the OUTPUT JCL statement, the system uses the value defined in the transaction program (TP) user's RACF profile when:
  - The user submitting the TP profile has a RACF profile defined for him, and
  - The transaction program profile includes TAILOR_SYSOUT(YES).

  Otherwise, the system uses the value defined on the transaction initiator's job statement.
- **In a non-APPC scheduling environment:** In a JES2 system, if you do not code the ROOM parameter on the OUTPUT JCL statement, the system uses the 4-character room field defined in the JES2 accounting parameter on the JOB statement.

  In a JES3 system, there is no default for the ROOM parameter on the OUTPUT JCL statement.

## Overrides

- **In an APPC scheduling environment:** In both JES2 and JES3 systems the ROOM parameter on the OUTPUT JCL statement overrides the room defined in the RACF profile. The room in the RACF profile overrides the room defined in the transaction initiator's job statement.
- **In a non-APPC scheduling environment:** In both JES2 and JES3 systems, the ROOM parameter on the OUTPUT JCL statement overrides the 4-character room field defined in the JES2 accounting parameter on the JOB statement.

## Example of the ROOM parameter

```
//OUTDS8   OUTPUT  ROOM='CONFERENCE ROOM'
```

In this example, CONFERENCE ROOM is printed on the line reserved for ROOM on the separator pages of any sysout data set that references OUTDS8.

# SYSAREA parameter

***Parameter type***

Keyword, optional

***Purpose***

Use the SYSAREA (system area) parameter to indicate whether the system should reserve an area on each page of printed output for the security label. The security label represents a security level and categories as defined to RACF.

**Note:** When a system area is reserved for a security label, the system shifts the printed output on each page. You cannot print output data in the system area.

***Reference***

For additional information on the system area, refer to *PSF for z/OS: Customization* and *PSF for z/OS: Security Guide*.

## Syntax

```
SYSAREA=  {YES}
          {Y  }
          {NO }
          {N  }
```

## Subparameter definition

**YES**
Requests that a system area be reserved on each page of printed output for the security label. This parameter can also be coded as Y.

**NO**
Requests that a system area not be reserved on each page of printed output for the security label. This parameter can also be coded as N.

## Defaults

If you do not code the SYSAREA parameter, an installation default determines if a system area is reserved for a security label.

## Relationship to other parameters

Use the SYSAREA parameter with the DPAGELBL parameter on the OUTPUT JCL statement and the SECLABEL parameter on the JOB statement as instructed by your security administrator.

The SYSAREA parameter can be coded with any other OUTPUT JCL statement parameters.

## Example of the SYSAREA parameter

```
//JOBB   JOB   1,'JIM WOOSTER',SECLABEL=CONF
                .
                .
//PRESRPT OUTPUT DPAGELBL=YES,SYSAREA=YES,FORMS=CSEC
```

In this example, the security label CONF (specified on the SECLABEL parameter of the JOB statement) is printed on each page of printed output in the system area. The sysout data set is printed on forms named CSEC.

# THRESHLD parameter

***Parameter type:*** Keyword, optional, JES3 only

***Purpose:*** Use the THRESHLD parameter to specify the maximum size for the sysout data set. JES3 calculates the sysout data set size as the number of records multiplied by the number of copies requested. When this size exceeds the THRESHLD value, JES3 creates a new unit of work, on a data set boundary, and queues it for printing. Consequently, copies of the sysout data set may be printed simultaneously by different printers.

Use the THRESHLD parameter for jobs that generate many large data sets or many copies of one large data set.

**Note:** THRESHLD is supported only on JES3 systems.

## Syntax

```
THRESHLD=limit
```

## Subparameter definition

**limit**
Specifies the maximum number of records for a single sysout data set. limit is a decimal number from 1 through 99999999.

## Defaults

If you do not code the THRESHLD parameter, JES3 uses an installation default specified at initialization.

## Example of the THRESHLD parameter

```
//STEPA  EXEC   PGM=RPTWRT
//SYSDS3 OUTPUT DEFAULT=YES,THRESHLD=10000
//RPT1   DD     SYSOUT=A,COPIES=10
//RPT2   DD     SYSOUT=A,COPIES=2
//RPT3   DD     SYSOUT=A,COPIES=5
```

In this example, the report data sets, RPT1, RPT2, and RPT3, are processed in sysout class A. All three DD statements implicitly reference the step-level default OUTPUT JCL statement SYSDS3; therefore, the THRESHLD value specified in the OUTPUT JCL statement applies to the three reports combined. JES3 is to print the following:

| Copies | Data Set | Records in Data Set | Total Records |
|---|---|---|---|
| 10 | RPT1 | 1000 | 10000 |
| 2 | RPT2 | 2000 | 4000 |
| 5 | RPT3 | 500 | 2500 |
| | | | Total 16500 |

Because the total exceeds the THRESHLD limit, JES3 divides the sysout data sets into two units of work. RPT1 is printed as one unit, and the other two data sets are printed together as another unit. If the THRESHLD limit had been 20000, all three data sets would have been printed as one unit of work.

# TITLE parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Use the TITLE parameter to print a description of the output on the separator pages of the output of a sysout data set. An installation can use the description to assist in sysout distribution.

## Syntax

```
TITLE=  {'description of output'}
        {description-of-output  }
```

See “Using enclosing apostrophes in OUTPUT parameters” on page 477 for the TITLE parameter.

## Subparameter definition

**description of output**
Specifies a description of output to be associated with a sysout data set. The description of output is 1 - 60 EBCDIC text characters. See “Character sets” on page 21 for a description of EBCDIC text characters.

## Example of the TITLE parameter

```
//OUTDS5   OUTPUT  TITLE='ANNUAL REPORT'
```

In this example, ANNUAL REPORT is printed on the line reserved for title on the separator pages of any sysout data set referencing OUTDS5.

# TRC parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Use the TRC parameter to specify whether the logical record for each output line in the sysout data set contains table reference character (TRC) codes or not. The TRC code identifies which font-name in the CHARS parameter is to be used to print the record.

If present, a TRC code in the output line record is:

- The first byte, if a carriage control character is not used.
- The second byte, immediately following the carriage control character, if used.

**Note:** TRC is supported only for a data set processed by the Print Services Facility (PSF).

## Syntax

```
TRC= {YES}
     {Y  }
     {NO }
     {N  }
```

## Subparameter definition

**YES**
Indicates that the data set contains TRC codes. This subparameter can also be coded as Y.

**NO**
Indicates that the data set does not contain TRC codes. This subparameter can also be coded as N.

**Note:** The data set DCB must not indicate that the data set contains TRC codes. DCB=(OPTCD=J) overrides TRC=NO when the data set is printed by PSF.

## Defaults

If you do not code the TRC parameter, the default is to use TRC characters only if the data set DD statement specified DCB=(OPTCD=J).

## Relationship to other parameters

A table reference character for the entire data set can be specified in the OUTPUT JCL MODIFY parameter.

## Example of the TRC parameter

```
//WRTR   JOB    ACNO77,MAEBIRD,MSGCLASS=B
//DS23   OUTPUT DEFAULT=YES,FORMS=STD,CONTROL=PROGRAM,TRC=YES
//STEP1  EXEC   PGM=DLYRPT
//DAILY  DD     SYSOUT=A,CHARS=(GT12,GB12,GI12)
```

In this example, sysout DD statement DAILY implicitly references the default job-level OUTPUT JCL statement DS23. This OUTPUT JCL statement directs PSF to print the daily report on standard forms, using table reference characters. The sysout data set defined by DD statement DAILY contains carriage control

characters in the first character of each logical record and a TRC code in the second character. The TRC characters in the records are 0 to use the font GT12; 1 to use GB12; and 2 to use GI12.

## UCS parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Use the UCS parameter to identify:

- The universal character set (UCS) image JES is to use in printing the sysout data set.
- A print train (print chain or print band) JES is to use in printing the sysout data set on an impact printer.
- A font for the sysout data set printed on an AFP printer in a JES2 system.

The UCS image specifies the special character set to be used. JES loads the image into the printer's buffer. The UCS image is stored in SYS1.IMAGELIB. IBM provides the special character set codes in Table 29.

***References:*** For more information on the UCS parameter, see *z/OS DFSMSdfp Advanced Services*.

### Syntax

```
UCS=character-set-code
```

### Subparameter definition

**character-set-code**
Identifies a universal character set. The character-set-code is 1 through 4 alphanumeric or national ($, #, @) characters. See Table 29 for IBM standard special character set codes.

### Defaults

If you do not code the UCS parameter, the system checks the UCS image in the printer's buffer; if it is a default image, as indicated by its first byte, JES uses it. If it is not a default image, JES loads the UCS image that is the installation default specified at JES initialization.

On an impact printer, if the chain or train does not contain a valid character set, JES asks the operator to specify a character set and to mount the corresponding chain or train.

*Table 29. Special Character Sets for the 1403, 3203 Model 5, and 3211 Printers*

| 1403 | 3203 Model 5 | 3211 | Characteristics |
|---|---|---|---|
| AN | AN | A11 | Arrangement A, standard EBCDIC character set, 48 characters |
| HN | HN | H11 | Arrangement H, EBCDIC character set for FORTRAN and COBOL, 48 characters |
| | | G11 | ASCII character set |
| PCAN | PCAN | | Preferred alphanumeric character set, arrangement A |

*Table 29. Special Character Sets for the 1403, 3203 Model 5, and 3211 Printers (continued)*

| 1403 | 3203 Model 5 | 3211 | Characteristics |
|---|---|---|---|
| PCHN | PCHN | | Preferred alphanumeric character set, arrangement H |
| PN | PN | P11 | PL/I alphanumeric character set |
| QN | QN | | PL/I preferred alphanumeric character set for scientific applications |
| QNC | QNC | | PL/1 preferred alphanumeric character set for commercial applications |
| RN | RN | | Preferred character set for commercial applications of FORTRAN and COBOL |
| SN | SN | | Preferred character set for text printing |
| TN | TN | T11 | Character set for text printing, 120 characters |
| XN | | | High-speed alphanumeric character set for 1403, Model 2 |
| YN | | | High-speed preferred alphanumeric character set for 1403, Model N1 |
| **Note:** Where three values exist (for the 1403, 3211, and 3203 Model 5 printers), code any one of them. JES selects the set corresponding to the device on which the data set is printed.<br><br>Not all of these character sets may be available at your installation. Also, an installation can design character sets to meet special needs and assign a unique code to them. Follow installation procedures for using character sets. | | | |

## Overrides

For printing on a printer with the UCS feature, a UCS parameter on the sysout DD statement overrides the OUTPUT JCL UCS parameter. For printing on a 3800, a CHARS parameter on the sysout DD statement or the OUTPUT JCL statement overrides all UCS parameters.

For a data set scheduled to the Print Services Facility (PSF), PSF uses the following parameters, in override order, to select the font list:

1. Font list in the library member specified by an OUTPUT JCL PAGEDEF parameter.
2. DD CHARS parameter.
3. OUTPUT JCL CHARS parameter.
4. DD UCS parameter.
5. OUTPUT JCL UCS parameter.
6. JES installation default for the device.
7. Font list on the PAGEDEF parameter in the PSF-cataloged procedure.

See “PAGEDEF parameter” on page 539 for more information.

## Using special characters sets

To use a special character set, SYS1.IMAGELIB must contain an image of the character set, and the chain or train for the character set must be available. IBM provides standard special character sets, and the installation may provide user-designed special character sets.

### Example of the UCS parameter

```
//PRTDS9 OUTPUT UCS=A11
```

In this example, JES uses standard EBCDIC character set arrangement A, with 48 characters, to print the sysout data set on a 3211 printer.

## USERDATA parameter

***Parameter type:*** Keyword, optional

***Purpose:*** The purpose and use of this keyword is defined by the installation. Refer to your installation's definition on the intent and use of this keyword.

If your installation does not define any use for this keyword, the information will be checked for syntax, stored as part of the output descriptor's information, and will then be ignored.

***Networking considerations:*** The use of the USERDATA keyword on one network node can be different from the use on another network node. An installation will have to coordinate any sending and receiving nodes to make use of the USERDATA keyword.

***References:*** Refer to the *z/OS MVS JCL User's Guide*, section "SYSOUT Resources—USERDATA OUTPUT JCL Keyword" for more details on how this keyword may be used.

### Syntax

```
USERDATA=value
         (value[,value]...)
```

- Your installation defines the intent and use of this keyword.
- You can omit the parentheses if you code only one value.
- Null positions in the USERDATA parameter are not allowed. For example, you cannot code USERDATA=(,value) or USERDATA=(value,,value).
- Each value may optionally be enclosed in apostrophes. See "Using enclosing apostrophes in OUTPUT parameters" on page 477 for the USERDATA parameter.

### Subparameter definition

**value**
Specifies the installation defined values for the installation's prescribed processing. You can code up to 16 installation-defined values. Each value may be from 1 to 60 EBCDIC text characters. See "Character sets" on page 21 for a description of EBCDIC text characters.

### Defaults

Determined by the installation.

### Overrides

Determined by the installation.

## Relationship to other keywords on this statement

Determined by the installation.

## Relationship to other control statements

Determined by the installation.

## Relationship to other system functions

Determined by the installation.

## Examples of the USERDATA parameter

The installation defines the intended content for each of the USERDATA values. The following examples are intended to provide samples of the allowable syntax for the USERDATA keyword. The resulting value (or portions of the value) are enclosed in parentheses to help distinguish them.

***Example 1***

```
//OUTDS1 OUTPUT USERDATA=USERVALUE
```

In this example, the USERDATA keyword contains a single parameter value (USERVALUE). Note that the value does not require enclosing apostrophes, because it contains only characters that are valid without them.

***Example 2***

```
//OUTDS2 OUTPUT USERDATA='Installation data'
```

In this example, the USERDATA keyword contains a single parameter value within apostrophes (Installation data).

***Example 3***

```
//OUTDS3 OUTPUT USERDATA='LOCALKEY=Installation data'
```

In this example, the USERDATA keyword contains a single parameter value within the apostrophes (LOCALKEY=Installation data). The single parameter value contains a string within the apostrophes that could be used to identify an installation-defined keyword (LOCALKEY) and its parameter value (Installation data).

***Example 4***

```
//OUTDS4 OUTPUT USERDATA='USERKEY1=User''s value'
```

In this example, the USERDATA keyword contains a single parameter value containing a string within the apostrophes that could be used to identify an installation defined keyword (USERKEY1) and its parameter value (User's value).

***Example 5***

```
//OUTDSA OUTPUT USERDATA=('non-keyword data',
//         'SOMEKEY=Some data',
//         'PARM3=Parm3''s value',
//         LASTVALUE)
```

In this example, the USERDATA keyword contains four parameter values.

- The first parameter value contains a string within the apostrophes (non-keyword data). An installation can consider this type of parameter a positional parameter.

It is recommended that positional parameters be clearly indicated by the installation to allow for easier specification, recognition, and processing.

- The second parameter value contains a string within the apostrophes that could be used to identify an installation defined keyword (SOMEKEY) and its parameter value (Some data).
- The third parameter value contains a string within the apostrophes that could be used to identify an installation defined keyword (PARM3) and its parameter value (Parm3's value).
- The fourth parameter value contains a string without any enclosing apostrophes (LASTVALUE).

*Example 6*

```
//OUTDSB OUTPUT USERDATA=('Installation_Keyword=Installation
//            defined keyword value',
//       'PARM2=Parm2''s value (second option)')
```

In this example, the USERDATA keyword contains two parameter values.

- The first parameter value contains a string within the apostrophes that could be used to identify an installation defined keyword (Installation_Keyword) and its parameter value (Installation defined keyword value), assuming the 'd' was specified in column 71 on the first statement.
- The second parameter value contains a string within the apostrophes that could be used to identify an installation defined keyword (PARM2) and its parameter value (Parm2's value (second option)).

*Example 7*

```
//PROCC  PROC PARM1=POSITIONAL,SOMEDATA=SOMETHING
//STEPC  EXEC PGM=MYPGM
//OUTDSC OUTPUT USERDATA=(&PARM1,
//          SOMEKEY-&SOMEDATA)
```

In this example, the USERDATA keyword contains two parameter values. If the installation allows a format of keyword-value, where the hyphen (-) is interpreted as an equal sign (=), then the parameter values do not need to be enclosed within apostrophes. Symbolic substitution of the parameter values is more straightforward.

- The first parameter value contains a string that could be used to identify an installation defined parameter value that is defined as a symbolic parameter. The procedure default for the value is taken from the PROC statement (POSITIONAL).
- The second parameter value contains a string that could be used to identify an installation defined keyword (SOMEKEY), the hyphen is considered an equal sign (by the installation), and the parameter value that is defined as a symbolic parameter. The procedure default for the value is taken from the PROC statement (SOMETHING).

*Example 8*

```
//PROCD  PROC PARM1=POSITIONAL,SOMEDATA=SOMETHING
//STEPD  EXEC PGM=MYPGM
//OUTDSD OUTPUT USERDATA=('&PARM1',
//          'SOMEKEY-&SOMEDATA')
```

In this example, the USERDATA keyword contains two parameter values. If the installation allows a format where an installation-defined keyword=value format

requires the entire parameter value to be enclosed within apostrophes, symbolic substitution of the parameter values is less straightforward than in the previous example.

- The first parameter value contains a string within the apostrophes that could be used to identify an installation defined parameter value that is defined as a symbolic parameter. Since the parameter is enclosed within apostrophes, the &PARM1 symbolic is not resolved so the parameter value is left unchanged (&PARM1).
- The second parameter value contains a string that could be used to identify an installation defined keyword (SOMEKEY), and the parameter value that is defined as a symbolic parameter. However, since the entire parameter is enclosed within apostrophes, the &SOMEDATA symbolic is not resolved so the entire parameter is left unchanged (SOMEKEY=&SOMEDATA).

## USERLIB parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Use the USERLIB parameter to identify libraries containing AFP resources to be used by Print Services Facility (PSF) when processing sysout data sets. The system searches libraries specified on the USERLIB parameter before using any system-defined resources. The resources specify how PSF is to print the sysout data set. They include:

- Fonts
- Page segments
- Overlays
- Page definitions

  To have PSF search the libraries specified on the USERLIB parameter for page definitions, you must code the membername on the PAGEDEF parameter of the OUTPUT JCL statement. If you do not code the PAGEDEF parameter, the system searches the system libraries.
- Form definitions

  To have PSF search the libraries specified on the USERLIB parameter for form definitions, you must code the membername on the FORMDEF parameter of the OUTPUT JCL statement. If you do not code the FORMDEF parameter, the system searches the system libraries.

### Syntax

```
USERLIB={data-set-name                                  }
        {(data-set-name1,data-set-name2,...data-set-name8)}
```

- You can omit the parentheses if you code only one data set name.
- If you code more than one data set name, each data set name *may* be enclosed in apostrophes. However, apostrophes around each data set name are not required.

### Subparameter definitions

**data-set-name**

Specifies from 1 to 8 library data set names containing AFP resources. The data set name must be a cataloged MVS data set. The library can contain any AFP resources. The libraries are searched in the order in which they are specified on the USERLIB statement.

## Defaults

If you do not code USERLIB, only the system and installation print resources are available. These resources include those available in the system libraries, and those specified inline in the print data set.

## Overrides

PSF obtains the system and installation resources in the following order:

1. Inline print data set resources
2. Libraries specified on the USERLIB statement
3. System libraries

## Requirements for USERLIB libraries

Data sets specified by USERLIB are concatenated to the system resource libraries, and are checked before the system libraries for requested resources. Unique member names should be defined for concatenated libraries. If the member names are not unique, the system uses the first member that it encounters.

## Examples of the USERLIB parameter

***Example 1***

```
//OUT1   OUTPUT  PAGEDEF=STNDRD,FORMDEF=CENTER,
//                USERLIB=(USER.PRIVATE.RESOURCE,GROUP.PRIVATE.RESOURCE)
```

In this example, PSF is to print the sysout data set using PAGEDEF=STNDRD and FORMDEF=CENTER.

When processing the sysout data set, PSF will search the user libraries before searching the system libraries for the specified PAGEDEF and FORMDEF. When searching the user libraries, PSF will search USER.PRIVATE.RESOURCE before searching GROUP.PRIVATE.RESOURCE.

***Example 2***

```
//OUT1   OUTPUT  PAGEDEF=STNDRD,FORMDEF=CENTER,
//         USERLIB=('USER.PRIVATE.RESOURCE','GROUP.PRIVATE.RESOURCE')
```

You may code apostrophes around the data set names, but apostrophes are not required. The system will process this example the same way it processes Example 1.

# USERPATH parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Names up to eight z/OS UNIX file system paths containing resources to be used by Print Services Facility (PSF) when processing sysout data sets. The system will search for resources in the paths specified on the USERPATH parameter before it searches paths specified at the system level. The paths specified on the USERPATH parameter can contain the following resources:

- TrueType fonts
- OpenType fonts

## Syntax

```
USERPATH={path                                          }
         {(path1,path2,...path8)}
```

- Path is the path name only. It cannot include the file name.
- You can omit the parentheses if you code only one path.
- A USERPATH parameter can specify from one to eight path subparameters.
- `USERPATH=(,path)` is invalid.
- If the path contains any special characters, blanks, or is continued to the next line, it must be enclosed in apostraphes.
- The first character in a path is a slash.
- A path can be specified as a maximum of 255 characters including any blank characters.
- See the PATH parameter on the DD statement for additional syntax rules.

## Subparameter definitions

**path**
Specifies the name of a z/OS UNIX System Services path which contains resources to be used for processing sysout data sets. Up to eight paths can be specified on the USERPATH parameter. PSF will search these paths for resources in the order the paths are specified.

## Defaults

If you do not code USERPATH, only the system paths and the sysout data set itself are searched for available resources.

## Overrides

Resources identified while processing a sysout data set are searched for in the following order:
- The sysout data set as an inline resource.
- The paths specified on the USERPATH parameter.
- The system path resource repositories.

## Relationship to other system functions

The USERPATH parameter is the companion parameter to the USERLIB parameter. Paths specified by the USERPATH parameter are logically concatenated to the system paths (see the FONTPATH parameter in the Print Services Facility for OS/390 & z/OS Customization). When looking for a resource, the USERPATH repositories are searched before the system path repositories. If two resources with the same file name exist in these paths, the system will use the first file encountered.

## Examples of the USERPATH parameter

***Example 1***

```
//OUT1   OUTPUT PAGEDEF=STNDRD,FORMDEF=CENTER,
//       USERLIB=(USER,PRIVATE,RESOURCE),
//       USERPATH=(/usr/fonts/ttfonts,/usr/fonts/otfonts)
```

In the prior example, two USERPATH paths have been specified. For any resource which can legally reside in a z/OS UNIX System Services path, PSF will first search path /usr/fonts/ttfonts and then path /usr/fonts/otfonts before searching for the resource in the system level path repositories.

# WRITER parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Use the WRITER parameter to name an external writer to process the sysout data set rather than JES. An external writer is an IBM- or installation-written program.

***References:*** For information about external writers, see *z/OS JES2 Initialization and Tuning Guide* or *z/OS JES3 Initialization and Tuning Guide*.

## Syntax

```
WRITER=name
```

## Subparameter definition

**name**
Identifies the member name (1 to 8 alphanumeric characters) of an installation-written program in the system library that the external writer loads to write the output data set.

Do not code INTRDR or STDWTR (and for JES3, NJERDR) as the writer name. These names are reserved for JES.

## Defaults

If you do not code the WRITER parameter, the installation's job entry subsystem processes the sysout data set.

## Overrides

The writer-name subparameter of the SYSOUT parameter on the sysout DD statement overrides the OUTPUT JCL WRITER parameter.

## Relationship to other parameters

For JES3, you can code the OUTPUT JCL DEST=nodename parameter with the WRITER=name parameter; however, do not code DEST=nodename.userid with WRITER=name.

## Starting an external writer

When a statement supplying processing options for a sysout data set specifies an external writer, the writer must be started before it can print or punch the data set. The writer is started by a system command from the operator or in the input stream. If the writer is not started before the job produces the sysout data set, the data set is retained until the writer is started.

## Examples of the WRITER parameter

***Example 1***

```
//MYOUT  JOB    ACCT928,MAEBIRD,MSGCLASS=B
// START XWTR
//MYDS   OUTPUT WRITER=MYPGM
//STEP1  EXEC   PGM=REPORT
//RPT1   DD     SYSOUT=A,OUTPUT=*.MYDS
```

The second statement is a JCL command statement to start the IBM-supplied external writer. This writer is a cataloged procedure in SYS1.PROCLIB. The sysout DD statement RPT1 explicitly references OUTPUT JCL statement MYDS, which specifies that the program MYPGM is to be loaded by XWTR and process the sysout data set.

***Example 2 (for a JES3 system)***

```
//**START XWTR
//MYOUT  JOB    ACCT928,MAEBIRD,MSGCLASS=B
//MYDS   OUTPUT WRITER=MYPGM
//STEP1  EXEC   PGM=REPORT
//RPT1   DD     SYSOUT=A,OUTPUT=*.MYDS
```

# Chapter 24. PEND statement

***Purpose:*** Use the PEND statement to mark the end of an in-stream procedure. You may end a cataloged procedure with a PEND statement, but it is not required.

## Description

### Syntax

```
//[name]  PEND  [comments]
```

The PEND statement consists of the characters // in columns 1 and 2 and three fields: name, operation (PEND), and comments. Do not continue a PEND statement.

### Name field

A name is optional on the PEND statement. If used, code it as follows:

- The name must begin in column 3.
- The name is 1 through 8 alphanumeric or national ($, #, @) characters.
- The first character must be alphabetic or national ($, #, @).
- The name must be followed by at least one blank.
- The name may be preceded by up to 8 alphanumeric or national characters, and then separated by a period. Coding the name in this way should not be confused with specifying an override, as can be done when coding DD statements.

If a name is not coded, column 3 must be blank.

### Operation field

The operation field consists of the characters PEND and must be preceded and followed by at least one blank. It can begin in any column.

### Comments field

The comments field follows PEND after at least one intervening blank.

### Location in the JCL

A PEND statement follows the statements of an in-stream procedure, and may follow the statements of a cataloged procedure.

### Examples of the PEND statement

***Example 1***

```
//PROCEND1 PEND THIS STATEMENT IS REQUIRED FOR IN-STREAM PROCEDURES
```

This PEND statement contains a comment.

***Example 2***

```
// PEND
```

© Copyright IBM Corp. 1988, 2015

This PEND statement contains only // and the operation field with the necessary blanks.

# Chapter 25. PROC statement

***Purpose:*** The PROC statement marks the beginning of a procedure. The PROC statement can assign default values to symbolic parameters, if coded, in the procedure.

## Description

### Syntax

```
For a cataloged procedure:
         [//[name]  PROC [parameter  [comments]]]
         //[name]  PROC

For an in-stream procedure:
         //name  PROC [parameter  [comments]]
         //name  PROC
```

A PROC statement consists of the characters // in columns 1 and 2 and four fields: name, operation (PROC), parameter, and comments.
**Note:** A PROC statement is optional in a cataloged procedure.

**Multiple parameters**: When more than one parameter is coded, separate parameters by commas. For example, //P1 PROC PARM1=OLD,PARM2=222001.

**Special characters**: When a parameter value contains special characters, enclose the value in apostrophes. The enclosing apostrophes are not considered part of the value. For example, //P2 PROC PARM3='3400-6'.

Code each apostrophe that is part of a value as two consecutive apostrophes. For example, //P3 PROC PARM4='O''DAY'.

However, if the symbolic parameter is enclosed within a matched pair of parentheses, you do not need to enclose the parentheses in apostrophes.

**Continuation onto another statement**: End each statement with a comma after a complete parameter. For example:

```
//P4 PROC  PARM5=OLD,PARM6='SYS1.LINKLIB(P40)',
//      PARM7=SYSDA,PARM8=(CYL,(10,1))
```

### Name field

A name is required on a PROC statement in an in-stream procedure and is optional on a PROC statement in a cataloged procedure. Code it as follows:

- When coded for an in-stream procedure, each name must be unique within the job. When coded for a cataloged procedure, the name need not be unique.
- The name must begin in column 3.
- The name is 1 through 8 alphanumeric or national ($, #, @) characters.
- The first character must be alphabetic or national ($, #, @).
- The name must be followed by at least one blank.

If a name is not coded, column 3 must be blank.

© Copyright IBM Corp. 1988, 2015

## Operation field

The operation field consists of the characters PROC and must be preceded and followed by at least one blank. It can begin in any column.

## Parameter field

The parameters on a PROC statement assign default values to symbolic parameters on procedure statements. An in-stream PROC statement requires parameters only if the procedure contains symbolic parameters. See “Using system symbols and JCL symbols” on page 38 for details on symbolic parameters and on how to assign values to them.

If coded, the parameter field must be preceded and followed by at least one blank.

## Comments field

The comments field follows the parameter field after at least one intervening blank. Do not code comments unless you code the parameter field.

## Overrides

To override a default parameter value on a PROC statement, code the same parameter on the EXEC statement that calls the procedure.

## Location in the JCL

A PROC statement must be the first statement in a procedure. An in-stream procedure must appear in the same job before the EXEC statement that calls it. A cataloged procedure appears in a procedure library, usually SYS1.PROCLIB.

## Examples of the PROC statement

***Example 1***

```
//DEF     PROC  STATUS=OLD,LIBRARY=SYSLIB,NUMBER=777777
//NOTIFY  EXEC  PGM=ACCUM
//DD1     DD    DSNAME=MGMT,DISP=(&STATUS,KEEP),UNIT=3390,
//            VOLUME=SER=888888
//DD2     DD    DSNAME=&LIBRARY,DISP=(OLD,KEEP),UNIT=3390,
//            VOLUME=SER=&NUMBER
```

Three symbolic parameters are defined in this cataloged procedure: &STATUS, &LIBRARY, and &NUMBER. Values are assigned to the symbolic parameters on the PROC statement. These values are used when the procedure is called and values are not assigned to the symbolic parameters on the calling EXEC statement.

***Example 2***

```
//CARDS  PROC
```

This PROC statement can be used to mark the beginning of an in-stream procedure named CARDS.

# Chapter 26. SCHEDULE statement

***Purpose:*** The SCHEDULE statement is used to specify scheduling attributes for a job such as the job group it is associated with and whether the job should be held until a specified time before execution. The SCHEDULE statement is ignored for the started tasks (STC) and TSO sessions (TSU).

**Note:** In z/OS Version 2 Release 2, the SCHEDULE statement is only supported by JES2.

The SCHEDULE statement must immediately follow the JOB statement. In other words, it must be the second statement in the job.

The parameters that you can specify for job processing are arranged alphabetically in the following sections.

For information about the JES initialization parameters that provide installation defaults, see *z/OS JES2 Initialization and Tuning Reference* .

## Description

### Syntax

The syntax of the SCHEDULE statement is:

```
//          SCHEDULE  positional-parameters[,keyword-parameter]...  [comments]
```

The SCHEDULE statement consists of the characters // in columns 1 and 2 and four fields: name, operation (SCHEDULE), parameter, and comments. Do not code comments if the parameter field is blank.

### Name field

A name is optional on the SCHEDULE JCL statement. If used, code it as follows:

- Each name must be unique within the job.
- The name must begin in column 3.
- The name is 1 through 8 alphanumeric or national ($, #, @) characters.
- The first character must be alphabetic or national ($, #, @).
- The name must be followed by at least one blank.

### Operation field

The operation field consists of the characters SCHEDULE and must be preceded and followed by at least one blank. It can begin in any column.

### Parameter field

The SCHEDULE statement has keyword parameters, but no positional parameters.

*Positional parameters*: None.

© Copyright IBM Corp. 1988, 2015

*Keyword parameters*: A SCHEDULE statement can contain the following keyword parameters:

## Comments field

The comments field follows the parameter field after at least one intervening blank space. If you do not code any parameters on a SCHEDULE statement, do not code any comments.

## Location in the JCL

If SCHEDULE statement is used, it must be the first statement after the JOB statement.

## Examples of SCHEDULE statement

```
// SCHEDULE JOBGROUP=MYGROUP


// SCHEDULE HOLDUNTL='+02:15'


// STARTBY HOLDUNTL=('12:50','12/13/2015'),WITH=OTHERJOB
```

# HOLDUNTL Parameter

***Parameter Type***

Keyword, optional

***Purpose***

Use the HOLDUNTL parameter to specify the date and time until which the job should be held.

## Syntax

```
HOLDUNTL = '+HH:MM'

HOLDUNTL = ( [ 'HH:MM' ] , [ 'MM/DD/YYYY' ] )

HOLDUNTL =  ( [ 'HH:MM' ] , [ 'YYYY/DDD' ]  )
```

## Subparameter definition

**+HH:MM**
Specifies that the job should be held for the specified number of hours and minutes starting from the time the job was submitted.

**HH:MM**
Specifies that the job should be held until the specified time in hours and minutes. The time is a 24-hour local time on the system where the job was submitted.

If time is omitted, but date is specified, the assumed time is midnight ('00:00'). When specified time is in the past, the job is not held.

**MM/DD/YYYY or YYYY/DDD**
Specifies that the job should be held until the specified day. The date can be specified as a calendar date (month/day/year) or as a Julian date (year/day of the year).

If the date is omitted but time is specified, then the following is assumed for the date:

- If the specified time has already passed when the job was submitted, then the next day is assumed.
- If specifies time has not yet passed when the job was submitted, then the current day is assumed.

### Relationship to other parameters

HOLDUNTL and STARTBY parameters can be specified together but they should both use the same time syntax - the **+HH:MM** syntax or time/date syntax. Also, it is an error when STARTBY time precedes the HOLDUNTL time.

## JOBGROUP Parameter

***Parameter Type***

Keyword, optional

***Purpose***

Use the JOBGROUP parameter to specify the name of a job group that this job is associated with. This can be specified as the job group name or as a job group name qualified with the job group identifier (for example, G1234.BUILD).

Use a job group name that is qualified with a job group identifier if you want to associate a job with a specific instance of this job group. This could be used when resubmitting a job to ensure that it is associated with the correct instance of a job group. However, it can be used any time that a job is submitted.

***References:*** For more information about job groups, see "JOBGROUP statement" on page 594.

### Syntax

```
JOBGROUP=job-group-name
```

### Subparameter definition

**job-group-name**
Identifies the name of a job group that this job is to be associated with.

### Defaults

If you do not specify the JOBGROUP parameter, the job is not associated with any job group.

### Relationship to other parameters

Do not use the JOBGROUP parameter with the STARTBY parameter.

### Example of the JOBGROUP parameter

```
//JOBA JOB  CLASS=A
//   SCHEDULE  JOBGROUP=GROUP1
```

In this example, JOBA is associated with the job group GROUP1.

# STARTBY Parameter

*Parameter Type*

Keyword, optional

*Purpose*

Use the STARTBY parameter to specify the preferred date and time when the job should enter execution. JES attempts to position this job in the job queue in such a way that the job becomes ready to be selected for the execution at the specified time.

However, JES does not guarantee that the job starts executing at the specified time. The ability of the job to start executing depends on the system environment, system affinity, availability of initiators, availability of resources, and so on.

## Syntax

```
STARTBY = '+HH:MM'

STARTBY = ( [ 'HH:MM' ] , [ 'MM/DD/YYYY' ] )

STARTBY =  ( [ 'HH:MM' ] , [ 'YYYY/DDD' ]  )
```

## Subparameter definition

**+HH:MM**
Specifies that the preferred time for the job to start executing is the specified number of hours and minutes from the time the job was submitted.

**HH:MM**
Specifies the preferred time for the job to start executing. The time is a 24-hour local time on the system where the job was submitted.

If the time is omitted, but the date is specified, the assumed time is midnight ('00:00').

**MM/DD/YYYY or YYYY/DDD**
Specifies the preferred date for the job to start executing. The date can be specified as a calendar date (month/day/year) or as a Julian date (year/day of the year).

If the date is omitted but time is specified, then the following is assumed for the date:

- If specifies time has already passed when the job was submitted, then the next day is assumed.
- If specifies time has not yet passed when the job was submitted, then the current day is assumed.

When specified combination of time and date is in the past, this is considered a JCL error.

## Relationship to other parameters

HOLDUNTL and STARTBY parameters can be specified together but they should both use the same time syntax - the **+HH:MM** syntax or time/date syntax. Also, it is an error when STARTBY time precedes the HOLDUNTL time.

STARTBY parameter is mutually exclusive with the JOBGROUP parameter.

# WITH Parameter

*Parameter Type*

Keyword, optional

*Purpose*

Use the WITH parameter to specify that the job must be executed on the same system where another reference job is active.

If the WITH parameter is used, the job is not eligible for execution until the reference job is active. In addition, the job can only be executed on the same system where the reference job is active.

Job having a WITH specification can be submitted before or after the reference job becomes active or submitted. However, it is recommended to submit a job after the reference job becomes active, because the reverse sequence causes additional processor overhead.

## Syntax

```
WITH = jobname
```

## Subparameter definition

**jobname**
Specifies the name of the reference job which must be active before this job can be selected for execution.

# Chapter 27. SET statement

***Purpose:*** Use the SET statement to:

- Define and assign initial values to symbolic parameters that are to be used when processing JCL statements.
- Change or nullify the values of defined symbolic parameters (those that are defined on previous SET statements) by assigning new values or nullifying current values.

The values that you assign to symbolic parameters on a SET statement are used in

- Subsequent JCL statements in the JCL stream, and
- Statements in subsequent procedures and nested procedures, when you:
  - Do not assign the values for the symbolic parameters on any PROC statements, or on any EXEC statements that call the procedures
  - Do not nullify the values for the symbolic parameters on any PROC statements, or on any EXEC statements that call the procedures.

Symbolic parameter values that are assigned or nullified by calling EXEC or PROC statements override the values you assign or nullify with the SET statement.

The rules for symbolic parameters apply to the symbolic parameters you specify on the SET statement. See the topics "Using system symbols and JCL symbols" on page 38 and "Using symbols in nested procedures" on page 51.

See also the EXEC and PROC statements, which also define and assign values to symbolic parameters.

## Description

### Syntax

```
//[name]  SET  symbolic-parameter=value
//      [,symbolic-parameter=value]... [comments]
```

© Copyright IBM Corp. 1988, 2015

The SET statement consists of the characters // in columns 1 and 2 and four fields: name, operation (SET), parameter(s), and comments.

**Multiple parameters**: When more than one parameter is coded, separate parameters by commas. For example:

```
//SP1  SET  PARM1=OLD,PARM2=222001
```

**Special characters and blanks**: When a parameter value contains special characters or blanks, enclose the value in apostrophes. The enclosing apostrophes are not considered part of the value. For example:

```
//SP2  SET  PARM3='3400-6'
```

Code each apostrophe that is part of a value as two consecutive apostrophes. For example:

```
//SP3  SET  PARM4='O''DAY'
```

However, if the symbolic parameter is enclosed within a matched pair of parentheses, you do not need to enclose the parentheses in apostrophes. For example:

```
//SET1  SET  DSP=(NEW,KEEP)
```

**Continuation onto another statement**: End each statement with a comma after a complete parameter and continue the parameter field on the next statement between columns 4 and 16. For example:

```
//SP4  SET  PARM5=OLD,PARM6='SYS1.LINKLIB(P40)',
//      PARM7=SYSDA,PARM8='(CYL,(10,1))'
```

## Name field

A name is optional on a SET statement. If used, code it as follows:

- The name should be unique within the job.
- The name must begin in column 3.
- The name is 1 through 8 alphanumeric or national ($, #, @) characters.
- The first character must be alphabetic or national ($, #, @).
- The name must be followed by at least one blank.
- The name can not be @GENSET@, as this is a reserved name.

If a name is not coded, column 3 must be blank.

## Operation field

The operation field consists of the characters SET and must be preceded and followed by at least one blank. It can begin in any column.

## Parameter field

The SET statement contains one or more parameters:

**symbolic-parameter=value[,symbolic parameter=value]...**
Defines a symbolic parameter and specifies the initial value to be assigned to the symbolic parameter appearing in subsequent JCL statements. Separate each assignment of a value to a symbolic parameter by commas.

To nullify a symbolic parameter, specify:

```
symbolic-parameter=
```

## Comments field

The comments field follows the parameter field after at least one intervening blank.

## Overrides

A value you assign to a symbolic parameter on a SET statement is overridden by:

- A subsequent SET statement
- Any default value assigned or nullified on a subsequent PROC statement for a procedure
- Any value assigned or nullified on a subsequent EXEC statement that calls a procedure.

When the target of an MVS START command contains a JOB statement, and the MVS START command specifies symbolic parameters, the system inserts a SET statement into the job to define those symbolic values. In contrast to the normal behavior of SET statements, values defined on this SET statement override:

- Other SET statements that occur before the first IF or EXEC statement in the job
- An EXEC statement that invokes an outer (non-nested) procedure
- A PROC statement in an outer procedure.

See "Defining and nullifying JCL symbols" on page 39 and "Using symbols in nested procedures" on page 51 for the complete set of rules for assigning values to symbolic parameters.

## Location in the JCL

A SET statement can appear anywhere in the job after the JOB statement with the following restrictions:

- It must appear in the job's JCL before the intended use of the symbolic parameter.
- It must follow a complete JCL statement.
- It cannot appear immediately after the first DD statement within a concatenation.

  ***Examples:*** The following JCL will work.

```
//DD1       DD  DSN=dsnA,DISP=SHR
//          DD  DSN=dsnB,DISP=SHR
//         SET
//          DD  DSN=dsnC,DISP=SHR
```

  The following JCL will fail.

```
//DD1       DD  DSN=dsnA,DISP=SHR
//         SET
//          DD  DSN=dsnB,DISP=SHR
//          DD  DSN=dsnC,DISP=SHR
```

## Relationship to other control statements

Symbolic parameters are also assigned values or nullified on PROC statements and EXEC statements that call procedures.

## Considerations for using the SET statement

- The symbolic parameters you define on the SET statement are assigned the specified values at the location in which the SET statement is encountered in processing the JCL.

  If you use SET to define a value for a symbolic parameter that does not appear in the JCL, the system does not issue message IEFC657I, and there is no JCL error.

- The SET statement is not executed conditionally. For example, if the SET statement appears in an IF/THEN/ELSE/ENDIF statement construct, the value is assigned to the symbolic parameter regardless of the logic of the construct.

## Examples of the SET statement

***Example 1:*** The symbolic parameter DSP is defined and initialized to the value NEW.

```
//SET1  SET  DSP=NEW
```

DSP is referenced by coding &DSP in the JCL, for example:

```
          .
          .
//DD1   DD  DSNAME=ALPHA.PGM1,DISP=(&DSP,KEEP)
```

In the example, &DSP is assigned the value NEW for execution:

```
//DD1   DD  DSNAME=ALPHA.PGM1,DISP=(NEW,KEEP)
```

***Example 2:*** The symbolic parameter DSP is defined and initialized to the value (NEW,DELETE,KEEP).

```
//SETA  SET  DSP=(NEW,DELETE,KEEP)
```

DSP is referenced by coding &DSP in the JCL, for example:

```
//PR2   PROC  DSP=(NEW,KEEP)
          .
          .
//DD6   DD  DSNAME=ALPHA.PGM2,DISP=&DSP
```

&DSP is assigned the value (NEW,KEEP) from PROC statement PR2 for execution:

```
//DD6   DD  DSNAME=ALPHA.PGM2,DISP=(NEW,KEEP)
```

In the example, the definition of DSP on SET statement SETA **does not** override the default definition of DSP on PROC statement PR2.

***Example 3:*** This example shows the SET statement spanning two records. The symbolic parameters are defined and initialized to the values shown on SET statement SETB. They are referenced by coding &AA, &BB, and &CC in the JCL, for example:

```
//SETB  SET   AA=BETA.PGM.RATE,BB=DCLAS03,
//         CC=(NEW,KEEP)
          .
          .
//PR3   PROC  ...
//S3    EXEC  PGM=...
//DD7   DD    DSNAME=&AA,DATACLAS=&BB,DISP=&CC
          .
//      PEND
          .
//S1    EXEC  PROC=PR3,BB=DCLAS0X
          .
```

In the example, the values assigned on DD statement DD7 for execution are:

```
//DD7   DD  DSNAME=BETA.PGM.RATE,DATACLAS=DCLAS0X,DISP=(NEW,KEEP)
```

The values defined for the symbolic parameters on SET statement SETB are assigned to the AA and CC symbolic parameters in procedure PR3 for execution. However, the value defined for symbolic parameter BB on EXEC statement S1 overrides the value defined on SET statement SETB.

***Example 4:*** The following example shows the use of the SET statement assigning values to symbolic parameters in an INCLUDE group.

```
//*  THIS INCLUDE GROUP IS CATALOGED AS...
//*  PUCHKOFF.SYSOUT.JCL(SYSOUT2)
//SYSOUT2  DD      SYSOUT=A
//OUT1     OUTPUT  DEST=POK,COPIES=3
//OUT2     OUTPUT  DEST=&AA,COPIES=&NC
//OUT3     OUTPUT  DEST=&BB,COPIES=10
//*  END OF INCLUDE GROUP...
//*  PUCHKOFF.SYSOUT.JCL(SYSOUT2)
```

The following program is executed.

```
//TESTJOB  JOB  ...
//LIBSRCH  JCLLIB  ORDER=PUCHKOFF.SYSOUT.JCL
//SET1     SET     AA=KINGSTON,BB=STL,NC=10
//STEP1    EXEC    PGM=OUTRTN
//OUTPUT1  INCLUDE MEMBER=SYSOUT2
//STEP2    EXEC    PGM=IEFBR14
```

The SET statement, which can be easily changed for different jobs, assigns values to the symbolic parameters in INCLUDE group SYSOUT2.

After the INCLUDE statement is processed, the JCL stream would be executed as:

```
//TESTJOB  JOB  ...
//LIBSRCH  JCLLIB  ORDER=PUCHKOFF.SYSOUT.JCL
//STEP1    EXEC    PGM=OUTRTN
//*  THIS INCLUDE GROUP IS CATALOGED AS...
//*  PUCHKOFF.SYSOUT.JCL(SYSOUT2)
//SYSOUT2  DD      SYSOUT=A
//OUT1     OUTPUT  DEST=POK,COPIES=3
//OUT2     OUTPUT  DEST=KINGSTON,COPIES=10
//OUT3     OUTPUT  DEST=STL,COPIES=10
//*  END OF INCLUDE GROUP...
//*  PUCHKOFF.SYSOUT.JCL(SYSOUT2)
//STEP2    EXEC    PGM=IEFBR14
```

The INCLUDE group has been imbedded in the JCL stream (replacing the INCLUDE statement) and values assigned to the symbolic parameters in the INCLUDE group.

# Chapter 28. XMIT JCL statement

*Support for the XMIT JCL statement:*

- The XMIT JCL statement has no function in an APPC scheduling environment. If you code XMIT, the system will check it for syntax and ignore it.
- The XMIT JCL statement is supported on both JES2 and JES3 systems. In JES2 systems, however, the SUBCHARS operand is not supported.

***Purpose:*** Use the XMIT JCL statement to transmit records from an MVS node to a JES3 node, a JES2 node, a VSE/POWER node, a VM/RSCS node, or an AS/400 node.

The sending system does not process or check the records for validity except when the JCL is processed by an internal reader (such as with TSO/E submit processing). In this case, the system recognizes /*EOF and /*DEL as internal reader control statements and errors can occur on the sending system if /*EOF or /*DEL are included in the XMIT JCL stream.

To transmit /*EOF and /*DEL statements as part of the XMIT JCL stream on a JES3 node, replace /* with two substitute characters and identify the substitute characters on the SUBCHARS parameter. Prior to transmission, the sending system converts the two substitute characters to /*. The receiving (execution) system can then process the /*EOF and /*DEL statements as internal reader control statements.

**Note:** Since JES2 does not support the SUBCHARS parameter, it also does not support sending /*EOF and /*DEL as part of the XMIT JCL stream.

Do not nest XMIT JCL statements. That is, do not include an XMIT JCL statement between an XMIT JCL statement and its delimiter.

The system builds network job header and trailer records from information on the JOB statement and any //*NETACCT statements, if included, preceding the XMIT JCL statement. Then the system transmits all the records between the XMIT JCL statement and a delimiter statement.

The records can consist of a job input stream, an in-stream DD * or DD DATA data set, or any job definition statements recognized by the destination node. If the records are a job input stream, and the destination node can process the JCL, the transmitted input stream is executed at the destination node. The records must be 80 characters long.

The records end when the system finds one of the following delimiters:

- /* in the input stream, if a DLM parameter is not coded on this XMIT JCL statement. (Refer to the Delimiter Statement information for an explanation of /*.)

  From TSO/E only, TSO/E inserts /* at the end-of-file if the default delimiter is not supplied.
- The two-character delimiter specified by a DLM parameter on this XMIT JCL statement.

© Copyright IBM Corp. 1988, 2015

# Description

## Syntax

```
//[name]  XMIT  parameter[,parameter]...  [comments]
```

The XMIT JCL statement consists of the characters // in columns 1 and 2 and four fields: name, operation (XMIT), parameter, and comments.

## Name field

A name is optional on the XMIT JCL statement. If used, code it as follows:

- Each name must be unique within the job.
- The name must begin in column 3.
- The name is 1 through 8 alphanumeric or national ($, #, @) characters.
- The first character must be alphabetic or national ($, #, @).
- The name must be followed by at least one blank.

## Operation field

The operation field consists of the characters XMIT and must be preceded and followed by at least one blank. It can begin in any column.

## Parameter field

The XMIT JCL statement contains only keyword parameters. A DEST parameter is required; the DLM and SUBCHARS parameters are optional. If your JCL is to be processed by an internal reader and /*EOF or /*DEL is part of the XMIT JCL stream, you must code the SUBCHARS parameter.

You can code the keyword parameters in any order in the parameter field.

| KEYWORD PARAMETERS | VALUES | PURPOSE |
|---|---|---|
| `DEST=nodename[.vmuserid]`<br><br>See section "DEST parameter" on page 586 | nodename: 1 - 8 alphanumeric or $, #, @ characters<br><br>vmuserid: 1 - 8 alphanumeric or $, #, @ characters | Identifies the destination for all following records until a delimiter stops transmission of the records. |
| `DLM=delimiter`<br><br>See section "DLM parameter" on page 587 | delimiter: 2 alphanumeric, $, #, @, or special characters | Specifies a delimiter to stop the transmission of records. |
| `SUBCHARS=substitute`<br><br>See section "SUBCHARS parameter" on page 588 | substitute: 2 alphanumeric, $, #, @, or special characters | Specifies a substitute for internal reader control statements. (JES3 only) |

## Comments field

The comments field follows the parameter field after at least one intervening blank.

## Location in the JCL

Place the XMIT JCL statement after a JOB statement and any //*NETACCT or /*NETACCT statements. (Other JES2 and JES3 JCL statements between the JOB and XMIT JCL statements are not supported and can cause unpredictable results.) The JOB statement must be valid for the submitting location.

Do not place any other MVS JCL statements between the JOB statement and the XMIT JCL statement. If any of these statements intervene, the system terminates the job.

## Error on XMIT JCL statement

For JES2, if the system finds an error on the XMIT JCL statement before a specified DLM parameter, the current job is flushed.

For JES3, if the system finds an error on the XMIT JCL statement before a specified DLM parameter, all jobs in the batch are flushed.

If the system finds an error on the XMIT JCL statement after a specified DLM parameter, the network job is flushed and local processing starts at the statement following the specified delimiter.

## Examples of the XMIT JCL statement

***Example 1***

```
//JOBA   JOB   25FA64,'KEN KAHN'
//X1     XMIT  DEST=KGNMVS45
          .
          .
        (records to be transmitted)
          .
/*
//JOBB   JOB   ...
          .
```

In this example, the records between the XMIT JCL statement and the delimiter statement (/* in columns 1 and 2) are transmitted to the node named KGNMVS45.

***Example 2***

```
//JOBC   JOB  PW19,'DEPT 53'
//X2     XMIT DEST=POKVMDD3.MVSGST34,DLM=AA
          .
          .
/*
        (records to be transmitted)
/*EOF
/*DEL
          .
          .
AA
//JOBB   JOB   ...
          .
```

In this example, processing is **not** through an internal reader on the sending system. The records between the XMIT JCL statement and the delimiter statement, which must contain AA in columns 1 and 2 as specified in the DLM parameter, are transmitted to the system, MVSGST34, running on the VM system at the node named POKVMDD3.

***Example 3 (JES3 only)***

```
//JOBE   JOB   NS37,'NYC BX'
//X3     XMIT  DEST=SANFRAN,DLM=AA,SUBCHARS='/+'
          .
          .
        (records to be transmitted)
          .
          .
```

```
/+EOF
         .
         .
/+DEL
AA
//JOBF   JOB   ...
         .
```

In this example, the JCL is processed through an internal reader on the sending system. The records between the XMIT JCL statement and the delimiter statement, which must contain AA in columns 1 and 2 as specified in the DLM parameter, are transmitted to the node named SANFRAN.

To transmit the /*EOF and /*DEL internal reader control statements, /* is replaced by /+ in columns 1 and 2 on both statements in the XMIT JCL stream and SUBCHARS='/+' is coded on the XMIT statement. The sending system does not recognize /+EOF and /+DEL as internal reader statements. Then prior to transmission, the sending system converts /+ to /* and sends /*EOF and /*DEL to the receiving node, which can then process the internal reader control statements.

# DEST parameter

***Parameter type:*** Keyword, required

***Purpose:*** Use the DEST parameter to specify a destination for the following input stream records. The DEST parameter can send the records to a node or, for a node that is a VM system, to a guest system running on the virtual machine.

## Syntax

```
DEST=nodename

DEST=nodename.vmuserid
```

## Subparameter definition

**nodename**
Identifies the destination node. The nodename identifies a JES2 system, a JES3 system, a VSE/POWER node, or a VM system. The nodename is 1 through 8 alphanumeric or national ($, #, @) characters specified during JES initialization. If the requested node is the same as the submitting node, the records following the XMIT JCL statement are processed by the local system.

**userid**
Identifies a destination guest system. The userid is 1 through 8 alphanumeric or national ($, #, @) characters.

## Examples of the DEST parameter

***Example 1***

```
//TRANS XMIT DEST=LAXSYS
```

This example sends the following records to a node named LAXSYS.

***Example 2***

```
//SEND  XMIT DEST=VMSYS3.GUEST7
```

This example sends the following records to a guest system, named GUEST7, running in the VM system at the node named VMSYS3.

## DLM parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Use the DLM parameter to specify a delimiter to stop transmission of input stream records. When the DLM parameter assigns a delimiter other than the standard delimiter (/* in columns 1 and 2), the records can include the standard delimiter.

If you use the DLM delimiter to define a delimiter, be sure to terminate the records with the specified DLM characters. Otherwise, all jobs between the XMIT JCL statement and the end-of-file will be transmitted, and processed at the node to which they are sent.

From TSO/E only, TSO/E inserts /* at the end-of-file if the default delimiter is not supplied.

### Syntax

```
DLM=delimiter
```

- If the specified delimiter contains any special characters, enclose the delimiter in apostrophes. In this case, a special character is any character that is neither alphanumeric nor national ($, #, @).
- If the delimiter contains an ampersand or an apostrophe, code each ampersand or apostrophe as two consecutive ampersands or apostrophes and enclose the delimiter in apostrophes. Each pair of consecutive ampersands or apostrophes counts as one character.

### Subparameter definition

**delimiter**
Specifies two characters that indicate the end of this data set in the input stream.

### Default

If you do not specify a DLM parameter, the default is the standard /* delimiter statement.

### Invalid delimiters

If the delimiter is not two characters, the system terminates the job and does not transmit any records.

### Examples of the DLM parameter

***Example 1***

```
//XX  XMIT  DEST=NYCNODE,DLM=AA
       .
       .
       (records to be transmitted)
       .
AA
```

The DLM parameter assigns the characters AA as the delimiter for the in-stream records to be transmitted.

***Example 2***

```
//XY  XMIT  DEST=ATL,DLM='A+'
//XZ  XMIT  DEST=BOST,DLM='&&7'
//XW  XMIT  DEST=CHI,DLM='B'''
```

These examples specify delimiters of A+, &7, and B'.

## SUBCHARS parameter

***Parameter type:*** Keyword, optional

***Purpose:*** Use the SUBCHARS parameter (supported by JES3 only) to specify a substitute (consisting of two characters) for the first two characters of /*EOF and /*DEL internal reader control statements. The substitute characters on the internal reader control statements must be in columns 1 and 2.

You can use the SUBCHARS parameter for any XMIT JCL job. However, SUBCHARS is required if you want to transmit internal reader control statements (/*EOF and /*DEL) and the job is processed by an internal reader on the sending system. Note that the system recognizes /*EOF and /*DEL as internal reader control statements and errors can occur on the sending system if /*EOF or /*DEL are included in the XMIT JCL stream.

To transmit internal reader control statements, replace /* on the /*EOF and /*DEL statements in the records to be transmitted with two substitute characters and identify the substitute characters on the SUBCHARS parameter. Prior to transmission, the system converts the substitute characters to /* and sends /*EOF and /*DEL to the receiving node for processing.

***Reference:*** The internal reader is described in *z/OS MVS Programming: Assembler Services Guide*.

### Syntax

```
SUBCHARS=substitute
```

- If the specified substitute contains any special characters, enclose the substitute in apostrophes. In this case, a special character is any character that is neither alphanumeric nor national ($, #, @).
- If the substitute contains an ampersand or an apostrophe, code each ampersand or apostrophe as two consecutive ampersands or apostrophes and enclose the substitute in apostrophes. Each pair of consecutive ampersands or apostrophes counts as one character.

### Subparameter definition

**substitute**
Specifies two characters that indicate the substitute characters for the first two characters of internal reader control statements. The substitute characters apply only to internal reader statements.

## Default

There is no default for SUBCHARS.

## Invalid substitute

If the substitute is not two characters, the system terminates the job and does not transmit any records.

## Examples of the SUBCHARS parameter

***Example 1***

```
//XX  XMIT  DEST=NYCNODE,SUBCHARS=MV
       .
       .
       (records to be transmitted)
MVEOF
       .
```

The SUBCHARS parameter identifies the characters MV as the substitute for the first two characters of the internal reader control statement to be transmitted. Prior to transmission, the system converts the MV substitute characters to /* and sends /*EOF to the receiving node for processing.

***Example 2***

```
//XY  XMIT  DEST=ATL,SUBCHARS='A+'
//XZ  XMIT  DEST=BOST,SUBCHARS='&&7'
//XW  XMIT  DEST=CHI,SUBCHARS='B'''
```

These examples specify substitutes of A+, &7, and B'.

# Chapter 29. JES2 Execution Control Statements

JES2 in z/OS 2.2 includes statements for defining a concept called a job group. A job group is a set of specifications (between a JOBGROUP and ENDGROUP statement) that define the execution sequencing of a group of jobs and the jobs themselves (submitted after the job group specification).

These statements include:

- JOBGROUP JCL statement ("JOBGROUP statement" on page 594)
- GJOB JCL statement ("GJOB statement" on page 598)
- JOBSET JCL statement ("JOBSET statement" on page 599)
- SJOB JCL statement ("SJOB statement" on page 600)
- ENDSET JCL statement ("ENDSET statement" on page 601)
- AFTER JCL statement ("AFTER statement" on page 605)
- BEFORE JCL statement ("BEFORE statement" on page 602)
- CONCURRENT JCL statement ("CONCURRENT statement" on page 607)
- ENDGROUP JCL statement ("ENDGROUP statement" on page 609)

Job groups also support the inclusion of comment cards (statements that start with //*) in the same manner as normal JCL does. However, job groups cannot include JECL (statements that start with /*), instream data, use of symbols, or any JCL statement not listed above.

In addition to the list of JCL statements above, a new SCHEDULE JCL statement was added in z/OS 2.2 to support associating a job with a job group. This statement is described below (including keywords not related to job group processing).

## Examples of a JOBGROUP and associated jobs

The following is an example of a simple four job JOBGROUP and the associated batch jobs.

***Example 1***

```
//MULTI    JOBGROUP            JOBGROUP describes a group of dependent jobs
//A      GJOB                  GJOB describes a job in a group
//B      GJOB
//       AFTER NAME=A,         AFTER is positioned after a GJOB and describes its
//        WHEN=(RC=0)          dependency to another named job
//C      GJOB
//       AFTER NAME=A,
//        WHEN=(RC=4)          WHEN= describes a condition associated with the dependency
//D      GJOB
//       AFTER NAME=B
//       AFTER NAME=C
//MULTI    ENDGROUP            ENDGROUP marks the end of the job group definition
//*
//A      JOB     TIME=NOLIMIT,REGION=0K,MSGCLASS=A,CLASS=B
//       SCHEDULE JOBGROUP=MULTI
//STEP1    EXEC PGM=IEFBR14
//*
//B      JOB     TIME=NOLIMIT,REGION=0K,MSGCLASS=A,CLASS=A
//       SCHEDULE JOBGROUP=MULTI
//STEP1    EXEC PGM=IEFBR14
//*
```

© Copyright IBM Corp. 1988, 2015

```
//C      JOB     TIME=NOLIMIT,REGION=0K,MSGCLASS=A,CLASS=A
//       SCHEDULE JOBGROUP=MULTI
//STEP1    EXEC PGM=IEFBR14
//*
//D      JOB     TIME=NOLIMIT,REGION=0K,MSGCLASS=A,CLASS=A
//       SCHEDULE JOBGROUP=MULTI
//STEP1    EXEC PGM=IEFBR14
```

## JOBGROUP logging job

Job groups can be submitted to JES2 from any input source, for example, internal reader, RJE device, over NJE, and more. As part of input processing for a job group, a new object is created within JES2 to represent the group as a whole. This object is call a job group logging job or just a logging job. The logging job owns resources that are managed at the group level such as a SYSOUT logging data set , a spool data set with the JOBGROUP JCL (from the JOBGROUP statement to the ENDGROUP statement) and any messages related to the JCL, and the checkpoint control blocks that represent the job group. This logging job does not execute nor convert, and in general, is not intended to print.

Note that at the moment the TSO SUBMIT command does not support JOBGROUPs very well. TSO submit assumes that objects being submitted must start with a JOB card. If there is no JOB card at the beginning, then it tries to add one. This causes a number of problems. A job group structure can be appended to the end of a job being submitted. This only works because SUBMIT does not see the end of the JOB and passes the JOBGROUP JCL into JES2. The result is that TSO submit thinks it only submitted one job object when in fact it submitted two.

Since the job group logging job owns resources that need to be protected, it will be validated using the same process as batch jobs. All authentication processing for normal batch jobs is performed for job group logging jobs. This includes password verification, user propagation, JESJOBS SUBMIT authorization checks, JESINPUT class checks, SECLABEL checks, and more. The result of the authentication process is a RACF security token that is used to protect the job and the JES spool data sets that it owns.

The logging job is assigned a job id that starts with the letter 'G' (as opposed to the letters 'J', 'S', and 'T' that existed before). This job id will be seen in JES2 commands, certain JES2 control blocks (for example, the JCT), and SMF 6 and 26 records.

Since the logging job does not execute or convert, the only SMF records that are associated with it are the 6 and 26 records. The SMF exits are not called for this job structure (IEFUJI, IEFUJV, IEFACTRT). The accounting information can be included on JOBGROUPs and will be included in SMF records created, but it is not passed to any of the run time processing exits (since the job does not run).

## Other attributes of a JOBGROUP logging job

As a job object in JES2, the logging job has other attributes that cannot be specified on the JOBGROUP JCL statement. These attributes are assigned the default values based on the input device (or source) of the job group. Attributes such as JOBCLASS, default SYSOUT destination, message class, and more are set to the default value.

**Note:** Some of these attributes have no effect on the job group (such as JOBCLASS) while others do (such as destination).

## JES2 exit processing for JOBGROUP JCL

Certain JES2 exits are called during input processing for a job group. Exits 2/52 are called for the JOBGROUP card, exits 3/53 are called for the accounting information on the JOBGROUP card, exits 4/54 are called for each JCL card after the JOBGROUP card (up to the ENDGROUP card), Exits 20/50 for the end of input, and exit 7/8 for CBIO. Data areas passed to these exits are the same as any other batch job. However, The JCL statements are the new statements detailed later in this section.

After input processing, the normal job phase exits are called for the job (for example, exit 51 for phase changes). JOBGROUP logging jobs move from the INPUT phase to the SETUP phase, then the OUTPUT and HARDCOPY phases, and finally to the purge phase.

## Associating jobs with a job group

Batch jobs are associated with job groups using the new SCHEDULE JCL statement. For more information see the Chapter 26, “SCHEDULE statement,” on page 571.

## Processing for jobs in a job group

Jobs that are part of a job group go through INPUT and CONVERSION processing like any other batch jobs (they have a new SCHEDULE JCL statement to identify the job group they are part of). After input, they are placed into the SETUP phase until they are eligible to run meaning all dependencies have been satisfied. At that point, they are either placed on the execution queue to run or canceled and queued to the OUTPUT phase. Jobs associated with a job group run normally, like any other batch job.

There is no propagation of attributes from a job group to the jobs within a job group. The affinity attributes of a job group (SYSAFF, SYSTEM, SCHENV) on the JOBGROUP statement do apply to all the jobs in the job group (they are ANDed into any affinity on the job). However, the accounting information and user identity associated with the job group do not apply to the jobs in the job group.

## Configuring and activating job groups

There is a new JES2 initialization statement and command, GRPDEF, that controls job group processing. Keywords on GRPDEF control the number of data areas available for group processing and the number of jobs that can run concurrently. Data for job groups is stored in a data area called a ZJC. The number of data areas needed to represent a group is dictated by the complexity of the group. One data area is needed for the group, one for each job in the group, and one for each dependency. The 4 job group described earlier would require 1 ZJC for the group, 4 for jobs in the group, and 4 for each of the dependencies, for a total of 9 data areas.

The number of ZJCs (ZJCNUM=) is defaulted to 1000, allowing limited usage of the function. The value can be configured from 6 (only useful for basic testing) to 500,000.

The other configuration keyword on GRPDEF is the number of concurrent jobs that can be configured in a single job group. This is the CONCURRENT_MAX=

keyword. By default, the limit is set to zero, disabling the function. To allow users to use concurrent execution, this needs to be configured to a higher value. Valid range is 0 to 200.

All the functions of job groups are only available when JES2 is in the z22 $ACTIVATE mode. The current $ACTIVATE mode can be displayed using the $D ACTIVATE command. The command also list any reasons why you cannot activate to z22, if you are in z11 mode. There is also a health check that reports if you are not in z22 mode and what is needed to activate to z22 mode. In general, all members must be running z/OS 2.2 or later. SPOOLDEF CYL_MANAGED=ALLOWED must be set, and the checkpoint data sets have to be large enough to hold the larger data areas.

Doing a $ACTIVATE to z22 mode also enables a number of other functions in JES2 2.2, such as increased limits for checkpoint data, dynamic changes to the checkpoint size, and a large cache for spool space allocations.

JES2 can be retroactivated to z11 mode by the $ACTIVATE command, but to do this, there can be no job groups defined in the system.

# JOBGROUP statement

***Purpose:*** The JOBGROUP statement defines a job group. This identifies the name and attributes of the group. JOBGROUP definition starts with a JOBGROUP statement and ends with an ENDGROUP statement.

Job groups and pre-execution jobs that are associated with a job group cannot be offloaded using the SPOOL offload facility. However, sending a job group (the static control statements and all the related jobs) to another NJE node is supported. To send a JOBGROUP statement to an NJE node, you need to use the XMIT JCL or JECL statement. Then, after that, all the jobs that are associated with the job group also need to be routed to that node.

NJE has a restriction of a single job for each NJE stream (a job header and job trailer). This restriction continues to apply to job groups. The job group is sent as one NJE stream, and each job is sent in individual NJE streams.

The parameters that you can specify for job processing are arranged alphabetically in the following sections.

***References:*** For information about the JES initialization parameters that provide installation defaults, see *z/OS JES2 Initialization and Tuning Reference*.

## Description

### Syntax

```
//grpname JOBGROUP  positional-parameters[,keyword-parameter]...  [comments]

//grpname JOBGROUP
```

The JOBGROUP statement consists of the characters // in columns 1 and 2 and four fields: name, operation (JOBGROUP), parameter, and comments. Do not code comments if the parameter field is blank.

A JOBGROUP statement is required for each job group.

## Name field

Code a grpname on every JOBGROUP statement, as follows:

- Each grpname must be unique.
- The grpname must begin in column 3.
- The grpname is 1 through 8 alphanumeric or national ($, #, @) characters.
- The first character must be alphabetic or national ($, #, @).
- The grpname must be followed by at least one blank.

## Operation field

The operation field consists of the characters JOBGROUP and must be preceded and followed by at least one blank. It can begin in any column.

## Parameter field

A JOBGROUP statement has two kinds of parameters - positional and keyword. All parameters are optional, however, your installation might require the accounting information parameter and the programmer's name parameter.

- GROUP=RACF_*group*
- PASSWORD=*password*
- SECLABEL=*seclabel*
- TYPE=SCAN,
- HOLD=NO | YES
- ERROR=(*condition*)
- ONERROR=STOP | SUSPEND | FLUSH
- SYSAFF=(*affinity_list*)
- SYSTEM=(*system_list*)
- SCHENV=*scheduling_environment*
- Accounting information (positional)
- Programmer name (positional)

***Positional Parameters:*** A JOBGROUP statement can contain two positional parameters. They must precede all keyword parameters. You must code the accounting parameter first, followed by the programmer's name parameter.

**Accounting information**
Use the accounting information parameter to enter an account number or other accounting information. Same usage as accounting information parameter on a JOB statement.

**Programmer name**
Use the programmer⌂ name parameter to identify the person or group responsible for a job group. Same usage as programmer name parameter on a JOB statement.

***Keyword Parameters:*** A JOBGROUP statement can contain the following keyword parameters. You can code any of the keyword parameters in any order in the parameter field after the positional parameters.

**OWNER=**
The user ID to be associated with the job group. Rules are the same as for USER parameter on a JOB statement.

**GROUP=**
The RACF group to be associated with job group. Rules are the same as for GROUP parameter on a JOB statement.

**PASSWORD=**
The password (if required) and optional new password for the user ID associated with the job group.

**SECLABEL=**
The security label to be associated with the job group. Rules are the same as for SECLABEL parameter on a JOB statement.

**TYPE=SCAN**
The job group is checked for validity but not processed. Any error is recorded in the logging job. Since the internal structures for this job group are never created, any jobs that are subsequently submitted for this job group will fail.

**HOLD=NO|YES**
The job group can be submitted in a held or a non-held state. If the job group is submitted in the hold state, none of the jobs that are associated with this job group will run until the job group is released.

**ERROR=(*condition*)**
This parameter defines a set of conditions that must be evaluated after each job in the job group completes execution to determine if an error condition is to be raised. If the condition is true, the job group is marked as in error and the action defined by ONERROR= parameter is taken.

The syntax of *condition* is the same as used for the conditional syntax on the IF statement. Supported keywords that can be tested are:

| Keyword | Use |
|---|---|
| RC | Indicates the return code of a job. |
| ABEND | Indicates that an ABEND condition occurred. |
| ¬ABEND | Indicates that no ABEND condition occurred. |
| ABENDCC | Indicates a specific system or user ABEND code. |
| RUN | Indicates that the job was executed. |
| ¬RUN | Indicates that the job was flushed from the job group. |

The operators that you can use are:

```
Operator                  Operation                   Order
--------                  ---------                   -----

NOT operator:

Not or ¬ or !             NOT                         first

Comparison operators:

GT  or  >                 Greater than                second
LT  or  <                 Less than                   second
NG or ¬> or !>            Not greater than            second
NL or ¬< or !<            Not less than               second
EQ  or  =                 Equal to                    second
NE or ¬= or !=            Not equal to                second
GE  or  >=                Greater than or equal to    second
LE  or  <=                Less than or equal to       second

Logical operators:

AND or  &                 AND                         third
OR  or  |                 OR                          third
```

Examples of **ERROR=**:

```
ERROR=(RC=4 | RC=8)
ERROR=(!ABEND AND RC=8)
ERROR=(ABENDCC=S0C4 OR ABENDCC=U1024)
```

**ONERROR=STOP|SUSPEND|FLUSH**
This is the action to take when a job group is determined to be in error. This applies when the condition defined on the ERROR= keyword is encountered or when a dependency is considered to fail.

**STOP**
No new jobs in the job group are started. Actively running jobs are allowed to complete. Jobs that are determined to be in error (based on the JOBGROUP ERROR= keyword or the condition in a dependency) can be resubmitted and the error state cleared if they run successfully.

**SUSPEND**
New jobs that have their dependencies satisfied are allowed to start. Jobs that are determined to be in error (based on the JOBGROUP ERROR= keyword or the condition in a dependency) are considered to have not run. These jobs can be resubmitted and the error state cleared if they run successfully.

**FLUSH**
All jobs that have not executed yet are canceled (flushed). No new jobs are started. When there are no longer any jobs running, the job group is marked completed.

**SYSAFF=**
Base system affinity for all jobs that are associated with this job group. Syntax is the same as SYSAFF= on the JOB card. This specification is combined (ANDed) with any affinity specification for each job in the group.

**SYSTEM=**
Indicates the systems that are eligible to process the jobs associated with this job group. Syntax is the same as SYSTEM= parameter on the JOB statement. This list is combined (ANDed) with any affinity specification for each job in the group.

**SCHENV=**
Default scheduling environment for all jobs that are associated with this job group. Syntax is the same as SCHENV= on the JOB card. The list of systems where this scheduling environment is available is combined (ANDed) with the other affinity specifications for the job. Note that this implies that a job in a job group can have two scheduling environments specified - the one for the job group and another one for the job in the group.

## Comments field

The comments field follows the parameter field after at least one intervening blank space. If you do not code any parameters on a JOBGROUP statement, do not code any comments.

## Location in the JCL

A JOBGROUP statement must be the first statement in each job group.

## Error on JOBGROUP statement

JOBGROUP statements terminate any currently active JCL being submitted (in much the same way as a JOB statement ends any current stream). A second JOBGROUP statement without an ENDGROUP statement is considered an error for the first JOBGROUP.

### Examples of JOBGROUP statements

```
//RUN00001 JOBGROUP OWNER=IBMUSER,PASSWORD=IBMUSER
```

# GJOB statement

***Purpose:*** All jobs in a job group must be defined with an appropriate GJOB or SJOB statement.

A GJOB is followed by as many dependencies (BEFORE, AFTER, CONCURRENT) as required for the job. All jobs referenced by the dependencies must be defined within the JOBGROUP. You cannot have dependencies on jobs outside of the JOBGROUP (or in another JOBGROUP). A job can be defined before or after the dependencies that reference the job.

***References:*** For information about the JES initialization parameters that provide installation defaults, see *z/OS JES2 Initialization and Tuning Reference* .

## Description

### Syntax

```
//gjobname GJOB FLUSHTYP=ALLFLUSH|ANYFLUSH  [comments]
```

The GJOB statement consists of the characters // in columns 1 and 2 and four fields: name, operation (GJOB), parameter, and comments. Do not code comments if the parameter field is blank.

### Name field

Each gjobname must be unique within the job group. This uniqueness spans other GJOBs, job sets, and SJOBs.

- Each gjobname must be unique.
- The gjobname must begin in column 3.
- The first character must be alphabetic or national ($, #, @).
- The gjobname is 1 through 8 alphanumeric or national ($, #, @) characters.
- The gjobname must be followed by at least one blank.
- gjobnames must be unique within a job group, gjobnames, job set names, SJOB names, and job group names all share the same namespace.

The gjobname identifies the name of a job that is part of the job group being defined. This name must match the name of a job that is submitted (that has a SCHEDULE statement identifying this job group) after the job group is defined.

### Operation field

The operation field consists of the characters GJOB and must be preceded and followed by at least one blank. It can begin in any column.

### Parameter field

A GJOB statement has one keyword parameter and no positional parameters.

***Keyword Parameters:*** A GJOB statement can contain the following keyword parameter.

**FLUSHTYP=ALLFLUSH|ANYFLUSH**
This job is flushed if all its parent jobs are flushed (ALLFLUSH) or if any one of its parent jobs are flushed (ANYFLUSH). The default is ALLFLUSH.

### Comments field

The comments field follows the parameter field after at least one intervening blank space. If you do not code any parameters on a GJOB statement, do not code any comments.

### Location in the JCL

A GJOB statement is only valid within a job group context. It cannot be located between a JOBSET and ENDSET statements.

### Error on GJOB statement

If the system finds an error on the GJOB statement, an HASP1116 message is issued which describes the error. See *z/OS JES2 Messages*. for more information about the message.

### Examples of GJOB statements

```
//MULTI    JOBGROUP            JOBGROUP describes a group of dependent jobs
//A        GJOB                GJOB describes a job in a group
//B        GJOB
//           AFTER NAME=A,      AFTER is positioned after a GJOB and describes its
//            WHEN=(RC=0)      dependency to another named job
//C        GJOB
//           AFTER NAME=A,
//            WHEN=(RC=4)      WHEN= describes a condition associated with the dependency
//D        GJOB
//           AFTER NAME=B
//           AFTER NAME=C
//MULTI    ENDGROUP            ENDGROUP marks the end of the job group definition
```

## JOBSET statement

***Purpose:*** A convenient method to define jobs with the same set of dependencies.

***References:*** For information about the JES initialization parameters that provide installation defaults, see *z/OS JES2 Initialization and Tuning Reference* .

## Description

### Syntax

| `//setname JOBSET FLUSHTYP=ALLFLUSH\|ANYFLUSH  [comments]` |
|---|
| The JOBSET statement consists of the characters // in columns 1 and 2 and four fields: name, operation (JOBSET), parameter, and comments. Do not code comments if the parameter field is blank. |

### Name field

Code a job set name on every JOBSET statement, as follows:

- Each setname must be unique within the job group. This uniqueness spans other job sets, GJOBs, and SJOBs.
- The setname must begin in column 3.
- The setname is 1 through 8 alphanumeric or national ($, #, @) characters.
- The first character must be alphabetic or national ($, #, @).
- The setname must be followed by at least one blank.

This is the name associated with the job set. It can be specified on NAME= keyword of the AFTER and BEFORE statements.

### Operation field

The operation field consists of the characters JOBSET and must be preceded and followed by at least one blank. It can begin in any column.

### Parameter field

A JOBSET statement has one keyword parameter and no positional parameters.

***Keyword Parameters:*** A JOBSET statement can contain the following keyword parameter.

**FLUSHTYP=ALLFLUSH|ANYFLUSH**
For each job in the set, the job is flushed if all parent jobs are flushed (ALLFLUSH) or if *any* parent jobs are flushed (ANYFLUSH).

### Comments field

The comments field follows the parameter field after at least one intervening blank space. If you do not code any parameters on a JOBSET statement, do not code any comments.

### Location in the JCL

A JOBSET statement is only valid within a job group context. A JOBSET statement must have a matching ENDSET statement. Job set definitions cannot be nested.

### Error on JOBSET statement

If the system finds an error on the JOBSET statement, an HASP1117 message is issued which describes the error. See *z/OS JES2 Messages*. for more information about the message.

### Examples of JOBSET statements

```
//SET1   JOBSET
//JOBA   SJOB
//JOBB   SJOB
//SET1   ENDSET
```

## SJOB statement

***Purpose:*** The SJOB JCL statement defines a job inside a job set.

***References:*** For information about the JES initialization parameters that provide installation defaults, see *z/OS JES2 Initialization and Tuning Reference* and *z/OS JES3 Initialization and Tuning Reference*.

## Description

### Syntax

```
//sjobname SJOB [comments]
```

The SJOB statement consists of the characters // in columns 1 and 2 and four fields: name and operation (SJOB).

An SJOB statement is required for each job within a job set. SJOB denotes one job. Name on an SJOB statement cannot be a name of a job set.

### Name field

Each job name that is specified on an SJOB must be unique within the job group. This uniqueness spans other GJOBs, job sets, and SJOBs.

Code a sjobname on every SJOB statement, as follows:

- Each sjobname must be unique.
- The sjobname must begin in column 3.
- The sjobname is 1 through 8 alphanumeric or national ($, #, @) characters.
- The first character must be alphabetic or national ($, #, @).
- The sjobname must be followed by at least one blank.

The sjobname identifies the name of a job that is included in the job set. This name must match the name of a job that is submitted (that has a SCHEDULE statement identifying this job group) after the job group is defined.

### Operation field

The operation field consists of the characters SJOB and must be preceded and followed by at least one blank. It can begin in any column.

### Parameter field

An SJOB statement has no parameters.

### Comments field

The SJOB statement has no parameters. Comments can be coded anywhere after the operation field.

### Location in the JCL

An SJOB statement is only valid within a job set context.

### Error on SJOB JCL statement

If the system finds an error on the SJOB statement, an HASP1115 message is issued which describes the error. See *z/OS JES2 Messages*. for more information about the message.

### Examples of SJOB statements

```
//SET1   JOBSET
//JOBA   SJOB
//JOBB   SJOB
//SET1   ENDSET
```

## ENDSET statement

***Purpose:*** The ENDSET JCL statement marks the end of a job set.

***References:*** For information about the JES initialization parameters that provide installation defaults, see *z/OS JES2 Initialization and Tuning Reference* and *z/OS JES3 Initialization and Tuning Reference*.

## Description

### Syntax

```
//setname ENDSET [comments]
```

The ENDSET statement consists of the characters // in columns 1 and 2 and two fields: name, operation (ENDSET).

An ENDSET statement is required for each JOBSET statement.

### Name field

Code a job set name on every ENDSET statement, as follows:

- Each setname must be unique.
- The setname that is specified on the ENDSET statement must match the setname that is specified on the matching JOBSET statement.

### Operation field

The operation field consists of the characters ENDSET and must be preceded and followed by at least one blank. It can begin in any column.

### Parameter field

An ENDSET statement has no parameters.

### Location in the JCL

An ENDSET statement marks the end of a job set and must be paired with a JOBSET statement.

### Error on ENDSET statement

If the system finds an error on the ENDSET statement, an HASP1118 message is issued which describes the error. See *z/OS JES2 Messages*. for more information about the message.

### Example of ENDSET statement

```
//SET1   JOBSET
//JOBA   SJOB
//JOBB   SJOB
//SET1   ENDSET
```

## BEFORE statement

***Purpose:*** Use the BEFORE statement, along with GJOB and JOBSET statements, to define jobs and sets of jobs that must execute in a particular sequence. AFTER and BEFORE statements have the same syntax and define the dependencies either as a dependent/parent or parent/dependent. Internally, all relationships are managed as parent/dependent.

***References:*** For information about the JES initialization parameters that provide installation defaults, see *z/OS JES2 Initialization and Tuning Reference* and *z/OS JES3 Initialization and Tuning Reference*.

## Description

### Syntax

| |
|---|
| `//  BEFORE NAME=name\|(name,name...),[keyword-parameter]... [comments]` |
| The BEFORE statement consists of the characters // in columns 1 and 2 and three fields: operation (BEFORE), parameter, and comments. Do not code comments if the parameter field is blank. The list may contain job(s) and/or job set(s). The list of names supports a maximum of 10 elements. |

### Name field

There is no name that is specified on a BEFORE statement.

## Operation field

The operation field consists of the characters BEFORE, and must be preceded and followed by at least one blank. It can begin in any column.

## Parameter field

The BEFORE statement has multiple keyword parameters, and the NAME keyword is required.

***Positional Parameters:*** None.

***Keyword Parameters:*** A BEFORE statement can contain the following keyword parameters:

**NAME=name | (name,name, ... )**
This specifies that the job specified on the preceding GJOB statement, or all of the jobs in the job set specified by the preceding JOBSET statement, must run before a job specified by NAME=. The list supports a maximum of 10 elements. If more names are wanted, code more BEFORE or AFTER statements. The NAME parameter must be the first parameter on the BEFORE statement. Each name in the list can be a name of a job specified by the GJOB statement or a name of a job set specified by the JOBSET statement.

**WHEN=(*condition*)**
This parameter defines a set of conditions that must be evaluated to determine what processing should occur next. The conditions apply to the parent job in the dependency (the job that is specified on the preceding GJOB statement or each of the jobs in the job set specified by the preceding JOBSET statement). Either the processing specified by ACTION= is taken (if the condition is true) or the processing specified by OTHERWISE is taken (if the condition is false).

The default for WHEN= is (RUN AND !ABEND).

The syntax of *condition* is the same as used for the conditional syntax on the IF statement. Supported keywords that can be tested are:

| Keyword | Use |
|---|---|
| RC | Indicates return code of a job. |
| ABEND | Indicates that an ABEND condition occurred. |
| ¬ABEND | Indicates that no ABEND condition occurred. |
| ABENDCC | Indicates a specific system or user ABEND code. |
| RUN | Indicates that the job was executed. |
| ¬RUN | Indicates that the job was flushed from the job group. |

The operators that you can use are:

```
Operator                Operation                Order
--------                ---------                -----

NOT operator:

Not or ¬ or !           NOT                      first

Comparison operators:

GT  or  >               Greater than             second
LT  or  <               Less than                second
NG or ¬> or !>          Not greater than         second
NL or ¬< or !<          Not less than            second
EQ  or  =               Equal to                 second
NE or ¬= or !=          Not equal to             second
GE  or  >=              Greater than or equal to  second
```

```
LE  or  <=                Less than or equal to          second

Logical operators:

AND or  &                 AND                            third
OR  or  |                 OR                             third
```

Examples of **WHEN=**:

```
WHEN=(RC=4 | RC=8)
WHEN=(!ABEND AND RC=8)
WHEN=(ABENDCC=S0C4 OR ABENDCC=U1024)
```

**ACTION=SATISFY|FLUSH|FAIL**
Processing to perform if the WHEN condition is true. The valid values are:

**SATISFY**
The dependency is considered satisfied. This is the default.

**FLUSH**
The dependency is considered flushed. The dependent job might be flushed, depending on the FLUSHTYP=ALLFLUSH/ANYFLUSH value of the dependent job.

**FAIL**
The failure of this dependency marks the job group in error. The ONERROR= action from the JOBGROUP statement is taken as a result of the failure.

**OTHERWISE=FLUSH|FAIL|SATISFY**
Processing to perform if the WHEN condition is false. The valid values are:

**FLUSH**
The dependency is considered flushed. The dependent job might be flushed, depending on the value FLUSHTYP=ALLFLUSH/ANYFLUSH of the dependent job. This is the default.

**FAIL**
The failure of this dependency marks the job group in error. The ONERROR action from the JOBGROUP statement is taken as a result of this failure.

**SATISFY**
The dependency is considered satisfied.

## Comments field

The comments field follows the parameter field after at least one intervening blank space.

## Location in the JCL

The BEFORE statement is valid within a job group context and must follow a GJOB or a JOBSET statement.

## Error on BEFORE statement

If the system finds an error on the BEFORE statement, an HASP1112 message is issued which describes the error. See *z/OS JES2 Messages*. for more information about the message.

## Examples of BEFORE statements

```
//GRP1    JOBGROUP
//JOB1    GJOB
//JOB2    GJOB
```

```
//         AFTER NAME=JOB1,WHEN=(RC=0)
//JOB3    GJOB
//         BEFORE NAME=JOB1
//GRP1    ENDGROUP
```

# AFTER statement

***Purpose:*** Use the AFTER JCL statement, along with GJOB and JOBSET statements, to define jobs and sets of jobs that must execute in a particular sequence. AFTER and BEFORE statements have the same syntax and define the dependencies either as a dependent/parent or parent/dependent. Internally, all relationships are managed as parent/dependent.

***References:*** For information about the JES initialization parameters that provide installation defaults, see *z/OS JES2 Initialization and Tuning Reference* and *z/OS JES3 Initialization and Tuning Reference*.

## Description

### Syntax

```
//      AFTER  NAME=name|(name,name...),[keyword-parameter]... [comments]
```

The AFTER statement consists of the characters // in columns 1 and 2 and three fields: operation (AFTER), parameter, and comments. Do not code comments if the parameter field is blank. The list of names supports a maximum of 10 elements.

### Name field

There is no name that is specified on a AFTER statement.

### Operation field

The operation field consists of the characters AFTER, and must be preceded and followed by at least one blank. It can begin in any column.

### Parameter field

The AFTER statement has multiple keyword parameters, and the NAME keyword is required. The NAME parameter must be the first parameter on the AFTER statement.

***Positional Parameters:*** None.

***Keyword Parameters:*** An AFTER statement can contain the following keyword parameters:

**NAME=name | (name,name, ... )**
This specifies that the job specified on the preceding GJOB statement, or all of the jobs in the job set specified by the preceding JOBSET statement, must run after a job specified by NAME=. Each name in the list can be a name of a job specified by the GJOB statement or a name of a job set specified by the JOBSET statement.

**WHEN=(*condition*)**
This specifies a set of conditions that must be evaluated to determine what processing should occur next. The conditions apply to the parent job in the dependency (the job that is specified on NAME=). Either the processing specified by ACTION= is taken (if the condition is true) or the processing specified on OTHERWISE is taken (if the condition is false).

The default for WHEN= is (RUN AND !ABEND).

The syntax of *condition* is the same as used for the condition syntax on the IF statement. Supported keywords that can be tested are:

| Keyword | Use |
|---|---|
| RC | Indicates return code of a job. |
| ABEND | Indicates that an ABEND condition occurred. |
| ¬ABEND | Indicates that no ABEND condition occurred. |
| ABENDCC | Indicates a specific system or user ABEND code. |
| RUN | Indicates that the job was executed. |
| ¬RUN | Indicates that the job was flushed from the job group. |

The operators that you can use are:

```
Operator                  Operation                  Order
--------                  ---------                  -----

NOT operator:

Not or ¬ or !             NOT                        first

Comparison operators:

GT  or  >                 Greater than               second
LT  or  <                 Less than                  second
NG or ¬> or !>            Not greater than           second
NL or ¬< or !<            Not less than              second
EQ  or  =                 Equal to                   second
NE or ¬= or !=            Not equal to               second
GE  or  >=                Greater than or equal to   second
LE  or  <=                Less than or equal to      second

Logical operators:

AND or  &                 AND                        third
OR  or  |                 OR                         third
```

Examples of **WHEN=**:

```
WHEN=(RC=4 | RC=8)
WHEN=(!ABEND AND RC=8)
WHEN=(ABENDCC=S0C4 OR ABENDCC=U1024)
```

**ACTION=SATISFY|FLUSH|FAIL**
Processing to perform if the WHEN condition is true. The valid values are:

**SATISFY**
The dependency is considered satisfied. This is the default.

**FLUSH**
The dependency is considered flushed. The dependent job might be flushed, depending on the FLUSHTYP=ALLFLUSH/ANYFLUSH value of the dependent job.

**FAIL**
The failure of this dependency marks the job group in error. The ONERROR= action from the JOBGROUP statement is taken as a result of the failure.

**OTHERWISE=FLUSH|FAIL|SATISFY**
Processing to perform if the WHEN condition is false. The valid values are:

**FLUSH**
The dependency is considered flushed. The dependent job might be

flushed, depending on the value FLUSHTYP=ALLFLUSH/ANYFLUSH of the dependent job. This is the default.

**FAIL**
The failure of this dependency marks the job group in error. The ONERROR action from the JOBGROUP statement is taken as a result of this failure.

**SATISFY**
The dependency is considered satisfied.

### Comments field
The comments field follows the parameter field after at least one intervening blank space.

### Location in the JCL
The AFTER statement is valid within a job group context and must follow a GJOB or a JOBSET statement.

### Error on AFTER statement
If the system finds an error on the AFTER statement, an HASP1112 message is issued which describes the error. See *z/OS JES2 Messages*. for more information about the message.

### Example of AFTER statements
```
//GRP1    JOBGROUP
//JOB1    GJOB
//JOB2    GJOB
//         AFTER NAME=JOB1,WHEN=(RC=0)
//JOB3    GJOB
//         BEFORE NAME=JOB1
//GRP1    ENDGROUP
```

## CONCURRENT statement

***Purpose:*** Use the CONCURRENT statement, together with a GJOB statement, to define jobs or job sets that must execute at the same time (simultaneously) on the same JES2 MAS member.

Jobs in a concurrent dependency can have other dependencies that are associated with them. However, all dependencies for all jobs that are to run concurrently, must be met before any of the concurrent jobs start. Jobs that run together are referred to as a concurrent set. JES uses WLM demand batch initiators to run jobs in the concurrent set when capacity is available. The example below illustrates this concept.

All jobs in a concurrent set must use the same WLM service class. You must define the attributes of jobs in a concurrent set in such a way so that the service class classification rules classify all jobs in the concurrent set to the same service class. If they are not classified to the same service class, then JES processing selects one job in the set at random and assigns its service class to all jobs in the concurrent set.

When a concurrent set goes into execution, an HASP1301 message is issued for one of the jobs in the set and is followed by an HASP1201 message when the concurrent set enters execution.

# Description

## Syntax

```
//         CONCURRENT NAME=name|(name,name,...)...  [comments]
```

The CONCURRENT statement consists of the characters // in columns 1 and 2 and three fields: operation (CONCURRENT), parameter, and comments. Do not code comments if the parameter field is blank.

The job specified by a preceding GJOB statement runs concurrently with the jobs specified on NAME= list. The list may contain job names or job set names. The list supports a maximum of 10 elements.

## Name field

There is no name that is specified on a CONCURRENT statement.

## Operation field

The operation field consists of the characters CONCURRENT and must be preceded and followed by at least one blank. It can begin in any column.

## Parameter field

A CONCURRENT statement has one keyword parameter NAME and is required.

***Positional Parameters:*** None.

***Keyword Parameters:*** A CONCURRENT statement can contain the following keyword parameter.

**NAME=name**

This specifies that the job specified on the preceding GJOB statement must run at the same time (simultaneously) with the jobs specified by the NAME parameter. These jobs must run at the same time on the same JES image. Job names or job set names may be specified.

Note that jobs in a concurrent dependency can have other dependencies that are associated with them. However, all dependencies for all jobs that run concurrently must be met before any of the jobs in the concurrent set start.

## Comments field

The comments field follows the parameter field after at least one intervening blank space. If you do not code any parameters on a JOB statement, do not code any comments.

## Location in the JCL

A CONCURRENT statement is only valid within a job group context and must follow a GJOB statement.

## Error on CONCURRENT statement

If the system finds an error on the CONCURRENT statement, an HASP1114 message is issued which describes the error. See *z/OS JES2 Messages*. for more information about the message.

## Examples of CONCURRENT statements

```
//GRP1        JOBGROUP
//JOB1         GJOB
//JOB2         GJOB
//              AFTER NAME=JOB1
```

```
//              CONCURRENT NAME=(JOB3,JOB4,JOB5)
//JOB6          GJOB
//              CONCURRENT NAME=(JOB3,JOB7)
//JOB3          GJOB
//JOB4          GJOB
//JOB5          GJOB
//JOB7          GJOB
//GRP1         ENDGROUP
```

# ENDGROUP statement

***Purpose:*** The ENDGROUP JCL statement defines the end of a job group.

***References:*** For information about the JES initialization parameters that provide installation defaults, see *z/OS JES2 Initialization and Tuning Reference* .

## Description

### Syntax

| `//grpname ENDGROUP` |
|---|
| The ENDGROUP statement consists of the characters // in columns 1 and 2 and two fields: name and operation (ENDGROUP). |

### Name field

Specify the grpname field that was specified on the corresponding JOBGROUP statement. This value is required and is checked against the grpname from the JOBGROUP statement. If they are not the same, the JOBGROUP is in error.

### Operation field

The operation field consists of the characters ENDGROUP and must be preceded and followed by at least one blank. It can begin in any column.

### Comments field

The ENDGROUP statement has no parameters.

### Location in the JCL

An ENDGROUP statement must be matched with a JOBGROUP statement of the same name.

### Error on ENDGROUP statement

If the system finds an error on the ENDGROUP statement, an HASP1111 message is issued which describes the error. See *z/OS JES2 Messages*.

### Examples of ENDGROUP statements

```
//MULTI    JOBGROUP           JOBGROUP describes a group of dependent jobs
//B        GJOB               Job within the group
//C        GJOB               Job within the group
//A        GJOB               A runs before B when ...
//           BEFORE NAME=B,     BEFORE is positioned after a GJOB and describes its
//            WHEN=(RC=0)     dependency to another named job
//           BEFORE NAME=C,    A runs before C when ...
//            WHEN=(RC=4)      WHEN= describes a condition associated with the dependency
//D        GJOB
//           AFTER NAME=B,
//           AFTER NAME=C
//MULTI    ENDGROUP           ENDGROUP marks the end of the job group definition
```

# Chapter 30. JES2 control statements

Code JES2 control statements with JCL statements to control the input and output processing of jobs. The rules for coding in Chapter 3, "Format of statements," on page 13, and Chapter 4, "Syntax of parameters," on page 19, apply to the JES2 control statements.

## Description

### Considerations for started tasks

The following statements are not supported for a started task:

- /*PRIORITY
- /*ROUTE XEQ
- /*SETUP
- /*XEQ
- /*XMIT
- /*$xxx

The /*PRIORITY statement is ignored. All other statements cause JES2 to fail the job.

### Considerations for an APPC scheduling environment

JES2 control statements have no function in an APPC scheduling environment. If you code them, the system will detect them as JCL errors.

### Location in the JCL

Place JES2 control statements, except the command and /*PRIORITY statements, after the JOB statement and its continuations. JES2 ignores JES2 control statements, except the command and /*PRIORITY statements, that appear before the JOB statement or between continued JOB statements.

Do not include JES2 control statements in a cataloged or in-stream procedure. JES2 ignores JES2 control statements in a procedure.

### Internal reader

Use the following control statements when submitting jobs to the internal reader. The internal reader is described in *z/OS MVS Programming: Assembler Services Guide*.

- /*DEL
- /*EOF
- /*PURGE
- /*SCAN

## JES2 command statement

***Purpose***

© Copyright IBM Corp. 1988, 2015

Use the command statement to enter a JES2 operator command through the input stream, the internal reader, or the system console.

**Note:** Do not specify this statement for a started task; if /*$xxx is specified, JES2 fails the job.

JES2 usually executes an in-stream command as soon as it is read. Therefore, the command will **not** be synchronized with the execution of any job or step in the input stream. To synchronize a command with the job processing, tell the operator the commands you want and when they should be issued, and let the operator enter them from the console.

Examples illustrate the format for commands entered through the input stream. Commands entered through an operator console should not have /* in columns 1 and 2.

*References*

For more information on the command statement and the JES2 verbs and operands, see *z/OS JES2 Commands*.

## Syntax

```
/*$command-verb,operand[,operand]...      [N]
```

The JES2 command statement consists of:

- The characters /* in columns 1 and 2.
- $ or a character chosen by the installation in column 3. For more information, see JES2 initialization statement CONDEF, RDRCHAR=.
- The command verb beginning in column 4.
- A comma.
- Operands up through column 71.
- N in column 72 if JES2 is **not** to write the command on the operator console.
- Blanks in columns 73 through 80. JES2 ignores these columns.

Do not continue command statements from one statement to the next, instead code as many command statements as you need.

## Parameter definition

**command-verb**
Specifies the operator command that JES2 is to perform. You can enter the following JES2 commands in the input stream.

```
$A  $E   $I   $O   $T
$B  $F   $L   $P   $RACE
$C  $G   $M   $R   $VS
$D  $H   $N   $S   $Z
```

**operand**
Specifies options for the command.

**N in column 72**
Indicates that JES2 is not to repeat the command on the operator console.

## Location in the JCL

Place JES2 command statements before jobs being entered through the input stream. JES2 ignores any JES2 command statements within a job.

Do not code JES2 commands in an NJE job stream. If you code JES2 commands in an NJE job stream, the system will not process them and will issue an error message.

If a job contains a JES2 /*XMIT statement, and you want JES2 to process and display the command at the input node only, place the command statement before the /*XMIT statement.

## Examples of the command statement

***Example 1***

```
/*$SI3-5
```

This command statement starts initiators three through five. The command is $S and the operand is I3-5. JES2 executes the command immediately and repeats the command on the operator console.

***Example 2***

```
/*$TRDR1,H=Y
```

In response to this command, JES2 places all jobs being read by reader 1 in a hold status. If a job contains a JES2 /*ROUTE XEQ or /*XEQ statement that specifies an execution node different from the input node, JES2 holds the job at the execution node, not the input node.

# /*JOBPARM statement

***Purpose:*** Use the /*JOBPARM statement to specify job-related parameters for JES2.

**Note:** For started tasks:

- The TIME parameter is ignored
- If RESTART=N, the parameter is ignored
- If RESTART=Y, JES2 fails the job
- For SYSAFF, the system on which the job is being started must be in the list of systems implied or specified, or JES2 will fail the job.

## Syntax

```
/*JOBPARM parameter[,parameter]...
```

The parameters are:

```
       {BURST} = {Y}
       {B    }   {N}

       {BYTES} = nnnnnn
       {M    }

       {CARDS} = nnnnnnnn
       {C    }

       {COPIES} = nnn
       {N     }

       {FORMS} =  {xxxxxxxx}
       {F    }    {STD     }

       {LINECT} = nnn
       {K     }

       {LINES} = nnnnnn
       {L    }

       {NOLOG}
       {J    }

       {PAGES} = nnnnnnnn
       {G    }

       {PROCLIB} = ddname
       {P      }

       {RESTART}  = {Y}
       {E      }    {N}

       {ROOM} = xxxx
       {R   }

       {SYSAFF} = {*                       }
       {S     }   {(*[,IND])               }
                  {ANY                     }
                  {(ANY[,IND])             }
                  {cccc                    }
                  {(cccc[,IND])            }
                  {(cccc[,cccc]...)        }
                  {((cccc[,cccc]...)[,IND])}

       {TIME} = nnnn
       {T   }
```

The /*JOBPARM statement consists of the characters /* in columns 1 and 2, JOBPARM in columns 3 through 9, a blank in column 10, and parameters in columns 11 through 71. JES2 ignores columns 72 through 80.

Do not continue a /*JOBPARM statement. Instead, code as many /*JOBPARM statements as necessary in an input stream.

Code any number of the parameters listed in this topic on a single /*JOBPARM statement.

## Parameter definition

**BURST=Y**

**BURST=N**
Specifies the default burst characteristic of all sysout data sets that JES2 produces for this job. BURST applies only when the data set is directed to a 3800 Printing Subsystem equipped with a burster-trimmer-stacker.

**Y** Requests that the 3800 output is to be burst into separate sheets.

**N** Requests that the 3800 output is to be in a continuous fanfold.

**BYTES=nnnnnn**
Specifies the maximum output, in thousands of bytes, the system is to produce from this job. The nnnnnn is 1 through 6 decimal numbers from 0 through 999999. When nnnnnn bytes are reached, JES2 gives control to an installation exit routine and the job might or might not be terminated.

**CARDS=nnnnnnnn**
Specifies the maximum number of output cards to be punched for this job's sysout data sets. The value is 1 through 8 decimal numbers from 0 through 99999999. When the specified number of cards is reached, JES2 gives control to an installation exit routine and the job might or might not be terminated.

**COPIES=nnn**
Specifies how many copies of the spool lines or bytes for this job's sysout data sets are to be printed or punched. The nnn is 1 through 3 decimal numbers from 1 through 255. An installation can reduce the upper limit of this value during JES2 initialization.

The COPIES parameter is ignored and only one copy is produced if any of the following is true:

- FREE=CLOSE is coded on the DD statement for the output data set.
- HOLD=YES is coded on any sysout DD statement in the job.
- The output class of the sysout data set is a held class, and the message class is also a held class. The message class is specified in the JOB statement MSGCLASS parameter.

**Note:**

The use of '/*JOBPARM COPIES= ' creates output groups which are special in a sense that they are a clone of the original output group. When these output groups are transmitted in an NJE network, the count of copies transmitted will not behave as expected. Another side effect of these clone JOEs is that their attributes cannot be changed, and will remain the same as that of the original output group.

It is recommended to avoid using the /*JOBPARM JECL statement to produce multiple copies, especially when those outputs will go through NJE network. Instead, use COPIES keyword at the DD level. When you use COPIES keyword at the DD level, clone output groups will not be produced and the outputs will be manipulated appropriately.

**FORMS=xxxxxxxx**
**FORMS=STD**
Specifies the print and/or punch forms JES2 is to use for sysout data sets for which FORMS is not specified on the DD statement or on a JES2 /*OUTPUT statement.

**xxxxxxxx**
Identifies the print or punch forms. The xxxxxxxx is 1 through 8 alphanumeric or national ($, #, @) characters.

**STD**
Indicates that JES2 is to use the default specified at JES2 initialization.

**LINECT=nnn**
Specifies the maximum number of lines that JES2 is to print on each output page for this job's sysout data sets. The nnn is 1 through 3 numbers from 0 through 254.

If you code LINECT=0, JES2 does not eject to a new page when the number of output lines exceeds the page limit that the installation specified during JES2 initialization.

The LINECT parameter on the /*OUTPUT statement overrides LINECT on the /*JOBPARM statement and the linect value in the accounting information parameter of the JOB statement.

**LINES=nnnnnn**
Specifies the maximum output, in thousands of lines, that JES2 is to place in the spool data sets for this job's sysout data sets. The number is 1 through 6 decimal numbers from 0 through 999999. When the specified number of lines is reached, JES2 gives control to an installation exit routine and the job might or might not be terminated.

The LINES parameter applies only to line-mode data. (See also the PAGES parameter.) If the sysout data set contains both line-mode and page-mode data, the lines and pages are counted separately and checked separately against the limit.

**NOLOG**
Requests that JES2 not print the job's hard-copy log. The job's hard-copy log contains the JES2 and operator messages about the job's processing.

**PAGES=nnnnnnnn**
Specifies the maximum number of output pages to be printed for this job's sysout data sets. The number is 1 through 8 decimal numbers from 0 through 99999999. When the specified number of pages is reached, JES2 gives control to an installation exit routine and the job might or might not be terminated.

The PAGES parameter applies only to page-mode data. (See also the LINES parameter.) If the sysout data set contains both page-mode and line-mode data, the pages and lines are counted separately and checked separately against the limit.

**PROCLIB=ddname**
Requests a JES2 procedure library by its ddname, as defined in the JES2 procedure used to initialize JES2. Typically, JES2 procedure library ddnames are in the format PROCnn, where nn is either 00 or 1 or 2 decimal numbers from 1 through 99. You can, however, use any valid ddname as long as the name matches the ddname in the JES2 procedure. The system retrieves called cataloged procedures from the requested JES2 procedure library.

If you omit the PROCLIB parameter, or the ddname cannot be found in the procedure used to start JES2, JES2 uses the procedure library specified on the PROC=nn parameter on one of the following JES2 initialization statements:

- JOBCLASS(v) for each job class
- JOBCLASS(STC) for all started tasks
- JOBCLASS(TSU) for all time-sharing tasks

If the PROC=nn parameter is not defined on the appropriate initialization statement, or if it is not valid, JES2 uses the default library, PROC00. See *z/OS JES2 Initialization and Tuning Guide* for information about creating the JES2

cataloged procedure and *z/OS JES2 Initialization and Tuning Reference* for information about defining JES2 initialization statements.

**RESTART=Y**
**RESTART=N**
Requests one of the following, if this job is executing before a re-IPL and JES2 warm start, and the job cannot restart from a step or checkpoint.

**Y** Requests that JES2 queue the job for re-execution from the beginning of the job.

**N** Requests that JES2 take no special action.

**Note:**

If you do not specify RESTART, JES2 assumes N. However, the installation may override this default in JES2 initialization parameters.

If the job is registered with the automatic restart manager (ARM) at the time of the IPL, ARM determines whether the job is restarted, regardless of whether RESTART=YES or NO is specified.

**ROOM=xxxx**
Indicates the programmer's room number. The xxxx is 1 through 4 alphanumeric characters. JES2 places the room number on the job's separators so that the installation can deliver the job's sysout data sets to the programmer.

**SYSAFF=***
**SYSAFF=(*[,IND])**
**SYSAFF=ANY**
**SYSAFF=(ANY[,IND])**
**SYSAFF=cccc**
**SYSAFF=(cccc[,IND])**
**SYSAFF=(cccc[,cccc]...)**
**SYSAFF=((cccc[,cccc]...)[,IND])**
Indicates the systems that are eligible to process the job. The parameter indicates from 1 system affinity representing a JES2 member name, up to the number of entries that can be coded on a JOBPARM statement, limited by the number of JES2 members that can exist in a MAS.

**Note:**

Use the SYSAFF parameter to ensure the conversion and execution of the job will be done on a specific system. If you code SYSAFF, both processes are done on the specified system.

For TSO-submitted jobs that specify NOTIFY in the JOB statement: after a job has completed execution, JES2 may change the SYSAFF specification for the job if the job executed on a processor other than the processor that the user is logged on. This is done by JES2 during output processing to allow NOTIFY processing to take place on the user's processor.

***** Indicates the system that read the job.

**ANY**
Indicates any system in the JES2 multi-access spool configuration.

**cccc**
Identifies a specific system, where cccc is the JES2 member name of the current system in the JES2 multi-access spool configuration. cccc is 1 through 4 alphanumeric characters. To specify more than one system,

separate the member names with commas and enclose the member name list in parentheses; for example, SYSAFF=(cccc,cccc,cccc).

**Note:** If you specify SYSAFF=*cccc* on the /*JOBPARM statement and also have a /*ROUTE XEQ or /*XEQ statement, the latter statement must appear before the /*JOBPARM statement.

**IND**
After any of the other SYSAFF specifications, indicates that JES2 is to use system scheduling in independent mode. When IND is coded, the subparameters must be enclosed in parentheses. IND cannot be coded by itself. It must be included with at least one JES2 member name, or a JCL error will be reported.

**TIME=nnnn**
Estimates the job execution time, in minutes of real time. The nnnn is 1 through 4 decimal numbers from 0 through 9999. If you omit a TIME parameter and a time subparameter in the JOB statement accounting information parameter, JES2 uses an installation default specified at initialization. If job execution exceeds the time, JES2 sends a message to the operator.

## Overrides

- The /*JOBPARM statement parameters override the installation defaults specified at JES2 initialization.

  **Note:** The /*JOBPARM statement parameters cannot override JES2 installation defaults when it is placed in a catalogued procedure for an STC.
- An OUTPUT JCL statement can override parameters on a /*JOBPARM statement.
- A JES2 /*OUTPUT statement can override parameters on a /*JOBPARM statement.
- Any /*JOBPARM statement parameter value overrides the equivalent parameter value from the JES2 accounting information on the JOB statement or from any preceding /*JOBPARM statement in this job.
- The JOB statement parameters BYTES, CARDS, LINES, and PAGES override the /*JOBPARM parameters BYTES, CARDS, LINES, and PAGES.

## Location in the JCL

Place the /*JOBPARM statement after the JOB statement.

## Execution node

JES2 normally processes /*JOBPARM statements at the node of execution.

When you place a /*JOBPARM statement **before** a /*ROUTE XEQ or /*XEQ statement, JES2 at the input node checks the /*JOBPARM statement for syntax and parameter validity. After processing the /*ROUTE XEQ or /*XEQ statement, JES2 then passes the /*JOBPARM statement to the execution node, where syntax and parameter validity are again checked.

When you place a /*JOBPARM statement **after** a /*ROUTE XEQ or /*XEQ statement, JES2 passes the /*JOBPARM to the execution node and performs all syntax and parameter validity processing at the execution node only.

***COPIES Parameter in Remote Processing:*** In remote processing, the COPIES parameter on the /*JOBPARM statement determines the number of output copies only when the execution node is a JES2 node. The /*JOBPARM COPIES parameter is not supported by RSCS, DOS/VSE POWER, or JES3.

## Examples of the /*JOBPARM statement

```
/*JOBPARM  LINES=60,ROOM=4222,TIME=50,PROCLIB=PROC03,COPIES=5
/*JOBPARM  L=60,R=4222,T=50,P=PROC03,N=5
```

The two statements specify the same parameters and values. The parameter specifications mean the following:

**LINES=60 or L=60**
The job's estimated output will be 60,000 lines.

**ROOM=4222 or R=4222**
The programmer's room is 4222. JES2 places this information in the separators for both printed and punched data sets.

**TIME=50 or T=50**
The job's estimated execution time is 50 minutes.

**PROCLIB=PROC03 or P=PROC03**
The procedure library that JES2 is to use to convert the JCL for this job is PROC03.

**COPIES=5 or N=5**
The estimated 60,000 lines of output will be printed five times.

# /*MESSAGE statement

***Purpose:*** Use the /*MESSAGE statement to send messages to the operator console when JES2 reads in the job.

## Syntax

| /*MESSAGE message |
|---|
| The /*MESSAGE statement consists of the characters /* in columns 1 and 2, MESSAGE in columns 3 through 9, a blank in column 10, and the message starting in any column from 11 through 71. JES2 ignores columns 72 through 80. |

## Relationship to the /*ROUTE XEQ statement

If the /*MESSAGE statement is in a job that also contains a JES2 /*ROUTE XEQ statement:

- Placing the /*MESSAGE statement before the /*ROUTE XEQ statement directs JES2 to send the message to the operators at the input node and the execution node.
- Placing the /*MESSAGE statement after the /*ROUTE XEQ statement directs JES2 to send the message only to the operator at the execution node.

## Location in the JCL

If the /*MESSAGE statement is after the JOB statement, JES2 appends the job number to the beginning of the message.

If the /*MESSAGE statement is not within a job, JES2 appends the input device name to the beginning of the message.

### Example of the /*MESSAGE statement

```
/*MESSAGE CALL DEPT 58 WHEN PAYROLL JOB IS FINISHED--EX.1946
```

JES2 sends this message to the operator console when the job is read in.

## /*NETACCT statement

***Purpose:*** Use the /*NETACCT statement to specify an account number that is available to all the nodes in a network. JES2 uses the account number as is or translates it to local account numbers.

### Syntax

```
/*NETACCT network-account-number
```

The /*NETACCT statement consists of the characters /* in columns 1 and 2, NETACCT in columns 3 through 9, a blank in column 10, and the network account number starting in any column from 11 through 71. JES2 ignores columns 72 through 80.

### Parameter definition

**network-account-number**
Specifies the job's accounting number. The network-account-number is 1 through 8 alphanumeric characters.

### Defaults

If no /*NETACCT statement is specified, JES2 uses the local account number to search a table for the network account number.

### Overrides

If you supply both a /*NETACCT and a local account number, JES2 uses the local account number on the input node.

### Location in the JCL

Place the /*NETACCT statement after the JOB statement.

If a job contains more than one /*NETACCT statement, JES2 uses the network account number from the last statement.

JES2 ignores the /*NETACCT statement on any node other than the input node.

### Example of the /*NETACCT statement

```
/*NETACCT NETNUM10
```

JES2 transmits the network account number, NETNUM10, with the job to the destination node.

# /*NOTIFY statement

***Purpose:*** Use the /*NOTIFY statement to direct a job's notification messages to a user.

**Note:** The /*NOTIFY statement does not affect where the job is executed or where output is printed or punched.

## Syntax

```
/*NOTIFY    {nodename.userid }
            {nodename:userid }
            {nodename/userid }
            {nodename(userid)}
            {userid          }
```

The /*NOTIFY statement consists of the characters /* in columns 1 and 2, NOTIFY in columns 3 through 8, a blank in column 9, and a parameter starting in any column from 10 through 71. JES2 ignores columns 72 through 80.

Do not code a comma, a right parenthesis, or a blank character in the nodename or userid.

## Parameter definition

**nodename.userid**
**nodename:userid**
**nodename/userid**
**nodename(userid)**

Identifies a node and a TSO/E or VM userid at that node. The nodename is a symbolic name defined by the installation during initialization; nodename is 1 through 8 alphanumeric or national ($, #, @) characters. The userid must be defined at the node; userid for TSO/E is 1 through 7 alphanumeric or national ($, #, @) characters and for VM is 1 through 8 alphanumeric or national ($, #, @) characters.

**userid**

Identifies a TSO/E or VM user. The userid for TSO/E is 1 through 7 alphanumeric or national ($, #, @) characters and for VM is 1 through 8 alphanumeric or national ($, #, @) characters. When you specify only a userid, JES2 assumes that the userid is at the origin node.

The userid may also be a valid remote ID in the form Rnnnn or a destid for a remote. If the userid is specified as R1-R9999, JES2 assumes the notify message is intended for a remote and not a userid. If the remote is defined to the system or is less than the highest defined remote for your system, the notify message is queued to the remote. If the remote value is greater than the highest defined remote but less than the maximum allowed remote, the notify message is discarded. If the Rxxxx value specified is greater than R9999, JES2 considers that a TSO/E userid and not a remote ID.

A valid remote ID is only found when the node specification is for the local node. A valid specification can be in the form of NxRy.

## Overrides

The JES2 /*NOTIFY statement overrides the NOTIFY parameter on the JOB statement.

## Location in the JCL

The /*NOTIFY statement directs the notification messages of the job in which it appears; place the /*NOTIFY statement after the JOB statement. Do not include the /*NOTIFY statement in an in-stream procedure.

## Examples of the NOTIFY statement

***Example 1***

```
/*NOTIFY VMNODE.VMUSER
```

JES2 sends notification messages to user VMUSER on node VMNODE.

***Example 2***

```
/*NOTIFY TSOUSER
```

JES2 sends notification messages to user TSOUSER on the job's origin node.

# /*OUTPUT statement

***Purpose:*** Use the /*OUTPUT statement to specify characteristics and options for one or more sysout data sets. This statement supplies processing options in addition to and in place of the options specified on the sysout DD statement.

**Note: You should use the OUTPUT JCL statement instead of the JES2 /*OUTPUT statement because of the OUTPUT JCL statement's enhanced output processing capabilities**.

## Syntax

```
/*OUTPUT code parameter[,parameter]...

The parameters are:

{BURST} =  {Y}
{B    }    {N}

{CHARS} =  {xxxx            }
{X    }    {(xxxx[,xxxx]...)}

{CKPTLNS}  =nnnnn
{E      }

{CKPTPGS}   =nnnnn
{P      }

{COMPACT}   =nn
{Z      }

{COPIES}  =  {nnn                                    }
{N     }     {(nnn[,(group-value[,group-value]...)])}

{COPYG} = {group-value                   }
{G    }   {(group-value[,group-value]...)}

{DEST} = {destination                    }         destination is:
{D   }   {(destination[,destination]...)}
                                                      {ANYLOCAL        }
                                                      {LOCAL           }
{FCB} =xxxx                                           {name            }
{C  }                                                 {Nnnnn           }
                                                      {NnnRmmmm        }
{FLASH} = {overlay-name          }                    {NnnnRmmm        }
{O    }   {(overlay-name[,count])}                    {NnnnnRmm        }
          {NONE                  }                    {nodename.userid }
                                                      {nodename:userid }
{FLASHC} =count                                       {nodename/userid }
{Q     }                                              {nodename(userid)}
                                                      {Rnnnn           }
{FORMS} = {xxxx}                                      {RMnnnn          }
{F    }   {STD }                                      {RMTnnnn         }
                                                      {Unnnn           }
                                                      {Userid          }
```

```
{{INDEX |I} =nn}
{{LINDEX|L} =nn}

{LINECT}  =nnn
{K     }

{MODIFY} = {module-name        }
{Y     }   {(module-name[,trc])}

{MODTRC} =trc
{M     }

{UCS} =xxxx
{T  }
```

The /*OUTPUT statement consists of the characters /* in columns 1 and 2, OUTPUT in columns 3 through 8, a blank in column 9, a code beginning in column 10, followed by a blank and the keyword parameters. JES2 ignores columns 72 through 80.

An * in column 10 indicates that this /*OUTPUT statement is a continuation of the previous /*OUTPUT statement: JES2 treats it as a continuation, even through the previous /*OUTPUT statement does not immediately precede the continuation.

Do not specify * in column 10 on the first /*OUTPUT statement in a job.

## Parameter definition

**code**
Identifies the /*OUTPUT statement. The code is 1 through 4 alphanumeric characters. To refer to a /*OUTPUT statement, the DD statement SYSOUT parameter must specify this code in its code-name subparameter. The referenced /*OUTPUT statement specifies processing options for the sysout data set defined in the referencing DD statement.

A code of * indicates that this /*OUTPUT statement is a continuation of the previous /*OUTPUT statement.

**Note:** If you specify the code-name subparameter on a DD statement SYSOUT parameter in a job or job step that contains a default OUTPUT JCL statement, JES2 uses the default OUTPUT JCL statement instead of the reference to the /*OUTPUT statement.

If more than one /*OUTPUT statement has the same code starting in column 10, JES2 uses the parameters from only the first /*OUTPUT statement.

**BURST=Y**
**BURST=N**
Indicates the default burst characteristic of all sysout data sets that JES2 produces for this job. BURST applies only when the data set is directed to a 3800 Printing Subsystem equipped with a burster-trimmer-stacker.

**Y** Requests that the 3800 output is to be burst into separate sheets.

**N** Requests that the 3800 output is to be in a continuous fanfold.

**CHARS=xxxx**
**CHARS=(xxxx[,xxxx]...)**
Names a font for all output that JES2 prints on an AFP printer in this job. The xxxx is 1 through 4 alphanumeric or national ($, #, @) characters. Code one to four names.

**CKPTLNS=nnnnn**
Specifies the maximum number of lines or cards contained in a logical page. The nnnnn is 1 through 5 decimal numbers from 0 through 32,767 for printers and 1 through 32,767 for punches. The default is specified in the JES2 initialization parameter for the device.

**CKPTPGS=nnnnn**
Specifies the number of logical pages to be printed before the next checkpoint is taken. The nnnnn is 1 through 5 decimal numbers from 1 through 32,767. The default is specified in the JES2 initialization parameter for the device.

**COMPACT=nn**
Specifies a compaction table for JES2 to use when sending this sysout data set, which must be a systems network architecture (SNA) data set, to a SNA remote terminal.

**Note:** The COMPACT parameter has no effect on compaction for NJE sessions; it applies only to SNA RJE sessions.

**COPIES=nnn**
**COPIES=(nnn[,(group-value[,group-value]...)])**
Specifies how many copies of the sysout data set are to be printed in page sequence order, or from an AFP printer, grouped by page.

If you route a job that has a COPIES parameter, the parameter will be used only if the receiving node is a JES2 node.

**nnn**
Specifies how many copies of the sysout data set are to be printed; each copy will be in page sequence order. The nnn is 1 through 3 decimal numbers from 1 through 255, subject to an installation-specified limit. For a data set printed on an AFP printer, JES2 ignores nnn if any group values are specified.

If you incorrectly code the nnn parameter of COPIES, JES2 terminates the JOB.

**group-value**
Specifies how many copies of each page are to be printed before the next page is printed. Each group-value is 1 through 3 decimal numbers from 1 through 255. You can code a maximum of eight group-values. Their sum must not exceed 255 or the installation-specified limit. The total copies of each page equals the sum of the group-values.

**Note:** This subparameter is valid only for output processed by PSF. For PSF-processed output, this subparameter overrides the nnn subparameter. The group-value subparameter of the COPIES parameter overrides the group-value subparameter of the COPYG parameter.

The following are not valid:

- A null group-value, for example, COPIES=(5(,)) or COPIES=(5,)
- A zero group-value, for example, COPIES=(5,(1,0,4))
- A null within a list of group-values, for example, COPIES=(5,(1,,4))

**COPYG=group-value**
**COPYG=(group-value[,group-value]...)**
Specifies how many copies of each page are to be printed before the next page is printed. Each group-value is 1 through 3 decimal numbers from 1 through 255. You can code a maximum of eight group-values. Their sum must not exceed 255. The total copies of each page equals the sum of the group-values.

**Note:** This parameter applies only for output processed by PSF. If you code COPYG and JES2 prints the data set on an impact printer, JES2 ignores COPYG. The group-value subparameter of the COPIES parameter overrides the group-value subparameter of the COPYG parameter.

**DEST=destination**
**DEST=(destination[,destination]...)**
**Note:** JES2 initialization statements determine whether or not the node name is

required when coding a userid. See your system programmer for information regarding how routings will be interpreted by JES2.

Specifies one to four different destinations for the sysout data set. The destination subparameters follow:

**ANYLOCAL**
**LOCAL**
Indicates a local node on a local device.

**name**
Identifies a local or remote device by a symbolic name defined by the installation during JES2 initialization. The name is 1 through 8 alphanumeric or national ($, #, @) characters.

**Nnnnn**
Identifies a node. nnnn is 1 through 4 decimal numbers from 1 through 1000. For example, N0103.

**NnnRmmmm**
**NnnnRmmm**
**NnnnnRmm**
Identifies a node and a remote work station connected to the node. The node number, indicated in the format by n, is 1 through 4 decimal numbers from 1 through 1000. The remote work station number, indicated in the format by m, is 1 through 4 decimal numbers from 1 through 9999. Do not code leading zeros in n or m. The maximum number of digits for n and m combined cannot exceed six.

**Note:** NnnR0 is equivalent to LOCAL specified at node Nn.

**nodename.userid**
**nodename:userid**
**nodename/userid**
**nodename(userid)**
Identifies a destination node and a TSO/E or VM userid at that node. Use this parameter to route a sysout data set between JES2 nodes and non-JES2 nodes. The nodename is a symbolic name defined by the installation during initialization; nodename is 1 through 8 alphanumeric or national ($, #, @) characters. The userid must be defined at the node; userid for TSO/E is 1 through 7 alphanumeric or national ($, #, @) characters and for VM is 1 through 8 alphanumeric or national ($, #, @) characters.

Use the form **nodename.userid** to specify up to four destinations using continuation statements. The continuation statement must contain the characters /* in columns 1 and 2, OUTPUT in columns 3 through 8, a blank in column 9, an * in or following column 10, followed by one or more blanks, and the characters DEST= with the specified destinations. For example:

```
/*OUTPUT ABCD DEST=(POK.USER27,NYC.USER31)
/*OUTPUT *  DEST=(BOCA.USER58,STL.USER22)
```

Use the form **nodename.userid** to send the output to the VM user's virtual reader.

**Rnnnn**
**RMnnnn**

**RMTnnnn**
Identifies a remote terminal. nnnn is 1 through 4 decimal numbers from 1 through 9999. Note that with remote pooling, the installation may translate this route code to another route code.

If you send a job to execute at a remote node and the job has a ROUTE PRINT RMTnnnn statement, JES2 returns the output to RMTnnnn at the node of origin. For JES2 to print the output at RMTnnnn at the executing node, code DEST=NnnnRmmm on an OUTPUT JCL statement or sysout DD statement.

**Note:** R0 indicates any local device.

**Unnnn**
Identifies a local terminal with special routing. nnnn is 1 through 4 decimal numbers from 1 through 9999.

If you send a job to execute and the job has a ROUTE PRINT Unnnn statement, JES2 returns the output to Unnnn at the node of origin.

**Userid**
Identifies a userid at the local node.

**FCB=xxxx**
Identifies the forms control buffer (FCB) image JES2 is to use to guide printing of the sysout data set. The xxxx is 1 through 4 alphanumeric or national ($, #, @) characters and is the last characters of a SYS1.IMAGELIB member name:

- FCB2xxxx member, for a 3211 Printer, a 3203 Printer Model 5, or a printer supported by systems network architecture (SNA).
- FCB3xxxx member, for a 3800 Printing Subsystem.
- FCB4xxxx member, for a 4248 Printer.

IBM provides two standard FCB images. Code STD1 or STD2 only to request them.

- STD1, which specifies 6 lines per inch on an 8.5-inch-long form. (3211 and 3203-5 only)
- STD2, which specifies 6 lines per inch on an 11-inch-long form. (3211 and 3203-5 only)

If the printer on which JES2 is to print the data set does not have the forms control buffer feature, JES2 sends the operator a message to mount the proper carriage control tape.

**FLASH=overlay-name**
**FLASH=(overlay-name[,count])**
**FLASH=NONE**
Identifies the forms overlay to be used in printing the sysout data set on a 3800 Printing Subsystem and, optionally, specifies the number of copies on which the forms overlay is to be printed.

**overlay-name**
Identifies the forms overlay frame that the operator is to insert into the printer before printing begins. The name is 1 through 4 alphanumeric or national ($, #, @) characters.

Do not omit the overlay-name. The count subparameter is optional. If you omit it, you can omit the parentheses. However, if you omit it, you must not code it as a null; for example, FLASH=(ABCD,) is invalid.

Before printing starts, JES2 does not verify that the operator inserted the correct forms overlay frame for flashing.

**count**
Specifies the number, 1 through 255, of copies that JES2 is to flash with the overlay, beginning with the first copy printed.

JES2 determines the maximum number of copies to flash with the forms overlay by the value of nnn or the group-value total on the COPIES parameter. If the FLASH count value is greater than the value from the COPIES parameter, JES2 prints with the forms overlay the lower value.

The count subparameter of the FLASH parameter overrides the count value of the FLASHC parameter.

**NONE**
Suppresses flashing for this sysout data set.

**Defaults**: If you omit this parameter and did not specify FLASH on the DD statement or FLASHC on the /*OUTPUT statement, JES2 uses the default specified at JES2 initialization.

If you specify an overlay-name without specifying a count, JES2 flashes all copies. That is, the default for count is 255. If you specify 0 for count, JES2 also flashes all copies.

**FLASHC=count**
Specifies the number, 0 through 255, of copies that JES2 is to flash with the overlay, beginning with the first copy printed.

**Note:** For the 3800 printer, if you specify FLASH and omit FLASHC, JES2 flashes all copies.

The count subparameter of the FLASH parameter overrides the count value of the FLASHC parameter.

**FORMS=xxxx**
**FORMS=STD**
Identifies the forms on which JES2 is to print or punch the sysout data set.

**xxxx**
Identifies the print or punch forms. form-name is 1 through 4 alphanumeric or national ($, #, @) characters.

**STD**
Indicates that JES2 is to use the default specified at JES2 initialization.

**INDEX=nn**
Sets the left margin for output on a 3211 Printer with the indexing feature. The width of the print line is reduced by the INDEX parameter value. The nn specifies how many print positions the left margin on the 3211 output is to be indented. nn is a decimal number from 1 through 31. n=1 indicates flush-left; n=2 through n=31 indent the print line by n-1 positions.

JES2 ignores the INDEX parameter if the printer is not a 3211 with the indexing feature.

**Note:** INDEX and LINDEX are mutually exclusive; if you code both, JES2 uses the value you specified in INDEX.

**LINDEX=nn**
Sets the right margin for output on a 3211 Printer with the indexing feature.

The width of the print line is reduced by the LINDEX parameter value. The nn specifies how many print positions the right margin on 3211 output is to be moved in from the full page width. nn is a decimal number from 1 through 31. n=1 indicates flush-right; n=2 through n=31 move the right margin over by n-1 positions.

JES2 ignores the LINDEX parameter on all printers except the 3211 with the indexing feature.

**Note:** INDEX and LINDEX are mutually exclusive; if you code both, JES2 uses the value you specified in INDEX.

**LINECT=nnn**

Specifies the maximum number of lines JES2 is to print on each output page. The nnn is a number from 0 through 255.

Specify LINECT=0 to keep JES2 from starting a new page when the number of lines exceeds the JES2 initialization parameter.

If you code LINECT on the /*OUTPUT statement, it overrides the LINECT value on the /*JOBPARM statement and the linect value in the accounting information parameter of the JOB statement.

If the LINECT parameter is omitted from the /*OUTPUT statement, JES2 obtains the value from one of the following sources, in order:

1. The LINECT parameter on the /*JOBPARM statement.
2. The linect field of the accounting information parameter on the JOB statement.
3. The installation default specified at JES2 initialization.

**MODIFY=module-name**
**MODIFY=(module-name[,trc])**

Specifies a copy-modification module that tells JES2 how to print the sysout data set on a 3800 Printing Subsystem. The module can specify legends, column headings, blanks, and where and on which copies the data is to be printed. The module is defined and stored in SYS1.IMAGELIB using the IEBIMAGE utility program.

**module-name**

Identifies a copy-modification module in SYS1.IMAGELIB. The module-name is 1 through 4 alphanumeric or national ($, #, @) characters.

Do not omit the module-name.

**trc**

Identifies which table-name in the CHARS parameter is to be used. This **table reference character** is 0 for the first table-name specified, 1 for the second, 2 for the third, or 3 for the fourth.

If you do not specify **trc**, the default is 0. If the trc value is greater than the number of table-names in the CHARS parameter, JES2 uses the first table named in the CHARS parameter.

The trc subparameter is optional. If you omit it, you can omit the parentheses. However, if you omit it, you must not code it as a null; for example, MODIFY=(TAB1,) is invalid. If you omit the trc subparameter, JES2 uses the first table-name.

The trc subparameter of the MODIFY parameter overrides the trc subparameter of the MODTRC parameter.

**MODTRC=trc**
Identifies which table-name in the CHARS parameter is to be used. This **table reference character** is 0 for the first table-name specified, 1 for the second, 2 for the third, or 3 for the fourth.

If you do not specify **trc**, the default is 0. If the trc value is greater than the number of table-names in the CHARS parameter, JES2 uses the first table named in the CHARS parameter.

The trc subparameter of the MODIFY parameter overrides the trc subparameter of the MODTRC parameter.

**UCS=xxxx**
Identifies the universal character set (UCS) image JES2 is to use in printing the sysout data set. The xxxx is 1 through 4 alphanumeric or national ($, #, @) characters. See Table 18 on page 291 for IBM standard special character set codes.

## Overrides

- /*OUTPUT statement parameters override all equivalent DD statement parameters.
- If a /*OUTPUT statement contains duplicate parameters, the last parameter overrides all preceding duplicates, except for the DEST parameter.
- Any parameter coded on subsequent /*OUTPUT statements overrides the same parameter on previous /*OUTPUT statements.
- JES2 adds any parameter you code on subsequent /*OUTPUT statements that you did not code on previous /*OUTPUT statements to the previous /*OUTPUT statement.
- If you code LINECT on the /*OUTPUT statement, it overrides the LINECT value on the /*JOBPARM statement and the linect value in the accounting information parameter of the JOB statement.

## Relationship to other control statements

- JES2 processes /*OUTPUT statements placed after a /*ROUTE XEQ statement at the execution node only.
- JES2 processes /*OUTPUT statements placed before a /*ROUTE XEQ statement at both the input node and the execution node.

## Location in the JCL

Place the /*OUTPUT statement after the JOB statement. Do not include the /*OUTPUT statement in an instream procedure.

## Example of the /*OUTPUT statement

```
/*OUTPUT ABCD COPIES=6,COPYG=(1,2,3),DEST=RMT23
```

This statement refers to all sysout data sets defined by a DD statement that specifies SYSOUT=(c,,ABCD). Six copies of each page of output are printed. If the printer is an AFP printer, first one copy of each page is printed, then two copies of each page, and finally, three copies of each page. If the printer is not an AFP printer, COPYG is ignored and six copies of the entire data set are printed. The output is sent to remote terminal 23.

# /*PRIORITY statement

***Purpose:*** Use the /*PRIORITY statement to assign a selection priority for your job. Within a job class, a job with a higher priority is selected for execution sooner.

**Note:** The /*PRIORITY statement is ignored for a started task.

In a JES2 system, there are a number of factors that determine the order in which a particular job is selected for execution. Therefore, you cannot be assured that job priority (based on the PRTY you assign a job), job class, or the order of job submission will guarantee that the jobs will execute in a particular order. If you need to submit jobs in a specific order, contact your JES2 system programmer for advice based on how your system honors such requests. (*z/OS JES2 Initialization and Tuning Guide* provides JES2 system programmer procedures concerning job queuing and how to control job execution sequence.)

## Syntax

```
/*PRIORITY p
```

The /*PRIORITY statement consists of the characters /* in columns 1 and 2, PRIORITY in columns 3 through 10, a blank in column 11, and the priority starting in any column from 12 through 71. JES2 ignores columns 72 through 80.

## Parameter definition

**p** Requests a priority. The p is 1 or 2 decimal numbers from 0 through 15. The highest priority is 15.

Follow your installation's rules in coding a priority.

## Overrides

A priority specified on a /*PRIORITY statement overrides a priority specified in the PRTY parameter on a JOB statement.

## Relationship to other control statements

The system derives the priority from the following, in override order:

1. JES2 /*PRIORITY statement.
2. The PRTY parameter on the JOB statement.
3. The accounting information on a /*JOBPARM statement.
4. The accounting information on the JOB statement.
5. An installation default specified at JES2 initialization.

## Location in the JCL

The /*PRIORITY statement must immediately precede the JOB statement. If not, or if **p** is not a number from 0 through 15, JES2 ignores the /*PRIORITY statement and flushes the input stream until the next JOB statement or another /*PRIORITY statement.

In a JES2 network, IBM recommends that the /*PRIORITY statement immediately follow the /*XMIT statement. If you code any other statement between /*XMIT and JOB, JES2 will ignore the statement and issue an error message.

### Example of the PRIORITY statement

```
/*PRIORITY 7
```

This statement assigns a job queue selection priority of 7. This value has meaning only in relation to other jobs in the system.

## /*ROUTE statement

***Purpose:*** Use the /*ROUTE statement to specify the destination of sysout data sets that are not routed by a DEST parameter or to identify the network node where the job is to execute.

**Note:** Do not specify the /*ROUTE XEQ statement for a started task; if /*ROUTE XEQ is specified, JES2 fails the job.

### Syntax

```
/*ROUTE  {PRINT}     {ANYLOCAL        }
         {PUNCH}     {LOCAL           }
                     {name            }
                     {Nnnnn           }
                     {NnnRmmmm        }
                     {NnnnRmmm        }
                     {NnnnnRmm        }
                     {nodename.userid }
                     {nodename:userid }
                     {nodename/userid }
                     {nodename(userid)}
                     {Rnnnn           }
                     {RMnnnn          }
                     {RMTnnnn         }
                     {Unnnn           }
                     {Userid          }

/*ROUTE XEQ    {name               }
               {Nnnnn              }
               {nodename.vmguestid }
               {nodename:vmguestid }
               {nodename/vmguestid }
               {nodename(vmguestid)}
```

The /*ROUTE statement consists of the characters /* in columns 1 and 2; ROUTE in columns 3 through 7; at least one blank followed by PRINT, PUNCH, or XEQ; at least one blank followed by one of the destinations or nodes; and at least one blank before column 72. JES2 ignores columns 72 through 80.

Code only one destination or node on each /*ROUTE statement.

### Parameter definition

**PRINT**
Requests that JES2 route the job's sysout data sets that are printed.

**PUNCH**
Requests that JES2 route the job's sysout data sets that are punched.

**XEQ**
Requests that JES2 route the job to a network node for execution.

**ANYLOCAL**

**LOCAL**
Indicates a local node on a local device.

**name**
Identifies a local or remote device or node by a symbolic name defined by the installation with the JES2 DESTID initialization statement. The name is 1 through 8 alphanumeric or national ($, #, @) characters.

**Nnnnn**
Identifies a node. nnnn is 1 through 4 decimal numbers from 1 through 1000. For example, N0103.

**NnnRmmmm**
**NnnnRmmm**
**NnnnnRmm**
Identifies a node and a remote work station connected to the node. The node number, indicated in the format by n, is 1 through 4 decimal numbers from 1 through 1000. The remote work station number, indicated in the format by m, is 1 through 4 decimal numbers from 1 through 9999. Do not code leading zeros in n or m. The maximum number of digits for n and m combined cannot exceed six.

**Note:** NnnR0 is equivalent to LOCAL specified at node Nn.

**nodename.userid**
**nodename:userid**
**nodename/userid**
**nodename(userid)**
Identifies a node and a VM or TSO/E userid, a remote workstation, or a symbolic name defined at the destination node. The node is a symbolic name defined by the installation during initialization; nodename is 1 through 8 alphanumeric or national ($, #, @) characters. The userid must be defined at the node; userid is 1 through 8 alphanumeric or national ($, #, @) characters.

A userid requires a node; therefore, code nodename.userid. You **cannot** code a userid without a nodename.

If you specify a TSO/E userid, do not specify a nodename that is the same as the origin node.

**Note:** If a data set is queued for transmission and an operator changes its destination, the userid portion of the routing is lost.

**Rnnnn**
**RMnnnn**
**RMTnnnn**
Identifies a remote terminal. nnnn is 1 through 4 decimal numbers from 1 through 9999. Note that with remote pooling, the installation may translate this route code to another route code.

If you send a job to execute at a remote node and the job has a ROUTE PRINT RMTnnnn statement, JES2 returns the output to RMTnnnn at the node of origin. For JES2 to print the output at RMTnnnn at the executing node, code DEST=NnnnRmmm on an OUTPUT JCL statement or sysout DD statement.

**Note:** R0 indicates any local device.

**Unnnn**
Identifies a local terminal with special routing. nnnn is 1 through 4 decimal numbers from 1 through 9999.

If you send a job to execute and the job has a ROUTE PRINT Unnnn statement, JES2 returns the output to Unnnn at the node of origin.

**Userid**
Identifies a userid at the local node.

**Note:** JES2 initialization statements determine whether or not the node name is required when coding a userid. See your System Programmer for information regarding how routings will be interpreted by JES2.

**nodename.vmguestid**
**nodename:vmguestid**
**nodename/vmguestid**
**nodename(vmguestid)**
Identifies the network node where the job is to execute. The nodename identifies an MVS JES2 system, an MVS JES3 system, a VSE POWER node, or a VM system. If nodename specifies the local node, the job executes locally. The nodename is 1 through 8 alphanumeric, national ($, #, @), or special characters specified during JES2 initialization.

The vmguestid identifies a guest system running in a virtual machine (VM), for example, an MVS system running under VM. Do not specify a work station or terminal in this parameter.

## Location in the JCL

Place the /*ROUTE statement after the JOB statement and either before or after the EXEC statements. Place a /*ROUTE XEQ statement before all DD * or DD DATA statements in the job.

## Processing of /*ROUTE statements

- The system ignores the /*ROUTE XEQ statement for NJE devices.
- If you do not specify a node on the /*ROUTE PRINT or PUNCH statement, printing or punching occurs at the input node.
- JES2 processes /*ROUTE XEQ statements on the input node only.
- When a /*ROUTE PRINT or PUNCH statement follows a /*ROUTE XEQ statement, JES2 processes the /*ROUTE PRINT or PUNCH statement on the execution node only. However, printing or punching occurs at the node specified on the /*ROUTE PRINT or PUNCH statement.
- When a /*ROUTE PRINT or PUNCH statement precedes a /*ROUTE XEQ statement, JES2 processes the /*ROUTE PRINT or PUNCH statement on both the input and execution nodes. However, printing or punching occurs at the node specified on the /*ROUTE PRINT or PUNCH statement.

## Multiple /*ROUTE statements

JES2 uses the last /*ROUTE statement of each category, if a job contains more than one /*ROUTE PRINT or PUNCH or XEQ statement.

## Examples of the ROUTE statement

***Example 1***

```
/*ROUTE  PRINT RMT6
```

This statement sends the printed output to remote terminal 6.

***Example 2***

```
/*ROUTE  PUNCH PUN2
```

This statement sends the punched output to device PUN2, which was identified to the system during initialization.

***Example 3***

```
//JOBB   JOB   ...
/*ROUTE  XEQ   DENVER
//STEP1  EXEC  ...
          .
          .
```

This statement sends the job to the node named DENVER for execution. The entire job is scanned for JCL errors on the input system before it is transmitted to the target system. The entire job is transmitted, which includes the JOBB JOB statement. Options on the JOBB JOB statement apply to both the input and target system.

***Example 4***

```
 //PAYROLL JOB  JONES,CLASS=C
 /*ROUTE  XEQ WSC
 /*JOBPARM  L=60,R=4222,T=50,P=PROC03,N=5
 //EXEC PROC=PROC489
 /*ROUTE  XEQ POK
```

These statements specify multiple routes and could cause JECL statements to be ignored.

# /*SETUP statement

***Purpose:*** Use the /*SETUP statement to identify volumes that the operator should mount before the job is executed. When the job enters the system, JES2 issues a message to the operator console, asking the operator to mount the identified volumes. JES2 then places the job in hold status until the operator mounts the volumes and releases the job.

**Note:** Do not specify this statement for a started task; if /*SETUP is specified, JES2 fails the job.

## Syntax

```
/*SETUP  serial-number[,serial-number]...
```

The /*SETUP statement consists of the characters /* in columns 1 and 2, SETUP in columns 3 through 7, a blank in column 10, and the volume serial number(s) starting in any column from 11 through 71. JES2 ignores columns 72 through 80.

Do not continue the /*SETUP statement; code as many /*SETUP statements as necessary.

## Parameter definition

**`serial-number`**
Identifies by serial number the volume(s). A volume serial number is 1 through 6 alphanumeric, national ($, #, @), or special characters; enclose a serial number that contains special characters, other than hyphens, in apostrophes. If the number is shorter than 6 characters, it is padded with trailing blanks.

## Location in the JCL

Place all /*SETUP statements after the JOB statement and before the first EXEC statement.

To prevent JES2 from requesting mounting of volumes on a node other than the node of execution, the /*SETUP statement should follow any /*ROUTE XEQ or /*XEQ statement. If JES2 processes the /*SETUP statement before processing a /*ROUTE XEQ or /*XEQ statement, JES2 requests the setup on both the input and execution nodes.

## Example of the /*SETUP statement

```
/*SETUP  666321,149658
```

This statement requests that volumes 666321 and 149658 be mounted for the job.

# /*SIGNOFF statement

***Purpose:*** Use the /*SIGNOFF statement to tell JES2 to end a remote job stream processing session. At the completion of the current print and/or punch streams, JES2 disconnects the remote work station from the system. If JES2 is reading jobs from the station when the output completes, JES2 disconnects the remote station when the input is completed.

**Note:** The remote terminal access processor processes the /*SIGNOFF statement if it appears in a job stream.

Both systems network architecture (SNA) and binary synchronous communication (BSC) remote work stations can use the /*SIGNOFF statement. SNA remote stations can also use the LOGOFF command to end a session with JES2. The LOGOFF command has some options that the /*SIGNOFF statement does not provide.

- /*SIGNOFF# = LOGOFF TYPE(COND) Conditional Disconnect
- LOGOFF = LOGOFF TYPE(UNCOND) Unconditional Disconnect

***References:*** For information on the LOGOFF command, see *z/OS V2R1.0 Communications Server: SNA Programming*.

## Syntax

| /*SIGNOFF |
|---|
| The /*SIGNOFF statement consists of the characters /* in columns 1 and 2, SIGNOFF in columns 3 through 9, and blanks in columns 10 through 80. |

## Location in the JCL

The /*SIGNOFF statement can appear anywhere in a local input stream or an input stream from a SNA or BSC remote work station.

## Example of the /*SIGNOFF statement

```
/*SIGNOFF
```

This statement requests that JES2 terminate a remote job stream processing session.

# /*SIGNON statement

***Purpose:*** Use the /*SIGNON statement to tell JES2 to begin a remote job stream processing session. For non-multi-leaving remote stations, the terminal transmits the /*SIGNON statement alone as part of the initial connection process.

**Note:** The remote terminal access processor processes the /*SIGNON statement if it appears in a job stream. When the terminal access processor processes the /*SIGNON statement, the line being processed is restarted.

Systems network architecture (SNA) remote work stations must use the LOGON command instead of the /*SIGNON statement to notify JES2 of a connection request.

***References:*** For information on the LOGON command, see *z/OS V2R1.0 Communications Server: SNA Programming*.

## Syntax

```
/*SIGNON      {REMOTEnnn}   [password1]  [new-password]  [password2]
              {RMTnnnn  }
              {RMnnnn   }
              {Rnnnn    }
              {NxxRnnnn }
              {dest-name}
```

The /*SIGNON statement consists of the following. Note that all the fields in this statement must appear in fixed locations.

| Column | Contents |
|---|---|
| **1-8** | /*SIGNON |
| **16-24** | REMOTEnnn, RMTnnnn, RMnnnn, Rnnnn, NxxRnnnn, or dest-name beginning in 16 |
| **25-32** | password1, beginning in 25 |
| **35-42** | new-password, beginning in 35 |
| **73-80** | password2, beginning in 73 |

## Parameter definition

**REMOTEnnn**
Specifies the identification number assigned to the remote station asking to sign on. The nnn is 1 through 3 decimal numbers.

Code REMOTEnnn with the same characters as RMTnnn on the /*ROUTE statement. If you code REMOTEnnn on the /*SIGNON statement, you are restricted to coding RMTnnn with only three numbers on the /*ROUTE statement.

**RMTnnnn**
**RMnnnn**
**Rnnnn**
Specifies the identification number assigned to the remote station. nnnn is one through four decimal numbers.

**NxxRnnnn**
Specifies the node number in the NJE network and the identification number assigned to the remote station. Nxx must specify the node to which the remote work station is connected. xx is 1 through 1000. nnnn is 1 through 4 decimal numbers. xx plus nnnn cannot exceed 6 numbers.

**dest-name**
Specifies the name (one through eight characters) that you use to refer to the JES2-defined destination. The dest-name must be defined as a remote work station on the system to which the terminal is connected.

**password1**
Specifies the password assigned to a nondedicated connection that allows the remote station access to JES2 for remote job stream processing. The installation assigns this password during system initialization. The operator can change or delete this password with the $T command.

**new-password**
Specifies a new password for the remote job entry (RJE) station that is signing on. If the installation is controlling the sign on with JES2 password support instead of RACF, the new password is ignored.

**password2**
Specifies the current password for the remote station that is signing on; this password identifies the remote station as a valid remote job entry (RJE) station. This parameter is assigned by either RACF, a JES2 initialization parameter (if JES2 password support is used), or the $T command.

## Location in the JCL

Place the /*SIGNON statement at the start of an input stream to be transmitted from a remote work station. The terminal transmits the /*SIGNON statement alone as part of the initial connection process.

Place the /*SIGNON statement at the end of the JES2/RTP input stream for multi-leaving remote stations.

## Examples of the /*SIGNON statement

*Example 1*

```
/*SIGNON       REMOTE123LINEPSWD
```

This statement requests that remote station 123 begin a remote job stream processing session. LINEPSWD, beginning in column 25, is the password assigned to the nondedicated connection.

*Example 2*

```
/*SIGNON       RMT1000  LINEPSWD
```

This statement requests that remote station 1000 begin a remote job stream processing session. LINEPSWD, beginning in column 25, is the password assigned to the non-dedicated connection.

*Example 3*

```
/*SIGNON       RMT1000  LINEPSWD  PSWDNEW                          PSWD2
```

This statement requests that remote station 1000 begin a remote job stream processing session. LINEPSWD, beginning in column 25, is the password assigned

to the nondedicated connection. PSWD2, beginning in column 73, is the password assigned to the remote station 1000. PSWDNEW, beginning in column 35, is the new password to be assigned to remote station 1000.

***Example 4***

```
/*SIGNON       N11R123  LINEPSWD
```

This statement requests that remote station 123 at node 11 begin a remote job stream processing session. LINEPSWD, beginning in column 25, is the password assigned to the switched connection.

# /*XEQ statement

***Purpose:*** Use the /*XEQ statement to identify the network node where the job is to execute. It performs the same function as the /*ROUTE XEQ statement.

**Note:**

1. Do not specify this statement for a started task; if /*XEQ is specified, JES2 fails the job.
2. The XEQ statement is ignored for NJE devices.

## Syntax

```
/*XEQ   {Nnnnn               }
       {nodename[.vmguestid]}
       {name                }
```

The /*XEQ statement consists of the characters /* in columns 1 and 2, XEQ in columns 3 through 5, a blank in column 6, and a node starting in any column starting with 7.

## Parameter definition

**Nnnnn**
Identifies a node. nnnn is 1 through 4 decimal numbers from 1 through 1000. For example, N0103.

**nodename**
Identifies the network node where the job is to execute. The nodename identifies an MVS JES2 system, an MVS JES3 system, a VSE POWER node, or a VM system. If nodename specifies the local node, the job executes locally. The nodename is 1 through 8 alphanumeric, national ($, #, @), or special characters specified during JES2 initialization.

**vmguestid**
Identifies a guest system running in a virtual machine (VM), for example, an MVS system running under VM. Do not specify a work station or terminal in this parameter.

**name**
Specifies the name (1 through 8 characters) that you use to refer to the JES2-defined destination. The name must be defined as a node and userid at the destination node.

## Location in the JCL

Place the /*XEQ statement after the JOB statement and either before or after the EXEC statements. Place a /*XEQ statement before all DD * or DD DATA statements in the job.

## Multiple /*XEQ statements

JES2 uses the last /*XEQ statement, if a job contains more than one /*XEQ statement.

## Example of the XEQ statement

```
//JOBB   JOB  ...
/*XEQ    ATLANTA
//STEP1  EXEC  ...
            .
            .
```

JES2 routes and executes this job on the node defined as ATLANTA. The entire job is transmitted, which includes the JOBB JOB statement. Options on the JOBB JOB statement apply to both the input and target system.

# /*XMIT statement

***Purpose:*** Use the /*XMIT statement to transmit records from a JES2 node to either another JES2 node or an eligible non-JES2 node, for example, a VM or JES3 node. JES2 does not process or check the records for JES2 validity. JES2 builds header and trailer records from information on the JOB statement immediately preceding the /*XMIT statement. Then JES2 transmits all records after the /*XMIT statement.

The records may consist of a job input stream or an in-stream DD * or DD DATA data set. If the records are in a job input stream and the destination node can process JCL (which means it is the ultimate node, not a store-and-forward node), the system will execute the transmitted input stream provided that: (a) the record immediately following the /*XMIT statement is a JOB statement valid at that node, and (b) the input stream consists only of the JCL and data for the one job headed by that JOB statement. The system will flush ALL jobs if an NJE (network job entry) receiver finds multiple JOB statements in the input stream.

The records end when JES2 finds one of the following:

- /* in the input stream
- The two-character delimiter specified by a DLM parameter on this /*XMIT statement
- The input stream runs out of records.
- If the records are being read from an internal reader, the internal reader is closed

**Note:** Do not specify this statement for a started task; if /*XMIT is specified, JES2 fails the job.

## Syntax

```
/*XMIT  {Nnnnn              }     [ DLM=xx]
       {nodename           }
       {nodename.userid    }
       {nodename:userid    }
       {nodename/userid    }
       {nodename(userid)   }
       {nodename.vmguestid }
       {nodename:vmguestid }
       {nodename/vmguestid }
       {nodename(vmguestid)}
       {name               }
```

The /*XMIT statement consists of the characters /* in columns 1 and 2, XMIT in columns 3 through 6, a blank in column 7, a nodename or node-number starting in any column starting with 8, and optionally followed, with an intervening blank, by a delimiter parameter.

Do not continue an /*XMIT statement.

## Parameter definition

**Nnnnn**
Identifies the destination node. nnnn is 1 through 4 decimal numbers from 1 through 1000. For example, N0103.

**nodename**
Identifies the destination node. The nodename identifies an MVS JES2 system, an MVS JES3 system, a VSE POWER node, or a VM system. The nodename is 1 through 8 alphanumeric, national ($, #, @), or special characters specified during JES2 initialization.

**userid**
Identifies a destination terminal or work station at the node. The userid must be defined at the node; userid for TSO/E is 1 through 7 alphanumeric or national ($, #, @) characters and for VM is 1 through 8 alphanumeric or national ($, #, @) characters.

**vmguestid**
Identifies a destination guest system running in a virtual machine (VM), for example, an MVS system running under VM. Do not specify a work station or terminal in this parameter.

**name**
Specifies the name (1 through 8 characters) that you use to refer to the JES2-defined destination. The name must be defined as a node and userid at the destination node.

**DLM=xx**
Specifies a two-character delimiter to terminate the data being transmitted.

Code any two characters for the delimiter. If the specified delimiter contains any special characters, enclose it in apostrophes. In this case, a special character is any character that is neither alphanumeric nor national ($, #, @).

Failing to code enclosing apostrophes produces unpredictable results.

If the delimiter contains an ampersand or an apostrophe, code each ampersand or apostrophe as two consecutive ampersands or apostrophes. Each pair of consecutive ampersands or apostrophes counts as one character.

If you specify a DLM parameter, you must terminate the transmitted records with the characters in the DLM parameter. The characters you assign as delimiters override any delimiter implied by the defaults.

The characters // are not valid delimiters unless specifically indicated by DLM=//.

## Defaults

For the end of the records to be transmitted, the default is /* in the input stream.

If you specify for DLM only one character or more than two characters, JES2 uses /*.

## Location in the JCL

Place the /*XMIT statement immediately after a JOB statement. If the records being transmitted are a job input stream, another JOB statement must follow the /*XMIT statement.

You can code only the /*PRIORITY statement between an /*XMIT statement and a JOB statement. If you code any other statement between /*XMIT and JOB, JES2 will ignore the statement and issue an error message.

Code only one /*XMIT statement in a job.

## Examples of the XMIT statement

***Example 1***

```
//JOBA  JOB  ...
/*XMIT  ATLANTA DLM=AA
         .
         records to be transmitted
         .
AA
```

JES2 transmits to the node ATLANTA all records following the /*XMIT statement up to the specified delimiter, AA.

***Example 2***

```
//JOBX  JOB  ...
/*XMIT  VMSYS1.MVS223
//JOBB  JOB  ...
         .
         job to be transmitted
         .
/*
```

JES2 transmits the JOBB job stream to the VM guest system, MVS223, running on node VMSYS1, which is a VM system. The job stream will be executed by the MVS223 system.

The information specified on the JOBX statement is processed on the submitting system and transmitted in the networking headers to the target system. The target system, if it is a JES2 node, uses the default routing in the network job header unless it is specifically overridden in the JCL for the transmitted job.

# Chapter 31. JES3 control statements

Code JES3 control statements with JCL statements to control the input and output processing of jobs. The rules for coding in Chapter 3, "Format of statements," on page 13, and Chapter 4, "Syntax of parameters," on page 19, apply to the JES3 control statements.

## Description

### Considerations for an APPC scheduling environment

JES3 control statements have no function in an APPC scheduling environment. If you code them, the system will ignore them, and they will appear as comments in the job listing.

### Considerations for started tasks

JES3 JECL statements are not supported for started tasks. Use of JECL statements will result in JES3 failing the job.

### Location in the JCL

Place JES3 control statements, except the command and //**PAUSE statements, after the JOB statement and its continuations. JES3 ignores JES3 control statements, except the command and //**PAUSE statements, that appear before the JOB statement or between continued JOB statements.

### Internal reader

Use the following control statements when submitting jobs to the internal reader. The internal reader is described in *z/OS MVS Programming: Assembler Services Guide*.

- /*DEL
- /*EOF

### Examples of JES3 control statements

The first example shows JES3 control statements in relation to each other and to JCL statements for a job entered from a remote work station. No actual job should require all of these statements.

The second example shows an ordinary job entered through the local input stream.

The following sections show the recommended syntax for each control statement, with examples. Note, however, that for some JES3 control statements (such as the //* MAIN statement) a single slash followed by an asterisk (/*), rather than two slashes and an asterisk (//*), will be processed as syntactically acceptable. Your installation may disallow this option by using the ALTJCL keyword parameter of the STANDARDS initialization statement. For further information see *z/OS JES3 Initialization and Tuning Reference*

***Example 1:***

© Copyright IBM Corp. 1988, 2015

```
//**MESSAGE,CN1,ENTER A START COMMAND FOR THIS JOB
//**PAUSE
//TEST1    JOB   ,,MSGCLASS=A
//*NETACCT PNAME=MAEBIRD,ACCT=2K14920
//*NET NETID=N1,NHOLD=0
//*PROCESS CI
//*PROCESS MAIN
//*PROCESS OUTSERV
//*DATASET DDNAME=STEP1.DD1
         .
         .
         data
         .
//*ENDDATASET
//*ENDPROCESS
//*OPERATOR THIS IS TEST JOB TEST1.
//*MAIN CLASS=C
//*FORMAT PR,DDNAME=STEP1.DD2,DEST=ANYLOCAL,COPIES=2
//*ROUTE XEQ NODE1
//FARJOB1  JOB   ,,MSGCLASS=A
//STEP1    EXEC  PGM=CHECKER
//DD1      DD    DSNAME=INPUT
//DD2      DD    SYSOUT=A
/*
```

***Example 2:***

```
//RUN2     JOB   ,,MSGCLASS=A
//*MAIN CLASS=B
//*FORMAT PR,DDNAME=STEPA.DD2,DEST=ANYLOCAL,COPIES=5
//STEPA    EXEC  PGM=WRITER
//DD1      DD    DSNAME=IN1,DISP=OLD,UNIT=3390,VOLUME=SER=MH2244
//DD2      DD    SYSOUT=A
/*
```

## JES3 control statement tracking

JES3 is instrumented to report the use of JES3 control statements (JECL) in jobs that have been submitted to the system. Occurrences of JES3 JECL statements in a job stream will be reported during input service using the Generic Tracker macro GTZTRACK. When GTZ tracking is enabled, JES3 will record GTZ data identifying the JES3 JECL statements found within a job stream. See JES3 control statement tracking in *z/OS MVS Diagnosis: Tools and Service Aids* for a description and the layout of the JES3 generated GTZ tracking records.

# JES3 command statement

***Purpose***

Use the command statement to enter a JES3 operator command through the input stream.

JES3 usually executes an in-stream command as soon as it is read. Therefore, the command will **not** be synchronized with the execution of any job or step in the input stream. To synchronize a command with job processing, tell the operator the commands you want and when they should be issued, then let the operator enter them from the console.

***References***

For more information on the command statement and the JES3 verbs and operands, see "Entering Commands through the Input Stream" in *z/OS JES3 Commands*.

## Syntax

```
//**command-verb[,operand]...
```

The JES3 command statement consists of the characters //** in columns 1 through 4, the command verb beginning in column 5, and, if the command requires operands, a comma followed by the operands up through column 72. JES3 ignores columns 73 through 80.

Do not continue command statements from one record to the next.

## Parameter definition

***command-verb**

Indicates one of the following JES3 commands. **Do not** specify a *DUMP or *RETURN command on a JES3 command statement.

| Command | Short Form |
|---|---|
| **CALL** | X |
| **CANCEL** | C |
| **DELAY** | D |
| **DISABLE** | H |
| **ENABLE** | N |
| **ERASE** | E |
| **FAIL** | |
| **FREE** | |
| **INQUIRY** | I |
| **MESSAGE** | Z |
| **MODIFY** | F |
| **RESTART** | R |
| **SEND** | T |
| **START** | S |

**SWITCH**

**VARY** V

**operand**
Specifies an operand that pertains to the command-verb.

## Location in the JCL

- Place JES3 command statements before the first JOB statement in the input stream, if you are also submitting jobs. JES3 treats any JES3 command statements that follow the JOB statement as comment statements.
- You may enter several command statements at one time.
- You may enter command statements through card, tape, or disk readers.
- You may place command statements as the first statements in an active card reader that you are restarting.
- You **may not** enter command statements through an internal reader (by issuing a TSO/E Submit command) or from another node.

## Examples of the command statement

***Example 1***

```
//**VARY,280,OFFLINE
//**V,281,OFFLINE
//**VARY,282,OFF

//**V,280-282,OFF
```

In this example, the first three statements each vary one device offline. Alternatively, the fourth statement varies all three devices offline. If you place these statements in card reader 01C, for example, and that card reader is currently not in use, the operator would enter through the operator console:

```
*X CR,IN=01C
```

***Example 2***

```
//**MESSAGE,CN1,OUTPUT FROM JOB X REQUIRES SPECIAL CONTROLS
```

This statement instructs the operator from a remote location. Place this statement before the first job in the input stream.

# //*DATASET statement

***Purpose:*** Use the //*DATASET statement to identify the beginning of an in-stream data set, which can contain JCL statements or data. (The //*ENDDATASET statement ends the in-stream data set.) The data set can be used as input to a dynamic support program (DSP), such as OUTSERV.

**Note:** Make sure the operator includes a **C** operand on the *CALL command for the reader that reads a job containing this statement if it contains a MODE=C parameter.

## Syntax

```
//*DATASET DDNAME=ddname[,parameter]...

The parameters are:

     MODE=   {E}
             {C}

     J=  {YES}
         {NO }

     CLASS=  {NO      }
             {MSGCLASS}
             {class   }
```

The //*DATASET statement consists of the characters //* in columns 1 through 3, DATASET in columns 4 through 10, a blank in column 11, and parameters in columns 12 through 72. JES3 ignores columns 73 through 80.

## Parameter definition

**DDNAME=ddname**
Specifies the name of the in-stream data set that follows the //*DATASET statement.

**MODE=E**
**MODE=C**
Defines the card-reading mode.

**E** Indicates that JES3 is to read the statements as EBCDIC with validity checking. E is the default if the MODE parameter is omitted.

**C** Indicates that JES3 is to read the statements in card image form, that is, in column binary or data mode 2.

MODE=C is not valid for jobs read from disk or tape, or for jobs submitted from remote work stations.

**J=YES**
**J=NO**
Indicates how JES3 is to recognize the end of the data set.

If you specify MODE=C, JES3 ignores the J parameter; therefore, use a //*ENDDATASET statement to end the data set

**YES**
Indicates that a //*ENDDATASET statement ends the data set. Specify YES when JOB statements appear in the data set.

**NO** Indicates that a JOB statement ends the data set. NO is the default if the J parameter is omitted, unless MODE=C is specified.

**CLASS=NO**
**CLASS=MSGCLASS**
**CLASS=class**
Identifies the output class JES3 is to use for the data set.

**NO** Indicates that the system is to assign an output class. If you omit the CLASS parameter, the default is NO.

**MSGCLASS**
Requests the output class in the MSGCLASS parameter on the JOB statement.

**class**
Specifies the output class.

## Location in the JCL

Place a //*DATASET statement immediately before the first record of an in-stream data set.

## Example of the //*DATASET statement

```
//*PROCESS OUTSERV
//*DATASET DDNAME=MYPRINT,J=YES
           .
           .
          data
           .
           .
//*ENDDATASET
//*FORMAT PR,DDNAME=MYPRINT,COPIES=5
//STEP1   EXEC ...
           .
           .
```

In this example, the //*DATASET statement marks the beginning of the in-stream data set MYPRINT. The //*FORMAT PR statement requests five copies of it. The //*ENDDATASET statement marks the end of the data set.

# //*ENDDATASET statement

***Purpose:*** Use the //*ENDDATASET statement to indicate the end of an in-stream data set that was begun with a //*DATASET statement.

## Syntax

| //*ENDDATASET |
|---|
| The //*ENDDATASET statement consists of the characters //* in columns 1 through 3 and ENDDATASET in columns 4 through 13. Columns 14 through 80 must be blank. |

## Location in the JCL

Place a //*ENDDATASET statement immediately after the last record of an in-stream data set that was begun with a //*DATASET statement.

## Example of the //*ENDDATASET statement

```
//*DATASET DDNAME=INFO,J=YES
           .
           .
           data
           .
           .
//*ENDDATASET
```

In this example, the //*ENDDATASET statement marks the end of the in-stream data set INFO.

# //*ENDPROCESS statement

***Purpose:*** Use the //*ENDPROCESS statement to indicate the end of a series of //*PROCESS statements in a job.

## Syntax

```
//*ENDPROCESS [comments]
```

The //*ENDPROCESS statement consists of the characters //* in columns 1 through 3, ENDPROCESS in columns 4 through 13, a blank in column 14, and, optionally, comments starting in any column beginning with 15. JES3 ignores columns 73 through 80.

## Location in the JCL

Place a //*ENDPROCESS statement immediately after the last //*PROCESS statement in a job. The //*ENDPROCESS statement is optional if a JCL statement follows the last //*PROCESS statement.

Do not place any //*PROCESS statements after the //*ENDPROCESS statement.

## Example of the //*ENDPROCESS statement

```
//*ENDPROCESS   END OF PROCESS STATEMENTS
```

# //*FORMAT PR statement

***Purpose:*** Use the //*FORMAT PR statement to specify to JES3 processing instructions for sysout data sets that are printed. These instructions permit special processing of sysout data sets, such as:

- Multiple destinations.
- Multiple copies of output with different attributes.
- Forced single or double space control.
- Printer overflow checking.

//*FORMAT PR statements can be either specific or nonspecific. A specific //*FORMAT PR statement contains a DDNAME parameter that specifies something other than a null value, such as DDNAME=ddname or DDNAME=JESYSMSG. A nonspecific //*FORMAT PR statement contains DDNAME= , with no value (null) specified for the DDNAME parameter.

You can code multiple specific //*FORMAT PR statements for a particular sysout data set to specify special requirements for different copies of the data set. In addition, you can code a //*FORMAT PU statement for the same sysout data set, thereby both printing and punching it.

You can also code multiple nonspecific //*FORMAT PR statements. In this case, the system produces only one copy of each data set, combining any parameter values specified on the statements. If you specify a given parameter on more than one of these statements, the system uses the parameter value specified on the last //*FORMAT PR statement containing that parameter.

**Note:** The //*FORMAT PR statement applies only to sysout data sets printed by JES3. The statement is ignored for data sets sent to a TSO/E userid or processed by an external writer.

***Reference:*** For examples of //*FORMAT statement processing on the JES3 hold queue and writer queue, see *z/OS JES3 Initialization and Tuning Guide*.

## Syntax

```
//*FORMAT  PR,DDNAME=   {ddname                       }   [,parameter]...
                        {stepname.ddname              }         ,parameter]...
                        {stepname.procstepname.ddname}
                        {JESYSMSG                     }
                        {JESJCL                       }
                        {JESMSGLG                     }

//*FORMAT PR,DDNAME=[,parameter]...

The parameters are:

{ CARRIAGE=  {carriage-tape-name} }
{            {6                 } }
{ FCB=  {image-name}               }
{       {6         }               }

 CHARS=  {STANDARD                    }
         {table-name                  }
         {(table-name[,table-name]...)}

 CHNSIZE=  {DS         }
           {(nnn[,mmm])}

 COMPACT=compaction-table-name

 CONTROL=  {PROGRAM}
           {SINGLE }
           {DOUBLE }
           {TRIPLE }

 COPIES=  {nnn                                }
          {(nnn,(group-value[,group-value]...))}
          {(group-value[,group-value]...)      }
```

```
DEST=  {ANYLOCAL              }
       {device-name           }
       {device-number         }
       {group-name            }
       {nodename[.remote]     }
       {(type[,device-name])  }
       {(type[,device-number])}
       {(type[,group-name])   }

EXTWTR=name

FLASH= {STANDARD              }
       {overlay-name          }
       {(overlay-name[,count])}

FORMS= {STANDARD }
       {form-name}

MODIFY=  {module-name        }
         {(module-name[,trc])}

OVFL=  {ON }
       {OFF}

PRTY=nnn

STACKER=  {STANDARD}
          {S       }
          {C       }

THRESHLD=limit

TRAIN=  {STANDARD  }
        {train-name}
```

The //*FORMAT PR statement consists of the characters //* in columns 1 through 3, FORMAT in columns 4 through 9, and a blank in column 10. PR begins in column 11 or beyond, followed by a comma, and parameters start after the command and can continue through column 72. JES3 ignores columns 73 through 80.

## Parameter definition

**PR**
Indicates that this statement is associated with a sysout data set that is printed.

**DDNAME=**
**DDNAME=ddname**
**DDNAME=stepname.ddname**
**DDNAME=stepname.procstepname.ddname**
**DDNAME=procstepname.ddname**
**DDNAME=JESYSMSG**
**DDNAME=JESJCL**
**DDNAME=JESMSGLG**

**(null)**
Specifies that the parameters on this //*FORMAT PR statement are the defaults for the job. These parameters then apply to all of the job's sysout data sets that are printed, except those covered by a //*FORMAT PR statement with a value other than (null) for DDNAME.

**Overrides**: Parameters coded on a nonspecific //*FORMAT PR statement are overridden by parameters coded on sysout DD statements or by parameters in the JES3 SYSOUT initialization statement.

**ddname**
**stepname.ddname**
**stepname.procstepname.ddname**
**procstepname.ddname**
Identifies the DD statement that defines the sysout data set to be printed; for example, **ddname** indicates all DD statements with the name, ddname, in this job. **Stepname.ddname** indicates DD statement, ddname, in step, stepname, in this job. **Stepname.procstepname.ddname** indicates DD statement, ddname, in procedure step, procstepname, of a procedure that is called by a step, stepname, in this job. The ddname must match exactly the ddname on the DD statement. (See the example for the //*DATASET statement.) If the identified DD statement does not contain a SYSOUT parameter, JES3 ignores the //*FORMAT PR statement.

**Note:** If a ddname matches more than one //*FORMAT PR statement, the //*FORMAT PR statement that has more qualifiers for the ddname will override the others. See "Examples of the //*FORMAT PR statement" on page 658.

**JESYSMSG**
Requests printing of system messages for this job.

**JESJCL**
Requests printing of JCL statements and messages for this job.

**JESMSGLG**
Requests printing of JES3 and operator messages for this job.

**CARRIAGE=carriage-tape-name**
**CARRIAGE=6**
Specifies the carriage tape for the 3211, 3203 Model 5, or 1403 Printer for printing this output class.

**carriage-tape-name**
Identifies the name of the carriage tape. The name is 1 through 8 characters. For the 3211 and 3203-5, SYS1.IMAGELIB must contain a module for each carriage tape name.

**6** Indicates the installation standard carriage tape.

**Note:** You cannot code both the CARRIAGE and FCB parameters on the same //*FORMAT PR statement.

**CHARS=STANDARD**
**CHARS=table-name**
**CHARS=(table-name[,table-name]...)**
Requests one or more fonts for printing the sysout data set on an AFP printer.

**STANDARD**
Indicates the standard character-arrangement table, which was specified at JES3 initialization.

**table-name**
Identifies a character-arrangement table. Each table-name is 1 through 4 alphanumeric or national ($, #, @) characters. When coding more than one table-name, parentheses are required around the list and null positions are invalid in the list.

**CHNSIZE=DS**
**CHNSIZE=(nnn[,mmm])**
Gives the number of logical records to be transmitted to a work station as a systems network architecture (SNA) chain and indicates whether normal output checkpoints are to be taken for this sysout data set.

**Note:** This parameter is valid only when transmitting to a SNA work station.

Be careful in selecting subparameters, because each affects performance differently. Sending the data set as a SNA chain provides the best performance, but can cause duplicate data to be written to the output device if operator intervention is required. The remote operator can eliminate duplicate data by issuing commands to reposition and restart the output writers.

When an end-of-chain indicator is sent in the data set, JES3 takes an output checkpoint. You can provide additional checkpoints for critical data by sending an end-of-chain indicator. For example, when printing bank checks, you can have an output checkpoint taken for each check by specifying each check as a SNA chain.

**DS** Indicates that the sysout data set is to be sent as a single SNA chain and that JES3 is not to take normal output checkpoints. DS is the default if the CHNSIZE parameter is omitted.

**nnn**
Specifies the SNA chain size in pages. nnn is a decimal number from 1 through 255. The size of a page is determined by:

- The value of mmm.
- The carriage control characters in the data that skip to channel 1.

**mmm**
Specifies the number of logical records in a page, when the data contains no carriage control characters. mmm is a decimal number from 1 through 255.

**COMPACT=compaction-table-name**
Specifies the compaction table for JES3 to use when sending a systems network architecture (SNA) data set to a SNA remote terminal. The compaction-table-name is a symbolic name defined by the installation during JES3 initialization. The name is 1 through 8 alphanumeric characters.

In the following cases, JES3 performs compaction using an installation default table, if defined, or sends the data without compacting it, if no table was defined. In all cases, JES3 writes a message to the console.

- No compaction table is specified.
- The specified compaction table is invalid.
- JES3 cannot find the specified compaction table.

If the remote printer does not support compaction, JES3 ignores the COMPACT parameter and sends the data without compacting it.

**CONTROL=PROGRAM**
**CONTROL=SINGLE**
**CONTROL=DOUBLE**
**CONTROL=TRIPLE**
Indicates either that the data records control printing or that the output is to be printed with single, double, or triple spacing.

**PROGRAM**
Indicates that each logical record in the data set begins with a carriage

control character. You can specify in the DD statement, the DCB macro, or the data set label that an optional control character is part of each record in the data set. The carriage control characters can be in either the extended USASCII code or can be the actual channel command code. The carriage control characters are given in *z/OS DFSMS Using Data Sets*.

**SINGLE**
Requests single spacing.

**DOUBLE**
Requests double spacing.

**TRIPLE**
Requests triple spacing.

**COPIES=nnn**
**COPIES=(nnn,(group-value[,group-value]...))**
**COPIES=(group-value[,group-value]...)**
Indicates how many copies of the sysout data set to print. If you do not specify a COPIES parameter, the default is 1.

You can omit the parentheses if you code only nnn.

**nnn**
A number from 1 through 254 that specifies how many copies of the data set to print. Each copy will be in page sequence order.

If you code COPIES=0 on the DD statement, the system uses a default of 1, which is the default for the DD COPIES parameter.

JES3 ignores nnn if any group-values are specified.

**group-value**
Specifies how many copies of each page are to be printed before the next page is printed. Each group-value is a number from 1 through 255. You can code a maximum of eight group-values. Their sum must not exceed 255. The total copies of each page equals the sum of the group-values.

This subparameter is valid only for output on a 3800 Printing Subsystem. Group values override an nnn subparameter.

**DEST=destination**
Routes the output from the sysout data set to a printer. This parameter overrides the //*MAIN statement ORG parameter.

If you omit DEST, JES3 assigns the first available printer that is in the origin group and that fulfills all processing requirements. The origin group is the group of printers defined for the local or remote submitting location. If the job originated at a remote job processing (RJP) terminal, JES3 returns the output to the originating terminal group.

If the job was submitted through TSO/E to the NJE network for execution, the default is the node from which the job was submitted, and the destination ANYLOCAL.

**ANYLOCAL**
Indicates any local printer that is being used for the output class specified in the SYSOUT parameter on the DD statement and that is attached to the global processor.

**device-name**
Requests a local device by a symbolic name defined by the installation during JES3 initialization. device-name is 1 through 8 alphanumeric or national ($, #, @) characters.

**device-number**
Identifies a specific device by a 3-digit or 4-digit hexadecimal number. Precede a 4-digit number with a slash (/). A 3-digit number can be specified with or without a slash.

**group-name**
Identifies a group of local devices, an individual remote station, or a group of remote stations by a symbolic name defined by the installation during JES3 initialization. group-name is 1 through 8 alphanumeric or national ($, #, @) characters.

**nodename**
Identifies a node by a symbolic name defined by the installation during JES3 initialization. nodename is 1 through 8 alphanumeric or national ($, #, @) characters.

**remote**
Identifies a remote work station or VM userid to which the receiving node directs output. remote is 1 through 8 characters.

**(type)**
Indicates a device classification. type is in the form **(gggssss)** where **ggg** is the general device classification and **ssss** is the specific device classification. The type must be enclosed in parentheses. The type must be defined by the installation during JES3 initialization. For example, type for a 3800 is (PRT3800).

**EXTWTR=name**
Identifies the external writer that is to process the sysout data set at the destination node. name is 1 through 8 alphanumeric characters and must identify a module defined to the remote JES3 node that is to execute the job. (Do not code NJERDR, it is reserved for JES3.)

**FCB=image-name**
**FCB=6**
Specifies the forms control buffer (FCB) image JES3 is to use to guide printing of the sysout data set by a 1403 Printer, 3211 Printer, 3203 Printer Model 5, 4245 Printer, 4248 Printer, or 3800 Printing Subsystem, or by a printer supported by systems network architecture (SNA) remote job processing (RJP).

If the data set is to be produced on some other device, JES3 ignores the FCB parameter.

**image-name**
Identifies the FCB image. The name is 1 through 4 alphanumeric or national ($, #, @) characters and is the last characters of a SYS1.IMAGELIB member name:

- FCB2xxxx member for a 3211, 3203 model 5, or printer supported by SNA.
- FCB3xxxx member for a 3800.
- FCB4xxxx member for a 4248.

**6** Indicates the standard FCB. JES3 uses the standard FCB specified at JES3 initialization.

**Note:** You cannot code both the CARRIAGE and FCB parameters on the same //*FORMAT PR statement.

**FLASH=STANDARD**
**FLASH=overlay-name**
**FLASH=(overlay-name[,count])**
Identifies the forms overlay to be used in printing the sysout data set on a 3800 Printing Subsystem and, optionally, to specify the number of copies on which the forms overlay is to be printed.

You can omit the parentheses if you code only an overlay-name. If you omit the count subparameter or specify a count of 0, JES3 flashes all copies with the specified overlay.

**STANDARD**
Indicates the standard forms flash overlay. JES3 uses the standard forms overlay specified at JES3 initialization.

**overlay-name**
Identifies the forms overlay frame that the operator is to insert into the printer before printing begins. The name is 1 through 4 alphanumeric or national ($, #, @) characters.

**count**
Specifies the number, 0 through 255, of copies that JES3 is to flash with the overlay, beginning with the first copy printed. Code a count of 0 to flash **all** copies.

**Note:** See the *Forms Design Reference Guide for the 3800* for information on designing and making forms overlays.

**FORMS=STANDARD**
**FORMS=form-name**
Indicates the forms on which the sysout data set is to be printed.

**STANDARD**
Indicates the standard form. JES3 uses the standard form specified at JES3 initialization.

**form-name**
Names the print forms. form-name is 1 through 8 alphanumeric characters.

**MODIFY=module-name**
**MODIFY=(module-name[,trc])**
Specifies a copy modification module that tells JES3 how to print the sysout data set on a 3800 Printing Subsystem. The module can specify how to replace blanks or data in the data set. You can omit the parentheses if you code only a module-name.

The module is defined and stored in SYS1.IMAGELIB using the IEBIMAGE utility program. See *z/OS DFSMSdfp Utilities* for more information.

If you omit the trc subparameter, JES3 prints the data set with the first character-arrangement table coded in the CHARS parameter.

**module-name**
Identifies a copy modification module in SYS1.IMAGELIB. module-name is 1 through 4 alphanumeric or national ($, #, @) characters.

**trc**
Identifies which table-name in the CHARS parameter is to be used. This

table reference character is 0 for the first table-name specified, 1 for the second, 2 for the third, or 3 for the fourth.

**OVFL=ON**
**OVFL=OFF**
Indicates whether or not the printer program should test for forms overflow.

Because the overflow test is a responsibility of the terminal package for the remote RJP terminal, JES3 ignores OVFL for remote job processing.

**ON** Indicates that the printer program should eject whenever the end-of-forms indicator (channel 12) is sensed. ON is the default if the OVFL parameter is omitted.

**OFF**
Indicates that forms overflow control is not to be used.

**PRTY=nnn**
Specifies the priority at which the sysout data set enters the output queue. nnn is a decimal number from 0 through 255; 0 is the lowest priority while 255 is the highest.

**STACKER=STANDARD**
**STACKER=S**
**STACKER=C**
Requests a stacker for output processed by PSF on any continuous forms AFP printer.

**STANDARD**
Indicates the standard installation default. This default is specified at JES3 initialization.

**S** Indicates the burster-trimmer-stacker, in which the output is burst into separate sheets.

**C** Indicates the continuous forms stacker, in which the output is left in continuous fanfold.

**THRESHLD=limit**
Specifies the maximum size for the sysout data set. JES3 calculates the sysout data set size as the number of records multiplied by the number of copies requested. When this size exceeds the THRESHLD value, JES3 creates a new unit of work, on a data set boundary, and queues it for printing. Consequently, copies of the sysout data set may be printed simultaneously by different printers.

Use the THRESHLD parameter for jobs that generate many large sysout data sets. Grouping data sets as a single unit of work for an output service writer may decrease the time required for the output service writer to process the data sets.

The value specified in this parameter overrides the value specified during JES3 initialization.

**limit**
Specifies the maximum records for a single sysout data set. limit is a decimal number from 1 through 99999999. The default is 99999999.

**TRAIN=STANDARD**
**TRAIN=train-name**
Indicates the printer train to be used in printing the sysout data set. See

Table 18 on page 291 for the IBM-supplied trains. Because these trains are not standard machine features, verify that the installation has the required printer train before specifying it.

Do not code the TRAIN parameter for output destined for a remote job processing (RJP) terminal.

**STANDARD**
Indicates the standard installation default. This default is specified at JES3 initialization.

**train-name**
Specifies an installation-supplied printer train. Check with your installation for the names of trains.

## Relationship to sysout DD and OUTPUT JCL statements

- JES3 ignores the processing options specified on a default //*FORMAT statement when a sysout DD statement explicitly or implicitly references an OUTPUT JCL statement.
- JES3 ignores the processing options specified on a default OUTPUT JCL statement when a //*FORMAT statement explicitly references a sysout DD statement.
- When a sysout DD statement explicitly references an OUTPUT JCL statement and a //*FORMAT statement explicitly references the same DD statement, the processing options from both the OUTPUT JCL and //*FORMAT statements apply. Two separate sets of output are created from the data set defined by the sysout DD statement; one according to the processing options on the OUTPUT JCL and DD statements, and the other according to the processing options on the //*FORMAT and DD statements.

## Relationship to //*PROCESS statement

JES3 accumulates //*FORMAT PR statements within a job and applies them to any JES3 //*PROCESS statement that is normally affected by a //*FORMAT PR statement.

## Location in the JCL

Place all //*FORMAT PR statements for the job after the JOB statement and before the first EXEC statement.

## Examples of the //*FORMAT PR statement

***Example 1***

```
//*FORMAT PR,DDNAME=STEP1.REPORT,COPIES=2
```

This statement requests two copies of the data set defined by sysout DD statement REPORT, which appears in STEP1 of this job. Any printer with standard forms, train, and carriage tape can be used.

***Example 2***

```
//*FORMAT PR,DDNAME=,DEST=ANYLOCAL
```

This statement specifies that all sysout data sets not referenced by //*FORMAT PR statements are to be printed on any local printer.

***Example 3***

```
//*FORMAT PR,DDNAME=STEP1.REPORT,DEST=A
//*FORMAT PR,DDNAME=REPORT,DEST=B
```

This statement requests one copy of the data set defined by sysout DD statement REPORT, which appears in STEP1 of this job, to be sent to destination A and one copy of the data set defined by sysout DD statement REPORT to be sent to destination B. The REPORT data set for STEP1 is sent to destination A because the //*FORMAT PR statement with more qualifiers for the same ddname overrides the other. The REPORT data set for any other step is sent to destination B.

## //*FORMAT PU statement

***Purpose:*** Use the //*FORMAT PU statement to specify to JES3 processing instructions for sysout data sets that are punched. These instructions permit special processing of sysout data sets, such as:

- Multiple destinations.
- Multiple copies of output with different attributes.

Use the //*FORMAT PU statement to specify to JES3 processing instructions for sysout data sets that are punched. These instructions permit special processing of sysout data sets, such as:

- Multiple destinations.
- Multiple copies of output with different attributes.

//*FORMAT PU statements can be either specific or nonspecific. A specific //*FORMAT PU statement contains a DDNAME parameter that specifies something other than a null value, such as DDNAME=ddname or DDNAME=JESYSMSG. A nonspecific //*FORMAT PU statement contains DDNAME= , with no value (null) specified for the DDNAME parameter.

You can code multiple specific //*FORMAT PU statements for a particular sysout data set to specify special requirements for different copies of the data set. In addition, you can code a //*FORMAT PR statement for the same sysout data set, thereby both printing and punching it.

You can also code multiple nonspecific //*FORMAT PU statements. In this case, the system produces only one copy of each data set, combining any parameter values specified on the statements. If you specify a given parameter on more than one of these statements, the system uses the parameter value specified on the last //*FORMAT PU statement containing that parameter.

**Note:** The //*FORMAT PU statement applies only to sysout data sets punched by JES3. The statement is ignored for data sets sent to a TSO/E userid or processed by an external writer.

***Reference:*** For examples of //*FORMAT statement processing on the JES3 hold queue and writer queue, see *z/OS JES3 Initialization and Tuning Guide*.

## Syntax

```
//*FORMAT PU,DDNAME=   {ddname                       }   [,parameter]...
                       {stepname.ddname             }          parameter]...
                       {stepname.procstepname.ddname}

//*FORMAT PU,DDNAME=[,parameter]...

The parameters are:

 CHNSIZE=   {DS         }
            {(nnn[,mmm])}

 COMPACT=compaction-table-name

 COPIES=nnn

 DEST=   {ANYLOCAL              }
         {device-name           }
         {device-number         }
         {group-name            }
         {nodename[.remote]     }
         {(type[,device-name])  }
         {(type[,device-number])}
         {(type[,group-name])   }

 EXTWTR=name

 FORMS= {STANDARD }
        {form-name}

 INT= {YES}
      {NO }
```

The //*FORMAT PU statement consists of the characters //* in columns 1 through 3, FORMAT in columns 4 through 9, and a blank in column 10. PU begins in column 11 or beyond, followed by a comma, and parameters start after the comma and continue through column 72. JES3 ignores columns 73 through 80.

## Parameter definition

**PU**
Indicates that this statement is associated with a sysout data set that is punched.

**DDNAME=**
**DDNAME=ddname**
**DDNAME=stepname.ddname**
**DDNAME=stepname.procstepname.ddname**

**(null)**
Specifies that the parameters on this //*FORMAT PU statement are the defaults for the job. These parameters then apply to all of the job's sysout data sets that are punched except those covered by a //*FORMAT PU statement with a value other than (null) for DDNAME.

**Overrides**: Parameters coded on a nonspecific //*FORMAT PU statement are overridden by parameters coded on sysout DD statements or by parameters in the JES3 SYSOUT initialization statement.

**ddname**
**stepname.ddname**

**stepname.procstepname.ddname**
Identifies the DD statement that defines the sysout data set to be punched. Use form **ddname** to indicate all DD statements with the name, ddname, in this job. Use form **stepname.ddname** to indicate DD statement, ddname, in step, stepname, in this job. Use form **stepname.procstepname.ddname** to indicate DD statement, ddname, in procedure step, procstepname, of a procedure that is called by a step, stepname, in this job. The ddname must match exactly the ddname on the DD statement. (See the example for the //*DATASET statement.) If the identified DD statement does not contain a SYSOUT parameter, JES3 ignores the //*FORMAT PU statement.

**Note:** If a ddname matches more than one //*FORMAT PU statement, the //*FORMAT PU statement that has more qualifiers for the ddname will override the others. See "Examples of the //*FORMAT PU statement" on page 664.

**CHNSIZE=DS**
**CHNSIZE=(nnn[,mmm])**
Gives the number of logical records to be transmitted to a work station as a systems network architecture (SNA) chain and indicates whether normal output checkpoints are to be taken for this sysout data set.

**Note:** This parameter is valid only when transmitting to a SNA work station.

Be careful in selecting subparameters, because each affects performance differently. Sending the data set as a SNA chain provides the best performance, but can cause duplicate data to be written to the output device if an operator intervention is required. The remote operator can eliminate duplicate data by issuing commands to reposition and restart the output writers.

When an end-of-chain indicator is sent in the data set, JES3 takes an output checkpoint. You can provide additional checkpoints for critical data by sending an end-of-chain indicator. For example, when punching bank checks, you can have an output checkpoint taken for each check by specifying each check as a SNA chain.

**DS** Indicates that the sysout data set is to be sent as a single SNA chain and that JES3 is not to take normal output checkpoints. DS is the default if the CHNSIZE parameter is omitted.

**nnn**
Specifies the SNA chain size in pages. nnn is a decimal number from 1 through 255. The size of a page is determined by the value you assign to mmm.

**mmm**
Specifies the number of logical records in a page. mmm is a decimal number from 1 through 255.

**COMPACT=compaction-table-name**
Specifies the compaction table for JES3 to use when sending a systems network architecture (SNA) data set to a SNA remote terminal. The compaction-table-name is a symbolic name defined by the installation during JES3 initialization. The name is 1 through 8 alphanumeric characters.

In the following cases, JES3 performs compaction using an installation default table, if defined, or sends the data without compacting it, if no table was defined. In all cases, JES3 writes a message to the console.

- No compaction table is specified.

- The specified compaction table is invalid.
- JES3 cannot find the specified compaction table.

If the remote punch does not support compaction, JES3 ignores the COMPACT parameter and sends the data without compacting it.

**COPIES=nnn**
Indicates how many copies of the sysout data set are to be punched. nnn is a number from 0 through 255. If you code COPIES=0, JES3 does not punch this data set. If a COPIES parameter is not specified, the default is 1.

**DEST=destination**
Routes the output from the sysout data set to a punch. This parameter overrides the //*MAIN statement ORG parameter.

If you omit DEST, JES3 assigns the first available punch that is in the origin group and that fulfills all processing requirements. The origin group is the group of punches defined for the local or remote submitting location. If the job originated at a remote job processing (RJP) terminal, JES3 returns the output to the originating terminal group.

If the job was submitted through TSO/E to the NJE network for execution, the default is the node from which the job was submitted, and the destination ANYLOCAL.

**ANYLOCAL**
Indicates any local punch that is being used for the output class specified in the SYSOUT parameter on the DD statement and that is attached to the global processor.

**device-name**
Requests a local device by a symbolic name defined by the installation during JES3 initialization. device-name is 1 through 8 alphanumeric or national ($, #, @) characters.

**device-number**
Specifies the 3-digit or 4-digit hexadecimal device number. Precede a 4-digit number with a slash (/). A 3-digit number can be specified with or without a slash.

**group-name**
Identifies a group of local devices, an individual remote station, or a group of remote stations by a symbolic name defined by the installation during JES3 initialization. group-name is 1 through 8 alphanumeric or national ($, #, @) characters.

**nodename**
Identifies node by a symbolic name defined by the installation during JES3 initialization. nodename is 1 through 8 alphanumeric or national ($, #, @) characters.

**remote**
Identifies a remote work station or VM userid to which the receiving node directs output. remote is 1 through 8 characters.

**(type)**
Indicates a device classification. type is in the form **(gggssss)** where **ggg** is the general device classification and **ssss** is the specific device classification. The type must be enclosed in parentheses. The type must be defined by the installation during JES3 initialization. For example, type for a 3525 is (PUN3525).

**EXTWTR=name**
Identifies the external writer that is to process the sysout data set at the destination node. name is 1 through 8 alphanumeric characters and must identify a module defined to the remote JES3 node that is to execute the job.

**FORMS=STANDARD**
**FORMS=form-name**
Indicates the forms on which the sysout data set is to be punched.

**STANDARD**
Indicates the standard form. JES3 uses the standard form specified at JES3 initialization.

**form-name**
Names the punch forms. form-name is 1 through 8 alphanumeric characters.

**INT=YES**
**INT=NO**
Specifies whether or not the output is to be interpreted. If the INT parameter is omitted, the default is NO.

**YES**
Requests that JES3 try to punch the sysout data set on a 3525 Card Punch (PUN3525I) with a Multiline Card Print feature.

**Note:** If the DEST parameter does not send output to a 3525I, JES3 ignores INT=YES, if specified.

**NO** Requests that the cards not be interpreted.

## Relationship to sysout DD and OUTPUT JCL statements

- JES3 ignores the processing options specified on a default //*FORMAT statement when a sysout DD statement explicitly or implicitly references an OUTPUT JCL statement.
- JES3 ignores the processing options specified on a default OUTPUT JCL statement when a //*FORMAT statement explicitly references a sysout DD statement.
- When a sysout DD statement explicitly references an OUTPUT JCL statement and a //*FORMAT statement explicitly references the same DD statement, the processing options from both the OUTPUT JCL and //*FORMAT statements apply. Two separate sets of output are created from the data set defined by the sysout DD statement; one according to the processing options on the OUTPUT JCL and DD statements, and the other according to the processing options on the //*FORMAT and DD statements.

## Relationship to //*PROCESS statement

JES3 accumulates //*FORMAT PU statements within a job and applies them to any JES3 //*PROCESS statement that is normally affected by a //*FORMAT PU statement.

## Location in the JCL

Place all //*FORMAT PU statements for the job after the JOB statement and before the first EXEC statement.

## Examples of the //*FORMAT PU statement

*Example 1*

```
//*FORMAT PU,DDNAME=STEP2.PUNCHOUT,DEST=PU1,FORMS=RED-STRP
```

This statement requests that one copy of the data set defined by sysout DD statement PUNCHOUT in STEP2 of this job be punched on device PU1. Before processing, the operator is requested to insert **RED-STRP** cards into the punch.

*Example 2*

```
//*FORMAT PU,DDNAME=STEP1.PUNCHOUT,DEST=DEVA
//*FORMAT PU,DDNAME=PUNCHOUT,DEST=DEVB
```

This statement requests one copy of the data set defined by sysout DD statement PUNCHOUT in STEP1 of this job to be punched on device DEVA and one copy of the data set defined by sysout DD statement PUNCHOUT to be punched on device DEVB. The PUNCHOUT data set for STEP1 is sent to DEVA because the //*FORMAT PU statement with more qualifiers for the same ddname overrides the other. The PUNCHOUT data set for any other step is sent to DEVB.

# //*MAIN statement

***Purpose:*** Use the //*MAIN statement to define the processor requirements for the current job. Many of the parameters are used to override parameters on the JES3 STANDARDS initialization statement.

**Note:** If any parameter is misspelled or contains an invalid value, JES3 writes the following to the JESMSG data set: the //*MAIN statement, the relative error position on the statement, and an error message. Then JES3 abnormally terminates the job.

## Syntax

```
//*MAIN parameter[,parameter]...

The parameters are:

ACMAIN=processor-id

BYTES= {([nnnnnn][,WARNING][,mmm])}
       {([nnnnnn][,W][,mmm])      }
       {([nnnnnn][,CANCEL])       }
       {([nnnnnn][,C])            }
       {([nnnnnn][,DUMP])         }
       {([nnnnnn][,D])            }

CARDS= {([nnnn][,WARNING][,mmm])}
       {([nnnn][,W][,mmm])      }
       {([nnnn][,CANCEL])       }
       {([nnnn][,C])            }
       {([nnnn][,DUMP])         }
       {([nnnn][,D])            }

CLASS=class-name

DEADLINE=  {(time,type[,date])     }
           {(time,type[,rel,cycle])}

EXPDTCHK=  {YES}
           {NO }

FAILURE=  {RESTART}
          {CANCEL }
          {HOLD   }
          {PRINT  }

FETCH=  {ALL                  }
        {NONE                 }
        {SETUP                }
        {(ddname[,ddname]...) }
        {/(ddname[,ddname]...)}

HOLD=  {YES}
       {NO }

IORATE=  {MED }
         {HIGH}
         {LOW }

JOURNAL=  {YES}
          {NO }
```

```
LINES= {([nnnn][,WARNING][,mmm])}
       {([nnnn][,W][,mmm])      }
       {([nnnn][,CANCEL])       }
       {([nnnn][,C])            }
       {([nnnn][,DUMP])         }
       {([nnnn][,D])            }

LREGION=nnnnK

ORG= {group-name       }
     {nodename[.remote]}

PAGES= {([nnnnnnnn][,WARNING][,mmm])}
       {([nnnnnnnn][,W][,mmm])      }
       {([nnnnnnnn][,CANCEL])       }
       {([nnnnnnnn][,C])            }
       {([nnnnnnnn][,DUMP])         }
       {([nnnnnnnn][,D])            }

PROC=  {ST}
       {xx}

RINGCHK=  {YES}
          {NO }

SETUP= {JOB                                                                }
       {HWS                                                                }
       {THWS                                                               }
       {DHWS                                                               }
       {(stepname.ddname[,stepname.ddname]...)                             }
       {(stepname.procstepname.ddname[,stepname.procstepname.ddname]...) }
       {/(stepname.ddname[,stepname.ddname]...)                            }
       {/(stepname.procstepname.ddname[,stepname.procstepname.ddname]...)}

SPART=partition-name

SYSTEM=  {ANY                       }
         {JGLOBAL                   }
         {JLOCAL                    }
         {(main-name[,main-name]...) }
         {/(main-name[,main-name]...)}

THWSSEP=  {IGNORE }
          {PREFER }
          {REQUIRE}

TRKGRPS=(primary-qty,second-qty)

TYPE=  {ANY}
       {VS2}

UPDATE=(dsname[,dsname]...)

USER=userid
```

The //*MAIN statement consists of the characters //* in columns 1 through 3, MAIN in columns 4 through 7, a blank in column 8, and parameters in columns 9 through 72. JES3 ignores columns 73 through 80.

## Parameter definition

**ACMAIN=processor-id**

Identifies the job with the specified processor, even though the job was not submitted from or run on that processor. ACMAIN allows:

- Sysout data sets to be sent to a userid attached to the specified processor. The userid must be named in the USER parameter. The ACMAIN parameter applies to all sysout data sets for the job.
- Receipt of notification that a job you submitted through batch processing has completed by coding the ACMAIN parameter on a JES3 //*MAIN statement in addition to the JOB statement NOTIFY parameter. The ACMAIN parameter names the processor that you, the TSO/E user, are logged onto.

**processor-id**
Requests a processor in the complex.

**BYTES=([nnnnnn][,WARNING][,mmm])**
**BYTES=([nnnnnn][,W][,mmm])**
**BYTES=([nnnnnn][,CANCEL])**
**BYTES=([nnnnnn][,C])**
**BYTES=([nnnnnn][.DUMP])**
**BYTES=([nnnnnn][,D])**
Specifies the maximum number of bytes of data to be spooled from this job's sysout data sets and the action to be taken if the maximum is exceeded.

If BYTES is not specified, the installation default for this job class applies.

**nnnnnn**
Specifies the number of bytes in thousands. nnnnnn is 1 through 6 decimal numbers from 1 through 999999.

**WARNING or W**
If the maximum is exceeded, requests that JES3 issue an operator warning message and continue processing.

Any messages about this parameter following the warning message will reflect the number specified on the STANDARD initialization statement or the system default, **not** the specified maximum.

**mmm**
Specifies the frequency that an operator warning message is to be issued after the maximum specified by nnnnnn is exceeded. mmm is a multiple of 10 in the range 10 to 100. mmm is a percentage of nnnnnn that is used to calculate the number of additional bytes between warning messages. For example, if BYTES=(100,W,20) is specified, the first warning message is sent to the operator when 100,000 bytes of sysout data is reached. Subsequent warning messages are sent when each additional 20 percent of 100,000 is reached (at 120,000 bytes, 140,000 bytes, and so on). Messages are sent until the job ends or the operator cancels the job.

**CANCEL or C**
If the maximum is exceeded, requests that JES3 cancel the job.

**DUMP or D**
If the maximum is exceeded, requests that JES3 cancel the job and ask for a storage dump.

**CARDS=([nnnn][,WARNING][,mmm])**
**CARDS=([nnnn][,W][,mmm])**
**CARDS=([nnnn][,CANCEL])**
**CARDS=([nnnn][,C])**
**CARDS=([nnnn][,DUMP])**
**CARDS=([nnnn][,D])**
Specifies the maximum number of cards to be punched from this job's sysout data sets and the action to be taken if the maximum is exceeded.

If you specify CARDS=0 the zero applies only to the quantity of punched output; it does **not** cancel the action to be taken if the maximum is exceeded. If a record is then sent to a punch, JES3 will warn, cancel, or dump, depending on the second parameter.

**Note:** When punching dump output, JES3 ignores CARDS=0.

If CARDS is not specified, the installation default for this job class is used.

**nnnn**
Specifies the number of cards in hundreds. nnnn is 1 through 4 decimal numbers from 1 through 9999.

**WARNING or W**
If the maximum is exceeded, requests that JES3 issue an operator warning message and continue processing.

Any subsequent messages about this parameter will reflect the number specified on the STANDARD initialization statement or the system default, **not** the maximum specified in the CARDS parameter.

**mmm**
Specifies the frequency that an operator warning message is to be issued after the maximum specified by nnnn is exceeded. mmm is a multiple of 10 in the range 10 to 100. mmm is a percentage of nnnn that is used to calculate the number of additional cards between warning messages. For example, if CARDS=(100,W,20) is specified, the first warning message is sent to the operator when 10,000 cards of sysout data is reached. Subsequent warning messages are sent when each additional 20 percent of 10,000 is reached (at 12,000 cards, 14,000 cards, and so on). Messages are sent until the job ends or the operator cancels the job.

**CANCEL or C**
If the maximum is exceeded, requests that JES3 cancel the job.

**DUMP or D**
If the maximum is exceeded, requests that JES3 cancel the job and ask for a storage dump.

**CLASS=class-name**
Specifies the job class for this job. class-name is 1 through 8 characters.

If the desired class-name is a single-character, you can specify it on the //*MAIN statement or the JOB statement.

JES3 uses the following, in override order, to assign the job to a class:

1. //*MAIN statement CLASS parameter
2. JOB statement CLASS parameter
3. The default class, which is defined during JES3 initialization.

If neither CLASS nor LREGION is specified, JES3 determines the logical region size based on initialization parameters.

**DEADLINE=(time,type[,date])**
**DEADLINE=(time,type[,rel,cycle])**
Specifies when the job is required.

When you specify the current date but submit the job after the specified time, JES3 changes the priorities to make the job the same priority level it would have if it had been submitted before the deadline but not completed.

**Attention**: Deadline scheduling can interfere with dumping a portion of the job queue. For example, if JOB A is waiting to be scheduled, has a priority of 7, and, in one minute, is due to have its priority increased to 9, JOB A could be missed by dump job processing, if the dump job facility is dumping the entire job queue and currently dumping priority 8 jobs. The dump job facility processes the jobs with the highest priority first. If the dump job facility does not finish processing priority 8 jobs before JOB A becomes priority 9, JOB A will not be dumped.

Deadline scheduling information is not sent with a job when the job is transferred via NJE to another node; the destination node may use different deadline scheduling algorithms, if any.

**time**
Specifies the deadline time, expressed as one of the following:

**nM** The job is to be scheduled within n minutes. n is 1 through 4 numbers from 0 through 1440.

**nH** The job is to be scheduled within n hours. n is 1 or 2 numbers from 0 through 24.

**hhhh**
The job is to be scheduled by the time of day, hhhh, in 24-hour clock time (0800 is 8:00 a.m.). hhhh is from 0000 (start of the day) through 2400 (end of the day).

**type**
Identifies the deadline algorithm. The deadline algorithm is defined by the installation, controls how the job's priority is increased, and is one character: A through Z or 0 through 9. If the specified algorithm is not defined, JES3 abnormally terminates the job.

**date**
Specifies the date, in one of the following formats, when the time parameter takes effect.

**mmddyy**
where mm is the month (01-12), dd the day (01-31), and yy the 2-digit year (01-99).

**mm/dd/yyyy**
where mm is the month (01-12), dd the day (01-31), and yyyy the 4-digit year (for example, 1999). Leading zeroes are required in the day and month fields.

**Note:**

1. For dates in the format of mmddyy, a century of '19' is assumed.
2. For dates in the format of mmddyy, a date of '00' is not allowed.
3. For dates of January 1, 2000 and later, you must use the form mm/dd/yyyy.
4. If both date and rel,cycle are omitted, JES3 assumes (1) the current date, if the deadline time is later in the day, or (2) the next day's date, if the deadline time has already past today.

**rel**
Specifies on which day within a cycle the deadline falls. rel is 1 through 3 numbers from 1 through 366. The value of rel depends on the specified cycle, as follows:

- WEEKLY: Sunday is day 1; Saturday is day 7. If rel is greater than 7, it defaults to 7.
- MONTHLY: Day 1 is the first day of the month. Days 29, 30, and 31 are treated as the last day of the month. If rel is greater than 31, it defaults to 31.
- YEARLY: Day 1 is January 1; day 365 is December 31, for non-leap years, and day 366 is December 31, for leap years. If rel is greater than 365, it defaults to 365 for non-leap years or 366 for leap years.

**cycle**
Specifies the length of a cycle. cycle is coded as WEEKLY, MONTHLY, or YEARLY.

For example, DEADLINE=(1200,B,1,WEEKLY) indicates that the job reaches its deadline at 12 noon on Sunday. This job would be submitted once a week for it to be processed every Sunday.

**EXPDTCHK=YES**
**EXPDTCHK=NO**
Indicates whether or not JES3 is to perform expiration date checking for scratch output tape volumes with IBM standard labels (SL).

**YES**
Requests expiration date checking. Tape volumes premounted for SL scratch requests must have expired dates.

**NO** Requests that expiration dates not be checked.

**FAILURE=RESTART**
**FAILURE=CANCEL**
**FAILURE=HOLD**
**FAILURE=PRINT**
Indicates the job recovery option to be used if the system fails. If you do not code a FAILURE parameter on the //*MAIN statement, JES3 assigns the job the default failure option, which is defined during JES3 initialization for each job class. (See also the RD parameter on the JOB statement.)

**Note:** If a job is registered with the automatic restart manager (ARM) at the time of a system failure, ARM determines whether to restart the job, regardless of the value specified on the FAILURE keyword.

If the ARM restarts the job, JES discards all non-spin sysout data sets created during the previous execution. (You can avoid losing that output by adding SPIN=UNALLOC to the DD statement for the SYSOUT data set.)

**RESTART**
Requests that JES3 restart the job when the failing processor is restarted. Do not specify RESTART for jobs that use the DEQ at DEMOUNT facility for tape volumes.

**CANCEL**
Requests that JES3 print the job and then cancel the job.

**HOLD**
Requests that JES3 hold the job for restart.

**PRINT**
Requests that JES3 print the job and then hold the job for restart.

**FETCH=ALL**
**FETCH=NONE**

**FETCH=SETUP**
**FETCH=(ddname[,ddname]...)**
**FETCH=/(ddname[,ddname]...)**
Determines the fetch messages that will be issued to the operator for disk and tape volumes for this job.

If FETCH is not specified, the installation default for this job class applies.

**ALL**
Requests that JES3 issue fetch messages to the operator for all removable volumes specified in DD statements that request JES3-setup devices. This subparameter does not apply to permanently resident volumes.

**NONE**
Requests that JES3 not issue fetch messages.

**SETUP**
Requests that JES3 issue fetch messages to the operator for the volumes specified in all DD statements identified in the //*MAIN SETUP parameter. If you code FETCH=SETUP without also coding the //*MAIN SETUP parameter, JES3 will issue fetch message as though you had specified FETCH=ALL.

**ddname**
Requests that JES3 issue fetch messages for only the volumes specified in DD statement ddname.

If you code a list of ddnames and the list cannot be contained on a single statement, FETCH= must be repeated on the continuation statement.

**/ddname**
Requests that JES3 **not** issue fetch messages for any volumes specified in DD statement ddname.

**HOLD=YES**
**HOLD=NO**

**YES**
Indicates that the job is to enter the system in operator-hold status and be withheld from processing until the operator requests its release. However, if an error occurs during input service processing, the job is not held for operator intervention.

This parameter has the same function as TYPRUN=HOLD on the JOB statement.

**NO** Indicates that the job is to enter the system normally. Processing does not require operator intervention. If the HOLD parameter is omitted, NO is the default.

**IORATE=MED**
**IORATE=HIGH**
**IORATE=LOW**
Indicates the I/O-to-processor ratio for a job. Use this parameter to balance the mixture of jobs selected for execution on the processor.

If you do not code an IORATE parameter on the //*MAIN statement, JES3 assigns the job the default I/O-to-processor ratio, which is defined during JES3 initialization for each job class.

**JOURNAL=YES**

**JOURNAL=NO**
Indicates whether or not JES3 is to create a job journal for the job.

If JOURNAL is omitted, JES3 uses an installation default specified at initialization. If you use the automatic restart manager (ARM) to restart a job, you do not need to save the journal because ARM does not use the job journal when restarting jobs.

**YES**
Indicates that the job is to have a job journal.

**NO** Indicates that the job is not to have a job journal.

**LINES=([nnnn][,WARNING][,mmm])**
**LINES=([nnnn][,W][,mmm])**
**LINES=([nnnn][,CANCEL)**
**LINES=([nnnn][,C])**
**LINES=([nnnn][,DUMP])**
**LINES=([nnnn][,D])**
Indicates the maximum number of lines of data to be printed from this job's sysout data sets and the action to be taken if the maximum is exceeded.

If you specify LINES=0 the zero applies only to the number of lines; it does **not** cancel the action to be taken if the maximum is exceeded. If a record is sent to be printed, JES3 will warn, cancel, or dump, depending on the second parameter.

**Note:** JES3 ignores any line count specification when printing the output for a SYSABEND or SYSUDUMP sysout data set.

If LINES is not specified, the installation default for this job class applies. The installation default is specified on the OUTLIM parameter of the OUTSERV JES3 initialization statement.

**nnnn**
Specifies the number of lines, in thousands. nnnn is 1 through 4 decimal numbers from 1 through 9999.

**WARNING or W**
If the maximum is exceeded, requests that JES3 issue an operator warning and continue processing.

Any messages about this parameter following the warning message will reflect the number specified on the STANDARD initialization statement or the system default, **not** the maximum specified in the LINES parameter.

**mmm**
Specifies the frequency that an operator warning message is to be issued after the maximum specified by nnnn is exceeded. mmm is a multiple of 10 in the range 10 to 100. mmm is a percentage of nnnn that is used to calculate the number of additional lines between warning messages. For example, if LINES=(100,W,20) is specified, the first warning message is sent to the operator when 100,000 lines of sysout data is reached. Subsequent warning messages are sent when each additional 20 percent of 100,000 is reached (at 120,000 lines, 140,000 lines, and so on). Messages are sent until the job ends or the operator cancels the job.

**CANCEL or C**
If the maximum is exceeded, requests that JES3 cancel the job.

**DUMP or D**
If the maximum is exceeded, requests that JES3 cancel the job and ask for a storage dump.

**LREGION=nnnnK**
Specifies the approximate size of the largest step's working set in real storage during execution. LREGION (logical region) is used by JES3 to improve scheduling on the processor. The nnnn is 1 through 4 decimal numbers that indicate the size in kilobytes (1 kilobyte = 1024 bytes).

If neither CLASS nor LREGION is coded, JES3 determines the logical region size based on initialization parameters.

Use the LREGION parameter carefully. If the values selected for LREGION are too small, the job may take longer to run.

**ORG=group-name**
**ORG=nodename[.remote]**
Indicates that the job's sysout data sets are to be directed to the named group or network node. Otherwise, the job's sysout data sets are directed to the group of devices or node from which the job originated.

**group-name**
Specifies an origin group.

**nodename**
Specifies a network node. nodename is 1 through 8 characters.

**remote**
Specifies a remote work station or VM userid. remote is 1 through 8 characters and must be separated from the nodename by a period.

**Overriding an ORG Parameter**: If you do not want a particular data set in the job to go to the destination on the ORG parameter, change its destination in one of the following ways:

- If the sysout data set is not scheduled to a **held** class, you can override the ORG parameter destination with the DEST parameter on a //*FORMAT, OUTPUT JCL, or DD statement.
- If the sysout data set is scheduled to a **held** class, you can override the ORG parameter destination with the DEST parameter on an OUTPUT JCL, or DD statement.

JES3 ignores the ORG parameter for a dynamically-allocated SYSOUT data set.

**PAGES=([nnnnnnnn][,WARNING][,mmm])**
**PAGES=([nnnnnnnn][,W][,mmm])**
**PAGES=([nnnnnnnn][,CANCEL])**
**PAGES=([nnnnnnnn][,C])**
**PAGES=([nnnnnnnn][,DUMP])**
**PAGES=([nnnnnnnn][,D])**
Indicates the maximum number of pages to be printed for this job's sysout data sets and the action to be taken if the maximum is exceeded.

If PAGES is not specified, the installation default for this job class applies.

**nnnnnnnn**
Specifies the number of pages. nnnnnnnn is 1 through 8 decimal numbers from 1 through 16777215.

**WARNING or W**
If the maximum is exceeded, requests that JES3 issue an operator warning message and continue processing.

Any messages about this parameter following the warning message will reflect the number specified on the STANDARD initialization statement or the system default value, **not** the maximum specified in the PAGES parameter.

**mmm**
Specifies the frequency that an operator warning message is to be issued after the maximum specified by nnnnnnnn is exceeded. mmm is a multiple of 10 in the range 10 to 100. mmm is a percentage of nnnnnnnn that is used to calculate the number of additional pages between warning messages. For example, if PAGES=(1000,W,20) is specified, the first warning message is sent to the operator when 1,000 pages of sysout data is reached. Subsequent warning messages are sent when each additional 20 percent of 1,000 is reached (at 1,200 pages, 1,400 pages, and so on). Messages are sent until the job ends or the operator cancels the job.

**CANCEL or C**
If the maximum is exceeded, requests that JES3 cancel the job.

**DUMP or D**
If the maximum is exceeded, requests that JES3 cancel the job and ask for a storage dump.

**PROC=ST**
**PROC=xx**
Names the procedure library that the system is to search for cataloged procedures called by EXEC statements in the job. If a procedure cannot be found in the named library, JES3 abnormally terminates the job.

If this parameter is omitted, the default depends on the source of the job. If the job is submitted as a batch job, the default is ST. If the job is submitted from an internal reader, the default can be another procedure library, as specified by the installation on the STANDARDS initialization statement (the INTPROC, STCPROC, or TSOPROC parameters).

**ST** Indicates the standard procedure library: SYS1.PROCLIB.

**xx** Identifies the last 2 characters of the ddname of a procedure library. xx is defined by the installation (IATPLBxx) in the procedure used to start JES3. If this parameter is coded, only the specified library is searched; SYS1.PROCLIB is not searched.

**RINGCHK=YES**
**RINGCHK=NO**
Indicates whether or not JES3 is to check the status of the tape reel ring for tape devices set up by JES3.

**YES**
Indicates that a validation check is to be made. If the RINGCHK parameter is omitted, YES is the default.

**NO** Indicates that ring checking is to be by-passed for this job.

**SETUP=JOB**
**SETUP=HWS**
**SETUP=THWS**
**SETUP=DHWS**

**SETUP=(stepname.ddname[,stepname.ddname]...)**
**SETUP=**
**(stepname.procstepname.ddname[,stepname.procstepname.ddname]...)**
**SETUP=/(stepname.ddname[,stepname.ddname]...)**
**SETUP=**
**/(stepname.procstepname.ddname[,stepname.procstepname.ddname]...)**

Modifies the standard setup algorithm used in assigning devices to a job before its execution.

If SETUP is omitted, JES3 assigns mountable tape and disk volumes based on an installation default defined at initialization.

**JOB**
Requests job setup, which is allocation of all JES3-managed devices required in the job before the job executes. JES3 mounts the initial volumes necessary to run all steps before the job executes. JOB overrides the SETUP parameter on the JES3 STANDARDS initialization statement.

**HWS**
Requests high watermark setup, which is allocation of the minimum number of devices required to run the job. The minimum number is equal to the greatest number of devices of each type needed for any one job step. High watermark setup does not cause premounting of all mountable volumes.

**THWS**
Requests high watermark setup for tapes but job setup for disks.

**DHWS**
Requests high watermark setup for disks but job setup for tapes.

**stepname.ddname**
**stepname.procstepname.ddname**
Specifies explicit setup, which is allocation of the volumes needed for a DD statement before the job executes. JES3 premounts the indicated volumes. When requesting explicit setup, specify enough devices so that JES3 can allocate all the required devices at any one time. If too few devices are specified, JES3 cancels the job.

Use form **stepname.ddname** to indicate DD statement, ddname, in step, stepname, in this job. Use form **stepname.procstepname.ddname** to indicate DD statement, ddname, in procedure step, procstepname, of a procedure that is called by a step, stepname, in this job. The ddname must match exactly the ddname on the DD statement. (See the example for the //*DATASET statement.)

If you code a list of ddnames and the list cannot be contained on a single statement, SETUP= must be repeated on the continuation statement.

**/stepname.ddname**
**/stepname.procstepname.ddname**
Requests that JES3 **not** explicitly set up any volumes specified in DD statement ddname.

**SPART=partition-name**
Indicates the spool partition in which JES3 is to allocate spool space to this job.

**partition-name**
Specifies the name of the spool partition. **partition-name** is 1 through 8 characters and must match a partition name specified during JES3

initialization. If the name does not match, JES3 ignores the SPART parameter and uses the installation default.

The SPART parameter does not affect allocation for the sysout data sets for the job; these data sets always go to the spool partitions specified during JES3 initialization for the output classes.

If SPART is not specified, JES3 allocates spool data sets to a partition, as follows, in override order:

1. The spool partition for the job's class.
2. The spool partition for the processor executing the job.
3. The default spool partition.

**SYSTEM=ANY**
**SYSTEM=JGLOBAL**
**SYSTEM=JLOCAL**
**SYSTEM=(main-name[,main-name]...)**
**SYSTEM=/(main-name[,main-name]...)**

Indicates the processor that is to execute this job. If a specific processor is named, the processor name must also be specified on the CLASS initialization statement for the job class.

**ANY**
Indicates any global or local system that satisfies the job's requirements.

**JGLOBAL**
Indicates that the job is to run on the global processor only.

**JLOCAL**
Indicates that the job is to run on a local processor only.

**main-name**
Indicates that the job is to run on the named processor or processors.

**/main-name**
Indicates that the job is not to run on the named processor or processors.

**Need for SYSTEM Parameter:** If you omit a SYSTEM parameter, the job runs on the processor used for the job's class. Usually a SYSTEM parameter is not needed. However, if any DD statement UNIT parameter in the job specifies a device-number, a SYSTEM parameter must be coded. JES3 ignores the SYSTEM parameter if either the SYSTEM or SYSAFF parameter is specified on the JOB statement.

**Parameter Agreements:** The following parameters must be consistent with the SYSTEM parameter or JES3 will terminate the job:

- CLASS parameter on the JOB or //*MAIN statement. The requested processor must be assigned to execute jobs in the specified class.
- All devices specified on DD statement UNIT parameters must be available to the requested processor.
- TYPE parameter on the //*MAIN statement must specify the system running on the requested processor.
- Dynamic support programs requested on //*PROCESS statements must be able to be executed on the requested processor.

**THWSSEP=IGNORE**
**THWSSEP=PREFER**
**THWSSEP=REQUIRE**

Indicates whether or not you want scratch tape requests and specific tape

requests separated and whether you want scratch tapes of different media types separated during high watermark processing. This parameter is valid only if high watermark setup (HWS or THWS) is specified on the SETUP parameter or defined at JES3 initialization.

Use this parameter to direct scratch and specific tape requests to different tape drives (for example, you may want JES3 to allocate only scratch tape requests to an IBM 3480 that is equipped with an automatic cartridge loader).

If you omit THWSSEP, JES3 uses an installation default defined at initialization.

**IGNORE**
Specifies that JES3 is not to separate scratch and specific tape requests and not separate scratch tape requests of different media types during high watermark processing. Both scratch and specific tape requests and scratch requests of different media types can be allocated on the same tape drive.

**PREFER**
Specifies that JES3 attempt to allocate scratch and specific tape requests on separate tape drives and attempt to allocate scratch tape requests of different media types on separate tape drives without allocating additional devices. If JES3 cannot separate the requests, scratch and specific tape requests and scratch tape requests of different media types are allocated on the same tape drive.

**REQUIRE**
Specifies that JES3 should not allocate scratch and specific tape requests on the same tape drive and not allocate scratch tape requests of different media types on the same tape drive, even if JES3 must allocate additional tape drives to satisfy the request.

**TRKGRPS=(primary-qty,second-qty)**
Specifies the number of track groups to be assigned to the job. A track group is a number of spool space allocation units. The size of the track group is defined in the GRPSZ parameter on the JES3 BUFFER or SPART initialization statement.

**primary-qty**
Specifies the number of track groups to be initially allocated. This quantity is one decimal number from 1 through 9.

**second-qty**
Specifies the number of track groups to be allocated when the currently allocated groups are filled and more space is needed. This quantity is one decimal number from 1 through 9.

The //*MAIN TRKGRPS parameter overrides a TRKGRPS parameter on the CLASS or MAINPROC initialization statement. However, when a sysout DD statement specifies an output class, the TRKGRPS parameter for that output class overrides the //*MAIN TRKGRPS parameter.

**TYPE=ANY**
**TYPE=VS2**
Indicates the control program that is to execute this job. If you omit a TYPE parameter, the job runs under the control program used for the job's class.

**ANY**
Indicates that JES3 is to use any control program that satisfies the job's requirements. In present systems, JES3 schedules the job on MVS.

**VS2**
Indicates that JES3 is to schedule the job on MVS.

**UPDATE=(dsname[,dsname]...)**
Identifies the procedure library data set(s) that this job is to update. This parameter causes all jobs using the identified data set and any concatenated data sets to be held until the update is complete. See *z/OS JES3 Initialization and Tuning Guide* for information about updating procedure libraries.

**dsname**
Specifies the data set name. The identified data set cannot be concatenated to another data set.

**Note:** If a data set is dynamically allocated as both a JES3 DISKRDR data set and a JES3 PROCLIB data set, the UPDATE = parameter (JES3 procedure library update facility) cannot be used to move the data set.

**USER=userid**
Identifies the job with the specified TSO/E user, even though the job was not submitted via TSO/E by that user. USER allows:

- The TSO/E userid, interacting with a global or local processor, to issue the TSO/E OUTPUT command to access sysout data sets from the job. If the job executes on one processor and the TSO/E userid is attached to another processor, the ACMAIN parameter must identify the processor for the TSO/E userid.
- The TSO/E userid, interacting with any processor, to inquire about the status of the job or to cancel the job.

**userid**
Identifies a TSO/E user. userid is 1 through 7 alphanumeric or national ($, #, @) characters.

## Location in the JCL

When you specify **ORG** on a //*MAIN statement, place the //*MAIN statement before all //*FORMAT statements that do not contain a DEST parameter. If JES3 does not process the ORG parameter before the //*FORMAT statements, JES3 uses the default destination for the //*FORMAT statements; their output is sent to the node where the job entered the system.

When you specify **ORG** on a //*MAIN statement that is part of a remote job, place the //*MAIN statement immediately after the second JOB statement.

## Examples of the //*MAIN statement

***Example 1***

```
//*MAIN SYSTEM=SY1,LINES=(5,C),SETUP=HWS,
//*FAILURE=RESTART,DEADLINE=(0800,A,3,WEEKLY)
```

The job executes on processor SY1. It is estimated to produce not more than 5000 lines of printed output; if the output exceeds 5000 lines, JES3 is to cancel the job. HWS specifies high watermark setup, so JES3 is to allocate the minimum number of devices required for this job. If the system fails, JES3 is to restart the job on the processor SY1. JES3 is to complete this job by 8 a.m. on Tuesday (Tuesday is day number 3) by adjusting the job's scheduling priority using the installation-defined A-type deadline scheduling parameters.

***Example 2***

```
//*MAIN ACMAIN=2,USER=GARYHIL
```

If this statement appears in a job entered from any TSO/E userid on any processor in the complex, then the job's sysout data sets would go to TSO/E userid GARYHIL on processor 2.

## //*NET statement

***Purpose:*** Use the //*NET statement to define the dependencies between jobs in a dependent job control (DJC) network. JES3 sets up a network of dependent jobs and executes them in a specific order. (Once set up, the structure of a DJC network cannot be changed unless all of the jobs in the network are resubmitted.) Jobs belonging to a DJC network cannot be registered with the automatic restart manager (ARM).

### Syntax

```
//*NET   {NETID} =name[,parameter]...
         {ID   }

The parameters are:

  {ABCMP} =  {NOKP}
  {AC   }    {KEEP}

{  {ABNORMAL|AB}  = {D} }
{                   {F} }
{                   {R} }
{  {NORMAL|NC} =    {D} }
{                   {F} }
{                   {R} }

 DEVPOOL=(  {ANY} [,device-name,n]...
            {NET}

 DEVRELSE=  {YES}
            {NO }

  {NETREL}  =(netid,jobname)
  {NR    }

  {NHOLD}  =n
  {HC   }

  {NRCMP}  = {HOLD}
  {PC   }    {NOHO}
             {FLSH}

  {OPHOLD}  = {NO }
  {OH    }    {YES}

  {RELEASE}  =(jobname[,jobname]...)
  {RL     }

  {RELSCHCT}  =n
  {RS      }
```

The //*NET statement consists of the characters //* in columns 1 through 3, NET in columns 4 through 6, a blank in column 7, and parameters in columns 8 through 72. JES3 ignores columns 73 through 80.

## Parameter definition

**NETID=name**
Specifies the name of the DJC network for this job. name is 1 through 8 characters; the first character must be alphabetic.

All jobs put into the system with the same NETID name form a DJC network. To add a job to an existing DJC network, specify the NETID name for that job.

**ABCMP=NOKP**
**ABCMP=KEEP**
Indicates what action JES3 is to take if the job abnormally terminates.

**NOKP**
Indicates that JES3 is to purge the DJC network if the job abnormally terminates and has not been resubmitted by the time the other jobs in the network have completed. JES3 purges the network unless successor jobs or subnetworks are missing. If the ABCMP parameter is omitted, NOKP is the default.

**KEEP**
Indicates that the DJC network is to be kept in the system until (1) the job is resubmitted and completes normally or (2) the operator forces the network from the system. Use KEEP to make sure that the network is not purged until the operator takes proper action.

**Note:** If the job abnormally terminates, you can resubmit it to the DJC network, and the network will be retained until the job completes.

**ABNORMAL=D**
**ABNORMAL=F**
**ABNORMAL=R**
**NORMAL=D**
**NORMAL=F**
**NORMAL=R**
Indicates the action JES3 is to take for this job when any predecessor job completes execution normally or abnormally. If the ABNORMAL parameter is omitted, the default is R, and, if the NORMAL parameter is omitted, the default is D.

**D** Requests that JES3 decrease this job's NHOLD count, which indicates the number of predecessors for this job. When the NHOLD count becomes zero, JES3 can schedule this job.

**F** Requests that JES3 flush this job and its successor jobs from the system. JES3 cancels the job, prints any output, and cancels all successor jobs presently in the system, regardless of their normal or abnormal specifications. However, JES3 admits into the system all successor jobs that enter after the DJC network has been flushed. To flush those jobs, the operator must cancel the jobs or the network.

**R** Requests that JES3 retain this job in the system and not decrease the NHOLD count. R suspends the job and its successor jobs from scheduling until either the predecessor job is resubmitted or the operator decreases the NHOLD count.

**DEVPOOL=(ANY[,device-name,n]...)**
**DEVPOOL=(NET[,device-name,n]...)**
Identifies devices to be dedicated to this DJC network. The system allocates

these devices only to jobs in the network. The DEVPOOL parameter should be coded on the //*NET statement that establishes the network; it is ignored on other //*NET statements.

**ANY**
Indicates that jobs in the network can use any dedicated or undedicated device. JES3 tries to allocate from the dedicated pool before allocating any undedicated devices.

**NET**
Indicates that jobs can use only devices dedicated to the network.

**device-name,n**
Identifies a dedicated device. Code as many device-names with numbers as will fit on one statement. **device-name** specifies (1) a device name defined to JES3 by the installation during initialization or (2) a device-type defined to the system in HCD. **n** is the number of named devices. n is a number from 1 through 32,767.

**DEVRELSE=YES**
**DEVRELSE=NO**
Indicates when devices dedicated to the DJC network are to be released. The DEVRELSE parameter can be coded in several jobs in the network, but must not be coded in the first job. If no network job containing DEVRELSE=YES completes, the system releases the devices when it purges the network.

**YES**
Requests that JES3 release all devices at the end of this job. Completion of any job that specified DEVRELSE=YES causes the devices dedicated to the network to be released.

**NO** Requests that JES3 release all devices only when the last job in the network ends.

**NETREL=(netid,jobname)**
Indicates that this job must be executed before the named job in another DJC network can be executed. The NETREL parameter can be specified only once for each job of a DJC network.

**netid**
Identifies the NETID for the successor job.

**jobname**
Names the JOB statement for the successor job.

**NHOLD=n**
Indicates the number of predecessor job completions required before this job can be released for scheduling. The predecessor number can include jobs from another DJC network. **n** is a number from 0 through 32,767.

When the predecessor number reaches 0, the job is scheduled for execution. The system reduces this number:

- When each predecessor job completes execution.
- By operator command.
- When a program in a predecessor job issues an assembler DJC WTO macro.

If you specify NHOLD=0 or omit the NHOLD parameter, this job has no predecessor jobs. JES3 can schedule it for immediate execution.

If the NHOLD count is incorrect, the following can occur:

- If n is greater than the actual number of predecessor jobs, JES3 does not release this job for execution when all of its predecessor jobs complete execution.
- If n is less than the actual number of predecessor jobs, JES3 prematurely releases the job for execution.

**NRCMP=HOLD**
**NRCMP=NOHO**
**NRCMP=FLSH**

Indicates that a network job that completed normally is being resubmitted and that JES3 must erase all references to the job before the job reenters the network.

**HOLD**
Indicates that JES3 is to hold the job until it is released by the operator.

**NOHO**
Indicates that JES3 is to allow the job to be scheduled as system resources become available.

**FLSH**
Indicates that JES3 is to flush the job from the system.

**OPHOLD=NO**
**OPHOLD=YES**

**NO** Indicates that the job is to be processed normally without operator intervention. If OPHOLD is omitted, NO is the default.

**YES**
Indicates that JES3 is to hold the job until it is released by the operator.

**RELEASE=(jobname[,jobname]...)**

Indicates that this job must be executed before the named job(s) in this DJC network can be executed.

**jobname**
Names the JOB statement for a successor job. You can specify from 1 through 50 successor jobnames.

RELEASE is the only parameter on the //*NET statement that can be split and continued on the next statement. To continue the RELEASE parameter, end the statement with the comma following a jobname and continue the next statement with the next jobname. The left parenthesis appears at the beginning of the jobname list and the right parenthesis appears at the end of the list. For example:

```
//*NET NETID=EXP1,RELEASE=(JOB35,JOB27Z,MYJOB,
//*WRITJB,JOBABC)
```

**RELSCHCT=n**

Controls early set up of a dependent job's resources. Set up begins when the NHOLD count becomes less than or equal to n. n is a number from 1 through 32,767.

If you specify RELSCHCT=0 or omit the RELSCHCT parameter, JES3 does not set up dependent jobs early.

**Note:** Use this parameter carefully; RELSCHCT can tie up devices and data sets for long times. Do not specify the RELSCHCT parameter:

- For a job that may have catalog dependencies.

- For a job that contains one or more //*PROCESS statements.

## Location in the JCL

Place the //*NET statement for a job after the JOB statement and before the first EXEC statement. Code only one //*NET statement for each job in a DJC network.

The //*NET statement must precede any //*PROCESS statements.

## Examples of the //*NET statement

***Example 1***

```
//*NET  NETID=NET01,NHOLD=0,DEVPOOL=(,3330,2)
```

This statement defines a DJC network named NET01. The network contains no predecessor jobs. The DEVPOOL parameter, which must be coded in the first job in the network, requests that JES3 establish a device pool of two 3330s for network NET01.

***Example 2***

```
//*NET NETID=N1,RELEASE=B,NETREL=(N2,B2)
```

This statement adds a job to the DJC network named N1. This job must be executed before job B, which is in N1, and before job B2, which is in the DJC network named N2.

# //*NETACCT statement

***Purpose:*** Use the //*NETACCT statement to specify accounting information that JES3 is to transmit with a job to another node in the network.

## Syntax

```
//*NETACCT parameter[,parameter]...

The parameters are:

      PNAME=programmer's-name
      ACCT=number
      BLDG=address
      DEPT=dept
      ROOM=room
      USERID=userid
```

- The //*NETACCT statement consists of the characters //* in columns 1 through 3, NETACCT in columns 4 through 10, a blank in column 11, and parameters in columns 9 through 72. JES3 ignores columns 73 through 80.
- Do not continue a //*NETACCT statement. If the parameters cannot fit on one statement, code more than one //*NETACCT statement.
- Enclose any parameter value that contains special characters, including embedded blanks, in apostrophes.

## Parameter definition

**PNAME=programmer's-name**
Identifies the programmer. programmer's-name is 1 through 20 characters.

**ACCT=number**
Gives the network account number. number is 1 through 8 characters.

**BLDG=address**
Gives the programmer's building address. address is 1 through 8 characters.

**DEPT=dept**
Gives the programmer's department number. dept is 1 through 8 characters.

**ROOM=room**
Gives the programmer's room number. room is 1 through 8 characters.

**USERID=userid**
Gives the programmer's network userid. userid is 1 through 8 characters.

## Defaults

For any //*NETACCT parameter that is omitted, JES3 uses an installation default specified at JES3 initialization.

## Location in the JCL

Place the //*NETACCT statement(s) for a job stream to be transmitted immediately after the first JOB statement and before any //*ROUTE XEQ or //XMIT statements.

Place the //*NETACCT statement(s) for a SYSOUT stream to be transmitted immediately after the first JOB statement and before any //*MAIN statements specifying ORG=nodename.

For jobs running at the submitting system and potentially having the destination changed to a network destination via an output service modify command (*MODIFY,U ...), place the //*NETACCT statement(s) for the SYSOUT immediately after the JOB statement.

## Example of the //*NETACCT statement

```
//*NETACCT    PNAME=COLLINS,ACCT=D58D921,USERID=NXT
```

# //*OPERATOR statement

***Purpose:*** Use the //*OPERATOR statement to issue a message to the operator. Columns 1 through 80 are written on the operator console and in the job's hard-copy log when JES3 reads in the job.

## Syntax

```
//*OPERATOR  message
```

The //*OPERATOR statement consists of the characters //* in columns 1 through 3, OPERATOR in columns 4 through 11, a blank in column 12, and the message for the operator in columns 13 through 80.

## Location in the JCL

Place the //*OPERATOR statement anywhere after the JOB statement.

### Example of the //*OPERATOR statement

```
//*OPERATOR CALL EXT. 55523 WHEN THIS JOB STARTS
```

## //**PAUSE statement

***Purpose:*** Use the //**PAUSE statement to halt an input reader temporarily. When you enter a //**PAUSE statement through an input reader, JES3 issues a message and waits for the operator to reply. To start the input reader, the system operator must issue a *START command or a remote work station with console level 15 must send a start message.

The //**PAUSE statement is intended primarily for system checkout and test. It should be issued only by remote work stations.

### Syntax

| |
|---|
| `//**PAUSE  [comments]` |
| The //**PAUSE statement consists of the characters //** in columns 1 through 4, PAUSE in columns 5 through 9, a blank in column 10, and, optionally, comments starting in any column beginning with 11. JES3 ignores columns 73 through 80. |

### Location in the JCL

Place the //**PAUSE statement before the first JOB statement in an input stream. If it appears after the first JOB statement, JES3 ignores it.

### Example of the //**PAUSE statement

```
//**PAUSE  THIS IS A TEST.
```

## //*PROCESS statement

***Purpose:*** Use the //*PROCESS statement to control how JES3 processes a job. A job that contains //*PROCESS statements receives only the JES3 processing specified on the //*PROCESS statements plus certain required processing.

Specifically, the //*PROCESS statement calls a dynamic support program (DSP) in the DSP dictionary. JES3 must be able to process the called DSP.

***Standard job processing:*** JES3 uses a series of processing functions to process a job. Standard processing consists of only the standard scheduler functions:

- Converter/interpreter service
- Main service
- Output service
- Purge service

***Nonstandard job processing:*** A nonstandard job uses one or more special processing functions in place of or in addition to standard processing or skips one or more of the standard functions. Specify a nonstandard job by following the JOB statement with a JES3 //*PROCESS statement for each processing function.

***Use of nonstandard job processing:*** Nonstandard job processing is useful in testing. For example, a //*PROCESS statement can make JES3 bypass program execution

so that the job's JCL can be checked. Another //*PROCESS statement can make JES3 bypass output processing; then the operator can check by inquiry command whether the job reached execution.

If the job generates spin data sets during main execution, the next scheduler element will not be processed until the spin data sets have been processed. To avoid long waits or system hangs, make sure that the OUTSERV scheduler element is the next scheduler element after main processing.

## Syntax

```
//*PROCESS dsp
[parameter[,parameter]...]
```

The //*PROCESS statement consists of the characters //* in columns 1 through 3, PROCESS in columns 4 through 10, a blank in column 11, and the DSP name beginning in column 12. The rest of the columns must be blank.

If the requested DSP requires parameters, code them on the following statement. The parameter statement consists of parameters in columns 1 through 72, separated by commas. Columns 73 through 80 must be blank. Only one parameter statement after a //*PROCESS statement is allowed, any others are ignored by JES3.

## Parameter definition

**dsp**

Identifies the DSP that JES3 is to use in processing the job. Table 30 lists the valid DSP names and whether parameters can follow.

*Table 30. DSPs for JES3 //*PROCESS Statements*

| DSP | DSP function | Parameters |
|---|---|---|
| **Standard processing functions:** | | |
| CI | JES3 Converter/Interpreter Service, which interprets the JCL and creates control blocks. | Yes (See *z/OS JES3 Commands*) |
| MAIN | Main Service, which processes the program. | No |
| OUTSERV | Output Service, which processes the job's output. | No |
| PURGE | Purge Service, which purges the job. This is the last function in any job. JES3 automatically creates this DSP. | No |
| **Nonstandard processing functions:** | | |
| CBPRNT | Control Block Print | Yes (See *z/OS JES3 Commands*) |
| DISPDJC | Display Dependent Job Control | Yes (See *z/OS JES3 Commands*) |
| DISPLAY | Display Job Queues | Yes (See *z/OS JES3 Commands*) |
| DJCPROC | Invoke Dependent Job Control Updating<br><br>**Note:** A //*PROCESS DJCPROC statement is required only when a //*PROCESS MAIN statement is not coded. | No |

*Table 30. DSPs for JES3 //*PROCESS Statements (continued)*

| DSP | DSP function | Parameters |
|---|---|---|
| **Standard processing functions:** | | |
| DR | Disk Reader | Yes (See *z/OS JES3 Commands*) |
| ISDRVR | Input Service Driver (JES3 Control Statement Processing) | Yes (Qualified ddname of input data set) |
| JESNEWS | Use JESNEWS Facility | Yes (See *z/OS JES3 Commands*) |
| xxx | User-written DSP | (See *z/OS JES3 Customization*) |

## Location in the JCL

- Place all //*PROCESS statements for a job immediately after the JOB statement and before the first EXEC statement. If the job includes a //*NET statement, the //*NET statement must appear between the JOB statement and the first //*PROCESS statement.
- The //*PROCESS statements can be separated only by their parameter statements.
- JES3 processes the //*PROCESS statements in the order in which they appear in the input stream.
- The first //*PROCESS statement must request an interpreter DSP if you want input service error messages, which indicate that a job is to be scheduled for interpreter processing before being purged.

## Examples of the //*PROCESS statement

***Example 1***

```
//EXAM1  JOB
//*PROCESS  CI
//*PROCESS  MAIN
//*PROCESS  OUTSERV
//S1     EXEC PGM=ANY
          .
          .
         JCL statements
          .
```

This example shows how to submit a simple job via //*PROCESS statements. It is processed like a standard job. The four standard scheduler functions are used for the job: CI, MAIN, OUTSERV, and PURGE. Note that PURGE is not specified; JES3 automatically creates this DSP.

***Example 2***

```
//EXAM2  JOB
//*PROCESS  CI
//*PROCESS  MAIN
//*PROCESS  OUTSERV
//*PROCESS  PLOT
//*ENDPROCESS
//S1     EXEC PGM=ANY
//DD1    DD  ...
          .
          .
         JCL statements
          .
```

This example shows how to request a user-written DSP: PLOT. PLOT is to be executed after output service has completed. Note that PURGE is again not specified but is automatically created.

***Example 3***

```
//EXAM3  JOB
//*PROCESS  OUTSERV
//*FORMAT   PR,DDNAME=S1.DS1,COPIES=5
//*DATASET  DDNAME=S1.DS1
      .
      .
      data
      .
      .
//*ENDDATASET
//S1  EXEC PGM=ANY
//DS1 DD   DSNAME=DATA1
      .
      .
```

This example uses JES3 output service and the //*DATASET statement. Five copies of data set DS1 are printed on any local printer.

## //*ROUTE XEQ statement

***Purpose:*** Use the //*ROUTE XEQ statement to send the following input stream to a network node where the job is then executed. JES3 stops transmitting input stream records when it finds one of the following:

- The second JOB statement after the //*ROUTE XEQ statement.
- The input stream runs out of records.

All output from the job is assumed to print/punch at the originating node unless otherwise specified on a DEST parameter.

The //*ROUTE XEQ statement must be given 80 character records.

### Syntax

```
//*ROUTE XEQ nodename[.vmguestid]
```

The //*ROUTE XEQ statement consists of the characters //* in columns 1 through 3, ROUTE in columns 4 through 8, a blank in column 9, and, starting in any column from 10 through 72: XEQ, followed by at least one blank and then parameters. JES3 ignores columns 73 through 80.

Do not imbed blanks in the nodename or vmguestid parameters.

### Parameter definition

**nodename**

Indicates the node. The nodename identifies an MVS JES2 system, an MVS JES3 (global) system, a VSE POWER node, or a VM system.

If nodename specifies a local node:

- The job executes locally if the job begins with a JOB statement.
- The job is terminated if the job begins with an NJE statement.

**.vmguestid**
Identifies a guest system running in a virtual machine (VM), for example, an MVS system running under VM.

**Note:** Do not specify a work station or terminal in this parameter.

## Location in the JCL

- Place the //*ROUTE XEQ statement after a JOB statement that is valid for the submitting location and any //*NETACCT statements.
- JES3 requires a MVS JOB statement immediately after the //*ROUTE XEQ statement.
- If the destination node is not a MVS system, any statement immediately following the MVS JOB statement must be a valid JOB statement for the executing node.

## JOB Statement after //*ROUTE XEQ

An error in the //*ROUTE XEQ statement can cause the JOB statement following the //*ROUTE XEQ to be processed at the submitting node. To prevent this, code NJB instead of JOB on the second JOB statement; JES3 changes the NJB to JOB before transmitting the job.

**Note:**

1. TSO/E users must code NJB instead of JOB on the second JOB statement.
2. If an MVS JOB statement is not immediately following the //*ROUTE XEQ statement, the XMIT JCL statement must be used instead of //*ROUTE XEQ.

## Example of the //*ROUTE XEQ statement

```
//JOBN1   JOB   options ...
//*ROUTE  XEQ   2
//JOBN2   JOB   options ...
//STEP1   EXEC  PGM=REPORTER
//DD1     DD    SYSOUT=A,DEST=N1R33
//DD2     DD    SYSOUT=A,DEST=N2R33
//DD3     DD    SYSOUT=B,DEST=R33
//DDIN    DD    *
          .
          .
          data
          .
/*
```

In this example, JOB statement JOBN1 is entered through the JES3 system at node 1. The //*ROUTE XEQ statement tells JES3 to send the following input stream to node 2. Transmission of the input stream is stopped by the /* delimiter statement. JOB statement JOBN2 and all following statements until the delimiter are read and executed by the system at node 2.

The sysout data sets are sent to two work stations:

- Sysout data set DD1 is produced at work station 33 attached to node 1.
- Sysout data set DD2 is produced at work station 33 attached to node 2.
- Sysout data set DD3 is produced at work station 33 attached to node 1. Because no node is specified, the originating node is assumed.

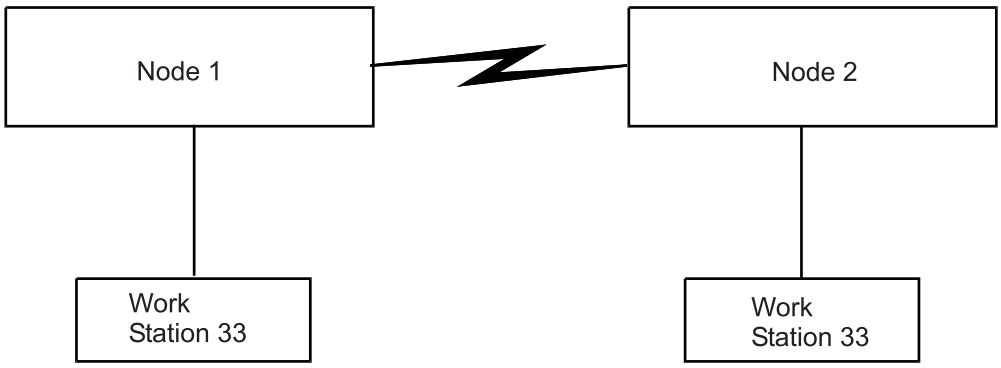


*Figure 2. Example //*ROUTE XEQ statement*

# /*SIGNOFF statement

***Purpose:*** Use the /*SIGNOFF statement to tell JES3 to end a remote job stream processing session. At the completion of the current print and/or punch streams, JES3 disconnects the remote work station from the system. If JES3 is reading jobs from the station when the output completes, JES3 disconnects the station when the input is completed.

Both systems network architecture (SNA) and binary synchronous communication (BSC) remote work stations use the /*SIGNOFF statement.

***References:*** For more information on the /*SIGNOFF command, see *z/OS JES3 Initialization and Tuning Reference*.

## Syntax

| /*SIGNOFF |
|---|
| The /*SIGNOFF statement consists of the characters /* in columns 1 and 2, SIGNOFF in columns 3 through 9, and blanks in columns 10 through 80.<br><br>Note that, unlike other JES3 statements, this statement starts with only one slash. |

## Location in the JCL

The /*SIGNOFF statement can appear anywhere in a local input stream or an input stream from a SNA or BSC remote work station.

## Example of the /*SIGNOFF statement

```
/*SIGNOFF
```

This statement requests that JES3 terminate a remote job stream processing session.

# /*SIGNON statement

***Purpose:*** Use the /*SIGNON statement to tell JES3 to begin a remote job stream processing session. The /*SIGNON statement can override the remote identification number normally assigned to the remote work station. This statement is optional for all work stations except non-multi-leaving remote stations on a switched line.

Systems network architecture (SNA) remote work stations must use the LOGON command instead of the /*SIGNON statement to notify JES3 of a connection request.

***References:*** For information on the LOGON command, see *z/OS JES3 Initialization and Tuning Reference* and *z/OS V2R1.0 Communications Server: SNA Programming*.

## Syntax

```
/*SIGNON  work-station-name {A|(blank)} {R|(blank)} passwd1  passwd2  new-passwd
```

The /*SIGNON statement consists of the following:

| Column | Contents |
|---|---|
| **1-2** | /* |
| **3-8** | SIGNON |
| **9-15** | blanks |
| **16-20** | work-station-name, beginning in 16 |
| **21** | blank |
| **22** | A or a blank |
| **23** | R or a blank |
| **24** | blank |
| **25-32** | password1, beginning in 25 |
| **33-34** | blanks |
| **35-42** | password2, beginning in 35 |
| **43** | blank |
| **44-51** | new-password, beginning in 44 |
| **52-80** | blanks |

Note that, unlike other JES3 statements, this statement starts with only one slash.

## Parameter definition

**work-station-name**
Specifies the name of the remote work station. The work-station-name is 1 through 5 characters and must have been defined on a JES3 RJPTERM initialization statement.

**A** Indicates an automatic reader. A can be coded only when the work station is a programmable terminal. Leave this column blank if you do not want to specify an automatic reader.

**R** Indicates that print or punch output will be rescheduled if the needed device is not ready. R can be coded only when the work station is a nonprogrammable terminal. Leave this column blank if you do not want to specify the R option.

**password1**
Specifies the password for the remote job processing (RJP) line. This parameter is one through eight characters and must have been initially defined at system initialization.

**password2**
Specifies the current password for the work station. This parameter is one through eight characters and must have been initially defined at system initialization.

**new-password**
Specifies a new password for the work station. This parameter is one through eight characters.

## Location in the JCL

Place the /*SIGNON statement at the start of an input stream to be transmitted from a remote work station.

## Example of the /*SIGNON statement

```
/*SIGNON       QUIN  A  PSWD1     PSWD2
```

This statement requests that remote work station QUIN begin a remote job stream processing session. The value **A** in column 22 specifies an automatic reader for the programmable terminal. PSWD1, beginning in column 25, is the password assigned to a dial line. PSWD2, beginning in column 35, is the password assigned to the remote work station.

To change the current password PSWD2 for the remote work station, the preceding /*SIGNON statement can be specified as:

```
/*SIGNON       QUIN  A  PSWD1     PSWD2    PSWDNEW
```

This statement assigns PSWDNEW, beginning in column 44, as the new password for the remote work station QUIN.

# JES2 processing of JES3 control statements

Starting in Version 2 Release 2 of z/OS, JES2 supports some JES3 control statements.

When relevant JES2 support is enabled, jobs containing JES3 control statements can be submitted to the JES2 subsystem. Normally, JES2 treats JES3 control statements as JCL comments. Depending on JES2 configuration, JES3 control statements can be ignored, recognized and flagged as an error, or processed. When JES2 is configured to process a JES3 control statement, JES2 will map the requested JES3 function to a similar function supported by JES2. For JES3 functions that do not have JES2 equivalent, a warning message is issued.

For details on now to configure JES2 to enable this support and which JES3 statements are currently supported by JES2, see the descriptions of INPUTDEF and JECLDEF JES2 initialization statements in *z/OS JES2 Initialization and Tuning Reference* or the descriptions of INPUTDEF and JECLDEF JES2 commands in *z/OS JES2 Commands*.

# Appendix. Accessibility

Accessible publications for this product are offered through IBM Knowledge Center (http://www.ibm.com/support/knowledgecenter/SSLTBW/welcome).

If you experience difficulty with the accessibility of any z/OS information, send a detailed message to the "Contact us" web page for z/OS (http://www.ibm.com/systems/z/os/zos/webqs.html) or use the following mailing address.

IBM Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
United States

## Accessibility features

Accessibility features help users who have physical disabilities such as restricted mobility or limited vision use software products successfully. The accessibility features in z/OS can help users do the following tasks:

- Run assistive technology such as screen readers and screen magnifier software.
- Operate specific or equivalent features by using the keyboard.
- Customize display attributes such as color, contrast, and font size.

## Consult assistive technologies

Assistive technology products such as screen readers function with the user interfaces found in z/OS. Consult the product information for the specific assistive technology product that is used to access z/OS interfaces.

## Keyboard navigation of the user interface

You can access z/OS user interfaces with TSO/E or ISPF. The following information describes how to use TSO/E and ISPF, including the use of keyboard shortcuts and function keys (PF keys). Each guide includes the default settings for the PF keys.

- *z/OS TSO/E Primer*
- *z/OS TSO/E User's Guide*
- *z/OS V2R2 ISPF User's Guide Vol I*

## Dotted decimal syntax diagrams

Syntax diagrams are provided in dotted decimal format for users who access IBM Knowledge Center with a screen reader. In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), they can appear on the same line because they are considered a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or 3.1 or 3.1.1. To hear these numbers correctly, make sure that the screen reader is set to read out

© Copyright IBM Corp. 1988, 2015

punctuation. All the syntax elements that have the same dotted decimal number (for example, all the syntax elements that have the number 3.1) are mutually exclusive alternatives. If you hear the lines 3.1 USERID and 3.1 SYSTEMID, your syntax can include either USERID or SYSTEMID, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, it is preceded by the backslash (\) character. The * symbol is placed next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element *FILE with dotted decimal number 3 is given the format 3 * FILE. Format 3* FILE indicates that syntax element FILE repeats. Format 3* * FILE indicates that syntax element * FILE repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol to provide information about the syntax elements. For example, the lines 5.1*, 5.1 LASTRUN, and 5.1 DELETE mean that if you use more than one of the LASTRUN and DELETE syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, it indicates a reference that is defined elsewhere. The string that follows the % symbol is the name of a syntax fragment rather than a literal. For example, the line 2.1 %OP1 means that you must refer to separate syntax fragment OP1.

The following symbols are used next to the dotted decimal numbers.

**? indicates an optional syntax element**
The question mark (?) symbol indicates an optional syntax element. A dotted decimal number followed by the question mark symbol (?) indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element, (for example 5? NOTIFY). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that the syntax elements NOTIFY and UPDATE are optional. That is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.

**! indicates a default syntax element**
The exclamation mark (!) symbol indicates a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicate that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the dotted decimal number can specify the ! symbol. For example, if you hear the lines 2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the

default option for the FILE keyword. In the example, if you include the FILE keyword, but do not specify an option, the default option KEEP is applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, the default FILE(KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1! (KEEP), and 2.1.1 (DELETE), the default option KEEP applies only to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.

*** indicates an optional syntax element that is repeatable**

The asterisk or glyph (*) symbol indicates a syntax element that can be repeated zero or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line 5.1* data area, you know that you can include one data area, more than one data area, or no data area. If you hear the lines 3* , 3 HOST, 3 STATE, you know that you can include HOST, STATE, both together, or nothing.

**Notes:**

1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.
2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you can write HOST STATE, but you cannot write HOST HOST.
3. The * symbol is equivalent to a loopback line in a railroad syntax diagram.

**+ indicates a syntax element that must be included**

The plus (+) symbol indicates a syntax element that must be included at least once. A dotted decimal number followed by the + symbol indicates that the syntax element must be included one or more times. That is, it must be included at least once and can be repeated. For example, if you hear the line 6.1+ data area, you must include at least one data area. If you hear the lines 2+, 2 HOST, and 2 STATE, you know that you must include HOST, STATE, or both. Similar to the * symbol, the + symbol can repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loopback line in a railroad syntax diagram.

# Notices

This information was developed for products and services offered in the U.S.A. or elsewhere.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

© Copyright IBM Corp. 1988, 2015

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Site Counsel
IBM Corporation
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

COPYRIGHT LICENSE:

This information might contain sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

## Policy for unsupported hardware

Various z/OS elements, such as DFSMS, HCD, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted

for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

## Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: IBM Lifecycle Support for z/OS (http://www.ibm.com/software/support/systemsz/lifecycle/)
- For information about currently-supported IBM hardware, contact your IBM representative.

## Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available at Copyright and Trademark information (http://www.ibm.com/legal/copytrade.shtml).

# Index

## Special characters


, (comma)
  in syntax 21
  use of in parameter field 15
  when coded in bracket or brace 20
/ (slash)
  in syntax 21
/ (slash) subparameter
  of //*MAIN FETCH parameter 671
  of //*MAIN SETUP parameter 675
  of //*MAIN SYSTEM parameter 676
//**PAUSE control statement
  description 685
  example 685
  in JES3 685
  location in JCL 685
//*DATASET control statement
  description 646
  example 648
  in JES3 646, 647, 648
  location in JCL 648
  parameter 647
//*ENDDATASET control statement
  description 648
  example 648
  in JES3 648
  location in JCL 648
//*ENDPROCESS control statement
  description 649
  example 649
  in JES3 649
  location in JCL 649
//*FORMAT PR control statement
  description 649
  example 658
  in JES3 649, 651, 658
  location in JCL 658
  parameter 651
  relationship to //*PROCESS statement 658
  relationship to sysout DD and OUTPUT JCL statement 658
//*FORMAT PU control statement
  description 659
  example 664
  in JES3 659, 660, 663, 664
  location in JCL 663
  parameter 660
  relationship to //*PROCESS statement 663
  relationship to sysout DD and OUTPUT JCL statement 663
//*MAIN control statement
  description 664
  example 678
  in JES3 664, 666, 678
  location in JCL 678
  parameter 666
//*NET control statement
  description 679
//*NET control statement *(continued)*
  DJC (dependent job control) network 679
  example 683
  in JES3 679, 680, 683
  location in JCL 683
  parameter 680
//*NETACCT control statement
  default 684
  description 683
  in JES3 683, 684
  location in JCL 684
  parameter 683
//*OPERATOR control statement
  description 684
  example 685
  in JES3 684, 685
  location in JCL 684
//*ROUTE XEQ control statement
  description 688
  example 689
  in JES3 688, 689
  location in JCL 689
  parameter 688
/* (slash asterisk)
  as delimiter statement 329
/*DEL control statement
  submitting jobs to internal reader 611, 643
  with XMIT JCL statement 583, 588
/*EOF control statement
  submitting jobs to internal reader 611, 643
  with XMIT JCL statement 583, 588
/*JOBPARM control statement
  description 613
  example 619
  in JES2 613, 614, 618, 619
  location in JCL 618
  override 618
  parameter 614
/*MESSAGE control statement
  description 619
  example 620
  in JES2 619, 620
  location in JCL 619
  relationship to /*ROUTE XEQ statement 619
/*NETACCT control statement
  default 620
  description 620
  example 620
  in JES2 620
  location in JCL 620
  override 620
  parameter 620
/*NOTIFY control statement
  description 621
  example 622
  in JES2 621, 622
  location in JCL 622
/*NOTIFY control statement *(continued)*
  override 621
  parameter 621
/*OUTPUT control statement
  description 622
  example 630
  in JES2 622, 624, 630
  location in JCL 630
  override 630
  parameter 624
  relationship to other control statement 630
/*PRIORITY control statement
  description 631
  example 632
  in JES2 631, 632
  location in JCL 631
  override 631
  parameter 631
  relationship to other control statement 631
/*PURGE control statement
  submitting jobs to internal reader 611
/*ROUTE control statement
  /*ROUTE PRINT 632, 634
  /*ROUTE PUNCH 632, 634
  /*ROUTE XEQ 632, 634
  description 632
  example 634
  in JES2 632, 634
  location in JCL 634
  multiple statement 634
  parameter 632
  processing 634
/*ROUTE PRINT
  description 632
/*ROUTE PUNCH
  description 632
/*ROUTE XEQ
  description 632
  relationship to /*MESSAGE statement 619
/*SCAN control statement
  submitting jobs to internal reader 611
/*SETUP control statement
  description 635
  example 636
  in JES2 635, 636
  location in JCL 636
  parameter 635
/*SIGNOFF control statement
  description 636, 690
  example 636, 690
  in JES2 636
  in JES3 690
  location in JCL 636, 690
/*SIGNON control statement
  description 637, 690
  example 638, 692


© Copyright IBM Corp. 1988, 2015

/*SIGNON control statement *(continued)*
in JES2 637, 638
in JES3 690, 691, 692
location in JCL 638, 692
parameter 637, 691
/*XEQ control statement
description 639
example 640
in JES2 639, 640
location in JCL 640
multiple statement 640
parameter 639
/*XMIT control statement
default 642
description 640
example 642
in JES2 640, 641, 642
location in JCL 642
parameter 641
. (period)
in syntax 21
.. (two consecutive periods)
in syntax 21
... (ellipsis)
in syntax 20
( ) (parentheses)
in syntax 21
* (asterisk)
as code parameter of JES2 /*OUTPUT statement 624
in syntax 21
relationship to DD DATA parameter 129
* parameter
default 101
description 101
example 103
location in JCL 102
of DD statement 101, 102, 103
relationship to other control statement 102
relationship to other parameter 101
unread record 103
* subparameter
of /*JOBPARM SYSAFF parameter 617
of DD SYSOUT parameter 284
of JOB RESTART parameter 451
of OUTPUT JCL CLASS parameter 490
*.ddname subparameter
description 23, 153
of DD DCB parameter 137
of DD DSNAME parameter 182
of DD REFDD parameter 257
of VOLUME=REF subparameter 305
*.label subparameter
of DD CNTL parameter 125
*.name subparameter
description 23
of DD OUTPUT parameter 234
*.procstepname.ddname subparameter
description 23
of VOLUME=REF subparameter 305
*.stepname.ddname subparameter
description 23, 153
of DD DCB parameter 137
*.stepname.ddname subparameter *(continued)*
of DD DSNAME parameter 182
of DD REFDD parameter 257
of EXEC PGM parameter 359
of VOLUME=REF subparameter 305
*.stepname.label subparameter
of DD CNTL parameter 125
*.stepname.name subparameter
description 23
of DD OUTPUT parameter 234
*.stepname.procstepname.ddname subparameter
description 23, 153
of DD DCB parameter 137
of DD DSNAME parameter 182
of DD REFDD parameter 257
of EXEC PGM parameter 359
of VOLUME=REF subparameter 305
*.stepname.procstepname.label subparameter
of DD CNTL parameter 125
*.stepname.procstepname.name subparameter
description 23
of DD OUTPUT parameter 234
[ ] (brackets)
in syntax 20
| (logical or)
in syntax 19
| (OR) operator
of IF/THEN/ELSE/ENDIF statement construct 381
& (ampersand)
in syntax 21
& (AND) operator
of IF/THEN/ELSE/ENDIF statement construct 381
&&dsname subparameter
of DD DSNAME parameter 181
+HH:MM subparameter
of HOLDUNTL parameter 572
of STARTBY parameter 574
= (equal sign)
in syntax 21
' (apostrophe)
use to enclose special character 22
use with special character 22
when not needed to enclose special character 22
{ } (braces)
in syntax 20

# Numerics

1440 subparameter
of EXEC TIME parameter 371
of JOB TIME parameter 460
3211 Printer with indexing feature
specifying indexing of left margin 519, 520
specifying indexing of right margin 523
3480 Magnetic Tape Subsystem
specifying in UNIT parameter 294
3540 diskette input/output unit
with DD * statement 102
3540 diskette input/output unit *(continued)*
with DD DATA parameter 130
with DD DCB parameter 139
with DD DSID parameter 174
with VOLUME=SER subparameter 309
3800 Printing Subsystem
DD BURST parameter 117
DD CCSID parameter 119
DD CHARS parameter 121
OUTPUT JCL BURST parameter 485
OUTPUT JCL CHARS parameter 486
specifying copy group 127, 495, 654

# A

A parameter
of JES3 /*SIGNON statement 691
A subparameter
of DCB BFTEK subparameter 140
of DCB OPTCD subparameter 146
of DCB PCI subparameter 148
of RECFM parameter 252, 253
A11 character set
for 3211 printer 290, 556
AB parameter
of JES3 //*NET statement 679
ABCMP parameter
of JES3 //*NET statement 680
ABE subparameter
of DCB EROPT subparameter 144
abend condition
with IF/THEN/ELSE/ENDIF statement construct 382
ABEND keyword
of IF/THEN/ELSE/ENDIF statement construct 382
ABENDCC keyword
of IF/THEN/ELSE/ENDIF statement construct 382
abnormal
dump 319
evaluating 383
ABNORMAL parameter
of JES3 //*NET statement 680
ABSTR subparameter
of DD SPACE parameter 270
AC parameter
of JES3 //*NET statement 679
ACB (access method control block) 105
ACC subparameter
of DCB EROPT subparameter 144
access method
for dummy data set 190
access method control block 105
access-code subparameter
of DD ACCODE parameter 104
accessibility 693
contact IBM 693
features 693
ACCODE parameter
default 105
description 104
example 105
of DD statement parameter 104, 105
override 105

ACCODE parameter *(continued)*
subparameter 104
account-number subparameter
of JOB accounting information parameter 410
accounting information 60
accounting-information parameter
description 409
example 411
JES2 format 410
JES2 processing of invalid subparameter 411
of JOB statement 409, 410, 411
overrides of subparameters in JES2 format 411
relationship to other control statement 410
specified on JES3 //*NETACCT statement 683
subparameter 410
subparameters for JES2 format 410
accounting-information subparameter
of EXEC ACCT parameter 339
of JOB accounting information parameter 410
ACCT parameter
description 338
example 339
of EXEC statement 338, 339
of JES3 //*NETACCT statement 684
subparameter 339
ACMAIN parameter
of JES3 //*MAIN statement 666
ACS routine
with DD DATACLAS parameter 134
with DD MGMTCLAS parameter 229
with DD STORCLAS parameter 276
ADDRESS parameter
description 480
of OUTPUT JCL statement 480, 481, 525, 526, 527, 528, 547
subparameter 481, 525, 526, 527, 528, 547
address subparameter
of //*NETACCT BLDG parameter 684
of DD SPACE parameter 270
ADDRSPC parameter
default 340, 412
description 340, 412
example 341, 413
of EXEC statement 340, 341
of JOB statement 412, 413
override 340, 412
relationship to REGION parameter 341, 412
subparameter 340, 412
administrator
with DD DATACLAS parameter 132
with DD MGMTCLAS parameter 228
with DD STORCLAS parameter 275
with DD UNIT parameter 293
with DD VOLUME=SER subparameter 305
AFF subparameter
of DD UNIT parameter 296
AFPSTATS parameter
default 483
description 483
example 484
of OUTPUT JCL statement 483, 484
override 483, 484
subparameter 483
AFTER statement
comments field 607
error on statement 607
example 607
in JCL 605, 607
location in JCL 607
name field 605
operation field 605
parameter field 605
AL subparameter
of DD LABEL parameter 218
ALIGN subparameter
of DD FCB parameter 196
alignment
of printing form 196
ALL subparameter
of JES3 //*MAIN FETCH parameter 671
of OUTPUT JCL JESDS parameter 521
ALX subparameter
of DD SPACE parameter 270
AMORG subparameter
of DD AMP parameter 107
AMP parameter
description 105
example 112
of DD statement 105, 107, 111, 112
relationship to other parameter 111
subparameter 107
with DSNAME parameter 182
AN character set
for 1403 printer 290, 556
for 3203 Model 5 printer 290, 556
AND (&) operator
of IF/THEN/ELSE/ENDIF statement construct 381
ANY subparameter
of //*MAIN SYSTEM parameter 676
of //*MAIN TYPE parameter 677
of //*NET DEVPOOL parameter 680
of /*JOBPARM SYSAFF parameter 617
ANYLOCAL subparameter
of //*FORMAT DEST parameter 654, 662
of DD DEST parameter 157
of OUTPUT JCL DEST parameter 505
ASCII tape record
converting to EBCDIC 147
assistive technologies 693
attribute
of data set 132, 135, 224, 256
on DD LIKE parameter 224
on DD REFDD parameter 256
specifying on DD LIKE parameter 224
attribute *(continued)*
specifying on DD REFDD parameter 256
specifying with DD DATACLAS parameter 132
specifying with DD DCB parameter 135
AUL subparameter
of DD LABEL parameter 218
automatic cartridge loader
use with THWSSEP subparameter of //*MAIN statement 676
average record length
specifying in DD SPACE parameter 266

# B

B subparameter
of DCB OPTCD subparameter 147
of RECFM parameter 252, 253
background or batch jobs
affect on DD TERM parameter 289
backward
coding 23, 153
example 24
to concatenated data set 99
with DD DUMMY statement 189
with EXEC COND parameter 346
BASIC subparameter
of DD DSNTYPE parameter 186
BCP (base control program)
in relation to JCL statement 5
BDAM (basic direct access method)
subparameters of DD DCB parameter 140
BEFORE statement
comments field 604
error on statement 604
example 604
in JCL 602, 603, 604
location in JCL 604
name field 602
operation field 603
parameter field 603
BFALN subparameter
of DD DCB parameter 140
BFTEK subparameter
of DD DCB parameter 140
BINARY subparameter
of DD FILEDATA parameter 198
blank
use in parameter 22
BLDG parameter
of JES3 //*NETACCT statement 684
BLKCHAR subparameter
of OUTPUT JCL DATACK parameter 499
blklgth subparameter
of DD SPACE parameter 266
BLKPOS subparameter
of OUTPUT JCL DATACK parameter 499
BLKSIZE parameter
coexistence consideration 116
default 115
description 114

BLKSIZE parameter *(continued)*
of DD statement 114, 115, 116
override 115
relationship to other control statement 116
subparameter 115
BLKSIZE subparameter
coded with DATA parameter 130
of DD DCB parameter 140
BLKSZLIM parameter
default 117
description 116
example 117
of DD statement 116, 117
override 117
relationship to other parameter 117
subparameter 117
block length
specifying in the DD SPACE parameter 266
BLOCK subparameter
of OUTPUT JCL DATACK parameter 498
blocks subparameter
of DCB LIMCT subparameter 145
BLP subparameter
of DD LABEL parameter 219
BPAM (basic partitioned access method)
subparameters of DD DCB parameter 140
BS subparameter
of DD RECFM parameter 253
BSAM (basic sequential access method)
subparameters of DD DCB parameter 140
with DD CHKPT parameter 124
BST subparameter
of DD RECFM parameter 253
BT subparameter
of DD RECFM parameter 252, 253
BTAM (basic telecommunications access method)
subparameters of DD DCB parameter 140
buffer
requirements with DD AMP parameter 112
buffer subparameter
of DCB BUFIN parameter 140
of DCB BUFNO subparameter 143
of DCB BUFOUT subparameter 143
BUFIN subparameter
of DD DCB parameter 140
BUFL subparameter
of DD DCB parameter 140
BUFMAX subparameter
of DD DCB parameter 142
BUFND subparameter
of DD AMP parameter 107
BUFNI subparameter
of DD AMP parameter 107
BUFNO subparameter
coded with DATA parameter 130
of DD DCB parameter 143
BUFOFF subparameter
of DD DCB parameter 143
BUFOUT subparameter
of DD DCB parameter 143
BUFSIZE subparameter
of DD DCB parameter 143
BUFSP subparameter
of DD AMP parameter 107
BUILDING parameter
description 484
of OUTPUT JCL statement 484
subparameter 484
BURST parameter
default 118, 485
description 117, 485
example 119, 486
of DD statement 117, 118, 119, 342, 417
of JES2 /*JOBPARM statement 615
of JES2 /*OUTPUT statement 624
of OUTPUT JCL statement 485, 486
override 118, 486
relationship to other control statement 118
relationship to other parameter 118, 119, 342, 417
subparameter 118, 485
BYTES parameter
of JES2 /*JOBPARM statement 615
of JES3 //*MAIN statement 667
of JOB statement 413
bytes subparameter
of AMP BUFSP parameter 107
of DD DCB BLKSIZE subparameter 140
of DD DCB BUFL subparameter 140
of DD DCB BUFSIZE subparameter 143
of DD DCB KEYLEN subparameter 145
of DD DCB LRECL subparameter 145
of DD KEYLEN parameter 214
of DD LRECL parameter 226

# C

C subparameter
of //*DATASET DDNAME parameter 647
of //*FORMAT STACKER parameter 657
of //*MAIN BYTES parameter 667
of //*MAIN CARDS parameter 667
of //*MAIN LINES parameter 672
of //*MAIN PAGES parameter 673
of DCB MODE subparameter 145
of DCB OPTCD subparameter 146, 147, 148
of DCB TRTCH subparameter 150
CANCEL subparameter
of //*MAIN BYTES parameter 667
of //*MAIN CARDS parameter 667
of //*MAIN FAILURE parameter 670
of //*MAIN LINES parameter 672
of //*MAIN PAGES parameter 673
of JOB BYTES parameter 414
of JOB CARDS parameter 415
CANCEL subparameter *(continued)*
of JOB LINES parameter 427
of JOB PAGES parameter 436
CARDS parameter
of JES2 /*JOBPARM statement 615
of JES3 //*MAIN statement 667
of JOB statement 415
cards subparameter
JES2 format of JOB accounting information 410
carriage control character
specifying 494
CARRIAGE parameter
of JES3 //*FORMAT PR statement 652
carriage-tape-name subparameter
of //*FORMAT CARRIAGE parameter 652
cataloged and in-stream
affect of parameters on calling EXEC statement 337
calling 360
cataloging 27
description 27
effect of PROC parameter on other parameters and following statement 360
example 33
example of symbols 50
in-stream data 102
indicating beginning 569
indicating end 567
JCL symbol 38
location of DD statements when overriding or adding to procedure 97
modifying DD statement 30
modifying OUTPUT JCL statement 30
overriding ACCT parameter 339
overriding ADDRSPC parameter 341
overriding COND parameter 345
overriding DYNAMNBR parameter 351
overriding EXEC statement parameter 29
overriding PARM parameter 353
overriding PERFORM parameter 358
overriding RD parameter 363
overriding REGION parameter 366
overriding REGIONX parameter 368, 450
overriding TIME parameter 371
statements as listed in job log 57
system symbol 38
testing 28
using 28
CATLG subparameter
of DD DISP parameter 165, 166
cccc subparameter
of /*JOBPARM SYSAFF parameter 617
CCSID parameter
description 119
examples 120
of DD statement 119, 120, 342, 417
of EXEC statement 341

CCSID parameter *(continued)*
of JOB statement 417
subparameter 119, 342, 417
character
with XMIT JCL statement 588
character set 516
chart 21
special character set 21
universal character set (UCS) 21
use in statement 21
character-arrangement table
specifying on DD CHARS parameter 121
specifying on OUTPUT JCL CHARS parameter 486
character-set-code subparameter
of DD UCS parameter 290
of OUTPUT JCL UCS parameter 556
CHARS parameter
affect of DD MODIFY parameter 231
affect of OUTPUT JCL MODIFY parameter 530
affect of OUTPUT JCL TRC parameter 555
default 122, 487
description 121, 486
example 123, 487
of DD statement 121, 122, 123
of JES2 /*OUTPUT statement 624
of JES3 //*FORMAT PR statement 652
of OUTPUT JCL statement 486, 487
override 122, 487
relationship to other control statement 123
relationship to other parameter 123
subparameter 122, 486
checkid subparameter
of JOB RESTART parameter 451
checkpoint
allowing and suppressing 361, 369, 443
for checkpointing data set 325
for checkpointing program 323
logical page size 488
of data set 325
of program 323
restart 450
written after specified number of logical pages 488
written after specified number of seconds 489
CHKPT parameter
description 123
example 124
for concatenated data set 124
of DD statement 123, 124, 325
override 124
relationship to other parameter 124
relationship to SYSCKEOV DD statement 124, 325
subparameter 124
CHNSIZE parameter
of JES3 //*FORMAT PR statement 653
of JES3 //*FORMAT PU statement 661
CKPTLINE parameter
default 488
description 488
example 488
of OUTPUT JCL statement 488
CKPTLNS parameter
of JES2 /*OUTPUT statement 624
CKPTPAGE parameter
default 489
description 488
example 489
of OUTPUT JCL statement 488, 489
relationship to other parameter 489
subparameter 489
CKPTPGS parameter
of JES2 /*OUTPUT statement 624
CKPTSEC parameter
default 490
description 489
example 490
of OUTPUT JCL statement 489, 490
relationship to other parameter 490
subparameter 489
class
assigning 418
assigning job log 430
held 491, 615
held in JES2 system 287
relationship to DD SYSOUT parameter 287
significance 287, 431, 491
specifying on OUTPUT JCL statement 490
specifying on sysout DD statement 283
CLASS parameter
assigning 418
default 419
description 418, 490
example 419, 492
of JES3 //*DATASET statement 647
of JES3 //*MAIN statement 668
of JOB statement 418, 419
of OUTPUT JCL statement 490, 492
override 419, 490
relationship to other control statement 419
subparameter 419, 490
class subparameter
of //*DATASET DDNAME parameter 647
of DD SYSOUT parameter 284
of JOB MSGCLASS parameter 430
of OUTPUT JCL CLASS parameter 490
class-name subparameter
of //*MAIN CLASS parameter 668
CLOSE macro instruction
with DD SPACE parameter 269
with the DD FREE parameter 203
CLOSE subparameter
of DD FREE parameter 202
CNTL statement
comments field 81
description 81, 125
example 82, 125
in JCL 81, 82
CNTL statement *(continued)*
label field 81
location in JCL 82
of DD statement 125
operation field 81
parameter field 81
subparameter 125
code parameter
parameter of JES2 /*OUTPUT statement 624
code subparameter
of EXEC COND parameter 344
of JOB COND parameter 420
code-name subparameter
of DD SYSOUT parameter 285
COLORMAP parameter
description 492
example 492
of OUTPUT JCL statement 492
subparameter 492
command statement
comments field 74
description 73, 611, 644
example 74, 613, 646
in JCL 73, 74
in JES2 611, 612, 613
in JES3 644, 645, 646
location in JCL 74, 613, 646
operand 612, 646
operation field 73
parameter 612, 645
parameter field 73
command-verb parameter
of JES2 command statement 612
of JES3 command statement 645
comment statement
description 79
example 79
in JCL 79
location in JCL 79
relationship to MSGLEVEL parameter 79
comments
format 13
rules for continuation 17
comments field
continuing 17
on JCL statement 13, 17
COMP subparameter
of DCB TRTCH subparameter 150
COMPACT parameter
default 493
description 492
example 493
of JES2 /*OUTPUT statement 625
of JES3 //*FORMAT PR statement 653
of JES3 //*FORMAT PU statement 661
of OUTPUT JCL statement 492, 493
override 493
subparameter 493
compaction
of data 150
compaction table
specifying 492

compaction-table-name subparameter
of //*FORMAT COMPACT parameter 653, 661
of OUTPUT JCL COMPACT parameter 493
comparison operator
on IF/THEN/ELSE/ENDIF statement construct 381
completion code
with IF/THEN/ELSE/ENDIF statement construct 382
COMSETUP parameter
description 493
example 494
of OUTPUT JCL statement 493, 494
subparameter 494
concatenation
of data 190
with dummy data set 190
CONCURRENT statement
comments field 608
error on statement 608
example 608
in JCL 607, 608
location in JCL 608
name field 608
operation field 608
COND parameter
caution 346
description 342, 419
effect on private job library specification 315
example 348, 421
of EXEC statement 342, 344, 345, 346, 348
of JOB statement 419, 420, 421
on EXEC statement 315
override 345, 420
subparameter 344, 420
summary chart 421
Consistent read 260
Consistent read explicit 261
contact
z/OS 693
CONTIG subparameter
of DD SPACE parameter 270
continuing statement
example 17
JCL statement 16
JES2 control statement 18
JES3 control statement 18
CONTROL parameter
default 494
description 494
example 495
of JES3 //*FORMAT PR statement 653
of OUTPUT JCL statement 494, 495
subparameter 494
converter/interpreter service
in job processing 685
COPIES parameter
default 127, 496, 497
description 126, 495
example 128, 496, 498
of DD statement 126, 127, 128
of JES2 /*JOBPARM statement 615
COPIES parameter *(continued)*
of JES2 /*OUTPUT statement 625
of JES3 //*FORMAT PR statement 654
of JES3 //*FORMAT PU statement 662
of OUTPUT JCL statement 495, 496, 497, 498
override 127, 496, 497
relationship to DD FLASH parameter 200
relationship to other control statement 128, 496, 497
relationship to other parameter 127, 496, 497
relationship to OUTPUT JCL COPIES parameter 128
subparameter 126, 495, 497
copies subparameter
JES2 format of JOB accounting information 411
copy
attributes from a model data set 224
jobstream to sysout 462
COPY subparameter
of JOB TYPRUN parameter 462
COPYG parameter
of JES2 /*OUTPUT statement 625
count subparameter
of //*FORMAT FLASH parameter 656
of /*OUTPUT FLASH parameter 627
of /*OUTPUT FLASHC parameter 628
of DD FLASH parameter 200
of OUTPUT JCL FLASH parameter 511
CPRI subparameter
of DD DCB parameter 143
CR subparameter
RLS parameter 260
CRE subparameter
RLS parameter 261
CROPS subparameter
of DD AMP parameter 107
CX subparameter
of DCB DSORG subparameter 143
cycle subparameter
of //*MAIN DEADLINE parameter 668
CYL subparameter
of DD SPACE parameter 266
cylinders
specifying in DD SPACE parameter 266
CYLOFL subparameter
of DD DCB parameter 143

# D

D subparameter
of //*MAIN BYTES parameter 667
of //*MAIN CARDS parameter 667
of //*MAIN LINES parameter 672
of //*MAIN PAGES parameter 673
of //*NET ABNORMAL parameter 680
D subparameter *(continued)*
of //*NET NORMAL parameter 680
of DCB BFALN subparameter 140
of DCB BFTEK subparameter 140
of DCB FUNC subparameter 144
of RECFM parameter 253
DA subparameter
of DCB DSORG subparameter 143
data control block
completing 138
completion during execution 135
copying attribute 137
DATA parameter
default 130
description 129
example 131
location in JCL 131
of DD statement 129, 130, 131
relationship to other control statement 131
relationship to other parameter 130
unread record 131
data set
attribute 229, 260
backup 229
by password 219
checkpoint 229, 260
copying attribute 137
deleting before 259
in generation group 229, 260
indexed sequential 229, 260
migration 229
multivolume 229, 260
organization 229, 260
partitioned (PDS) 229, 260
partitioned data set extended(PDSE) 229, 260
passed 229, 260
permanent 229, 260
record-level sharing, VSAM 260
requesting resource 6
sequence number 229, 260
specifying in DD LABEL parameter 218
specifying in DD RETPD parameter 258
specifying on DCB KEYLEN subparameter 145
specifying on DD KEYLEN parameter 213
specifying on DD KEYOFF parameter 215
system-managed 229, 260
temporary 229, 260
through DD PROTECT parameter 249
through DD SECMODEL parameter 262
type copied when DD statement referenced 309
data-class-name subparameter
of DD DATACLAS parameter 134
data-set-name subparameter
of DD LGSTREAM parameter 223
of DD LIKE parameter 225
data-set-sequence-number subparameter
of DD LABEL parameter 217

DATACK parameter
default 499
description 498
example 499
of OUTPUT JCL statement 498, 499
relationship to other parameter 499
subparameter 498
DATACLAS parameter
default 134
description 132
example 135
of DD statement 132, 134, 135
override 134
relationship to other parameter 135
subparameter 134
date subparameter
of //*MAIN DEADLINE parameter 668
DAU subparameter
of DCB DSORG subparameter 143
DCB macro 135
DCB parameter
description 135
example 139
macro instruction 135
of DD statement 135, 137, 138, 139, 140
relationship to other parameter 138
subparameter 137, 140
DD statement
comments field 97
ddname 83
description 83
example 100
in JCL 83, 85, 97, 100
location in JCL 97
maximum number per job 83
name field 83
number per STEP 83
operation field 85
parameter field 85
special DD statement 83, 85, 97, 100
ddname field
example 100
on DD statement 83
reserved for special use 313
special ddname 84
DDNAME parameter
description 150, 499
example 154
location in JCL 151
location of referenced statement 151
of DD statement 150, 151, 152, 154
of JES3 //*DATASET statement 647
of JES3 //*FORMAT PR statement 651
of JES3 //*FORMAT PU statement 660
of JOB statement 499
override 151
parameters not permitted on referenced DD statement 152
referenced DD statement 152
relationship to other parameter 151
subparameter 151
subparameter definition 499
ddname subparameter
of //*MAIN FETCH parameter 671
of /*JOBPARM PROCLIB parameter 616
of DD DDNAME parameter 151
DEADLINE parameter
of JES3 //*MAIN statement 668
default 482
location of default OUTPUT JCL statement 479
OUTPUT JCL statement 477
specifying statement 500
DEFAULT parameter
default 500
description 500
example 501
location in JCL 501
of OUTPUT JCL statement 500, 501
references to default OUTPUT JCL statement 501
subparameter 500
DEFER subparameter
of DD UNIT parameter 296
DELETE subparameter
of DD DISP parameter 163, 165
delimiter
for JES3 in-stream data set 648
for transmission of input stream 583
processing when invalid 174, 587
with DD * statement 101
with DD DATA statement 129
with XMIT JCL statement 587
delimiter statement
comments field 329
description 329
example 330
in JCL 329, 330
location in JCL 330
relationship to DLM parameter 329
delimiter subparameter
of DD DLM parameter 173
subparameter of XMIT JCL DLM parameter 587
DEN subparameter
of DD DCB parameter 143
dependent
specifying in JES3 system 679
DEPT parameter
description 502
of JES3 //*NETACCT statement 684
OUTPUT JCL statement parameter 502
subparameter 502
dept subparameter
of //*NETACCT DEPT parameter 684
DEST parameter
default 158, 506
description 155, 503, 586
example 159, 506, 586
of DD statement 155, 156, 157, 158, 159
of JES2 /*OUTPUT statement 626
of JES3 //*FORMAT PR statement 654
of JES3 //*FORMAT PU statement 662
DEST parameter *(continued)*
of OUTPUT JCL statement 503, 504, 505, 506
of XMIT JCL statement 586
override 159, 506
relationship to other control statement 159
relationship to other parameter 159, 506
subparameter 586
subparameter for JES2 504
subparameters for JES2 156
subparameters for JES3 157, 505
dest-name parameter
of JES2 /*SIGNON statement 638
destination
for data set 155
specifying on OUTPUT JCL statement 155
device
sharing through unit affinity 296
specifying in DD UNIT parameter 293
device-name subparameter
of //*FORMAT DEST parameter 655, 662
of DD DEST parameter 158
of OUTPUT JCL DEST parameter 505
subparameter of //*NET DEVPOOL parameter 680
device-number subparameter
of //*FORMAT DEST parameter 655, 662
of DD DEST parameter 158
of DD UNIT parameter 293
device-type subparameter
of DD UNIT parameter 294
DEVPOOL parameter
of JES3 //*NET statement 680
DEVRELSE parameter
of JES3 //*NET statement 681
DHWS subparameter
of //*MAIN SETUP parameter 675
DIAGNS subparameter
of DD DCB parameter 143
directory subparameter
for system assignment 268
of DD SPACE parameter 268
DISP parameter
default 166
description 159
example 171
of DD statement 159, 161, 166, 167, 171
relationship to other parameter 167
subparameter 161
with DSNAME parameter 183
disposition
DISP=MOD for multivolume data set 168
of data set 167, 168
of generation data set 167
of partitioned data set 167
of QSAM data set 167
of sysout data set 167, 168
of temporary data set 167

DJC (dependent job control)
network 679
DLM parameter
default 173, 587
description 172, 587
example 174, 587
of DD statement 172, 173, 174
of JES2 /*XMIT statement 641
of XMIT JCL statement 587
relationship to other parameter 173
subparameter 173, 587
DOUBLE subparameter
of //*FORMAT CONTROL parameter 653
of OUTPUT JCL CONTROL parameter 494
DPAGELBL parameter
default 508
description 507
example 508
of OUTPUT JCL statement 507, 508
relationship to other parameter 508
subparameter definition 508
DS subparameter
of //*FORMAT CHNSIZE parameter 653, 661
DSID parameter
description 174
example 175
of DD statement 174, 175
relationship to other parameter 175
subparameter 175
DSNAME parameter
description 176
example 183
of DD statement 176, 177, 178, 179, 182, 183
relationship to other parameter 182
subparameter 177
subparameter for dummy data set 182
subparameters for permanent data set 178
subparameters for temporary data set 179
subparameters when copying data set name 182
dsname subparameter
of //*MAIN UPDATE parameter 678
of DD DCB parameter 137
of DD DSNAME parameter 178
of VOLUME=REF subparameter 305
DSNTYPE parameter
description 184
example 187
of DD statement 184, 185, 186, 187
override 186
relationship to other parameter 187
subparameter 185
DSORG subparameter
of DD DCB parameter 143
DSP (dynamic support program)
calling 685
dsp parameter
of JES3 //*PROCESS statement 686
dummy file
for OS/390 UNIX System Services 240
DUMMY parameter
description 188
example 190
for HFS file 240
in concatenation 100
of DD statement 188, 189, 190
parameter 189
referenced in VOLUME=REF subparameter 309
relationship to other control statement 189
relationship to other parameter 189
same effect with NULLFILE 182
dump
duplicate request 322
high-density 122, 197, 487, 510
printing 321
request on DD statement 123, 197
request on OUTPUT JCL statement 487, 510
specification by SYSABEND, SYSMDUMP, and SYSUDUMP DD statement 319
storage 320
DUMP subparameter
of //*MAIN BYTES parameter 667
of //*MAIN CARDS parameter 667
of //*MAIN LINES parameter 672
of //*MAIN PAGES parameter 673
of JOB BYTES parameter 414
of JOB CARDS parameter 415
of JOB LINES parameter 427
of JOB PAGES parameter 436
on DD CHARS parameter 122
on OUTPUT JCL CHARS parameter 487
DUPLEX parameter
description 508
example 509
of OUTPUT JCL statement 508, 509
relationship to other parameter 509
subparameter definition 508
DYNAM parameter
description 191
example 191
of DD statement 191
relationship to other control statement 191
relationship to other parameter 191
dynamic system symbol 38
DYNAMNBR parameter
default 351
description 350
example 351
of EXEC statement 350, 351
subparameter 350

# E

E subparameter
of //*DATASET DDNAME parameter 647
of DCB BFTEK subparameter 140
of DCB CPRI subparameter 143
E subparameter *(continued)*
of DCB MODE subparameter 145
of DCB OPTCD subparameter 146
of DCB TRTCH subparameter 150
EBCDIC character
converting to ASCII code 147
EBCDIC text
description 21
END subparameter
of DD FREE parameter 202
ENDCNTL statement
comments field 331
description 331
example 331
in JCL 331
label field 331
location in JCL 331
operation field 331
ENDGROUP statement
comments field 609
error on statement 609
example 609
in JCL 609
location in JCL 609
name field 609
operation field 609
ENDSET statement
error on statement 602
example 602
in JCL 601, 602
location in JCL 602
name field 602
operation field 602
parameter field 602
EOV subparameter
of DD CHKPT parameter 124
EQ subparameter
of EXEC COND parameter 344
of JOB COND parameter 420
EROPT subparameter
of DD DCB parameter 144
error
coding DD OUTPUT parameter 236
error messages
in reading or writing a data set 498
printing with PSF 498
ES subparameter
of DD RECORG parameter 255
ET subparameter
of DCB TRTCH subparameter 150
EVEN subparameter
of EXEC COND parameter 344
EXCP (execute channel program)
subparameters of DD DCB parameter 140
EXEC statement
comments field 337
description 333
example 337
in JCL 333, 334, 337
location in JCL 337
name field 333
operation field 334
parameter field 334
RLSTMOUT parameter 369
default 370
example 370

execution
at checkpoint 450
at step 450
bypassing 342, 419
holding 462
of job at network node 688
requesting for all steps in job 443
requesting for step 361
restarting step 361, 443
specifying 450
specifying program 358
timing 370, 459
with EXEC COND parameter 347
EXPDT parameter
description 193
example 194
of DD statement 193, 194
override 194
relationship to other parameter 194
subparameter 193
EXPDTCHK parameter
of JES3 //*MAIN statement 670
explicit
to OUTPUT JCL statement 478, 501
EXPORT statement 375
examples 377
field
Comments 376
Label 375
Operation 375
Parameter 375
Location in JCL 376
parameter
SYMLIST 376
parameter syntax
SYMLIST 376
subparameter definition 377
SYMLIST parameter 376
syntax 375
extents
allocation 268
external
specifying on OUTPUT JCL statement 564
specifying on sysout DD statement 283
starting 287, 564
with OUTPUT JCL WRITER parameter 564
with SYSOUT writer-name parameter 287
EXTPREF subparameter
of DD DSNTYPE parameter 186
EXTREQ subparameter
of DD DSNTYPE parameter 185
EXTWTR parameter
of JES3 //*FORMAT PR statement 655
of JES3 //*FORMAT PU statement 663

# F

F subparameter
of //*NET ABNORMAL parameter 680
of //*NET NORMAL parameter 680
F subparameter *(continued)*
of AMP RECFM subparameter 109
of DCB BFALN subparameter 140
of DCB OPTCD subparameter 146
of RECFM parameter 252, 253
FAILURE parameter
of JES3 //*MAIN statement 670
FB subparameter
of AMP RECFM subparameter 109
FCB parameter
default 196, 510
defining for work station 197
description 195, 509
example 197, 511
of DD statement 195, 196, 197
of JES2 /*OUTPUT statement 627
of JES3 //*FORMAT PR statement 655
of OUTPUT JCL statement 509, 510, 511
override 196, 510
relationship to other control statement 197
relationship to other parameter 196, 510
subparameter 196, 510
fcb-name subparameter
of DD FCB parameter 196
of OUTPUT JCL FCB parameter 510
FETCH parameter
of JES3 //*MAIN statement 671
file
HFS
dummy 240
file definition statement 135
FILEDATA parameter
default 199
description 198
example 199
of DD statement 198, 199
override 199
relationship to other parameter 199
subparameter 198
syntax 198
FLASH parameter
default 200, 512
description 199, 511
example 201, 512
of DD statement 199, 200, 201
of JES2 /*OUTPUT statement 627
of JES3 //*FORMAT PR statement 656
of OUTPUT JCL statement 511, 512
override 200, 512
relationship to other control statement 201
relationship to other parameter 200, 512
subparameter 200, 511
FLASHC
of JES2 /*OUTPUT statement 628
flashing
printing with 199, 511
printing without 201, 512
relationship to DD COPIES parameter 127
flashing *(continued)*
relationship to OUTPUT JCL COPIES parameter 496, 497
FLSH subparameter
of //*NET NRCMP parameter 682
FOLD subparameter
of DD UCS parameter 291
for data set
specifying on OUTPUT JCL statement 503
specifying on XMIT JCL statement 586
foreground jobs
affect on DD TERM parameter 289
form
for printing or punching 195, 509
JES2 format subparameter of JOB accounting information 411
specifying on OUTPUT JCL FORMS parameter 515
specifying on sysout DD statement 283
form-name subparameter
of //*FORMAT FORMS parameter 656, 663
of DD SYSOUT parameter 285
of OUTPUT JCL FORMS parameter 515
FORMDEF parameter
description 513
example 514
of OUTPUT JCL statement 513, 514
override 513
subparameter 513
FORMLEN parameter
description 514
of OUTPUT JCL statement 514
FORMS parameter
default 515
description 515
example 515
of JES2 /*JOBPARM statement 615
of JES2 /*OUTPUT statement 628
of JES3 //*FORMAT PR statement 656
of JES3 //*FORMAT PU statement 663
of OUTPUT JCL statement 515
override 515
subparameter 515
forward
to concatenated data set 99
FREE parameter
affect on JES2 /*JOBPARM COPIES parameter 615
default 202
description 201
example 203
of DD statement 201, 202, 203
override 202
relationship to other control statement 203
relationship to other parameter 202
subparameter 202
FREEVOL parameter
of DD statement
description 204

FRLOG subparameter 108
FSSDATA parameter
description 515
of OUTPUT JCL statement 515
FUNC subparameter
of DD DCB parameter 144
with LABEL parameter 221

# G

G11 character set
for 3211 printer 290, 556
GAM (graphics access method)
subparameters of DD DCB parameter 140
GDGORDER parameter
example 205
of DD statement 205
description 205
subparameter 205
GE subparameter
of EXEC COND parameter 344
of JOB COND parameter 420
generation subparameter
of DD DSNAME parameter 178
generations
specifying maximum 228
GENERIC subparameter
of DD SECMODEL parameter 263
GJOB statement
comments field 599
error on statement 599
example 599
in JCL 598, 599
location in JCL 599
name field 598
operation field 598
parameter field 598
started tasks 598
GNCP subparameter
of DD DCB parameter 144
GROUP parameter
default 424
description 423
example 424
of JOB statement 423, 424
subparameter 424
group-name parameter
of //*FORMAT DEST parameter 655, 662
of //*MAIN ORG parameter 673
of DD DEST parameter 158
of DD UNIT parameter 294
of JOB GROUP parameter 424
of OUTPUT JCL DEST parameter 505
group-value subparameter
of //*FORMAT COPIES parameter 654
of /*OUTPUT COPIES parameter 625
of /*OUTPUT COPYG parameter 625
of DD COPIES parameter 127
of OUTPUT JCL COPIES parameter 495
GROUPID parameter
description 518
example 518
of OUTPUT JCL statement 518
relationship to other control statement 518
subparameter 518
GS subparameter
of DCB DSORG subparameter 143
GT subparameter
of EXEC COND parameter 344
of JOB COND parameter 420
GTF (generalized trace facility)
use 143

# H

H subparameter
of DCB OPTCD subparameter 146, 147
H11 character set code
for 3211 printer 290, 556
halt reading
in JES3 system 685
HC parameter
abbreviation of NHOLD parameter of JES3 //*NET statement 679
HFS subparameter
of DD DSNTYPE parameter 185
HH:MM subparameter
of HOLDUNTL parameter 572
of STARTBY parameter 574
HIGH subparameter
of //*MAIN IORATE parameter 671
HN character set code
for 1403 and 3203 Model 5 printer 290, 556
HOLD parameter
affect on JES2 /*JOBPARM COPIES parameter 615
default 207
description 206
examples 208
of DD statement 206, 207, 208
of JES3 //*MAIN statement 671
override 207
relationship to other control statement 208
relationship to other parameter 208
subparameter 206
HOLD subparameter
of //*MAIN FAILURE parameter 670
of //*NET NRCMP parameter 682
of JOB TYPRUN parameter 462
HOLDUNTL parameter
SCHEDULE statement 572
HOLDUNTL parameter syntax
SCHEDULE statement 572
HOLDUNTL relationship to other parameters
SCHEDULE statement 573
HOLDUNTL subparameter definition
SCHEDULE statement 572
HWS subparameter
of //*MAIN SETUP parameter 675

# I

I subparameter
of AMP OPTCD subparameter 108
of DCB FUNC subparameter 144
of DCB OPTCD subparameter 148
IAZSYMBL 56
ID parameter
abbreviation of NETID parameter of JES3 //*NET statement 679
id subparameter
of DD DSID parameter 175
identifier
format 13
IEFSJSYM 55
IGNORE subparameter
of //*MAIN THWSSEP parameter 676
IL subparameter
of AMP OPTCD subparameter 108
image-name subparameter
of //*FORMAT FCB parameter 655
implicit
to OUTPUT JCL statement 478, 501
using OUTPUT JCL DEFAULT parameter 500
in generation group
in restarted job 452
labels for 218
naming 178
in JCL
comments field 76, 385, 394, 399, 478, 578
comparison operator 381
consideration 400, 579
considerations 386
considerations for using 394
default definition 477
description 75, 379, 393, 397, 467, 469, 571, 577
ELSE clause 386
example 77, 395, 400, 467, 580
example of job and step-level statement 479
examples of schedule statement 572
job-level statement 478
location in JCL 76, 467, 478
location in procedure 479
location in the JCL 385, 394, 399, 572, 579
location of default statement 479
logical operator 381
name field 76, 379, 393, 397, 469, 571, 578
NOT operator 382
operation 571
operation field 76, 380, 393, 398, 469, 578
override 479, 579
parameter field 76, 393, 398, 470, 571, 578
relational-expression 380
relationship to DD statement COPIES parameter 128
relationship to JES2 /*OUTPUT statement 480
relationship to JES3 //*FORMAT statement 480

in JCL *(continued)*
relationship to other control statement 579
relationship to other parameter 385
relationship to sysout DD statement 480
step-level statement 479
syntax 571
THEN clause 386
use of parenthesis 385
in JES3
description 685
ending 649
example 687
location in JCL 687
parameter 686
in printed output
limiting length 411, 524
specification 653
specifying 494
in reading or writing a data set
specifying options for 144
IN subparameter
of DD LABEL parameter 220
of OUTPUT JCL OFFSETXB parameter 533
in-stream data
for procedure 102
multiple in-stream data sets in a step 103
with DD * statement 101
with DD DATA statement 131
with DSNAME parameter 181
with JES3 //*DATASET statement 646
with SYSIN DD statement 326
INCLUDE group
considerations for using 394
description 393
IND subparameter
of /*JOBPARM SYSAFF parameter 617
INDEX parameter
default 520
description 519
example 520
of OUTPUT JCL statement 519, 520
parameter of JES2 /*OUTPUT statement 628
relationship to other parameter 520
subparameter 520
information subparameter
of EXEC PARM parameter 353
initiation or selection
specifying 442
input stream
description 5
INT parameter
of JES3 //*FORMAT PU statement 663
internal
submitting job 611, 643
INTRAY parameter
description 520
example 521
of OUTPUT JCL statement 520, 521
relationship to other parameter 520
INTRAY parameter *(continued)*
subparameter 520
INTRDR subparameter
of OUTPUT JCL WRITER parameter 564
INTVL subparameter
of DD DCB parameter 144
IORATE parameter
of JES3 //*MAIN statement 671
IPCS (interactive problem control system)
to print dump 321
IPLTXID subparameter
of DD DCB parameter 144
IS subparameter
of DCB DSORG subparameter 143
ISO/ANSI/FIPS Version 1 or 3 tape data set
indicating in DD LABEL parameter 218
restriction on DD DISP parameter 162
with DD ACCODE parameter 104
ISU subparameter
of DCB DSORG subparameter 143

# J

J parameter
of JES3 //*DATASET statement 647
J subparameter
of DCB OPTCD subparameter 147
JCL (job control language)
format 13
statement 1
JCL subparameter
of OUTPUT JCL JESDS parameter 521
JCL symbol 38
JCL Symbol Service (IEFSJSYM) 55
JCLHOLD subparameter
of JOB TYPRUN parameter 462
JCLTEST subparameter
of EXEC PGM parameter 359
JECL statements 2
JES (job entry subsystem)
running a started task 63
JES in-stream data 54
JES Symbol Service (IAZSYMBL) 56
JES2
format 16
location in JCL 611
statement 1, 16, 591, 611
JES3
example 643
format 16
location in JCL 643
statement 1, 16, 643
JES3 control statements
JES2 processing 692
JESDS parameter
description 521
example 522
location in JCL 522
location of statement containing 479
of OUTPUT JCL statement 521, 522
override 522
subparameter 521
JESJCL subparameter
of //*FORMAT DDNAME parameter 651
JESLOG parameter of JOB statement 424
JESMSGLG subparameter
of //*FORMAT DDNAME parameter 651
JESYSMSG
of //*FORMAT DDNAME parameter 651
JGLOBAL subparameter
of //*MAIN SYSTEM parameter 676
JLOCAL subparameter
of //*MAIN SYSTEM parameter 676
job
background or batch jobs 403
beginning 403
class 403
dependent 403
description 5
entering 5
foreground jobs 403
nonstandard processing 403
processing 6, 403
request to not print 616
restarting 403
send messages to in JES3 system 684
specifying 462
specifying processing 521
standard processing 403
job group
activating 593
associating jobs 593
configuring 593
defining a static execution zone 594
processing for jobs 593
job log
assigning to an output class 430
cataloged procedure statement 57
controlling listing 432
in-stream procedure statement 57
job control statement 57
listing 57
specifying processing 521
statements in listing 57
symbolic parameter 57
Job Statement
DSENQSHR Parameter 421
JOB statement
comments field 408
description 403
example 408, 601
in JCL 403, 404, 408, 601, 608
location in JCL 408
name field 403
operation field 404
parameter field 404, 608
started tasks 404
JOB subparameter
of //*MAIN SETUP parameter 675
job-level
OUTPUT JCL statement level 478
job-level output
control of 60
jobclass subparameter
of JOB CLASS parameter 419

JOBGROUP
examples 591
logging job 592
JOBGROUP example
SCHEDULE statement 573
JOBGROUP JCL
exit processing 593
JES2 593
JOBGROUP parameter
SCHEDULE statement 573
JOBGROUP parameter syntax
SCHEDULE statement 573
JOBGROUP relationship to other parameters
SCHEDULE statement 573
JOBGROUP statement
comments field 597
description 594
error on statement 597
example 598
in JCL 594, 595, 597, 598
location in JCL 597
name field 595
operation field 595
parameter field 595
JOBGROUP subparameter definition
SCHEDULE statement 573
JOBGROUPother attributes
logging job 592
JOBLIB DD statement
description 313
example 315
location in JCL 315
overriding for a step 315, 318
parameter 313
relationship to other control statement 315
relationship to STEPLIB 315
with COND=ONLY parameter 346
jobname
coding 403, 600
jobname subparameter
of //*NET NETREL parameter 681
of //*NET RELEASE parameter 682
of WITH parameter 575
JOBRC parameter
default 426
example 427
of JOB JOBRC parameter 426
of JOB statement 426, 427
override 426
Relationship to other control statements 426
jobset
coding 599
JOBSET statement
comments field 600
error on statement 600
example 600
in JCL 599, 600
location in JCL 600
name field 599
operation field 600
parameter field 600
started tasks 600
JOURNAL parameter
of JES3 //*MAIN statement 672
JSTTEST subparameter
of EXEC PGM parameter 359

# K

K subparameter
of DD AVGREC parameter 113
KEEP subparameter
of //*NET ABCMP parameter 680
of DD DISP parameter 164, 166
keyboard
navigation 693
PF keys 693
shortcut keys 693
KEYENCD1 PARAMETER
of DD statement
description 211
examples 212
override 212
relationship to other parameter 212
subparameter 212
KEYENCD2 PARAMETER
of DD statement
description 212
examples 213
override 213
relationship to other parameter 213
subparameter 213
KEYLABL1 PARAMETER
of DD statement
default 209
description 209
examples 210
override 209
relationship to other parameter 210
subparameter 209
KEYLABL2 PARAMETER
of DD statement
default 211
description 210
examples 211
override 211
relationship to other parameter 211
subparameter 210
KEYLEN parameter
description 213
example 214
of DD statement 213, 214
override 214
relationship to other parameter 214
subparameter 214
KEYLEN subparameter
of DD DCB parameter 145
KEYOFF parameter
description 215
example 216
of DD statement 215, 216
override 215
relationship to other parameter 215
subparameter 215
keyword
on AFTER statement 605
on DD statement 86
keyword *(continued)*
on EXEC statement 334
on EXEC statement that calls procedure 337
on GJOB statement 598
on JOB statement 404, 595, 603, 608
on JOBSET statement 600
on OUTPUT JCL statement 470
syntax 15
usage warning 14
KS subparameter
of DD RECORG parameter 255

# L

L subparameter
of AMP OPTCD subparameter 108
of DCB BUFOFF subparameter 143
of DCB OPTCD subparameter 146, 148
LABEL parameter
default 220
description 216
example 221
of DD statement 216, 217, 220, 221
relationship to other control statement 221
relationship to other parameter 220
subparameter 217
LARGE subparameter
of DD DSNTYPE parameter 186
LE subparameter
of EXEC COND parameter 344
of JOB COND parameter 420
LGSTREAM parameter
defaults 223
description 222
example 224
of DD statement 222, 223, 224
override 223
relationship to other parameter 223
subparameter 223
library
procedure
use for procedure 5
LIBRARY subparameter
of DD DSNTYPE parameter 185
LIBRARY,1 subparameter
of DD DSNTYPE parameter 185
LIBRARY,2 subparameter
of DD DSNTYPE parameter 185
LIKE parameter
description 224
example 226
of DD statement 224, 225, 226
override 225
relationship to other parameter 226
subparameter 225
LIMCT subparameter
of DD DCB parameter 145
limit subparameter
of //*FORMAT THRESHLD parameter 657
of OUTPUT JCL THRESHLD parameter 553
limiting
of lines per printed page 411

limiting *(continued)*
output 411
LINDEX parameter
default 523
description 523
example 523
of JES2 /*OUTPUT statement 628
of OUTPUT JCL statement 523
relationship to other parameter 523
subparameter 523
LINE subparameter
of OUTPUT JCL PRMODE parameter 542
LINECT parameter
of JES2 /*JOBPARM statement 616
of JES2 /*OUTPUT statement 629
linect subparameter
default 524
description 524
example 524
JES2 format of JOB accounting information 411
of OUTPUT JCL statement 524
subparameter 524
LINES parameter
of JES2 /*JOBPARM statement 616
of JES3 //*MAIN statement 672
of JOB statement 427
lines subparameter
JES2 format of JOB accounting information 410
LOCAL parameter
of JES2 /*ROUTE statement 633
LOCAL subparameter
of /*OUTPUT DEST parameter 626
of DD DEST parameter 156
of OUTPUT JCL DEST parameter 504
log subparameter
JES2 format of JOB accounting information 411
LOG subparameter
of OUTPUT JCL JESDS parameter 521
logical operator
on IF/THEN/ELSE/ENDIF statement construct 381
LOW subparameter
of //*MAIN IORATE parameter 671
lowercase
in syntax 19
LRECL parameter
description 226
example 227
of DD statement 226, 227
override 227
relationship to other parameter 227
subparameter 226
LRECL subparameter
of DD DCB parameter 145
LREGION parameter
of JES3 //*MAIN statement 673
LS subparameter
of DD RECORG parameter 255
LT subparameter
of EXEC COND parameter 344
of JOB COND parameter 420
LTM subparameter
of DD LABEL parameter 219

# M

m subparameter
of //*FORMAT CHNSIZE parameter 653, 661
M subparameter
of DCB OPTCD subparameter 148
of DD AVGREC parameter 114
of DD RECFM parameter 252, 253
magnetic
specification for tape data set 143
MAILBCC
of OUTPUT JCL statement description 524
MAILCC
of OUTPUT JCL statement description 525
MAILFILE
of OUTPUT JCL statement description 526
MAILFROM
of OUTPUT JCL statement description 527
MAILTO
of OUTPUT JCL statement description 528
main service
in job processing 685
main-name subparameter
of //*MAIN SYSTEM parameter 676
management-class-name subparameter
of DD MGMTCLAS parameter 229
Master subsystem
JCL restrictions with START SUB=MSTR 64
restrictions with a started task 64
running a started task 64
MAXGENS parameter
description 228
example 228
of DD statement 228
relationship to other parameter 228
subparameter 228
MAXIMUM subparameter
of the JOB TIME parameter 460
on the EXEC TIME parameter 371
maximum-generations subparameter
of DD MAXGENS parameter 228
MED subparameter
of //*MAIN IORATE parameter 671
member subparameter
of DCB INTVL subparameter 144
of DD DSNAME parameter 178, 181
membername subparameter
of OUTPUT JCL FORMDEF parameter 513
of OUTPUT JCL PAGEDEF parameter 540
members
maximum generations for 228
MERGE parameter
default 529
description 529
examples 500, 529
MERGE parameter *(continued)*
of JOB statement 500, 529
subparameter definition 529
message
from functional subsystem 540
specifying processing 521
to operator in JES3 system 684
messages subparameter
of JOB MSGLEVEL parameter 433
MGMTCLAS parameter
default 229
description 228
example 230
of DD statement 228, 229, 230
override 229
relationship to other parameter 230
subparameter 229
minutes subparameter
of EXEC TIME parameter 370
of JOB TIME parameter 459
MM/DD/YYYY or YYYY/DDD subparameter
of HOLDUNTL parameter 572
of STARTBY parameter 574
mmm subparameter
of //*MAIN BYTES parameter 667
of //*MAIN CARDS parameter 667
of //*MAIN LINES parameter 672
of //*MAIN PAGES parameter 673
of OUTPUT JCL OFFSETXB parameter 533
MOD subparameter
of DD DISP parameter 162
MODE parameter
of JES3 //*DATASET statement 647
MODE subparameter
of DD DCB parameter 145
modification
by specifying copy-modification module 230, 529
coding 30
of procedure DD statement 30
of procedure DD statements 30
MODIFY parameter
default 231, 530
description 230, 529
example 232, 531
of DD statement 230, 231, 232
of JES2 /*OUTPUT statement 629
of JES3 //*FORMAT PR statement 656
of OUTPUT JCL statement 529, 530, 531
override 231, 531
relationship to other control statement 232
relationship to other parameter 231, 531
subparameter 231, 530
MODTRC parameter
of JES2 /*OUTPUT statement 630
module subparameter
of AMP SYNAD subparameter 109
module-name subparameter
of //*FORMAT MODIFY parameter 656

module-name subparameter *(continued)*
of /*OUTPUT MODIFY parameter 629
of DD MODIFY parameter 231
of OUTPUT JCL MODIFY parameter 530
MSG subparameter
of OUTPUT JCL JESDS parameter 522
msg-count subparameter
of OUTPUT JCL PIMSG parameter 541
MSGCLASS parameter
default 431
description 430
example 431
of JOB statement 430, 431
subparameter 430
MSGCLASS subparameter
of //*DATASET DDNAME parameter 647
MSGLEVEL parameter
default 433
description 432
example 433
of JOB statement 432, 433
subparameter 432
multivolume
referenced in VOLUME=REF subparameter 307
specifying volume 303
MXIG subparameter
of DD SPACE parameter 270

# N

n or number subparameter
of //*FORMAT CHNSIZE parameter 653, 661
of //*FORMAT COPIES parameter 654, 662
of //*FORMAT PRTY parameter 657
of //*MAIN BYTES parameter 667
of //*MAIN CARDS parameter 667
of //*MAIN LINES parameter 672
of //*MAIN PAGES parameter 673
of //*NET DEVPOOL parameter 680
of //*NET NHOLD parameter 681
of //*NET RELSCHCT parameter 682
of //*NETACCT ACCT parameter 684
of /*JOBPARM BYTES parameter 615
of /*JOBPARM CARDS parameter 615
of /*JOBPARM COPIES parameter 615
of /*JOBPARM LINECT parameter 616
of /*JOBPARM LINES parameter 616
of /*JOBPARM PAGES parameter 616
of /*JOBPARM TIME parameter 618
of /*OUTPUT CKPTLNS parameter 624
n or number subparameter *(continued)*
of /*OUTPUT CKPTPGS parameter 624
of /*OUTPUT COMPACT parameter 625
of /*OUTPUT COPIES parameter 625
of /*OUTPUT INDEX parameter 628
of /*OUTPUT LINDEX parameter 628
of /*OUTPUT LINECT parameter 629
of AMP BUFND parameter 107
of AMP BUFNI parameter 107
of AMP STRNO parameter 109
of DCB BUFOFF subparameter 143
of DCB GNCP subparameter 144
of DCB INTVL subparameter 144
of DCB NCP parameter 145
of DCB THRESH subparameter 149
of DD COPIES parameter 126
of DD OUTLIM parameter 232
of DD RETPD parameter 259
of EXEC DYNAMNBR parameter 350
of EXEC PERFORM parameter 357
of JOB PERFORM parameter 439
of OUTPUT JCL CKPTPAGE parameter 489
of OUTPUT JCL CKPTSEC parameter 489
of OUTPUT JCL COPIES parameter 495
of OUTPUT JCL INDEX parameter 520
of OUTPUT JCL INTRAY parameter 520
of OUTPUT JCL LINDEX parameter 523
of OUTPUT JCL LINECT parameter 524
of OUTPUT JCL PRTY parameter 547
N subparameter
of /*JOBPARM BURST parameter 615
of /*JOBPARM RESTART parameter 617
of /*OUTPUT BURST parameter 624
of DCB PCI subparameter 148
name
format 13
in name field of OUTPUT JCL statement 469
qualified 469
unqualified 469
name parameter
of JES2 /*ROUTE statement 633
NAME parameter
description 531
of OUTPUT JCL statement 531
subparameter 531
name subparameter
of //*FORMAT EXTWTR parameter 655, 663
of //*NET NETID parameter 680
of /*OUTPUT DEST parameter 626
name subparameter *(continued)*
of DD DEST parameter 156
of OUTPUT JCL DEST parameter 504
of OUTPUT JCL WRITER parameter 564
navigation
keyboard 693
NC parameter
abbreviation of NORMAL parameter of JES3 //*NET statement 679
NC subparameter
of EXEC RD parameter 363
of JOB RD parameter 445
NCK subparameter
of AMP CROPS subparameter 107
NCP subparameter
of DD DCB parameter 145
NE subparameter
of EXEC COND parameter 344
of JOB COND parameter 420
nested
description 35
example 35
modifying procedure statement 36
symbolic parameter 51
NET subparameter
of //*NET DEVPOOL parameter 680
NETID parameter
of JES3 //*NET statement 680
netid subparameter
of //*NET NETREL parameter 681
NETREL parameter
of JES3 //*NET statement 681
network-account-number parameter
of JES2 /*NETACCT statement 620
NEW subparameter
of DD DISP parameter 161
new-password subparameter
of JOB PASSWORD parameter 438
parameter of JES2 /*SIGNON statement 638
parameter of JES3 /*SIGNON statement 692
NHOLD parameter
of JES3 //*NET statement 681
NL subparameter
of DD LABEL parameter 218
Nn parameter
of JES2 /*ROUTE statement 633
of JES2 /*XEQ statement 639
of JES2 /*XMIT statement 641
Nn subparameter
of /*OUTPUT DEST parameter 626
of DD DEST parameter 156
of OUTPUT JCL DEST parameter 504
nnnnK subparameter
of //*MAIN LREGION parameter 673
No read integrity 260
NO subparameter
of //*DATASET DDNAME parameter 647
of //*FORMAT FORMS parameter 663

NO subparameter *(continued)*
of //*MAIN EXPDTCHK parameter 670
of //*MAIN HOLD parameter 671
of //*MAIN JOURNAL parameter 672
of //*MAIN RINGCHK parameter 674
of //*NET DEVRELSE parameter 681
of //*NET OPHOLD parameter 682
of DD BURST parameter 118
of DD HOLD parameter 207
of OUTPUT JCL AFPSTATS parameter 483
of OUTPUT JCL BURST parameter 485
of OUTPUT JCL DEFAULT parameter 500
of OUTPUT JCL DPAGELBL parameter 508
of OUTPUT JCL DUPLEX parameter 508
of OUTPUT JCL PIMSG parameter 541
of OUTPUT JCL SYSAREA parameter 552
of OUTPUT JCL TRC parameter 555
NOCOMP subparameter
of DCB TRTCH subparameter 150
node
affect on JES2 /*JOBPARM COPIES parameter 619
of execution 618
node subparameter
of DD DEST parameter 157, 158
nodename parameter
of JES2 /*NOTIFY statement 621
of JES2 /*ROUTE statement 633, 634
of JES2 /*XEQ statement 639
of JES2 /*XMIT statement 641
of JES3 //*ROUTE XEQ 688
nodename subparameter
of //*FORMAT DEST parameter 655, 662
of //*MAIN ORG parameter 673
of /*OUTPUT DEST parameter 626
of DD DEST parameter 158
of OUTPUT JCL DEST parameter 505, 506
of XMIT JCL DEST parameter 586
NOHO subparameter
of //*NET NRCMP parameter 682
NOKP subparameter
of //*NET ABCMP parameter 680
NOLIMIT subparameter
of the JOB TIME parameter 460
on the EXEC TIME parameter 371
NOLOG parameter
of JES2 /*JOBPARM statement 616
non-SMS-managed data set
with DD VOLUME=REF subparameter 308
NONE subparameter
of //*MAIN FETCH parameter 671
of /*OUTPUT FLASH parameter 627
of DD FLASH parameter 200
NONE subparameter *(continued)*
of OUTPUT JCL FLASH parameter 511
nonspecific request
allocation 268
specifying 309
volume count 303
nonstandard processing
description 685
NOPWREAD subparameter
of DD LABEL parameter 220
normal
dump 319
NORMAL parameter
of JES3 //*NET statement 680
NORMAL subparameter
of OUTPUT JCL DUPLEX parameter 509
NOT (¬) operator
of IF/THEN/ELSE/ENDIF statement construct 382
notation
for syntax 19
Notices 697
notification
of job completion 433
receiving 434
NOTIFY parameter
description 433, 532
example 435
of JOB statement 433, 434, 435
of OUTPUT JCL statement 532, 533
subparameter 533
subparameter for JES2 434
subparameter for JES3 434
NR parameter
abbreviation of NETREL parameter of JES3 //*NET statement 679
NR subparameter
of EXEC RD parameter 362
of JOB RD parameter 444
NRC subparameter
of AMP CROPS subparameter 107
NRCMP parameter
of JES3 //*NET statement 682
NRE parameter
of AMP CROPS subparameter 107
NRI subparameter
RLS parameter 260
NSL subparameter
of DD LABEL parameter 218
NTM subparameter
of DD DCB parameter 146
null subparameter
of //*FORMAT DDNAME parameter 651, 660
NULLFILE subparameter
of DD DSNAME parameter 182
number
affect on number of devices allocated 295
specifying by volume-count subparameter 303
NxxRnnnn
parameter of JES2 /*SIGNON statement 638

# O

O subparameter
of DCB MODE subparameter 145
of cataloged and in-stream procedures
of DD statement 30
of EXEC statement parameter 29
of OUTPUT JCL statement 30
with DD DUMMY statement 190
of data
block size 98
checkpointing 124
coding concatenation 98
description 97
device 98
logical record length 99
of job library 315
of step library 318
reference 99
with dummy data set 100
of data set
at abnormal termination 165
at normal termination 163
deleting before 194, 259
for input or output 220
holding for reuse 350
specifying in DD EXPDT parameter 193
specifying in DD LABEL parameter 220
with multiple references in DD OUTPUT parameter 236
of DCB macro instruction
when coding DUMMY 190
of DD SPACE parameter
for specific request 270, 271
for system assignment 266
of DD statement
defaults 238, 241, 244, 247
description 113, 237, 240, 242, 245
dummy HFS file 240
example 114, 240, 241, 245, 249
file status 248
override 114
relationship to other parameter 114, 238, 241, 244, 247
relationship to other statements 239
subparameter 113, 238, 241, 242, 246
with DD AVGREC parameter 113
with DD SPACE reclgth subparameter 266
of device
from group 294
number 295
when unit affinity is specified 296
of label
specified by DD LABEL parameter 221
of operator
of operator 381
of print margins
specifying on OUTPUT JCL statement 519, 520, 523
of statement
field 13
of sysout data set
at abnormal termination 535
at normal termination 535

of volumes
deferred 296
parallel 295
OFF subparameter
of //*FORMAT OVFL parameter 657
offset-to-key subparameter
of DD KEYOFF parameter 215
OFFSETXB parameter
description 533
example 534
of OUTPUT JCL statement 533, 534
subparameter 533
OFFSETXF parameter
description 534
of OUTPUT JCL statement 534
OFFSETYB parameter
description 534
of OUTPUT JCL statement 534
OFFSETYF parameter
description 534
of OUTPUT JCL statement 534
OH parameter
abbreviation of OPHOLD parameter
of JES3 //*NET statement 679
OLD subparameter
of DD DISP parameter 162
omission
of ddname from DD statement 84
on direct access
request for specific track 270
system assignment 266
on IF/THEN/ELSE/ENDIF statement construct
continuing 380
description 380
keyword 382
operator 381
ON subparameter
of //*FORMAT OVFL parameter 657
ONLY subparameter
of EXEC COND parameter 345
Open/Close/EOV trace option 144
operating system
content 5
operation
format 13
operator
messages to in JES3 system 684
on IF/THEN/ELSE/ENDIF statement construct 381
operator commands
entered with JCL COMMAND statement 75
entering through JCL command statement 73
entering through JES2 command statement 611
entering through JES3 command statement 644
operator subparameter
of EXEC COND parameter 344
of JOB COND parameter 420
OPHOLD parameter
of JES3 //*NET statement 682
OPTCD subparameter
of DD AMP parameter 108
of DD DCB parameter 146
OR (|) operator
of IF/THEN/ELSE/ENDIF statement construct 381
ORG parameter
of JES3 //*MAIN statement 673
organization
with DSORG subparameter 143
organization subparameter
of DCB DSORG subparameter 143
OUT subparameter
of DD LABEL parameter 220
OUTBIN parameter
of OUTPUT JCL statement 534
OUTDISP parameter
description 535
of OUTPUT JCL statement 535, 536
subparameter 536
OUTLIM parameter
default 233
description 232
example 233
of DD statement 232, 233
relationship to other control statement 233
relationship to other parameter 233
subparameter 232
output
by specifying DD OUTLIM parameter 232
class 126, 410, 495, 654
limiting from job 410
maximum size of sysout data set 553
of lines per printed page 524
specifying copy number 126, 495, 654
specifying on the OUTPUT JCL statement 518
with JOB statement BYTES parameter 413
with JOB statement CARDS parameter 415
with JOB statement LINES parameter 427
with JOB statement PAGES parameter 435
output data set
controlling spacing in output 494, 653
processing instructions in JES3 system 649
processing options in JES3 system 659
OUTPUT parameter
default 234
description 233
example 236
location in JCL 235
of DD statement 233, 234, 235, 236
override 235
relationship to other parameter 235
subparameter 234
output queue
for sysout data set 546
output service
in job processing 685
output-group subparameter
of OUTPUT JCL GROUPID parameter 518
overflow
holding 143
overlay-name subparameter
of //*FORMAT FLASH parameter 656
of /*OUTPUT FLASH parameter 627
of DD FLASH parameter 200
of OUTPUT JCL FLASH parameter 511
OVERLAYB parameter
of OUTPUT JCL statement 537
OVERLAYF parameter
of OUTPUT JCL statement 538
OVFL parameter
of JES3 //*FORMAT PR statement 657
of OUTPUT JCL statement 538

# P

p parameter
of /*PRIORITY statement 631
P subparameter
of DCB FUNC subparameter 144
of DD UNIT parameter 295
P11 character set
for 3211 printer 290, 556
PAGE subparameter
of OUTPUT JCL PRMODE parameter 543
page-mode printer
on OUTPUT JCL FORMDEF parameter 513
on OUTPUT JCL PAGEDEF parameter 539
on OUTPUT JCL PRMODE parameter 542
PAGEDEF parameter
description 539
example 540
of OUTPUT JCL statement 539, 540
override 540
subparameter 540
PAGES parameter
of JES2 /*JOBPARM statement 616
of JES3 //*MAIN statement 673
of JOB statement 435
pano subparameter
JES2 format of JOB accounting information 410
parameter
detailed syntax 15
format 13
rules for continuation 17
symbolic
overriding a system symbol 41
parentheses
with relational-expression 385
PARM parameter
description 352
example 353
of EXEC statement 352, 353
subparameter 353

PARMDD parameter
data set requirements 355
description 354
example 356
of EXEC statement 354, 355, 356
parameter string requirements 355
record length requirements 355
relationship to other control statement 355
partition-name subparameter
of //*MAIN SPART parameter 675
partitioned (PDS)
naming 178, 181
partitioned data set extended(PDSE)
naming 178, 181
PASS subparameter
of DD DISP parameter 164
passed
unit count 295
password
for protection of data set 219
PASSWORD parameter
description 437
example 438
of JOB statement 437, 438
relationship to other parameter 438
subparameter 438
password subparameter
of JOB PASSWORD parameter 438
PASSWORD subparameter
of DD LABEL parameter 220
password1 parameter
of JES2 /*SIGNON statement 638
of JES3 /*SIGNON statement 692
password2 parameter
of JES2 /*SIGNON statement 638
of JES3 /*SIGNON statement 692
PATH parameter
of DD statement 237
PATHDISP parameter
of DD statement 240
PATHMODE parameter
of DD statement 242
PATHOPTS parameter
of DD statement 245
PC parameter
abbreviation of NRCMP parameter of JES3 //*NET statement 679
PCAN character set
for 1403 and 3203 Model 5 printer 290, 556
PCHN character set
for 1403 and 3203 Model 5 printer 290, 556
PCI subparameter
of DD DCB parameter 148
PDS subparameter
of DD DSNTYPE parameter 185
PEND statement
comments field 567
description 567
example 567
in JCL 567
location in JCL 567
name field 567
operation field 567
PERFORM parameter
default 357, 439
description 356, 438
example 358, 440
of EXEC statement 356, 357, 358
of JOB statement 438, 439, 440
override 357, 439
subparameter 357, 439
permanent
naming 178
PGM parameter
description 358
example 359
of EXEC statement 358, 359
subparameter 358
PIMSG parameter
default 541
description 540
example 541
of OUTPUT JCL statement 540, 541
subparameter 541
PIPE subparameter
of DD DSNTYPE parameter 185
PN character set
for 1403 and 3203 Model 5 printer 290, 556
PNAME parameter
of JES3 //*NETACCT statement 683
PO subparameter
of DCB DSORG subparameter 143
PORTNO parameter
of OUTPUT JCL statement 542
positional
on AFTER statement 605
on DD statement 85
on EXEC statement 334
on JOB statement 404, 608
on JOBGROUP statement 595
optionally required by installation 409, 440
syntax 15
POU subparameter
of DCB DSORG subparameter 143
PR parameter
of JES3 //*FORMAT PR statement 651
PREFER subparameter
of //*MAIN THWSSEP parameter 676
primary-qty subparameter
of //*MAIN TRKGRPS parameter 677
of DD SPACE parameter 677
PRINT parameter
of JES2 /*ROUTE statement 632
PRINT subparameter
of //*MAIN FAILURE parameter 670
PRINTDEV statement
DATACK default 499
defined resource libraries 548
example 82
PIMSG default 541
printed output
with OUTPUT JCL DPAGELBL parameter 507
printed output *(continued)*
with OUTPUT JCL DUPLEX parameter 508
with OUTPUT JCL SYSAREA parameter 552
printing
specifying on OUTPUT JCL statement 542
priority
APG (automatic priority group) 143
dispatching 143
initiation or selection 143
of lines for transmission 143
of operator 143
output queue 143
queue selection 143
priority subparameter
of JOB PRTY parameter 442
private
cataloging procedure 27
retrieving procedure 28
specifying for job 313, 397
specifying for step 316
specifying in PRIVATE subparameter 302
PRIVATE subparameter
of DD VOLUME parameter 302
PRMODE parameter
default 543
description 542
example 543
of OUTPUT JCL statement 542, 543
subparameter 542
PROC parameter
description 360
example 360
of EXEC statement 360
of JES3 //*MAIN statement 674
subparameter 360
PROC statement
comments field 570
description 569
example 570
in JCL 569, 570
location in JCL 570
name field 569
operation field 570
override 570
parameter field 570
procedure
adding 27
calling search order 28
cataloged and in-stream 27, 28
description 5
testing 5
nested 27, 28
private library 27
search order 28
procedure-name subparameter
of EXEC PROC parameter 360
process-mode subparameter
of OUTPUT JCL PRMODE parameter 543
processing
controlling in JES3 system 685
specifying control 403
specifying control in JES3 system 664

processor-id subparameter
of //*MAIN ACMAIN parameter 666
PROCLIB parameter
of JES2 /*JOBPARM statement 616
PROCLIB, ordering searches with JCLLIB statement 397
procstepname subparameter
of EXEC ACCT parameter 339
of EXEC ADDRSPC parameter 341
of EXEC COND parameter 344, 345
of EXEC DYNAMNBR parameter 351
of EXEC PARM parameter 353
of EXEC PERFORM parameter 358
of EXEC RD parameter 363
of EXEC TIME parameter 371
subparameter of EXEC REGION parameter 366
subparameter of EXEC REGIONX parameter 368
subparameter of JOB REGIONX parameter 450
profile-name subparameter
of DD SECMODEL parameter 262
program
control 81, 82, 331
end 331
execution 358
location of executable program 313, 316
statement 82
PROGRAM subparameter
of //*FORMAT CONTROL parameter 653
of OUTPUT JCL CONTROL parameter 494
program-name subparameter
of EXEC PGM parameter 358
programmer's-name parameter
description 440
example 441
of JOB statement 440, 441
parameter 441
programmer's-name subparameter
of //*NETACCT PNAME parameter 683
PROTECT parameter
description 249
example 251
of DD statement 249, 250, 251
override 250
relationship to other parameter 250
requirements for protecting direct access data set 251
requirements for protecting tape data set 250
requirements for protecting tape volume 250
subparameter 250
protection
through DD PROTECT parameter 249
through DD SECMODEL parameter 262
with OUTPUT JCL DPAGELBL parameter 507
protection *(continued)*
with OUTPUT JCL DUPLEX parameter 508
with OUTPUT JCL SYSAREA parameter 552
PRTOPTNS parameter
description 545
of OUTPUT JCL statement 545, 546
subparameter 546
PRTQUEUE parameter
description 546
of OUTPUT JCL statement 546
subparameter 546
PRTY parameter
default 442, 547
description 442, 546
example 443, 547
of JES3 //*FORMAT PR statement 657
of JOB statement 442, 443
of OUTPUT JCL statement 546, 547
override 547
subparameter 442, 547
PS subparameter
of DCB DSORG subparameter 143
PSF (Print Services Facility)
printing data set 513, 514, 539
printing line-mode data 543
table reference character codes in JES2 system 555
with DD CHARS parameter 122
with DD UCS parameter 292
with OUTPUT JCL CHARS parameter 487
with OUTPUT JCL DATACK parameter 498
with OUTPUT JCL PIMSG parameter 540
with OUTPUT JCL UCS parameter 557
PSU subparameter
of DCB DSORG subparameter 143
PU parameter
of JES3 //*FORMAT PU statement 660
PUNCH parameter
of JES2 /*ROUTE statement 632
purge service
in job processing 685

# Q

Q subparameter
of DCB OPTCD subparameter 147
QN character set
for 1403 and 3203 Model 5 printer 290, 556
QNC character set
for 1403 and 3203 Model 5 printer 290, 556
QSAM (queued sequential access method)
subparameters of DD DCB parameter 140
with DD CHKPT parameter 124
qualified
for data set 178
queue selection
requested on JES2 /*PRIORITY statement 631
specifying 442

# R

R parameter
of JES3 /*SIGNON statement 691
R subparameter
of //*NET ABNORMAL parameter 680
of //*NET NORMAL parameter 680
of DCB BFTEK subparameter 140
of DCB CPRI subparameter 143
of DCB FUNC subparameter 144
of DCB MODE subparameter 145
of DCB OPTCD subparameter 146, 148
of DCB PCI subparameter 148
of EXEC RD parameter 362
of JOB RD parameter 444
RACF (Resource Access Control Facility)
discrete profile 249
new password 437
protection 179, 181, 249, 423, 437, 453, 465, 507, 508, 552
RACF-defined group 423
RACF-defined password 437
RACF-defined user 465
with in-stream data set 181
with JOB SECLABEL parameter 453
with OUTPUT JCL DPAGELBL parameter 507
with OUTPUT JCL DUPLEX parameter 508
with OUTPUT JCL SYSAREA parameter 552
with sysout data set 181
with temporary data set 179
RC keyword
of IF/THEN/ELSE/ENDIF statement construct 382
RCK subparameter
of AMP CROPS subparameter 107
RD parameter
default 363, 445
description 361, 443
example 363, 445
of EXEC statement 361, 362, 363
of JOB statement 443, 444, 445
override 363, 445
relationship to other control statement 363, 445
subparameter 362, 444
reader
internal 583, 588
description 5
with JES3 XMIT JCL statement 583, 588
real
requesting for job 412
requesting for step 340
REAL subparameter
of EXEC ADDRSPC parameter 340
of JOB ADDRSPC parameter 412

RECFM parameter
description 251
example 254
of DD statement 251, 254
override 254
relationship to other parameter 254
RECFM subparameter
of DD AMP parameter 109
of records 109
reclgth subparameter
of DD SPACE parameter 266
record
specifying length 113, 145, 226
specifying organization 254
record length
of new data set 226
specifying in the DD SPACE parameter 266
RECORD subparameter
of DD FILEDATA parameter 198
record-level sharing, VSAM 260
RECORG parameter
default 255
description 254
example 255
of DD statement 254, 255
override 255
relationship to other parameter 255
subparameter 255
REF subparameter
of DD VOLUME parameter 305
REFDD parameter
description 256
example 258
of DD statement 256, 257, 258
override 257
relationship to other parameter 257
subparameter 257
region
default 365, 367, 447, 449
size 365, 367, 447, 449
REGION parameter
considerations 366, 448
default 365, 447
description 364, 446
example 366, 448
of EXEC statement 364, 365, 366
of JOB statement 366, 446, 447, 448
override 365, 447
relationship to the EXEC ADDRSPC parameter 366
relationship to the JOB ADDRSPC parameter 447
relationship to the MEMLIMIT parameter 366, 448
relationship to the REGIONX parameter 366, 448
subparameter 364, 446
REGIONX parameter
default 367, 449
description 366, 448
example 369, 450
of EXEC statement 366, 367, 368, 369
of JOB statement 448, 449, 450
override 368, 449
relationship to the EXEC ADDRSPC parameter 368
REGIONX parameter *(continued)*
relationship to the JOB ADDRSPC parameter 450
relationship to the MEMLIMIT parameter 368, 450
relationship to the REGION parameter 369
subparameter 367, 449
syntax 367, 448
rel subparameter
of //*MAIN DEADLINE parameter 668
RELEASE parameter
of JES3 //*NET statement 682
RELSCHCT parameter
of JES3 //*NET statement 682
remote subparameter
of //*FORMAT DEST parameter 655, 662
of //*MAIN ORG parameter 673
REMOTEnnn parameter
of JES2 /*SIGNON statement 637
REPLYTO
of OUTPUT JCL statement description 547
requesting resources 8
tasks 8
task chart 8
REQUIRE subparameter
of //*MAIN THWSSEP parameter 676
reserved name
on DSNAME parameter 182
RESFMT parameter
description 548
example 548
of OUTPUT JCL statement 548
subparameter 548
RESTART parameter
cautions with coding 452
description 450
example 452
of JES2 /*JOBPARM statement 617
of JOB statement 450, 451, 452
relationship to other control statement 452
subparameter 451
RESTART subparameter
subparameter of //*MAIN FAILURE parameter 670
restarting
with EXEC COND parameter 347
RETAIN subparameter
of DD VOLUME parameter 302
RETAINS|RETAINF parameter
description 548
of OUTPUT JCL statement 548
retention
specifying by RETAIN subparameter 302
RETPD parameter
description 258
example 259
of DD statement 258, 259
override 259
relationship to other parameter 259
subparameter 259
RETRYL|RETRYT parameter
description 550
of OUTPUT JCL statement 550
return code
specifying 342, 382, 419
with IF/THEN/ELSE/ENDIF statement construct 382
RINGCHK parameter
of JES3 //*MAIN statement 674
RJE (remote job entry)
DD * statement 102
RJP (remote job processing)
DD * statement 102
RL parameter
abbreviation of RELEASE parameter of JES3 //*NET statement 679
RLS parameter
CR subparameter 260
CRE subparameter 261
description 260
example 261
NRI subparameter 260
of DD statement 260, 261
override 261
relationship to other parameters 261
subparameter 260
syntax 260
RLSE subparameter
of DD SPACE parameter 269
RLSTMOUT parameter
EXEC statement 369
of EXEC statement description 369
Rm parameter
of JES2 /*ROUTE statement 633
Rm subparameter
of /*OUTPUT DEST parameter 626
of DD DEST parameter 156
of OUTPUT JCL DEST parameter 504
RMn parameter
of JES2 /*ROUTE statement 633
of JES2 /*SIGNON statement 637
RMn subparameter
of /*OUTPUT DEST parameter 627
of DD DEST parameter 157
of OUTPUT JCL DEST parameter 505
RMTn parameter
of JES2 /*ROUTE statement 633
of JES2 /*SIGNON statement 637
RMTn subparameter
of /*OUTPUT DEST parameter 627
of DD DEST parameter 157
of OUTPUT JCL DEST parameter 505
RN character set
for 1403 and 3203 Model 5 printer 290, 556
Rn parameter
of JES2 /*ROUTE statement 633
Rn subparameter
of /*OUTPUT DEST parameter 627
of DD DEST parameter 157
of OUTPUT JCL DEST parameter 505

RNC subparameter
of EXEC RD parameter 362
of JOB RD parameter 444
Rnnnn parameter
of JES2 /*SIGNON statement 637
ROOM parameter
description 551
of JES2 /*JOBPARM statement 617
of JES3 //*NETACCT statement 684
of OUTPUT JCL statement 551
subparameter 551
room subparameter
JES2 format of JOB accounting information 410
of //*NETACCT ROOM parameter 684
ROUND subparameter
of DD SPACE parameter 270
RR subparameter
of DD RECORG parameter 255
RS parameter
abbreviation of RELSCHCT parameter
of JES3 //*NET statement 679
RUN keyword
of IF/THEN/ELSE/ENDIF statement construct 382

# S

S subparameter
of //*FORMAT STACKER parameter 657
of DCB BFTEK subparameter 140
of DCB CPRI subparameter 143
of RECFM parameter 252, 253
SCAN subparameter
of JOB TYPRUN parameter 463
scanning for errors
without execution 359
SCHEDULE statement 571
scheduling environment, WLM 454
SCHENV parameter
default 455
description 454
example 455
of JOB statement 454, 455
relationship to other control statements 455
subparameter definition 455
search order
calling a procedure 28
SECLABEL parameter
default 454
description 453
example 454
of JOB statement 453, 454
relationship to other parameter 454
subparameter definition 454
seclabel-name subparameter
of JOB SECLABEL parameter 454
SECMODEL parameter
description 262
example 263
of DD statement 262, 263
override 263
relationship to other parameter 263
subparameter 262
second-qty subparameter
of //*MAIN TRKGRPS parameter 677
of DD SPACE parameter 267
seconds subparameter
of EXEC TIME parameter 370
of JOB TIME parameter 460
security label
on DPAGELBL parameter 507
on DUPLEX parameter 508
on SECLABEL parameter 453
on SYSAREA parameter 552
SEGMENT parameter
description 263
of DD statement 263, 264
override 264
relationship to other parameter 264
subparameter 264
sending comments to IBM xxv
sequence number
specifying in DD LABEL parameter 217
SER subparameter
of DD VOLUME parameter 304
serial numbers
specifying by SER subparameter 304
serial-number parameter
of JES2 /*SETUP statement 635
serial-number subparameter
of VOLUME=SER subparameter 304
setname
coding 602
SETUP parameter
of JES3 //*MAIN statement 675
SETUP subparameter
of //*MAIN FETCH parameter 671
shortcut keys 693
SHR subparameter
of DD DISP parameter 162
SINGLE subparameter
of //*FORMAT CONTROL parameter 653
of OUTPUT JCL CONTROL parameter 494
size
specifying 364, 366, 446, 448
SJOB statement
comments field 601
error on statement 601
in JCL 600, 601
location in JCL 601
name field 600
operation field 601
parameter field 601
SKP subparameter
of DCB EROPT subparameter 144
SL subparameter
of DD LABEL parameter 218
SMS (Storage Management Subsystem)
with data set password protection 219
with DD AMP parameter 105
with DD AVGREC parameter 113
with DD DATACLAS parameter 132
with DD DCB parameter 135
with DD DSNTYPE parameter 184
with DD EXPDT parameter 193
SMS (Storage Management Subsystem) *(continued)*
with DD KEYLEN parameter 213
with DD KEYOFF parameter 215
with DD LIKE parameter 224
with DD LRECL parameter 226
with DD MAXGENS parameter 228
with DD MGMTCLAS parameter 228
with DD RECFM parameter 251
with DD RECORG parameter 254
with DD REFDD parameter 256
with DD RETPD parameter 258
with DD SECMODEL parameter 262
with DD SPACE parameter 271
with DD SPACE reclgth subparameter 266
with DD STORCLAS parameter 275
with DD UNIT parameter 293
with DD VOLUME=REF subparameter 308
SMS-managed data set
definition 276
with data set password protection 219
with DD MGMTCLAS parameter 228
with DD STORCLAS parameter 275
with DD VOLUME=REF subparameter 308
with temporary data set 179
SMSHONOR subparameter
of DD UNIT parameter 296
SN character set
for 1403 and 3203 Model 5 printer 290, 556
SPACE parameter
description 264
example 271
of DD statement 264, 266, 271
override 271
relationship to other parameter 271
specifying for data sets with SMS 271
subparameter 266
SPART parameter
of JES3 //*MAIN statement 675
special character set
specifying 290, 556
use 292, 557
use in parameter 22
use in syntax 21
special DD statement
description 313
specifying
on the EXEC statement 356
on the JOB statement 438
SPIN parameter
default 274
description 273
of DD statement 273, 274
override 274
relationship to other parameter 274
subparameter 273
ST subparameter
of //*MAIN PROC parameter 674

STACK subparameter
of DD DCB parameter 149
STACKER parameter
of JES3 //*FORMAT PR statement 657
standard processing
description 685
STANDARD subparameter
of //*FORMAT CHARS parameter 652
of //*FORMAT FLASH parameter 656
of //*FORMAT FORMS parameter 656, 663
of //*FORMAT STACKER parameter 657
of //*FORMAT TRAIN parameter 657
START command
processing
when member is job 60
when member is procedure 60
STARTBY parameter
SCHEDULE statement 574
STARTBY parameter syntax
SCHEDULE statement 574
STARTBY relationship to other parameters
SCHEDULE statement 574
STARTBY subparameter definition
SCHEDULE statement 574
started task
determining source JCL 59
determining when to use 59
START command processing
when member is job 60
when member is procedure 60
started tasks
GJOB statement 598
JES2 considerations 611
JES3 considerations 643
JOB statement 404
JOBSET statement 600
statement
EXPORT 375
syntax 375
statement fields
chart 14
comments 14, 16
continuation to following statement 16
identifier 14, 16
location in statement 14
name 14, 16
operation 14, 16
parameter 14, 16
statements subparameter
of JOB MSGLEVEL parameter 432
static system symbol 38
status
coded in DD DISP parameter 161
STD subparameter
of /*JOBPARM FORMS parameter 615
of /*OUTPUT FORMS parameter 628
of OUTPUT JCL FCB parameter 510
STD subparameter *(continued)*
of OUTPUT JCL FLASH parameter 512
of OUTPUT JCL FORMS parameter 515
on OUTPUT JCL CHARS parameter 486
STD1 forms control buffer image
on DD FCB parameter 195
on OUTPUT JCL FCB parameter 509
STD2 forms control buffer image
on DD FCB parameter 195
on OUTPUT JCL FCB parameter 509
STD3 forms control buffer image
on DD FCB parameter 195
on OUTPUT JCL FCB parameter 509
step
beginning 333
description 5
maximum number 5
step-level
OUTPUT JCL statement 479
STEPLIB DD statement
description 316
example 318
location in JCL 318
parameter 317
relationship to JOBLIB 318
relationship to other control statement 318
stepname
coding 333
stepname subparameter
of EXEC COND parameter 344
of JOB RESTART parameter 451
stepname.ddname
on DD statement referenced by DD DDNAME parameter 152
stepname.ddname subparameter
of //*FORMAT DDNAME parameter 651, 660
of //*MAIN SETUP parameter 675
stepname.procstepname subparameter
of JOB RESTART parameter 451
stepname.procstepname.ddname subparameter
of //*FORMAT DDNAME parameter 651, 660
of //*MAIN SETUP parameter 675
storage-class-name subparameter
of DD STORCLAS parameter 276
STORCLAS parameter
default 276
description 275
example 277
of DD statement 275, 276, 277
override 276
relationship to other parameter 276
subparameter 276
with DD UNIT parameter 293
STRNO subparameter
of DD AMP parameter 109
SUB=MSTR option
started task 64
SUBCHARS parameter 588
default 589
description 588
SUBCHARS parameter *(continued)*
example 589
processing when invalid 589
subparameter 588
XMIT JCL statement parameter 588, 589
subparameter
coding when multiple 15
syntax 15
subparameter subparameter
of DD DCB parameter 137
substitute subparameter
of XMIT JCL SUBCHARS parameter 588
SUBSYS parameter
description 277
example 279
of DD statement 277, 278, 279
relationship to other parameter 278
running under the master subsystem 64
subparameter 278, 279
subsystem-name subparameter
of DD SUBSYS parameter 278
subsystem-subparameter subparameter
of DD SUBSYS parameter 278
SUL subparameter
of DD LABEL parameter 218
summary of changes xxvii
as updated December 2013 xxviii
Summary of changes xxix
symbol
JCL symbol 38
system symbol 38
symbolic
coding 38, 43
default substitution text 41
defining 39
example 50
in nested procedure 51
location 43
nullifying 39, 42
purpose 38
syntax 40
symbols
in-stream data
JES 54
SYMBOLS parameter 280
SYMLIST parameter 281, 376
SYNAD subparameter
of DD AMP parameter 109
syntax
for continuing statement 16
format of statement 13
notation 19
of parameter 19
scanning for error 462
scanning for errors 13, 16, 19, 462
SYS1.PROCLIB system procedure library
use for procedure 5
SYSABEND DD statement
description 319
example 322
location in JCL 320
overriding 321
SYSAFF parameter
default 457

SYSAFF parameter *(continued)*
description 455
examples 457
of JES2 /*JOBPARM statement 617
of JOB statement 455, 456, 457
relationship to other control statements 457
subparameter definition 456
SYSALLDA group name
assumed when in-stream data set referenced 309
SYSAREA parameter
default 553
description 552
example 553
of OUTPUT JCL statement 552, 553
relationship to other parameter 553
subparameter definition 552
SYSCHK DD statement
description 323
example 325
location in JCL 325
parameter 323
relationship to other control statement 324
SYSCKEOV DD statement
description 325
example 326
location in JCL 326
parameter 325
relationship to DD CHKPT parameter 124
SYSIN DD statement
description 326
example 326
location in JCL 326
parameter 326
SYSMDUMP DD statement
description 319
example 322
location in JCL 320
overriding 321
sysout (system output data set)
associating with an OUTPUT JCL statement 233
references to OUTPUT JCL statement 478
specifying through DD SYSOUT parameter 283
with DSNAME parameter 181
SYSOUT parameter
default 285
description 283
example 287
of DD statement 283, 284, 285, 286, 287
override 285
relationship to DD COPIES parameter 128
relationship to other control statement 286
relationship to other parameter 286
subparameter 284
with DEST=(node) subparameter 158
system completion code
with IF/THEN/ELSE/ENDIF statement construct 382
SYSTEM parameter
description 457
examples 458
of JES3 //*MAIN statement 676
of JOB statement 457, 458
relationship to other control statements 458
system symbol
coding 39, 43
in started task 56, 66, 69
overridden by JCL symbol 41
using in JCL 38
system-managed
sending to other destination 522
specifying processing 521
SystemName parameter
of JOB statement 458
subparameter definition 458
SYSUDUMP DD statement
description 319
example 322
location in JCL 320
overriding 321
SYSUID system symbolic parameter
description 49
restriction 50
use in transaction program profile 49

# T

T subparameter
of DCB EROPT parameter 144
of DCB FUNC subparameter 144
of DCB OPTCD subparameter 147
of DCB TRTCH subparameter 150
of RECFM parameter 252, 253
T11 character set
for 3211 printer 290, 556
table-name subparameter
of //*FORMAT CHARS parameter 652
on DD CHARS parameter 122
on OUTPUT JCL CHARS parameter 486
task
chart 6
description 5
for entering jobs
chart 6
for processing jobs
chart 8
for requesting sysout data set resources
chart 10
temporary
naming 181
temporary data set 179
TERM parameter
description 288
example 289
location in JCL 289
of DD statement 288, 289
relationship to other parameter 289
subparameter 289
terminal
data coming from or going to a terminal 288
termination
abnormal 342, 382, 419, 433
normal 342, 382, 419, 433
notification 433
testing return code 342, 382, 419
test
of on-line terminal 144
return code 144
TEXT subparameter
of DD FILEDATA parameter 198
test 198
THRESH subparameter
of DD DCB parameter 149
THRESHLD parameter
default 553
description 553
example 554
of JES3 //*FORMAT PR statement 657
of OUTPUT JCL statement 553, 554
subparameter 553
THWS subparameter
of //*MAIN SETUP parameter 675
THWSSEP parameter
of JES3 //*MAIN statement 676
time
use by job 459
use by job step 370
TIME parameter
default 371, 460
description 370, 459
example 372, 460
of EXEC statement 370, 371, 372
of JES2 /*JOBPARM statement 618
of JOB statement 459, 460
override 371, 460
subparameter 370, 459
time subparameter
of //*MAIN DEADLINE parameter 668
TIME subparameter
JES2 format of JOB accounting information 410
TITLE parameter
description 554
of OUTPUT JCL statement 554
subparameter 554
TN character set
for 1403 and 3203 Model 5 printer 290, 556
to remote node
input stream for execution 688
TRACE subparameter
of DCB DIAGNS subparameter 143
of DD AMP parameter 109
trace
of OPEN/CLOSE/EOV 143
track
specifying in DD SPACE parameter 266
track subparameter
of DCB CYLOFL subparameter 143
of DCB LIMCT subparameter 145
of DCB NCP parameter 146
trademarks 699

TRAIN parameter
of JES3 //*FORMAT PR statement 657
train-name subparameter
of //*FORMAT TRAIN parameter 657
transmission
of input stream to network node 688
of input stream using XMIT JCL statement 583
TRC parameter
default 555
description 555
example 555
of OUTPUT JCL statement 555
relationship to other parameter 555
subparameter 555
trc subparameter
of //*FORMAT FORMS parameter 656
of /*OUTPUT MODIFY parameter 629
of /*OUTPUT MODTRC parameter 630
of DD MODIFY parameter 231
of OUTPUT JCL MODIFY parameter 530
TRIPLE subparameter
of //*FORMAT CONTROL parameter 653
of OUTPUT JCL CONTROL parameter 494
TRK subparameter
of DD SPACE parameter 266
TRKGRPS parameter
of JES3 //*MAIN statement 677
TRTCH subparameter
of DD DCB parameter 150
TS subparameter
of DD TERM parameter 289
TUMBLE subparameter
of OUTPUT JCL DUPLEX parameter 509
TYPE parameter
of JES3 //*MAIN statement 677
type subparameter
of //*FORMAT DEST parameter 655, 662
of //*MAIN DEADLINE parameter 668
TYPRUN parameter
description 462
example 463
of JOB statement 462, 463
relationship to other control statement 463
subparameter 462

# U

U subparameter
of DCB OPTCD subparameter 147, 148
of DD AVGREC parameter 113
of DD RECFM parameter 252, 253
UCS parameter
default 291, 556
UCS parameter *(continued)*
description 290, 556
example 292, 558
of DD statement 290, 291, 292
of JES2 /*OUTPUT statement 630
of OUTPUT JCL statement 556, 557, 558
override 292, 557
relationship to other parameter 292
subparameter 290, 556
UJOBCORR parameter
description 464
examples 465
of JOB statement 464, 465
subparameter definition 465
Un parameter
of JES2 /*ROUTE statement 633
Un subparameter
of /*OUTPUT DEST parameter 627
of DD DEST parameter 157
of OUTPUT JCL DEST parameter 505
UNBLOCK subparameter
of OUTPUT JCL DATACK parameter 498
UNCATLG subparameter
of DD DISP parameter 165, 166
unit affinity
specifying in AFF subparameter 296
when DDNAME parameter is also coded 152
UNIT parameter
description 293
example 298
location in JCL 298
of DD statement 293, 297, 298
override 297
relationship to DD COPIES parameter 128
relationship to other control statements 298
relationship to other parameter 298
subparameter 293
unit-count subparameter
of DD UNIT parameter 295
universal character set (UCS)
specifying 290, 556
unqualified
for data set 178
UPDATE parameter
of JES3 //*MAIN statement 678
uppercase
in syntax 19
user completion code
with IF/THEN/ELSE/ENDIF statement construct 382
user interface
ISPF 693
TSO/E 693
USER parameter
default 466
description 465
example 466
of JES3 //*MAIN statement 678
of JOB statement 465, 466
relationship to other parameter 466
subparameter 466
USERDATA parameter
of OUTPUT JCL statement 558
userid
data coming from or going to a user 288
userid parameter
of JES2 /*NOTIFY statement 621
of JES2 /*ROUTE statement 633
of JES2 /*XMIT statement 641
userid subparameter
of //*MAIN USER parameter 678
of //*NETACCT USERID parameter 684
of /*OUTPUT DEST parameter 626
of DD DEST parameter 157, 158
of JES3 //*NETACCT statement 684
of JOB NOTIFY parameter 434
of JOB USER parameter 466
of OUTPUT JCL DEST parameter 505
USERLIB parameter
of OUTPUT JCL statement 561
userpath
of OUTPUT JCL statement 562

# V

V subparameter
of AMP RECFM subparameter 109
of DD DSID parameter 175
of DD RECFM parameter 252, 253
value1 subparameter
subparameter of EXEC REGIONX parameter 367
subparameter of JOB REGIONX parameter 449
value2 subparameter
subparameter of EXEC REGIONX parameter 367
subparameter of JOB REGIONX parameter 449
valueK subparameter
subparameter of EXEC REGION parameter 364
subparameter of JOB REGION parameter 446
valueM subparameter
subparameter of EXEC REGION parameter 365
subparameter of JOB REGION parameter 446
VB subparameter
of AMP RECFM subparameter 109
verification
of FCB image 196
of forms overlay frame 201, 512
VERIFY subparameter
of DD FCB parameter 196
of DD UCS parameter 291
VIRT subparameter
of EXEC ADDRSPC parameter 340
of JOB ADDRSPC parameter 412
virtual
requesting for job 412
requesting for step 340
vmguestid parameter
of JES2 /*ROUTE statement 634

vmguestid parameter *(continued)*
of JES2 /*XEQ statement 639
of JES2 /*XMIT statement 641
of JES3 //*ROUTE XEQ 689
vmguestid subparameter
of XMIT JCL DEST parameter 586
volume
specifying by RETAIN subparameter 302
VOLUME parameter
description 300
example 310
in JES3 system 309
of DD statement 300, 302, 309, 310
override 309
relationship to other parameter 309
subparameter 302
volume-count subparameter
of DD VOLUME parameter 303
volume-sequence-number subparameter
of DD VOLUME parameter 303
VS2 subparameter
of //*MAIN TYPE parameter 677
VSAM (virtual storage access method)
record-level sharing 260
referenced in VOLUME=REF subparameter 308
with DD AMP parameter 105
with DD DATACLAS parameter 133
with DD RECORG parameter 254
with DD RLS parameter 260

# W

W subparameter
of //*MAIN BYTES parameter 667
of //*MAIN CARDS parameter 667
of //*MAIN LINES parameter 672
of //*MAIN PAGES parameter 673
of DCB FUNC subparameter 144
of DCB OPTCD subparameter 146, 147, 148
WARNING subparameter
of //*MAIN BYTES parameter 667
of //*MAIN CARDS parameter 667
of //*MAIN LINES parameter 672
of //*MAIN PAGES parameter 673
of JOB BYTES parameter 414
of JOB CARDS parameter 415
of JOB LINES parameter 428
of JOB PAGES parameter 436
WITH parameter
SCHEDULE statement 575
WITH parameter syntax
SCHEDULE statement 575
WITH subparameter definition
SCHEDULE statement 575
WLM scheduling environment 454
work-station-name parameter
of JES3 /*SIGNON statement 691
WRITER parameter
default 564
description 564
example 564
of OUTPUT JCL statement 564
override 564
relationship to other parameter 564
WRITER parameter *(continued)*
subparameter 564
writer-name subparameter
of DD SYSOUT parameter 284
with DEST=(node) subparameter 158

# X

x subparameter
of //*MAIN PROC parameter 674
of /*JOBPARM FORMS parameter 615
of /*JOBPARM ROOM parameter 617
of /*OUTPUT CHARS parameter 624
of /*OUTPUT FCB parameter 627
of /*OUTPUT FORMS parameter 628
of /*OUTPUT UCS parameter 630
of /*XMIT DLM parameter 641
of DCB EROPT subparameter 144
X subparameter
of DCB FUNC subparameter 144
of DCB PCI subparameter 148
of LRECL parameter 227
XEQ parameter
of JES2 /*ROUTE statement 632
XMIT statement
comments field 584
description 583
error on statement 585
example 585
in JCL 583, 584, 585
location in JCL 584
name field 584
operation field 584
parameter field 584
support 583
XN character set
for 1403 and 3203 Model 5 printer 290, 556

# Y

Y subparameter
of /*JOBPARM BURST parameter 615
of /*JOBPARM RESTART parameter 617
of /*OUTPUT BURST parameter 624
of DCB OPTCD subparameter 148
of DD PROTECT parameter 250
YES subparameter
of //*DATASET DDNAME parameter 647
of //*FORMAT FORMS parameter 663
of //*MAIN EXPDTCHK parameter 670
of //*MAIN HOLD parameter 671
of //*MAIN JOURNAL parameter 672
of //*MAIN RINGCHK parameter 674
YES subparameter *(continued)*
of //*NET DEVRELSE parameter 681
of //*NET OPHOLD parameter 682
of DD BURST parameter 118
of DD HOLD parameter 207
of DD PROTECT parameter 250
of OUTPUT JCL AFPSTATS parameter 483
of OUTPUT JCL BURST parameter 485
of OUTPUT JCL DEFAULT parameter 500
of OUTPUT JCL DPAGELBL parameter 508
of OUTPUT JCL PIMSG parameter 541
of OUTPUT JCL SYSAREA parameter 552
of OUTPUT JCL TRC parameter 555
YN character set
for 1403 and 3203 Model 5 printer 290, 556
yyddd subparameter
of DD EXPDT parameter 193
yyyy/ddd subparameter
of DD EXPDT parameter 193

# Z

Z subparameter
of DCB OPTCD subparameter 147

Product Number: 5650-ZOS

Printed in USA

SA23-1385-03