

Db2 13 for z/OS

Troubleshooting for Db2
Last updated: 2025-05-07



Notes

Before using this information and the product it supports, be sure to read the general information under "Notices" at the end of this information.

2025-05-07 edition

This edition applies to Db2[®] 13 for z/OS[®] (product number 5698-DB2), Db2 13 for z/OS Value Unit Edition (product number 5698-DBV), and to any subsequent releases until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Specific changes are indicated by a vertical bar to the left of a change. A vertical bar to the left of a figure caption indicates that the figure has changed. Editorial changes that have no technical significance are not noted.

© **Copyright International Business Machines Corporation 1983, 2025.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

| | |
|--|-----------|
| About this information..... | ix |
| Who should read this information..... | ix |
| Db2 Utilities Suite for z/OS..... | x |
| Terminology and citations..... | x |
| Accessibility features for Db2 for z/OS..... | x |
| How to send comments..... | xi |
| How to read syntax diagrams..... | xi |
| | |
| Chapter 1. Introduction to troubleshooting for Db2..... | 1 |
| | |
| Chapter 2. Searching for known problems and solutions for Db2 for z/OS..... | 3 |
| Building a keyword string..... | 3 |
| Db2 format (free format)..... | 4 |
| Structured database format..... | 6 |
| Component identifier keyword..... | 10 |
| Release level keyword..... | 12 |
| Type-of-failure keywords..... | 12 |
| Dependency keywords..... | 49 |
| Techniques for varying the search..... | 50 |
| Getting fixes..... | 51 |
| | |
| Chapter 3. Recovering from different Db2 for z/OS problems..... | 53 |
| Recovering from IRLM failure..... | 53 |
| Recovering from z/OS or power failure..... | 53 |
| Recovering from disk failure..... | 54 |
| Recovering from application errors..... | 56 |
| Backing out incorrect application changes (with a quiesce point)..... | 56 |
| Backing out incorrect application changes (without a quiesce point)..... | 57 |
| Recovering from IMS-related failures | 57 |
| Recovering from IMS control region failure | 58 |
| Recovering from IMS indoubt units of recovery..... | 58 |
| Recovering from IMS application failure..... | 60 |
| Recovering from a Db2 failure in an IMS environment..... | 61 |
| Recovering from CICS-related failure | 61 |
| Recovering from CICS application failures..... | 62 |
| Recovering Db2 when CICS is not operational | 62 |
| Recovering Db2 when the CICS attachment facility cannot connect to Db2 | 63 |
| Recovering CICS indoubt units of recovery..... | 64 |
| Recovering from CICS attachment facility failure | 66 |
| Recovering from a QMF query failure..... | 66 |
| Recovering from subsystem termination | 67 |
| Recovering from temporary resource failure | 68 |
| Recovering from active log failures | 69 |
| Recovering from being out of space in active logs | 69 |
| Recovering from a write I/O error on an active log data set | 70 |
| Recovering from a loss of dual active logging | 71 |
| Recovering from I/O errors while reading the active log | 72 |
| Recovering from archive log failures | 73 |
| Recovering from allocation problems with the archive log | 73 |
| Recovering from write I/O errors during archive log offload | 74 |

| | |
|---|------------|
| Recovering from read I/O errors on an archive data set during recovery | 75 |
| Recovering from insufficient disk space for offload processing | 75 |
| Recovering from BSDS failures..... | 76 |
| Recovering from an I/O error on the BSDS | 76 |
| Recovering from an error that occurs while opening the BSDS | 77 |
| Recovering from unequal timestamps on BSDSs | 77 |
| Recovering the BSDS from a backup copy..... | 78 |
| Recovering from BSDS or log failures during restart..... | 80 |
| Recovering from failure during log initialization or current status rebuild..... | 83 |
| Recovering from a failure during forward log recovery..... | 93 |
| Recovering from a failure during backward log recovery..... | 98 |
| Recovering from a failure during a log RBA read request..... | 100 |
| Recovering from unresolvable BSDS or log data set problem during restart..... | 101 |
| Recovering from a failure resulting from total or excessive loss of log data..... | 103 |
| Resolving inconsistencies resulting from a conditional restart..... | 107 |
| Recovering from Db2 database failure | 111 |
| Recovering a Db2 subsystem to a prior point in time..... | 113 |
| Recovering the catalog and directory to a point in time before a CATMAINT or a function level upgrade in a data sharing environment..... | 114 |
| Recovering the catalog and directory to a point in time before a CATMAINT or a function level upgrade in a non-data sharing environment..... | 115 |
| Recovering from a down-level page set problem | 116 |
| Recovering from a problem with invalid LOBs..... | 117 |
| Recovering from table space I/O errors | 118 |
| Recovering from Db2 catalog or directory I/O errors | 119 |
| Recovering from integrated catalog facility failure | 120 |
| Recovering VSAM volume data sets that are out of space or destroyed..... | 120 |
| Recovering from out-of-disk-space or extent limit problems | 121 |
| Recovering from referential constraint violation | 125 |
| Recovering from distributed data facility failure | 126 |
| Recovering from conversation failure | 126 |
| Recovering from communications database failure..... | 127 |
| Recovering from database access thread failure | 128 |
| Recovering from VTAM failure | 128 |
| Recovering from VTAM ACB OPEN problems..... | 129 |
| Recovering from TCP/IP failure | 130 |
| Recovering from remote logical unit failure | 130 |
| Recovering from an indefinite wait condition..... | 131 |
| Recovering database access threads after security failure | 131 |
| Performing remote-site disaster recovery | 132 |
| Recovering from a disaster by using system-level backups..... | 132 |
| Restoring data from image copies and archive logs..... | 133 |
| Recovering from disasters by using a tracker site..... | 147 |
| Using data mirroring for disaster recovery..... | 156 |
| Scenarios for resolving problems with indoubt threads..... | 162 |
| Scenario: Recovering from communication failure | 163 |
| Scenario: Making a heuristic decision about whether to commit or abort an indoubt thread..... | 165 |
| Scenario: Recovering from an IMS outage that results in an IMS cold start..... | 167 |
| Scenario: Recovering from a Db2 outage at a requester that results in a Db2 cold start..... | 168 |
| Scenario: What happens when the wrong Db2 subsystem is cold started..... | 171 |
| Scenario: Correcting damage from an incorrect heuristic decision about an indoubt thread..... | 173 |
| Troubleshooting Db2 stored procedure problems..... | 174 |
| Identifying Db2 data inconsistency problems..... | 174 |
| Data inconsistency symptoms and actions..... | 175 |
| Chapter 4. Diagnostic aids for single systems and data sharing..... | 181 |
| Printing and analyzing dumps..... | 181 |

| | |
|---|------------|
| Printing dumps..... | 182 |
| Format dumps by using IPCS options..... | 182 |
| Format dumps by using the DSNWDMP statement..... | 183 |
| Analyze Db2 storage by using the SM options..... | 193 |
| SVC dumps..... | 196 |
| Redacting buffer pool data in SVC dumps for data privacy..... | 223 |
| Suppression of SVC dumps by using z/OS DAE..... | 223 |
| When SVC dumps are not produced..... | 224 |
| SYSUDUMP dumps..... | 224 |
| SYS1.LOGREC..... | 226 |
| Printing and analyzing global traces..... | 238 |
| Global trace facility..... | 239 |
| Starting the global trace..... | 240 |
| Displaying global trace activity..... | 243 |
| Modifying global trace activity..... | 243 |
| Stopping global trace activity..... | 243 |
| Using global trace output..... | 244 |
| Call attachment facility traces..... | 252 |
| IMS attachment facility traces and IMS log record..... | 254 |
| Resource Recovery Services attachment facility traces..... | 261 |
| TSO attachment facility traces..... | 263 |
| IRLM - Db2 activity trace..... | 268 |
| Diagnosing EDM pool space problems using traces..... | 274 |
| z/OS traces..... | 275 |
| Writing Db2 log buffers to IFI..... | 275 |
| Db2 stand-alone log services: change log inventory and print log map..... | 275 |
| Diagnostic information in Db2 catalog tables..... | 276 |
| SQLDA extension for binary XML data..... | 277 |
| Db2 utilities for troubleshooting..... | 278 |
| Db2 commands for troubleshooting..... | 280 |
| Program call linkages..... | 281 |
| TSO attachment facility diagnostic aids..... | 282 |
| SPUFI diagnostic panels..... | 282 |
| ABEND subcommand of the DSN command processor..... | 283 |
| SYS1 service aids..... | 284 |
| SYS1.DUMPXX..... | 284 |
| SYS1.LOGREC..... | 284 |
| Output from the MODIFY command..... | 284 |
| Data sharing problem diagnosis..... | 285 |
| Hangs in a data sharing environment..... | 285 |
| Timeouts and deadlocks in a data sharing environment..... | 286 |
| IRLM delays in a data sharing environment..... | 286 |
| Inconsistent data in a data sharing environment..... | 287 |
| Query parallelism problem diagnosis..... | 288 |
| Determine if a query problem is related to parallelism..... | 288 |
| Types of parallelism problems..... | 289 |
| Diagnose parallelism problems by using traces..... | 290 |
| Chapter 5. Diagnostic aids for distributed data..... | 293 |
| Diagnosing distributed data facility (DDF) failures..... | 293 |
| Distributed SQL application flow for VTAM connections..... | 293 |
| Distributed SQL application flow for TCP/IP connections..... | 294 |
| DDF error messages..... | 294 |
| TCP/IP startup problems..... | 297 |
| Db2 hangs during distributed processing..... | 298 |
| Problem determination procedures..... | 299 |
| Diagnostic tools for DDF and VTAM..... | 307 |

| | |
|---|------------|
| VTAM traces..... | 307 |
| Storage Management Services (buffer use) trace..... | 315 |
| Exception condition diagnostic procedures..... | 316 |
| DRDA exception condition diagnostic procedures..... | 316 |
| DRDA exception event notification..... | 316 |
| DRDA summary..... | 318 |
| DDIS IFCID 0191 trace record structure..... | 324 |
| DDIS IFCID 0191 common diagnostic procedures..... | 325 |
| IFCID 0191 trace record common diagnostic procedures..... | 330 |
| Distributed two-phase commit error conditions..... | 331 |
| Chapter 6. Data management..... | 335 |
| Resolving inconsistencies manually..... | 335 |
| Analyzing 00C9010X or 00C902XX abends..... | 335 |
| Analyzing 00C90102 abends..... | 338 |
| Running REPAIR..... | 342 |
| Inconsistency resolution with RECOVER TABLESPACE and RECOVER INDEX..... | 343 |
| RECOVER preparation..... | 343 |
| Running RECOVER TABLESPACE and REBUILD INDEX..... | 344 |
| Diagnosing DBD inconsistencies..... | 344 |
| Using REPAIR DBD..... | 344 |
| Finding a DBD in a dump..... | 345 |
| Analyzing a DBD..... | 349 |
| Resolve the inconsistent DBD..... | 360 |
| Chapter 7. Trace messages and codes..... | 361 |
| Db2 trace codes..... | 361 |
| Loading EID descriptions into a table..... | 361 |
| Retrieving EID descriptions..... | 362 |
| TSO attachment facility trace messages..... | 363 |
| Call attachment facility trace messages..... | 452 |
| SPUFI trace messages..... | 457 |
| Chapter 8. Collecting diagnostic data..... | 467 |
| Setting up the z/OS environment to collect diagnostic data..... | 467 |
| Maximizing the size of the z/OS system trace table..... | 468 |
| Increasing the size of the master trace table..... | 468 |
| Increasing the storage capacity for an SVC dump..... | 469 |
| Ensuring that dump data sets are available and automatically updated..... | 469 |
| Setting up a customized Db2 SVC dump..... | 470 |
| Setting up Db2 SLIP traps for a data sharing environment..... | 471 |
| Preserving standard diagnostic documentation..... | 471 |
| Preserving the z/OS console (SYSLOG)..... | 472 |
| Preserving LOGREC data..... | 472 |
| Preserving the JES job logs for key Db2 address spaces..... | 473 |
| Retention of Db2 dump data sets..... | 473 |
| Retention of Db2 logs..... | 473 |
| Requesting Db2 SVC dumps..... | 474 |
| Requesting data sharing environment Db2 SLIP traps..... | 474 |
| Collecting service SQL documentation..... | 475 |
| Collecting data for specific types of Db2 problems..... | 476 |
| Collecting data for general performance problems..... | 477 |
| Collecting data for access path performance problems..... | 478 |
| Collecting data for data access problems..... | 479 |
| Collecting data for incorrect output from an SQL statement..... | 479 |
| Collecting data when an abend occurs for an SQL query..... | 480 |
| Collecting data for stored procedure problems..... | 480 |

| | |
|--|------------|
| Collecting data for authorization problems..... | 481 |
| Collecting data for CCSID problems..... | 481 |
| Collecting data for corruption and inconsistency problems..... | 483 |
| Collecting data for IBM Db2 Analytics Accelerator for z/OS problems..... | 483 |
| Collecting data for application programming problems..... | 484 |
| Collecting data for operational problems..... | 486 |
| Collecting data for distributed data facility problems..... | 489 |
| Collecting data for Db2 utility problems..... | 491 |
| Collecting data for IRLM problems..... | 494 |
| Collecting data for a Db2 hang..... | 497 |
| Collecting data for SQL data definition language statement errors..... | 498 |
| Collecting data for RDS problems..... | 498 |
| Collecting data for a cached dynamic statement that blocks another statement with reason code 00E70081..... | 499 |
| Chapter 9. Preparing to transfer data sets with program objects to IBM..... | 503 |
| Chapter 10. Requesting product support for problems in Db2 for z/OS..... | 505 |
| Sending diagnostic data to IBM..... | 505 |
| Receiving support information from IBM..... | 506 |
| Information resources for Db2 for z/OS and related products..... | 509 |
| Notices..... | 511 |
| Programming interface information..... | 512 |
| Trademarks..... | 513 |
| Terms and conditions for product documentation..... | 513 |
| Privacy policy considerations..... | 513 |
| Glossary..... | 515 |
| Index..... | 517 |

About this information

This information provides a variety of information that can help you when you have problems associated with the Db2 13 for z/OS product. IBM® Support personnel might ask you to refer to this information when they help you with a specific problem.

Throughout this content, "Db2" means "Db2 13 for z/OS." References to other Db2 products use complete names or more-specific abbreviations.

Important: To find the most up to date content for Db2 13 for z/OS, always use [IBM Documentation](#) or download the latest PDF file from [PDF format manuals for Db2 13 for z/OS \(Db2 for z/OS in IBM Documentation\)](#).

This Db2 13 for z/OS product documentation generally assumes that the highest available function level is activated and that your applications are running with the highest available application compatibility level, with the following exceptions:

- The following documentation sections describe the Db2 13 migration process and how to activate new capabilities in function levels:
 - [Migrating to Db2 13 \(Db2 Installation and Migration\)](#)
 - [What's new in Db2 13 \(Db2 for z/OS What's New?\)](#)
 - [Adopting new capabilities in Db2 13 continuous delivery \(Db2 for z/OS What's New?\)](#)
- [FL 500 A](#) label like this one usually marks documentation changed for function level 500 or higher, with a link to the description of the function level that introduces the change in Db2 13. For more information, see [How Db2 function levels are documented \(Db2 for z/OS What's New?\)](#).

Continuous delivery in Db2 13

The availability of most new-function application capabilities in Db2 13 depends on the type of enhancement, the activated function level, and the application compatibility levels of each application. For a list of all available function levels that are currently available in Db2 13, see [Db2 13 function levels \(Db2 for z/OS What's New?\)](#).

Many new-function enhancements take effect at any function level when you apply a PTF in each Db2 subsystem or data sharing member. For more information, see [New-function APARs for Db2 13 \(Db2 for z/OS What's New?\)](#).

Availability of virtual storage and optimization enhancements

Some virtual storage and optimization enhancements take effect in function level 100. Optimization enhancements become available after full prepare of the SQL statements, depending on the statement type:

- For static SQL statements, after bind or rebind of the package.
- For non-stabilized dynamic SQL statements, immediately, unless the statement is in the dynamic statement cache.
- For stabilized dynamic SQL statements, after invalidation, free, or changed application compatibility level.

Who should read this information

This information is primarily intended for people who are responsible for diagnosing problems in a Db2 for z/OS environment.

Db2 Utilities Suite for z/OS

Important: Db2 Utilities Suite for z/OS is available as an optional product. You must separately order and purchase a license to such utilities, and discussion of those utility functions in this publication is not intended to otherwise imply that you have a license to them.

Db2 13 utilities can use the DFSORT program regardless of whether you purchased a license for DFSORT on your system. For more information about DFSORT, see <https://www.ibm.com/support/pages/dfsor>.

Db2 utilities can use IBM Db2 Sort for z/OS as an alternative to DFSORT for utility SORT and MERGE functions. Use of Db2 Sort for z/OS requires the purchase of a Db2 Sort for z/OS license. For more information about Db2 Sort for z/OS, see [Db2 Sort for z/OS documentation](#).

Related concepts

[Db2 utilities packaging \(Db2 Utilities\)](#)

Terminology and citations

When referring to a Db2 product other than Db2 for z/OS, this information uses the product's full name to avoid ambiguity.

The following terms are used as indicated:

Db2

Represents either the Db2 licensed program or a particular Db2 subsystem.

IBM OMEGAMON® for Db2 Performance Expert on z/OS

Refers to any of the following products:

- IBM IBM OMEGAMON for Db2 Performance Expert on z/OS
- IBM Db2 Performance Monitor on z/OS
- IBM Db2 Performance Expert for Multiplatforms and Workgroups
- IBM Db2 Buffer Pool Analyzer for z/OS

C, C++, and C language

Represent the C or C++ programming language.

CICS®

Represents CICS Transaction Server for z/OS.

IMS

Represents the IMS Database Manager or IMS Transaction Manager.

MVS

Represents the MVS element of the z/OS operating system, which is equivalent to the Base Control Program (BCP) component of the z/OS operating system.

RACF®

Represents the functions that are provided by the RACF component of the z/OS Security Server.

Accessibility features for Db2 for z/OS

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use information technology products successfully.

Accessibility features

The following list includes the major accessibility features in z/OS products, including Db2 for z/OS. These features support:

- Keyboard-only operation.
- Interfaces that are commonly used by screen readers and screen magnifiers.
- Customization of display attributes such as color, contrast, and font size

Tip: [IBM Documentation](#) (which includes information for Db2 for z/OS) and its related publications are accessibility-enabled for the IBM Home Page Reader. You can operate all features using the keyboard instead of the mouse.

Keyboard navigation

For information about navigating the Db2 for z/OS ISPF panels using TSO/E or ISPF, refer to the *z/OS TSO/E Primer*, the *z/OS TSO/E User's Guide*, and the *z/OS ISPF User's Guide*. These guides describe how to navigate each interface, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

Related accessibility information

IBM and accessibility

For more information about the commitment that IBM has to accessibility, see the *IBM Accessibility Center* at <http://www.ibm.com/able>.

How to send your comments about Db2 for z/OS documentation

Your feedback helps IBM to provide quality documentation.

Send any comments about Db2 for z/OS and related product documentation by email to db2zinfo@us.ibm.com.

To help us respond to your comment, include the following information in your email:

- The product name and version
- The address (URL) of the page, for comments about online documentation
- The book name and publication date, for comments about PDF manuals
- The topic or section title
- The specific text that you are commenting about and your comment

Related concepts

[About Db2 13 for z/OS product documentation \(Db2 for z/OS in IBM Documentation\)](#)

Related reference

[PDF format manuals for Db2 13 for z/OS \(Db2 for z/OS in IBM Documentation\)](#)

How to read syntax diagrams

Certain conventions apply to the syntax diagrams that are used in IBM documentation.

Apply the following rules when reading the syntax diagrams that are used in Db2 for z/OS documentation:

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

The  symbol indicates the beginning of a statement.

The  symbol indicates that the statement syntax is continued on the next line.

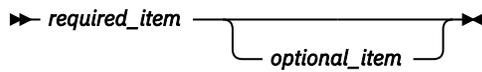
The  symbol indicates that a statement is continued from the previous line.

The  symbol indicates the end of a statement.

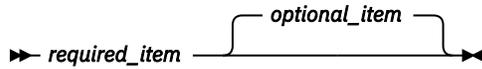
- Required items appear on the horizontal line (the main path).

 *required_item* 

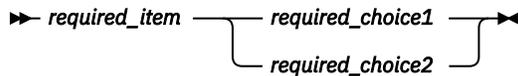
- Optional items appear below the main path.



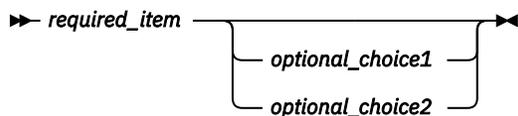
If an optional item appears above the main path, that item has no effect on the execution of the statement and is used only for readability.



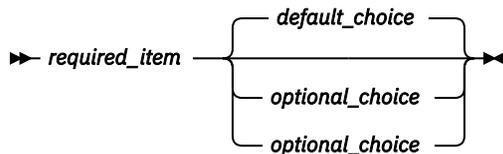
- If you can choose from two or more items, they appear vertically, in a stack. If you *must* choose one of the items, one item of the stack appears on the main path.



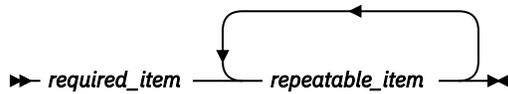
If choosing one of the items is optional, the entire stack appears below the main path.



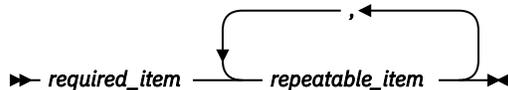
If one of the items is the default, it appears above the main path and the remaining choices are shown below.



- An arrow returning to the left, above the main line, indicates an item that can be repeated.

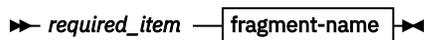


If the repeat arrow contains a comma, you must separate repeated items with a comma.

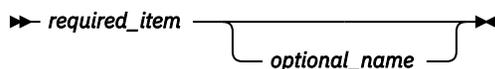


A repeat arrow above a stack indicates that you can repeat the items in the stack.

- Sometimes a diagram must be split into fragments. The syntax fragment is shown separately from the main syntax diagram, but the contents of the fragment should be read as if they are on the main path of the diagram.



fragment-name



- For some references in syntax diagrams, you must follow any rules described in the description for that diagram, and also rules that are described in other syntax diagrams. For example:

- For *expression*, you must also follow the rules described in [Expressions \(Db2 SQL\)](#).
- For references to *fullselect*, you must also follow the rules described in [fullselect \(Db2 SQL\)](#).
- For references to *search-condition*, you must also follow the rules described in [Search conditions \(Db2 SQL\)](#).
- With the exception of XPath keywords, keywords appear in uppercase (for example, FROM). Keywords must be spelled exactly as shown.
- XPath keywords are defined as lowercase names, and must be spelled exactly as shown.
- Variables appear in all lowercase letters (for example, *column-name*). They represent user-supplied names or values.
- If punctuation marks, parentheses, arithmetic operators, or other such symbols are shown, you must enter them as part of the syntax.

Related concepts

[About commands in Db2 for z/OS \(Db2 Commands\)](#)

[Db2 online utilities \(Db2 Utilities\)](#)

[Db2 stand-alone utilities \(Db2 Utilities\)](#)

Chapter 1. Introduction to troubleshooting for Db2

Troubleshooting is a systematic approach to solving a problem. The goal of troubleshooting is to determine why something does not work as expected and how to resolve the problem.

The first step in the troubleshooting process is to describe the problem completely. Problem descriptions help you and the IBM technical-support representative know where to start to find the cause of the problem. This step includes asking yourself basic questions:

- What are the symptoms of the problem?
- Where does the problem occur?
- When does the problem occur?
- Under which conditions does the problem occur?
- Can the problem be reproduced?

The answers to these questions typically lead to a good description of the problem, which can then lead you a problem resolution.

What are the symptoms of the problem?

When starting to describe a problem, the most obvious question is "What is the problem?" This question might seem straightforward; however, you can break it down into several more-focused questions that create a more descriptive picture of the problem. These questions can include:

- Who, or what, is reporting the problem?
- What are the error codes and messages?
- How does the system fail? For example, is it a loop, hang, crash, performance degradation, or incorrect result?

Where does the problem occur?

Determining where the problem originates is not always easy, but it is one of the most important steps in resolving a problem. Many layers of technology can exist between the reporting and failing components. Networks, disks, and drivers are only a few of the components to consider when you are investigating problems.

The following questions help you to focus on where the problem occurs to isolate the problem layer:

- Is the problem specific to one platform or operating system, or is it common across multiple platforms or operating systems?
- Is the current environment and configuration supported?

If one layer reports the problem, the problem does not necessarily originate in that layer. Part of identifying where a problem originates is understanding the environment in which it exists. Take some time to completely describe the problem environment, including the operating system and version, all corresponding software and versions, and hardware information. Confirm that you are running within an environment that is a supported configuration; many problems can be traced back to incompatible levels of software that are not intended to run together or have not been fully tested together.

When does the problem occur?

Develop a detailed timeline of events leading up to a failure, especially for those cases that are one-time occurrences. You can most easily develop a timeline by working backward: Start at the time an error was reported (as precisely as possible, even down to the millisecond), and work backward through the available logs and information. Sometimes, you need to look only as far as the first suspicious event that you find in a diagnostic log.

To develop a detailed timeline of events, answer these questions:

- Does the problem happen only at a certain time of day or night?
- How often does the problem happen?
- What sequence of events leads up to the time that the problem is reported?
- Does the problem happen after an environment change, such as upgrading or installing software or hardware?

Responding to these types of questions can give you a frame of reference in which to investigate the problem.

Under which conditions does the problem occur?

Knowing which systems and applications are running at the time that a problem occurs is an important part of troubleshooting. These questions about your environment can help you to identify the root cause of the problem:

- Does the problem always occur when the same task is being performed?
- Does a certain sequence of events need to occur for the problem to surface?
- Do any other applications fail at the same time?

Answering these types of questions can help you explain the environment in which the problem occurs and correlate any dependencies. Remember that just because multiple problems might have occurred around the same time, the problems are not necessarily related.

Can the problem be reproduced?

From a troubleshooting standpoint, the ideal problem is one that can be reproduced. Typically, when a problem can be reproduced you have a larger set of tools or procedures at your disposal to help you investigate. Consequently, problems that you can reproduce are often easier to debug and solve. However, problems that you can reproduce can have a disadvantage: If the problem is of significant business impact, you do not want it to recur. If possible, re-create the problem in a test or development environment, which typically offers you more flexibility and control during your investigation.

- Can the problem be re-created on a test system?
- Are multiple users or applications encountering the same type of problem?
- Can the problem be re-created by running a single command, a set of commands, or a particular application?

Chapter 2. Searching for known problems and solutions for Db2 for z/OS

Searching the IBM Support site is most effective if you include all the appropriate keywords in your search.

Procedure

To search the IBM Support site:

1. Search the [IBM Support website](#) by using the keywords that you developed. Otherwise, contact IBM Support.

Do not use both the CSECT keyword and the load module modifier keyword at the same time for the first search.

2. Compare each matching PTF or APAR closing description with the current failure symptoms.
 - If you find an appropriate PTF or APAR, apply the correction.
 - If you do not find an appropriate PTF or APAR, vary the search argument.
 - If you still cannot find a similar problem, prepare for an APAR.

Related concepts

[Load module modifier keyword](#)

Use this keyword if the search was unsuccessful when you used the CSECT keyword or the search yielded too many possible matches when you used the CSECT keyword.

Building a keyword string

You can systematically select *keywords* to describe a failure in Db2 for z/OS (Db2) or internal resource lock manager (IRLM). Keywords are predefined words or abbreviations that identify aspects of a program failure.

A string of keywords can be used to pinpoint a similar known problem in the [IBM Support website](#). If the search is successful, a similar problem description and, usually, a fix exists. If the search is unsuccessful, use these keywords when you communicate with IBM Support for more assistance or when you are documenting a possible APAR (Authorized Program Analysis Report).

The keywords are in two distinct formats: Db2 format (or free format) and structured database format.

Getting started

Before you build a keyword string, thoroughly check the application programs, JCL, and data set definitions for user errors.

The first keyword specifies a component identification number for Db2 or internal resource lock manager. A search of the IBM Support site with this keyword alone would detect all reported problems for that component (Db2 or internal resource lock manager). Each added keyword makes the search argument more specific and reduces the number of problem descriptions.

Sometimes, a similar problem can be found by using an incomplete set of keywords. To some extent, then, each keyword after the first is optional. If too many matches occur, keywords can be added to narrow the search.

When you communicate with IBM Support, identify the program failure with these keywords:

- Component identification keyword
- Release level keyword
- Type-of-failure keywords, including:

- CSECT keyword (required for certain failure types)
- Modifier keywords (optional).
- Dependency keyword

These keywords can be specified either in the Db2 format or in the "structured database" (SDB) format. For both formats, the choice of keywords depends on the type of failure that occurred.

Building a keyword string in Db2

Procedure:

1. Locate the three-digit release identifier in the title area of the formatted dump. It follows COMP=XYR00. For example:

```
COMP=XYR00.131.SSSC-SSI CALL
```

The highlighted digits signify:

13

Version

1

Release level

2. Build a keyword similar to Rxxx, replacing xxx with the version, release, and modification level numbers. Add this information to the component identifier keyword and turn to [“Type-of-failure keywords”](#) on page 12.

```
5740XYR00 R131 (free format)
PIDS/5740XYR00 LVLS/131 (structured format)
```

Db2 format (free format)

You can build a keyword string by using Db2 format (free format).

To begin selecting keywords:

1. Follow the procedures in [“Component identifier keyword”](#) on page 10 and [“Release level keyword”](#) on page 12. Do this task for all failures.
2. Follow one of the type-of-failure keyword procedures. If uncertain about which to use, see [Figure 3](#) on page 13. This chart describes procedures to follow based on common symptoms.
3. Identify the area of the failure by using CSECT and modifier keywords when appropriate. The procedures refer you to these steps as needed.
4. Follow [Chapter 2, “Searching for known problems and solutions for Db2 for z/OS,”](#) on page 3 to learn how to search with your set of keywords. Do this task for all failures.
5. If the search is unsuccessful, contact IBM Support as described in [Requesting product support for problems in Db2 for z/OS \(Troubleshooting problems in Db2\)](#)

The following flowchart helps you determine which Db2 keywords to use and the procedures for selecting them.

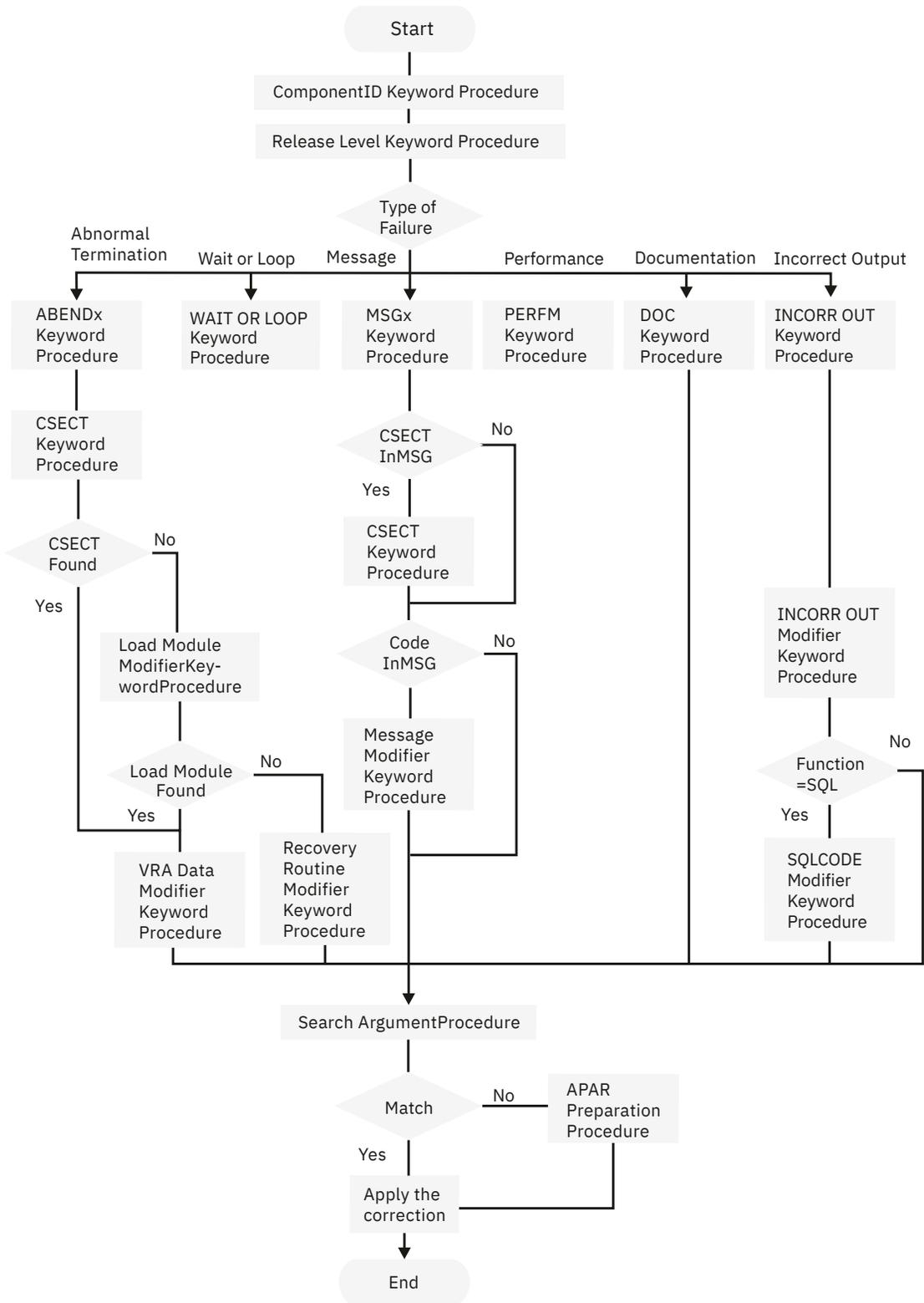


Figure 1. High-level flowchart of various sets of keywords

Structured database format

Use the structured database (SDB) format if your installation has a tool for performing structured searches.

Structured symptoms are also called RETAIN symptoms and "failure keywords". The structured symptoms consist of a prefix keyword, which identifies the type of symptom, followed by a slash (/) and the data portion of the symptom.

- The prefix keyword has one to eight characters.
- All characters must be alphanumeric, or one of the following characters: "#" "@" "\$".
- At least one character of data is required.
- The maximum length, including the prefix, is 15 characters.

For example, the following is a structured search symptom for a message identifier of IEC0201:

```
MS/IEC0201
```

The examples in the following sections show both the free format and the structured alternatives. The most commonly used structured search symptoms follow.

Prefix keyword **Meaning**

AB

Abend code

FLDS

Field or control block name that is involved with the problem

LVLS

Level of the base system or licensed program

MS

Message ID

OPCS

Operation code (opcode) for software, such as an assembler opcode

PCSS

Program command or other software statement, such as JCL, a parameter, or a data set name

PIDS

Program ID for a component that is involved in the problem

PRCS

Program return code, generated by software, including reason codes and condition codes

PTFS

Program temporary fix (PTF) for software that is associated with a problem

PUBS

Publication ID associated with a problem

RECS

Record that is associated with a problem

REGS

Register for a software program that is associated with a problem. The value can be the register/PSW difference (rrddd), which the STATUS FAILEDATA subcommand of IPCS provides for abends. The difference (ddd) is a hexadecimal offset from a probable base register or branch register (rr)

RIDS

Routine ID, such as the name of a CSECT or subroutine. If the RIDS/value has no suffix, the value is a CSECT name. The following suffixes are supported:

#L: for a load module

#R: for a recovery routine

VALU

Value in a field or register. One of the following qualifiers is required as the first character of the value:

- B: for a bit value'
- C: for a character value'
- H: for a hexadecimal value'

WS

Wait state code that is issued by the system or device-issued wait code. One of the following qualifiers is required as the first character of the value:

- D: for disabled wait (system that is disabled for I/O or external interrupts)
- E: for enabled wait

A complete list of prefix keywords follows. It provides a symptom-to-keyword cross-reference to aid in selecting SDB keywords.

| Symptom | Prefix Keyword |
|-----------------------|-----------------------|
| abend | AB/ |
| access methods | RIDS/ |
| address | ADRS/ |
| APARs | PTFS/ |
| assembler macros | RIDS/ |
| assembler messages | MS/ |
| clist | RIDS/ |
| commands | PCSS/ |
| compiler messages | MS/ |
| completion codes | PRCS/ |
| component | PIDS/ |
| condition codes | PRCS/ |
| control block | FLDS/ |
| control block offset | ADRS/ |
| control register | REGS/ |
| CSECTs | RIDS/ |
| data set name | PCSS/ |
| dependent component | PIDS/ |
| device error code | PRCS/ |
| diagnose command | PCSS/ |
| disabled wait (coded) | WS/ |
| displacements | ADRS/ |
| displays | DEVS/ |
| documents | PUBS/ |
| DSECTs | FLDS/ |
| enabled wait (coded) | WS/ |

| Symptom | Prefix Keyword |
|----------------------|-------------------------------------|
| error code | PRCS/ |
| EXECs | RIDS/ |
| feedback code | PRCS/ |
| field | FLDS/ |
| field value | VALU/ |
| file mode | PCSS/ |
| file name | PCSS/ |
| file type | PCSS/ |
| flag | FLDS/ |
| floating-point regs | REGS/ |
| fullscreen mode | PCSS/ |
| function keys | PCSS/ |
| general-purpose regs | REGS/ |
| hangs | WS/ |
| hung user/task | WS/ |
| I/O op codes | OPCS/ |
| incorrect output | INCORROUT ^{“1” on page 10} |
| JCL cards | PCSS/ |
| JCL parameters | PCSS/ |
| job step codes | PRCS/ |
| keys | PCSS/ |
| labels, code | FLDS/ |
| language statements | PCSS/ |
| level | LVLS/ |
| library names | PCSS/ |
| line command | PCSS/ |
| loops | LOOP ^{“1” on page 10} |
| low core address | ADRS/ |
| machine checks | SIG/ |
| macro as a routine | RIDS/ |
| macro as a statement | PCSS/ |
| maintenance levels | PTFS/ |
| message | MS/ |
| module | RIDS/ |
| offsets | ADRS/ |

| Symptom | Prefix Keyword |
|-----------------------|---------------------------------|
| opcode | OPCS/ |
| operator commands | PCSS/ |
| operator keys | PCSS/ |
| operator messages | MS/ |
| options | PCSS/ |
| overlay | OVS/ |
| PA key | PCSS/ |
| panel | RIDS/ |
| parameters | PCSS/ |
| performance | PERFM ^{“1”} on page 10 |
| PF key | PCSS/ |
| procedure names | PCSS/ |
| process names | PCSS/ |
| profile options | PCSS/ |
| program checks | AB/ |
| program id | RIDS/ |
| program keys | PCSS/ |
| program statement | PCSS/ |
| PSW | FLDS/ |
| PTFs, PE or otherwise | PTFS/ |
| publications | PUBS/ |
| PUT level | PTFS/ |
| reason codes | PRCS/ |
| register value | VALU/ |
| registers | REGS/ |
| release level | LVLS/ |
| replies to messages | PCSS/ |
| replies to prompts | PCSS/ |
| request codes | OPCS/ |
| responses to msgs | PCSS/ |
| responses to prompts | PCSS/ |
| return codes | PRCS/ |
| routines | RIDS/ |
| service level | PTFS/ |
| special characters | PCSS/ |

| Symptom | Prefix Keyword |
|-------------------|--------------------------------|
| SRLs | PUBS/ |
| statements | PCSS/ |
| status codes | PRCS/ |
| step codes | PRCS/ |
| structure word | FLDS/ |
| subroutines | RIDS/ |
| SVC | OPCS/ |
| SYSGEN parameters | PCSS/ |
| system check | PRCS/ |
| tables | FLDS/ |
| terminal keys | PCSS/ |
| value | VALU/ |
| variable | FLDS/ |
| waits (coded) | WS/ |
| waits (uncoded) | WAIT ^{“1”} on page 10 |

Note:

1. No prefix keyword. Use the failure keyword that is shown for searches.

Component identifier keyword

The component identifier is the first keyword in the search argument.

You use the component identifier keyword to identify the Db2 and related subcomponents when you search for APARs (Authorized Program Analysis Reports) and PTFs (Program Temporary Fixes) on the [IBM Support website](#).

The component identifier keyword is derived from a four-digit number (5740) and a five-digit identifier. This information describes how to determine the nine-digit component identifier keyword for the failure.

The following component identifiers are for Db2 and its subcomponents:

5740XYR00

Db2

569516401

IRLM

5740IX100

CICS attachment

5740IY100

IMS attachment

5740XYR01

Subsystem initialization subcomponent

Try to locate the component identifier by following the steps under [“Locating the component identifier by using the SVC dump”](#) on page 11.

If a dump title is not available, follow [“Locating the component identifier by using the SYS1.LOGREC entry”](#) on page 11 or [“Locating the component identifier by using the first page of an SVC dump”](#) on page 12.

Locating the component identifier by using the SVC dump

One way to locate the component identifier is by using the SVC dump title.

Procedure

To locate the component identifier by using the SVC dump title:

1. Locate the C= label in the dump title, and the first five characters that follow that label.
2. Append those five characters to 5740 and use this string as the first keyword. Then, turn to [“Release level keyword”](#) on page 12.

```
5740XYR00 (free format)
PIDS/5740XYR00 (structured format)
```

Example

1. No Distributed Data Processing:
For threads with **no remote activity**:

```
ssid,ABND=compltn-reason,U=authid,
C=compid.release.comp-function,M=module,
LOC=loadmod.csect+csect_offset
```

2. Distributed Data Processing:
For allied thread with **remote activity**
(requesting location threads):

```
ssid,ABND=compltn-reason,U=authid,
C=compid.release.comp-function,
DISTRBUTED, LOC=loadmod.csect+csect_offset
```

- For database access threads
(responding location threads):

```
ssid,ABND=compltn-reason,U=authid,
C=compid.release,
LOCN=16_char_loc_name,
LOC=loadmod.csect+csect_offset
```

Figure 2. Sample SVC dump titles

Locating the component identifier by using the SYS1.LOGREC entry

One way to locate the component identifier is by using the SYS1.LOGREC entry.

Procedure

To locate the component identifier by using the SYS1.LOGREC entry:

1. Locate the SYS1.LOGREC entry that is related to the error by comparing the date, time, program name, and ERRORID string in the dump with the information in the SYS1.LOGREC entry.
2. Locate the COMP ID INVOLVED column at the top of the first page and the five characters beside it.
3. Append those five characters to 5740 and turn to [“Release level keyword”](#) on page 12.

```
5740XYR00 (free format)
PIDS/5740XYR00 (structured format)
```

Locating the component identifier by using the first page of an SVC dump

One way to locate the component identifier is by using the first page of an SVC dump.

Procedure

To locate the component identifier by using the first page of an SVC dump:

1. Locate the Symptom String heading toward the middle of the first page and the nine characters that follow the PIDS/ label.

For an example, see [Figure 33 on page 197](#).

2. Use those characters as the first keyword and turn to [“Release level keyword” on page 12](#).

```
5740XYR00 (free format)
PIDS/5740XYR00 (structured format)
```

Release level keyword

The release level keyword narrows the symptom search to a specific release level.

Using this keyword is optional, but recommended, when you search the [IBM Support website](#). It is required, however, when you submit an APAR.

Type-of-failure keywords

A type-of-failure keyword describes an external symptom of a program failure.

The following table displays the various types of failures. Use this table to find the name and page number of the procedure that best matches the problem.

Table 1. Types of Db2 failures

| Problem | Procedure |
|---|---|
| Abend of the subsystem or task. | “ABENDx keyword” on page 14 |
| Unexpected program suspension. | “WAIT/LOOP keywords” on page 19 |
| Uncontrolled program that is looping (often signaled by repeating messages or trace entries). | “WAIT/LOOP keywords” on page 19 |
| Errors signaled by or associated with messages. | “MSGx keyword” on page 33 |
| Performance degradation. Use this option only when other keywords are inappropriate. | “PERFM keyword” on page 35 |
| Unexpected or missing output (not related to a message). | “INCORROUT modifier keyword” on page 42 |

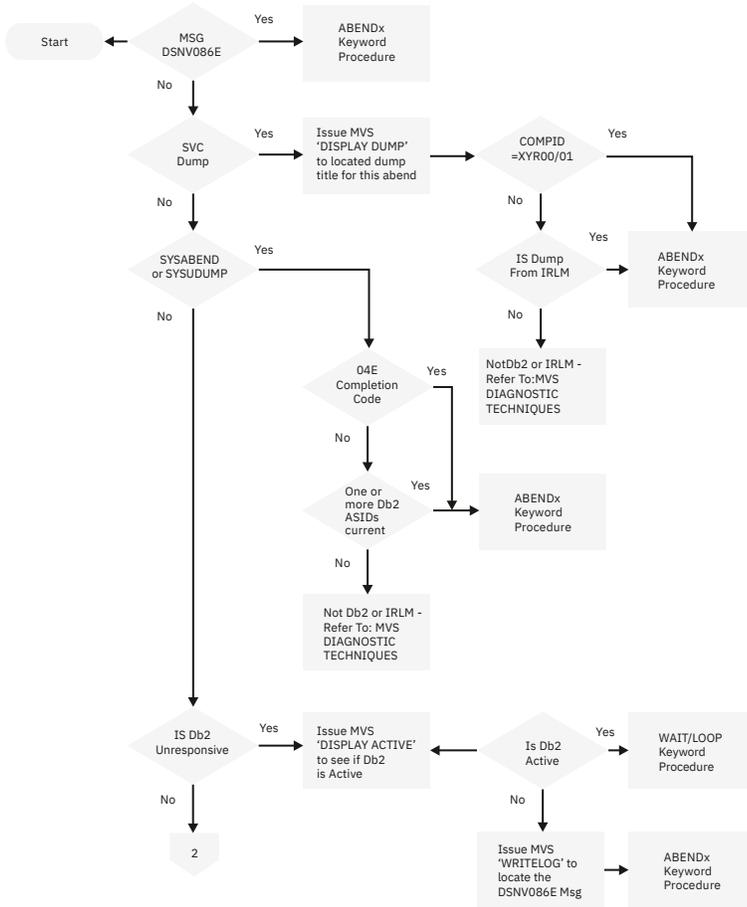


Figure 3. Determining the type of failure

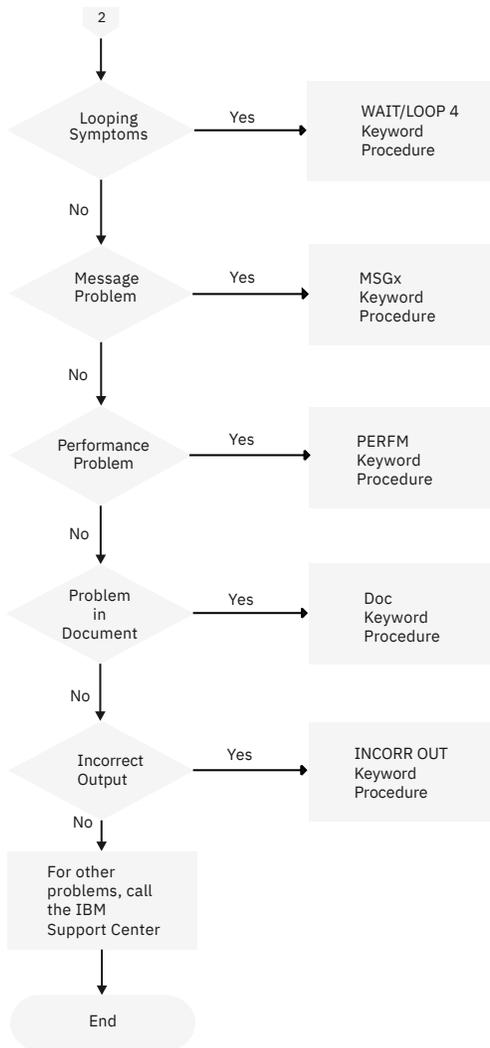


Figure 4. Determining the type of failure continued

ABENDx keyword

Use the ABENDx keyword procedure when the subsystem or task terminates abnormally.

You can locate the abend completion code and the abend reason code, if there is one, and use the code in a set of keywords.

Related reference

[Db2 abend completion codes \(X'04E' and X'04F\) \(Db2 Codes\)](#)

SVC dump message

When a Db2 or IRLM abend occurred, you might receive an IEA911E message from z/OS, indicating an SVC dump occurred.

Procedure

To find the name of the failing CSECT by using the first page of an SVC dump:

1. Locate the SVC dump title for this abend by using one of these methods:
 - Issue the z/OS **DISPLAY DUMP** command.
 - The dump title is always at the top of each page of the formatted copy of an SVC dump.
2. If COMP=XYR00 or COMP=XYR01

a) Locate the three-character completion code that follows the word ABND.

- If the completion code is X'071' or X'122', the operator pressed the RESTART key or canceled the job, probably to break a loop. Turn to [“WAIT/LOOP keywords” on page 19](#).
- If the completion code is anything except X'04E', build a keyword similar to ABENDxxx, replacing xxx with the completion code. Add this information to the keyword string.

```
:5740XYR00 R131 ABEND0C4: (free format)
:PIDS/5740XYR00 LVLS/131 AB/S00C4 (structured
format)
```

Some abends that are not X'04E' also have reason codes. These reason codes are found in register 15 at the time of the abend. Locate the reason code for the abend in either the 4-byte reason code field in a dump title that is generated by Db2, in the registers at time of error in the abstract information section of the dump, or from the value of register 15 in the error summary display that is obtained by issuing the z/OS command DISPLAY DUMP,ERRDATA,DSN=nn. Check the value against the description of the abend code in the z/OS system codes publication. If the value is a valid reason code for the abend completion code, add it to the keyword string.

```
ABEND058 RC0000000C (free format)
AB/S0058 PRCS/0000000C (structured format)
```

After you add the completion code and the reason code, when the abend has a reason code, to the keyword string, turn to [“CSECT keyword” on page 35](#).

- If the completion code is X'04E', find the 4-byte reason code that follows it. For an example, see [Figure 2 on page 11](#).
- b) Review the diagnostic information for this reason code. Follow any recommended procedures.
- c) Build a keyword similar to ABEND04E RCxxxxxxx, replacing xxxxxxxx with the reason code. Add this information to the keyword string and turn to [“CSECT keyword” on page 35](#).

```
5740XYR00 R131 ABEND04E RC00E20015.
(free format)
PIDS/5740XYR00 LVLS/131 AB/S00E4
PRCS/00E20015 (structured format)
```

3. If COMP is not XYR00 or XYR01.

Review the dump heading. If it reads DB2 SUBSYSTEM TERMINATION REQUESTED – REASON=xxxxxxx, follow the steps below. Otherwise, go to [Step “4” on page 15](#).

- a) Locate the 4-byte reason code in the message or dump heading following the word REASON.
- b) Build a keyword similar to ABEND04F RCxxxxxxx, replacing xxxxxxxx with the reason code. Add this information to the keyword string, and turn to [Chapter 2, “Searching for known problems and solutions for Db2 for z/OS,” on page 3](#).

```
5740XYR00 R131 ABEND04F RC00D93001:
(free format)
PIDS/5740XYR00 LVLS/131 AB/S004F
PRCS/00D93001 (structured format)
```

4. Scan the dump heading for IRLM (an IRLM-produced SVC dump title contains the character-string 'IRLM' only if the subsystem name of the failing IRLM is 'IRLM'). If found, follow the steps below. Otherwise, refer to the z/OS diagnostic techniques publication to determine the failing product.

- a) Print the SYS1.LOGREC entries. Refer to the z/OS diagnostic techniques publication, if necessary.
- b) Locate the appropriate SYS1.LOGREC entry. It should have the same time of day and program name as those appearing in the dump. For an example of a SYS1.LOGREC entry, see [Figure 66 on page 227](#).
- c) Locate the three-character completion code in the SYS1.LOGREC entry. To do this task, locate the SDWACMPC field of the SDWA (system diagnostic work area). This field contains the completion code. [Figure 66 on page 227](#) illustrates its location.

- If the completion code is X'071' or X'122', the operator pressed the RESTART key or canceled the job, probably to break a loop. Turn to [“WAIT/LOOP keywords” on page 19](#).
- Otherwise, build a keyword similar to ABENDxxx, replacing xxx with the completion code. Add this information to the keyword string, which should identify IRLM as the failing component. Turn to [“CSECT keyword” on page 35](#).

```
569516401 AR220 ABEND52A (free format)
PIDS/569516401 LVLS/220 AB/S052A
(structured format)
```

SYSABEND or SYSUDUMP

When a Db2 or IRLM abend occurred, a SYSABEND or SYSUDUMP dump might occur.

Procedure

To locate the abend completion code and the abend reason code, if there is one, and use the code in a set of keywords:

1. Determine whether there is also an SVC dump for this abend.
 - a) Issue the z/OS **DISPLAY DUMP** command. Refer to the z/OS system commands publication, if necessary.
 - b) Verify that the SVC dump has the same date, time stamp, and program name as those in the SYSABEND or SYSUDUMP. For information about printing the summary dump, see [“Printing dumps” on page 182](#).
2. If there is a corresponding SVC dump, continue with the steps below. Otherwise, go to Step [“3” on page 16](#).
 - a) Review the dump title and verify that COMP=XYR00 or COMP=XYR01. If it does not, the problem might not be Db2. Consult the z/OS diagnostic techniques publication to determine the failing product.
 - b) Locate the three-character completion code that follows the word ABND.
 - If the completion code is X'071' or X'122', the operator pressed the RESTART key or canceled the job, probably to break a loop. Turn to [“WAIT/LOOP keywords” on page 19](#).
 - If the completion code is anything except X'04E', build a keyword similar to ABENDxxx, replacing xxx with the completion code. Add this information to the keyword string and turn to [“CSECT keyword” on page 35](#).

```
5740XYR00 R131 ABEND0C4
(free format)
PIDS/5740XYR00 LVLS/131 AB/S00C4
(structured format)
```

- If the completion code is X'04E', find the 4-byte reason code follows it. For an example, see [“SVC dump titles that are issued by Db2” on page 199](#).
- c) Review the diagnostic information for this reason code, and follow any recommended procedures.
 - d) Build a keyword similar to ABEND04E RCxxxxxxxx, replacing xxxxxxxx with the reason code. Add this information to the keyword string and turn to [“CSECT keyword” on page 35](#).

```
5740XYR00 R131 ABEND04E RC00E20015
(free format)
PIDS/5740XYR00 LVLS/131 AB/S004E
PRCS/00E20015 (structured format)
```

3. If there is no corresponding SVC dump, follow the steps below.
 - a) Locate the three-character completion code. This information follows the words COMPLETION CODE on the first page of the SYSABEND or SYSUDUMP. For an example, see [Figure 64 on page 225](#).
 - If the completion code is X'071' or X'122', the operator pressed the RESTART key or canceled the job, probably to break a loop. Turn to [“WAIT/LOOP keywords” on page 19](#).

- If the completion code is X'04E', locate the registers at time of error (identified by REGS AT TIME OF ERROR). See [“The RTM2WA summary in a SYSUDUMP” on page 225](#) for an example.

Locate the reason code in register 15, the last register shown. Build a keyword similar to ABEND04E RCxxxxxxx, replacing xxxxxxxx with the reason code. Add this keyword to the string and turn to [“CSECT keyword” on page 35](#) and use [“Procedure using a SYS1.LOGREC entry” on page 37](#).

```
5740XYR00 R131 ABEND04E RC00E50063
  (free format)
PIDS/5740XYR00 LVLS/131 AB/S004E
  PRCS/00E50063 (structured format)
```

- If the completion code is not X'071', X'122', or XX'04E', determine whether the problem occurred in Db2. (SYSUDUMP and SYSABEND dumps are rarely taken for Db2 internal errors; invalid user data is usually the cause.)
 - Check the failing PSW address in the list of CDE entries for the task. If the failing address falls within the range of a module name beginning with "DSN", the failure occurred in a Db2 attachment module.
 - If the failing PSW address is outside the normal addressing range (such as zero), check the address in register 14 at time of failure. If this address falls within the range of a module name beginning with "DSN", the failure occurred in a Db2 attachment module or stored procedure module.
 - Compare the following items:
 - The PASID value in XSB that is associated with the active RB at the time of failure
 - The first four digits of the IDSQ field in the ASCB.
 If they are unequal and the application was running SQL statements, the failure could have occurred in a Db2 address space.
 - Check register 15 at time of failure. If it contains a reason code of X'00F300'nn, then one of the following situations occurred:
 - The abend took place in Db2 while processing a parameter supplied by the attachment subcomponent, and the passed parameter is in error.
 - The SQLCA address is invalid, and the problem is in the application program.
 - An invalid reason code or authorization ID was returned by an authorization exit routine, and the problem is in the application program.
- b) If the problem is not in Db2, check the application program. Often, there is invalid user data and, upon retry, an SQL error code is returned.
- c) If the problem appears to be in Db2, build a keyword similar to ABENDxxx, replacing xxx with the completion code. Add this information to the keyword string and turn to [“CSECT keyword” on page 35](#) and use [“Procedure using a SYS1.LOGREC entry” on page 37](#).

```
5740XYR00 R131 ABEND0C4
  (free format)
PIDS/5740XYR00 LVLS/131 AB/S00C4
  (structured format)
```

Db2 unresponsive

When a Db2 or IRLM abend occurred, Db2 might be unresponsive, but no messages or dumps were detected.

Procedure

To locate the abend completion code and the abend reason code, if there is one, and use the code in a set of keywords:

1. Issue the z/OS **DISPLAY ACTIVE** command to see whether Db2 is active. Refer to [z/OS MVS System Commands](#), if necessary.
2. If Db2 is active, turn to [“WAIT/LOOP keywords” on page 19](#).
3. Build a keyword similar to RCxxxxxxx, replacing xxxxxxxx with the reason code. Add this keyword to the string, and check for more diagnostic information. Next, turn to [Chapter 2, “Searching for known problems and solutions for Db2 for z/OS,” on page 3](#).

```
5740XYR00 R131 ABEND04F RC00D93001
  (free format)
PIDS/5740XYR00 LVLS/131 AB/S004F
  PRCS/00D93001 (structured format)
```

Db2 abnormal termination message

When a Db2 or IRLM abend occurs, you might receive message DSNV086E, which indicates that DB2 abended, and includes a reason code for the abend.

Procedure

To locate the abend completion code and the abend reason code, if there is one, and use the code in a set of keywords:

1. Issue the z/OS **DISPLAY DUMP** command to see whether any SVC dumps occurred near the time the message appeared. Refer to the [z/OS. system commands publication](#), if necessary.
2. If there were no SVC dumps for the abend, follow the substeps in this step. Otherwise, go to [Step “3” on page 18](#).
 - a) Locate the 4-byte reason code in the message.
 - b) Review the diagnostic information. Follow any recommended procedures.
 - c) Build a keyword similar to ABEND04F RCxxxxxxx, replacing xxxxxxxx with the reason code. Add this keyword to the string, and turn to [Chapter 2, “Searching for known problems and solutions for Db2 for z/OS,” on page 3](#).

```
5740XYR00 R131 ABEND04F RC00D93001
  (free format)
PIDS/5740XYR00 LVLS/131 AB/S004F
  PRCS/00D93001 (structured format)
```

3. If there was only one SVC dump for the abend, follow the substeps in this step. Otherwise, go to [Step “4” on page 19](#).
 - a) Review the dump title and verify that COMP=XYR00 or COMP=XYR01. If not, the problem might not be Db2. Use the [z/OS diagnostic techniques publication](#) to determine the failing product.
 - b) Locate the three-character completion code that follows the word ABND.
 - If the completion code is X'071' or X'122', the operator pressed the RESTART key or canceled the job, probably to break a loop. Turn to [“WAIT/LOOP keywords” on page 19](#).
 - If the completion code is anything except X'04E', build a keyword similar to ABENDxxx, replacing xxx with the completion code. Add this information to the keyword string and turn to [“CSECT keyword” on page 35](#).

```
5740XYR00 R131 ABEND0C4
  (free format)
PIDS/5740XYR00 LVLS/131 AB/S00C4
  (structured format)
```

- If the completion code is X'04E', find the 4-byte reason code that follows it. For an example, see [“SVC dump titles that are issued by Db2” on page 199](#).
- Review the diagnostic information for this reason code in [Db2 reason codes \(Db2 Codes\)](#). Follow any recommended procedures.

- If the reason code is not in the ranges that are shown above, build a keyword similar to ABEND04F RCxxxxxxx, replacing xxxxxxxx with the reason code. Add this information to the keyword string and turn to “CSECT keyword” on page 35.

```
5740XYR00 R131 ABEND04E RC00E20015
  (free format)
PIDS/5740XYR00 LVLS/131 AB/S004E
PRCS/00E20015 (structured format)
```

4. If there were two or more SVC dumps, follow the substeps in this step. Otherwise, go to Step “5” on page 19.
 - a) Read the sections in [Db2 reason codes \(Db2 Codes\)](#) that describe (1) the reason code appearing in the message, and (2) any reason codes appearing in the SVC dump titles. Reason codes appear after the completion code in the SVC dump title. For an example, see [“SVC dump titles that are issued by Db2”](#) on page 199.
 - b) Compare the reason codes in the SVC dumps to determine which dump relates to the DSNV086E message. Correlate the SYS1.LOGREC entry with console log time stamps.
 - c) Use that SVC dump and follow Step “3” on page 18.
5. If there were two or more different abends, follow the substeps below:
 - a) Determine which abend was the original cause by reviewing the time stamps in the SYS1.LOGREC entries.
 - b) Follow the substeps in Step “3” on page 18.

Related reference

[Db2 abend completion codes \(X'04E' and X'04F\) \(Db2 Codes\)](#)

WAIT/LOOP keywords

The symptoms for WAIT and LOOP keywords might not be distinguishable at first.

If you are unable to complete one or more of the steps that are listed, perhaps because a command does not function, continue with the others to gather as much information as possible.

When Db2 is "hung", waiting or looping, some z/OS diagnostic techniques can aid in analysis. Refer to [z/OS MVS Diagnosis: Tools and Service Aids](#) for z/OS diagnosis and command syntax information. In addition, refer to [Command types for Db2 for z/OS \(Db2 Commands\)](#) for the syntax of Db2 commands. Refer to [“Printing and analyzing dumps”](#) on page 181 for information about locating Db2 control blocks and formatting dumps. Finally, keep track of the current log and archive information.

Guidelines for good operational procedures

Good operational procedures make problem diagnosis easier and minimize the risk of damaging data.

Consider the following guidelines:

- If Db2 hangs during distributed processing, it is possible that the problem is with VTAM® or the network, rather than Db2. For more information about procedures on diagnosing problems in a distributed environment, see [“Diagnosing distributed data facility \(DDF\) failures”](#) on page 293.
- If a Db2 hangs in the data sharing environment, the steps that are listed in this section can be used to resolve the problem. Follow the extra steps for data sharing, where indicated.
- Keep space available on the active logs. If all active log data sets are full and archiving has not been completed, Db2 functions do not complete. Db2 cannot be stopped in this situation.
- Verify that all archive mount requests have been properly satisfied.
- Be sure that IRLM address spaces have a higher dispatching priority than any allied, Db2, or IMS address spaces. If IRLM does not get enough CPU time, a Db2 subsystem can experience bad performance or wait conditions.
- Whether the entire operating system seems to be affected or only Db2 and some of its users, request an z/OS console dump. It should include:
 - The common service area (CSA)

- The Db2 system services and database services address spaces and, for distributed processing only, the distributed data facility address space.

If the problem occurs in the data sharing environment and involves more than one member of the data sharing group, take dumps of these address spaces for each member of the data sharing group.

- z/OS master scheduler address space
- IRLM address space.

If the problem occurs in the data sharing environment and involves more than one member of the data sharing group, create a dump file of this address space for each member of the data sharing group. Always include the XESDATA keyword in the SDATA parameters when IRLM is a member of a SYSPLEX GROUP.

- The address space of at least one involved user
- GRSTRACE (global resource serialization trace)
- Cross-system extended services (XES), if the problem occurs in the data sharing environment and involves more than one member of the data sharing group.

To obtain a z/OS console dump:

1. If an allied address space is hung:

- Determine whether the relevant allied ASID getting CPU or if it is swapped out. If it is swapped out:

Take a console dump of the z/OS master (ASID=0001). Order is important. The z/OS master must be dumped first to ensure good dump data. After the z/OS master ASID=0001 is dumped, then dump:

- Allied ASID
- DB2MSTR
- DB2DBM1
- IRLM
- If the relevant ASID is not swapped out, the z/OS master is not required, so dump only:
 - Allied ASID
 - DB2MSTR
 - DB2DBM1
 - IRLM

2. If Db2 is hung, take a console dump of:

- DB2MSTR
- DB2DBM1
- IRLM
- XES, if the problem occurs in the data sharing environment and involves more than one member of the data sharing group.

SDATA information should include GRSQ and RGN added to the SDATA PARAMETER defaults on the dump command.

Request the console dump as soon as it becomes evident that Db2 or a transaction is not functioning normally.

- If you suspect that Db2 is looping, start the Db2 global trace, if it is not already active. Refer to [“Printing and analyzing global traces”](#) on page 238 for information about the Db2 global trace facility.
- If a performance problem is suspected, consider using the Db2 performance trace to obtain more information. Refer to [Performance trace \(Db2 Performance\)](#) for information about running the performance trace and choosing the events to be traced. Refer to [“Using the performance trace for diagnosis”](#) on page 251 for an example of using the performance trace to diagnose a WAIT problem.

- After the console dump completed, try to break the wait or loop by issuing the command

```
-CANCEL THREAD(token)
```

to cancel any "hung" or looping threads.

If the hung thread cannot be canceled, or if Db2 is hung, try the following commands, in order, until one of them succeeds (*ssnm* is the Db2 subsystem name):

1. -STOP DB2 MODE(QUIESCE)
2. -STOP DB2 MODE(FORCE)
3. MODIFY *irlmproc*,ABEND
4. CANCEL *ssnm*MSTR
5. CANCEL *ssnm*DBM1
6. FORCE *ssnm*MSTR

Use the FORCE command only as a last resort. FORCE can put IRLM in an indefinite state, and you might need to re-IPL z/OS before you can restart Db2.

If all attempts to clear the problem fail, then a re-IPL of z/OS is necessary. Taking a stand-alone dump at this point provides little diagnostic value, because the preceding attempts to clear the problem might delete important diagnostic information.

It is very important to take a console dump BEFORE attempting to clear the problem.

- If X'04E' or X'04F' abends that occurred before the WAIT/LOOP condition, obtain their corresponding SVC dumps. Follow the instructions in [“Printing and analyzing dumps” on page 181](#) and any recommendations of IBM Support.
- It is important to save the SYS1.LOGREC entries during the abnormal condition and for at least one hour that proceeds it. It is likewise important to save and print the console log for a similar period, preferably back to the -START DB2 command if this is not excessively long.

Initial procedure for the WAIT/LOOP keywords

While the documentation needed for a WAIT/LOOP might appear excessive, this failure typically does not produce any documentation specific to its occurrence. Determining its cause often requires considerable examination.

1. Determine the scope of the problem and the users and environments that are affected.
2. If the problem occurs in a sharing environment and involves more than one member of the data sharing group, do the following:

- Issue the command

```
-DISPLAY GROUP
```

to get information about the status of the data sharing group:

- Status of each member: ACTIVE, QUIESCED, or FAILED.
- Lock structure and SCA storage: the percentage of lock entries in use and the percentage of SCA in use indicates whether a resource availability problem might be the reason for the hang.

- Issue the IRLM status command:

```
F irlmproc,STATUS
```

on each member of the data sharing group to determine which IRLMs are responsive. If any IRLM does not respond, it might indicate that a wait on a lock is causing the hang.

In some cases, Db2 uses the IRLM notify service to send XCF messages between data sharing members. The sender of the message might be waiting indefinitely for the receivers to respond.

The message from the command `F irlmproc,STATUS` does not indicate if a system is suspended on a notify message. The buffer manager drain service, data definition language, and data set extend services use IRLM notify.

If the hung user is draining a page set, the user task might be suspended waiting for the claim count to reach zero. Issue the command:

```
-DISPLAY DATABASE(dbname) CLAIMERS
```

to determine which users across the data sharing group hold a claim on the page set. If claimers exist, the drainer will be resumed when the claim count reaches zero, or the drainer will time out.

3. If Db2 'hangs' during distributed processing, see [“Diagnosing distributed data facility \(DDF\) failures” on page 293](#) for procedures to follow.
4. Collect information about active Db2 threads. You can do this by issuing the Db2 command `-DISPLAY THREAD TYPE(ACTIVE)`, or by formatting your console dump using `DSNWDMP` with the option `DS=1`. For information about `DSNWDMP`, see [“Format dumps by using the DSNWDMP statement” on page 183](#). Review the output generated and the data in the following three fields. The field labels in `-DISPLAY THREAD` and `DSNWDMP` output are shown in parentheses.

- Status (ST or first part of Status):

For each thread, these status codes are possible:

N

The thread is in either identify or sign-on status.

ND

The thread is in either IDENTIFY or SIGNON status. The thread is not currently associated with any TCB.

QT

The create thread request is queued. The associated allied task is in a wait state.

T

An allied, non-distributed thread has been established.

TD

An allied thread was established (plan allocated). The thread is currently not associated with any TCB.

PT

The thread is a parallel task.

TR

The thread (an allied distributed thread) is requesting data from another database management system.

RA

The thread (database access agent) is performing a remote access on behalf of a request from another DBMS.

RN

A distributed thread is performing a remote access on behalf of a request from a partner location. The thread was suspended because Db2 must first connect to the partner location. The Db2 command `DISPLAY LOCATION(*)` shows conversation activity for this Db2 system conversation (SYSCON-O) service task.

RQ

A distributed thread is performing a remote access on behalf of a request from another location. The thread was suspended because the maximum number of active database access threads (as described by the `MAX REMOTE ACTIVE` value of the `DSN6SYSP` macro in the Db2 startup parameter, usually `DSNZPARM`) was reached. Database access agents (DBAAs) are queued until a slot becomes available. Consider increasing the `MAX REMOTE ACTIVE` value.

SP

A thread is executing within a stored procedure.

SW

A thread is waiting for a stored procedure to be scheduled.

TN

An allied thread was distributed to access data at another Db2 location, but was suspended because Db2 system conversations have not been established. The Db2 command DISPLAY LOCATION(*) shows conversation activity for this Db2 system conversation (SYSCON-O) service task.

QD

The thread is queued for deferred termination because the associated allied task terminated. The allied task is placed in a WAIT state if this Db2 thread is the last (or only) one for the address space.

D

The thread is being terminated because the associated allied task terminated. The allied task is placed in a WAIT state if this Db2 thread is the last (or only) one for the address space.

If a QT status is reported on the connection identifier (job name for batch), you are in a WAIT for a thread to become available. If you receive an excessive number of QT status codes, the limit specified for Db2 subsystem parameter CTHREAD can be increased. CTHREAD corresponds to field MAX USERS on installation panel DSNTIPE.

If after repeating the -DISPLAY THREAD command several times, different threads have the "D" status and the ones previously having a "D" have terminated, then there is NO wait or loop. It is simply taking a long time to terminate all threads. Since deferred termination is serialized across all threads whose allied tasks have abended, and since abend processing must be performed, the total length of elapsed time can be excessive.

However, if after considerable time, the same thread still has a status of "D", a wait or loop is the probable cause. Software or hardware monitors can be used to see if one or more of the Db2 address spaces is consuming processor or I/O resources. (The user's allied task has been in a wait state and cannot be using any processor or I/O resources.) If the processor utilization is high, then a loop should be assumed. If processor utilization is not high and there is ongoing I/O activity to the Db2 database and logging data sets, then wait a little longer. The allied thread is probably still in abend processing. If the processor and I/O utilization are low, it is safe to assume that Db2 is in an "endless" wait.

If the allied thread with the status of "D" that appears to be waiting or looping is distributed, that is, the message, "ACCESSING DATA AT location" appears on the display thread command, attempt to terminate the thread by issuing the 'VTAM VARY NET, TERM' command. If this does not solve the problem, continue with the following sections.

After confirming a wait or loop condition, take an z/OS console dump as described in the previous section, "[Guidelines for good operational procedures](#)" on page 19. After the dump is complete, canceling one of the Db2 address spaces is likely to clear the problem. Canceling the allied address space of the "hung" thread or issuing -STOP DB2 has no effect.

- Db2 activity indicator (A or second part of Status):

If the activity indicator is on (if column A contains an asterisk), the thread's allied task is executing in one of the Db2 address spaces. If the activity indicator is blank, the thread's allied task is not executing in a Db2 address space.

- Wraparound request counter (REQ or Req):

If several -DISPLAY THREAD commands are issued and the request counter is incrementing, the thread is not in a WAIT but may be in a LOOP.

If the counter is incrementing and the activity indicator is changing from blank to asterisk, the loop is NOT in Db2 code, but in the application. The combined state of the activity indicator and the REQ counter indicate that control of the task is continuously transferring between Db2 and the application.

If the activity indicator consistently contains an asterisk and the REQ counter is not incrementing, it is safe to assume there is a wait or loop in Db2 code.

5. Verify that the problem is not in an application process.
 - If there is no asterisk in the Db2 activity indicator (column A), the thread is not active and the problem is in the application.
 - If the problem is in the application, determine where the processor cycles are being used. The z/OS DISPLAY ACTIVE command or other tools can help accomplish this. This can help to determine if the application is in a wait or a loop. Applications that fail to check SQL return codes properly after each SQL call might go into a loop, making it appear that Db2 is looping.
6. Issue the Db2 command -DISPLAY DATABASE to determine if any Db2 resources are unavailable to the transactions involved in the wait or loop.
7. If Db2 resources are restricted because of utilities, issue the Db2 -DISPLAY UTILITY command to determine which utilities are operating.

Utilities holding exclusive use of Db2 objects during normal processing include REORG (except when SHRLEVEL CHANGE is specified), LOAD, REBUILD, RECOVER, CHECK, MODIFY, and REPAIR.
8. Check for long-running jobs and for SPUFI users running with AUTOCOMMIT=NO. This determines if needed resources are being held.
9. Determine if the Db2 catalog is being updated.
 - The catalog can be updated through BIND, DDL, and certain utilities, including COPY, RUNSTATS, and STOSPACE.
 - Many updates can lock out other users, causing what appears to be a wait. The problem disappears when the catalog updates are completed.
10. Check for transactions using both DL/I and SQL. These transactions can wait for extended periods if IRLM has a large deadlock timeout specified and if a deadlock occurs over DL/I and SQL.
11. If a user is holding needed resources, wait for termination of that user, or cancel the user and request a dump. Reassess the scope of the problem.
12. If only one user is affected, determine if any indexes have been dropped that would extend the application's execution time.
13. If you receive no response from Db2 commands, Db2 is in a wait state. A problem can have occurred during -STOP DB2 processing or abnormal subsystem termination.
14. Review the list below to determine the appropriate procedure to follow next, based upon the environments that are affected. The first procedure has the largest scope.
 - [“If Db2 is waiting during Db2 startup” on page 24](#)
 - [“If Db2 and/or z/OS are not operational” on page 25](#)
 - [“If users in more than one environment cannot issue SQL statements” on page 28](#)
 - [“If IMS dependent regions cannot issue SQL statements” on page 29](#)
 - [“If DSN users cannot issue SQL statements” on page 30](#)

If Db2 is waiting during Db2 startup

After the -START DB2 command is entered on the console, the system services address space is started, then the database services address space is started, and, finally, the distributed data facility address space is started.

The system services address space waits for a signal that the database services address space gained control. If the wrong program name is specified in the JCL, the program runs but the database services job step task does not gain control and the system services address space continues to wait indefinitely.

1. Cancel the system services address space from the console.
2. After Db2 is stopped, check the start procedures for the Db2 address spaces (system services, database services, and distributed data facility) to ensure that the JCL is correct.

3. Verify that the data set names on the //STEPLIB card are correct and that the region size is correct. Check the JES JOBLOG, if needed, for more information helpful to diagnosis.

If Db2 is waiting during distributed processing with another Db2

If Db2 hangs during distributed processing, the condition that causes the problem could be at the requesting Db2 location, the responding Db2 location, in VTAM, in TCP/IP, or in the network.

Capture enough information from all the participants to allow for effective diagnosis of the problem. In general, the SYSLOG, SYS1.LOGREC, and SVC dumps are needed from both the requesting and responding locations. (VTAM documentation can also be used.)

Related concepts

Diagnosing distributed data facility (DDF) failures

It is important to understand the flow of distributed requests and the procedures that can be used for doing problem determination for Db2 for z/OS in a distributed environment.

If Db2 and/or z/OS are not operational

If Db2 and/or z/OS are not operational, you can try using the diagnostic techniques for looping or waiting.

Looping

If Db2 or z/OS is not operational, you can try the diagnostic techniques for looping.

Procedure

1. If z/OS is operational, take a z/OS console dump as directed in [“Guidelines for good operational procedures”](#) on page 19, before you attempt to break the loop. If z/OS is not operational, take a stand-alone dump and IPL z/OS again.
2. Use standard z/OS diagnostic techniques to determine the location and scope of the loop. Refer to the z/OS diagnostic techniques and debugging handbook publications.
3. If Db2 is causing the loop, try to determine the CSECT or load module involved. Turn to [“CSECT keyword”](#) on page 35 and then to [“Load module modifier keyword”](#) on page 38.
4. Build a keyword similar to LOOP xxxxxx, replacing xxxxxx with the CSECT name. If the CSECT name is unavailable, or if a search with this information proves unsuccessful, replace it with the load module identifier. Add this information to the keyword string, and turn to [Chapter 2, “Searching for known problems and solutions for Db2 for z/OS,”](#) on page 3.

```
5740XYR00 R131 LOOP DSNXCR2
  (free format)
PIDS/5740XYR00 LVLS/131 RIDSDSNXCRS LOOP
  (structured format)
```

Diagnosing a wait by using the SYSZDSN3.ERLYOLRH ENQ

If Db2 and, possibly, z/OS are unresponsive, the holding of ENQ resource SYSZDSN3.ERLYOLRH can provide important clues about the location, but not the cause, of a wait. Normally this resource is held for short periods of time. If a number of tasks are holding or waiting for the resource longer than a few seconds, then the tasks should be examined.

About this task

SYSZDSN3.ERLYOLRH.ERLY_block_address is the full name of the ENQ resource. ERLY_block_address is a 32-bit binary address. If more than one Db2 subsystem is active, the ERLY_block_address value is unique for each subsystem. Be sure to look at the correct ENQ resource name.

Procedure

To locate the resource and analyze the status of each holding task:

1. The holders of the ENQ resource can be located by the following methods:
 - Various software monitor products can display the status of ENQ resources. However, if TSO or the z/OS consoles are unresponsive, not all of them are working.

- If z/OS is unresponsive, a stand-alone dump should have been taken before you restart the system. Find the SYSZDSN3.ERLYOLRH ENQ resource in the GRSTRACE portion of the dump. If you are printing the dump, include the GRSTRACE verb. Locate the ENQ resources in the section that is entitled, OUTPUT FROM GRSTRACE VERB. If you are using the dump online using IPCS, then invoke the GRSTRACE verb directly.
- Issue the following z/OS console command, which can also be issued by way of the TSO OPER command:

```
D GRS,RES=(SYSZDSN3,*)
```

2. If the ENQ resource is not held, skip the rest of this section. If it is held, the next step should be to determine if a problem exists in GRS. Review any messages and accompanying return and reason codes you receive.

If any match those codes in the following table, it indicates a possible GRS problem.

Table 2. Messages and codes for GRS problems

| Message | Return Code | Reason Code |
|----------|-------------|-------------|
| IDC3009I | 4 | Any |
| | 184 | Any |
| IEC161 | 28 | 184 |
| | 52 | Any |

3. In the GRSTRACE display of the SYSZDSN3.ERLYOLRH ENQ resource, an exclusive request at the top of the list prevents any shared request from getting the resource.
 - Shared holders that are higher in the list than an exclusive request must all release the resource before the exclusive request can be satisfied.
 - In Db2, a task holds this resource while it is running in the Db2 subsystem interface code. Tasks hold the resource in SHARED mode, except for the Db2 job step tasks and a z/OS master scheduler task, which is transmitting an End-of-Memory (EOM) condition on a Db2 address space. These tasks request the ENQ in EXCLUSIVE mode.
4. If all of the tasks are holding the resource in SHARED mode, then one or more are waiting in some Db2 module.
 - If you are viewing the ENQ resource online with a software monitor, the -DISPLAY THREAD command might be used to determine the status of the corresponding threads. Match the ASIDs in the list of tasks that are holding the resource with the ASIDs of the threads. For an interpretation of the -DISPLAY THREAD fields and the actions to take, refer to [“Initial procedure for the WAIT/LOOP keywords”](#) on page 21. Discontinue using this procedure unless the wait state cannot be cleared.
 - If you are viewing the ENQ resource from a dump, it should be a stand-alone dump. It is necessary to look at the status of each task that holds the resource. The address and module name of the wait point must be determined for each task. Refer to [“Locating the waiting CSECT”](#) on page 27 below.
5. If any task is requesting or holding the resource in EXCLUSIVE mode, then the Db2 subsystem is in the process of abnormal termination. Remember the jobname, ASID, and TCB address of the task. As stated above, it is either a Db2 job step or z/OS master scheduler task.
 - If the task is the first on the list, it is holding the ENQ resource. The Db2 subsystem interface code is attempting to complete subsystem abend termination, but has been unable to do so.
 - If the task is not first on the list, then it is waiting to obtain the resource in EXCLUSIVE mode, but is unable to do so because of one or more SHARED holders. The process of Db2 subsystem abend issues an ABEND with code X'04F' against the allied task of each thread that runs Db2 code. The presence of SHARED holders indicates that the abend was unsuccessful on those holders.
 - In any case, under this condition, Db2 cannot be recovered. If a stand-alone dump and restart the system has not been requested, do so now.

- Using the stand-alone dump, locate the address space and task of the EXCLUSIVE holding or waiting task and any shared holding tasks. Determine the address and module name of the wait point for each task.

If the dump is printed, use the z/OS TCB Summary to locate each address space and task in the dump. If you are viewing the dump by using IPCS, use IPCS procedures to locate each.

Any shared waiting tasks that follow the EXCLUSIVE holding task need not be examined. They have not caused the wait state.

Locating the waiting CSECT

If Db2 or z/OS are not operational, you can try locating the waiting CSECT.

Procedure

To locate the waiting CSECT:

- Determine whether an SVRB (supervisor request block) is beneath the lowest PRB (program request block) of a task.

- If not, check its PSW to see whether it points into DSNVSR.

If this PSW points into DSNVSR, use save area sets to determine the CSECT and offset from which suspend was called. Refer to [“If Db2 and/or z/OS are not operational”](#) on page 25 and follow the procedure that is given for waits.

- If so, look in the field that is labeled "W-L-I-C" in the PRB; the last byte in this field indicates the SVC ran. The PSW points to the instruction that follows the SVC.

If the interrupt code is X'2F', the SVC was a STIMER. Db2 uses STIMER loops in several places to wait for the occurrence of an event. The PSW in the PRB is in the CSECT in which the STIMER loop is occurring.

- Find the CSECT and load module names by using one of the following methods.

- Using the module entry point list (MEPL) enables positive verification of an entry point, which is helpful in determining the offset of an instruction within a CSECT. (The MEPL can also be obtained by specifying IFCID 186 in the IFCID() parameter of the -START TRACE command.) Refer to [“Printing and analyzing dumps”](#) on page 181 for detailed information about MEPLs.

There are three MEPLs:

- Batch Utilities MEPL

A batch utilities MEPL is found in all SVC dumps that are issued by Db2 involving batch utility abends. This list contains the CSECTs for the batch load module, DSNUTILB. Use the batch utilities MEPL only when message DSNU0171 is issued or when the dump title lists DSNUTILB as the failing load module.

- "Early Code" MEPL

If the component ID in a dump title is XYR01, an "early code" MEPL is included in the summary dump for load module DSN3EP. The eyecatcher is EEP MEPL-LIKE FOR DSN3EP. Use this MEPL when the failure occurred in a CSECT of load module DSN3EP; its format is the same as the other MEPLs. Locate the failing CSECT by following the procedure that is described under [“Finding the name of the failing CSECT in the MEPL”](#) on page 212. The ASID of load module DSN3EP is always 0000.

- "Standard" MEPL

The "standard" MEPL is included in all dumps of the Db2 address spaces. If the dump is initiated by component 5740XYR00, that MEPL is included in the summary section of the dump. The standard MEPL contains entries for all load modules and CSECTs in the Db2 address spaces. It can be recognized by the "MEPL" eyecatcher or by several pages of EBCDIC CSECT or load module names on the interpreted area of the dump at the right. Use the standard MEPL when the criteria for using the other two MEPLs do not apply.

Refer to [“The module entry point list \(MEPL\)” on page 209](#) and [“The batch utilities MEPL” on page 213](#) for detailed information about how to find the CSECT and load module in the MEPL.

- Using the PSW
 - a. Locate the instruction to which the PSW points and back up to the EBCDIC eyecatcher for the CSECT. This contains the function modification identifier (FMID), which indicates the fix level.
 - b. Verify that you are in the correct address space. Look at the XSB just under the PRB from which the PSW was obtained. The PASID field indicates the ASID of the current primary address space, from which instructions are being fetched.
- 3. Add **WAIT xxxxxx** to the keyword string, replacing **xxxxxx** with the CSECT name. If the CSECT name is unavailable or if a search with this name is unsuccessful, replace it with the load module identifier. For more information, turn to [“Load module modifier keyword” on page 38](#). To conduct a search, turn to Chapter 2, [“Searching for known problems and solutions for Db2 for z/OS,” on page 3](#).

```
5740XYR00 R131 WAIT DSNXCR2
  (free format)
PIDS/5740XYR00 LVLS/131 RIDSDSNXCR2 WAIT
  (structured format)
```

If users in more than one environment cannot issue SQL statements

This situation is when users of two or three of the attachment facilities cannot complete transactions, but Db2 is not "hung"; Db2 still responds to commands, or one attachment facility still operates normally.

Procedure

1. If the z/OS GTF trace facility is not active, issue the z/OS START GTF command with the USR and TIME=YES options.
2. Verify that dump data sets are available.
3. Issue the z/OS DUMP command to dump the Db2 address spaces (database services, system services, distributed data facility address space (for distributed processing only) and any user address spaces) and the IRLM address space. Many large dumps are produced that can help to diagnose the problem if other methods fail.
4. Determine if the problem is a WAIT or LOOP.
 - Examine the trace tables: GTF, Db2, TSO, GRS, CICS, IMS attachment facility, or IMS. Be aware of any repeating patterns, which might indicate a LOOP.

To use the GRS (global resource serialization) trace for a stand-alone dump, use the GRSTRACE keyword when the dump is printed.
 - Investigate what some users were doing before the problem occurred. If several users were doing the same thing, such as trying to access one resource, this might indicate a WAIT.
5. Add **WAIT** or **LOOP** to the keyword string. If unsure about which is appropriate, create two strings, one with each keyword. Turn to Chapter 2, [“Searching for known problems and solutions for Db2 for z/OS,” on page 3](#).

```
(free format):
5740XYR00 R131 WAIT
5740XYR00 R131 LOOP
(structured format):
PIDS/5740XYR00 LVLS/131 WAIT
PIDS/5740XYR00 LVLS/131 LOOP
```

If IMS dependent regions cannot issue SQL statements

This is a situation in which users communicating to Db2 through IMS cannot complete transactions, but Db2 is still operational, still responding to commands, or operating normally through an attachment facility.

1. Issue the IMS /DISPLAY ACTIVE command repetitively to identify the transactions being processed by each dependent region. Any dependent regions whose transaction codes are not changing might be involved in a WAIT or LOOP.
2. Issue the IMS /DISPLAY TRAN xx to verify that the program/transactions are not stopped.
3. Issue the z/OS command DISPLAY ACTIVE *jobname* repetitively, once for each message region that appears to be in a WAIT or LOOP.
4. Determine if the processor time remains the same. If so, the dependent region is probably in a WAIT state. Follow the steps in [“Gathering the information” on page 29](#).
5. Otherwise, determine if the dependent region is in a loop.
 - IMS detects loops that occur within IMS dependent regions. The user specifies the timeout value using the TRANSACTION macro PROCLIM parameter.
 - If the dependent region has a long timeout, issue the IMS /DISPLAY TRAN y command. If this shows the IMS transaction is decreasing, the region is processing messages and the application is not in an endless loop.
 - If you want you can issue the Db2 command -DISPLAY THREAD and review the "REQ" column, which often indicates loops.
6. Issue the IMS /DISPLAY SUBSYSTEM command to determine the status of the connection to Db2.
7. Issue the IMS /TRACE SET ON SUBS OPTION LOG command to trace IMS calls to Db2 on the IMS log.
8. Issue the following IMS commands in order until one breaks the connection. If successful, skip the next step. If not successful, go to the next step.

```
/STOP region reg# abdump  
/STOP region reg# CANCEL  
/STOP SUBSYS subsysname
```

If none of these commands break the connection, the IMS control region might need to be forced into an abnormal abend. Before doing this, you must understand that all the IMS dependent regions and not only the one that has the connection problems with Db2 are terminated, and that an emergency restart of IMS(/ERE) has to be performed. The commands used to terminate the control region are explained in the next step.

9. Issue the following MODIFY commands in this order until one successfully breaks the subsystem connection. Do not confuse these with the /MODIFY command or the MODIFY utility.

```
F IMS,DUMPname  
F IMS,STOPname  
F IMS,FORCEname
```

name refers to the attached external subsystem name defined in the IMS external subsystem PROCLIB member.

Gathering the information

1. If the reason for the wait or loop has not been determined, issue the IMS /STOP REGION x ABDUMP yyyy command to terminate the specific region involved in the wait or loop. If you receive a dump, the problem is in the application's address space and is probably the result of a user error. Analyze the dump and correct the problem. Do not continue with this procedure.
2. If the z/OS generalized trace facility (GTF) is not active, issue the z/OS START GTF command with the USR and TIME=YES options.
3. Verify that dump data sets are available.

4. Use the z/OS DUMP command to dump the Db2 address spaces (database services, system services, the distributed data facility address space for distributed processing only, and any user address spaces) and the IRLM address space. This generates many very large dumps, which can help to diagnose the problem if other methods fail.
5. Terminate the dependent region and produce a SYSABEND or SYSUDUMP. Be aware that doing so can cause the IMS control to ABEND, with an ABENDU113, or cause Db2 to ABEND. With a dump data set available, issue the IMS /STOP REGION x CANCEL command.
6. Print the entries in the SYS1.LOGREC data set for the estimated time that the loop or wait began. For more information on printing SYS1.LOGREC entries, refer to [“SYS1.LOGREC” on page 226](#).
7. Analyze the TOD (time of day) information in SYS1.LOGREC to determine if any failures occurred around the time the wait or loop occurred. Find entries related to the Db2 or dependent region address spaces.
8. Determine if users are doing something in the IMS attachment facility that might prevent them from accessing Db2. This might indicate a wait or loop in another component or in the attachment facility.
9. Determine if control is in IMS or in Db2. Review the information in the SYSABEND dump, IMS attachment facility trace table, and Db2 trace table. The trace tables are described in [“Printing and analyzing global traces” on page 238](#).
10. If control is in IMS, refer to [IMS diagnosis information](#). Otherwise, keep the SYS1.LOGREC entries, the GTF trace, and the dumps you obtained in Step 4.
11. Add WAIT or LOOP to the keyword string. If unsure about which is appropriate, create two strings, one with each keyword. Turn to [Chapter 2, “Searching for known problems and solutions for Db2 for z/OS,” on page 3](#).

```
(free format):
5740XYR00 R131 WAIT
5740XYR00 R131 LOOP
(structured format):
PIDS/5740XYR00 LVLS/131 WAIT
PIDS/5740XYR00 LVLS/131 LOOP
```

If DSN users cannot issue SQL statements

This is a situation in which users communicating with Db2 through the DSN command processor cannot complete transactions, but Db2 is still operational (still responding to commands, or operating normally through other attachments).

1. If you are a SPUFI user, determine if the SQL statement just issued, and for which you received no response, is one that can take longer than usual to execute.

For example, a statement that updates 80,000 records undoubtedly takes longer than one that updates one record.
2. Verify that the application programs are checking SQL return codes properly.

A return code of +100 is received when work completes successfully.
3. If the problem is in a batch environment, refer to [“In a batch environment” on page 32](#). If the problem is in a foreground environment, continue below with [“In a foreground environment” on page 30](#).

The TSO attachment facility provides several different tracing mechanisms, only one of which, the DSN trace stream, is mentioned here. For information about other tracing mechanisms, refer to [“TSO attachment facility traces” on page 263](#).

In a foreground environment

1. Try to terminate the current DSN session. Enter 'END' or Enter 'END' press the ATTENTION key twice.
2. If the z/OS generalized trace facility (GTF) is not active, issue the z/OS START GTF command with the USR and TIME=YES options.
3. Issue the Db2 command -START TRACE(GLOBAL) DEST(GTF).

4. Allocate a DSNTRACE data set to collect a copy of the DSN trace messages that are going to appear on your terminal.
5. Restart the DSN session using the DSN command DSN SYSTEM(*subsystem-name*) TEST(255), where *subsystem-name* is the site-defined name of the subsystem.
6. Review the DSN trace messages produced. Find any trace messages containing

```
BEFORE . . . =====  
AFTER  . . . =====
```

These indicate control is being passed between the application and Db2. For an example, see the following figure.

7. If the last trace stream message contains BEFORE with no corresponding AFTER message following, the problem is in Db2. Follow the steps below:
 - a. Verify that dump data sets are available.
 - b. Issue the z/OS DUMP command to dump the Db2 address spaces and the address space of the affected TSO user. This generates very large dumps, which can help to diagnose the problem if other methods fail.
8. If the last trace message does not contain BEFORE, the problem is in the application or the DSN command processor. Follow the steps below:
 - a. Verify that the application program is not in error. If necessary, have the operator cancel the TSO users involved in the problem. Request dumps for each user whose DSN session is canceled.
 - b. If you see an unending series of messages similar to those shown below, the application program is in a loop. Verify that it is checking the SQL return codes properly.

```
BEFORE SQL CALL =====  
AFTER SQL CALL =====
```


4. Cancel the job.
5. Resubmit the job with a SYSUDUMP DD card, using the DSN command DSN SYSTEM(*subsystem-name*)(TEST(255). Replace *subsystem-name* with the site-defined name of the subsystem.
6. Let the job run until you think the problem has occurred again. Then have the operator cancel the job with a dump.
7. Review the trace output. In batch DSN sessions, the trace messages go to the SYSTSPRT data set. [Figure 5 on page 32](#) provides an example of these trace messages. Find any trace messages similar to

```
BEFORE . . . =====
AFTER  . . . =====
```

These indicate control is being passed between the application and Db2.

8. Verify that the application program is not causing a loop.
 - a. Submit the job two or more times and compare the amount of trace message output. Determine if the trace messages seem to continue until the job times out or the operator cancels it.
 - b. If necessary, have the operator cancel the TSO users involved. Request dumps for each user whose DSN session is canceled.
 - c. Check for an unending sequence of messages similar to those shown below. If this occurs, the application program is causing the loop. Verify that it is checking the SQL return codes properly.

```
BEFORE SQL CALL =====
AFTER SQL CALL =====
```

9. If the problem appears to be in the DSN command processor, add WAIT TSOATTACH or LOOP TSOATTACH to the keyword string. If unsure which is appropriate, build two keyword strings, one with each keyword. Then turn to [Chapter 2, “Searching for known problems and solutions for Db2 for z/OS,” on page 3](#).

```
(free format):
5740XYR00 R131 WAIT TSOATTACH
5740XYR00 R131 LOOP TSOATTACH
(structured format):
PIDS/5740XYR00 LVLS/131 TSOATTACH WAIT
PIDS/5740XYR00 LVLS/131 TSOATTACH LOOP
```

10. If the problem appears to be in Db2, add WAIT or LOOP to the keyword string. If unsure which is appropriate, build two keyword strings, one with each keyword. Then turn to [Chapter 2, “Searching for known problems and solutions for Db2 for z/OS,” on page 3](#).

```
(free format):
5740XYR00 R131 WAIT
5740XYR00 R131 LOOP
(structured format):
PIDS/5740XYR00 LVLS/131 WAIT
PIDS/5740XYR00 LVLS/131 LOOP
```

Related concepts

Call attachment facility traces

The call attachment facility, which enables users to connect to Db2 through TSO foreground, TSO background, or z/OS batch, provides diagnostic trace messages intended primarily for use by IBM service personnel.

MSGx keyword

Use this keyword if an error is associated with a Db2 or an IRLM message. If you receive more than one message for one error, search the database using the first message issued. If unsuccessful, search the database using the next message, then the next, and so on.

To see if other messages related to the problem have been issued, check the console sheets that contain Db2 messages, as well as messages issued by other products. If any messages are prefixed with

"IEC", indicating it was issued by media manager services, check the SYSLOG for messages that identify associated VSAM problems. SYSLOG can also help to diagnose user errors.

Compare the message prefix with those shown in the table below to determine the appropriate procedure to follow.

Table 3. Message prefixes

| Prefix | Component | Procedure |
|--------|------------------------|---|
| DSN | Db2 | Follow “Procedure for Db2 messages” on page 34 on this page |
| DXR | IRLM | Follow “Procedure for IRLM messages” on page 34 |
| DFH | CICS | Consult CICS messages |
| DFS | IMS | Consult IMS messages and codes |
| IDC | Access method services | Consult z/OS MVS IDC messages |
| IKJ | TSO/E | Consult z/OS MVS IKJ messages |
| IST | VTAM | Consult z/OS Communications Server: SNA Messages |

Procedure for Db2 messages

1. Check if the name of the CSECT issuing the message appears. This name follows the message number, as shown in the following example.

```
DSNW413I s csect_name - ACCOUNTING FACILITY HAS LOST
      DATA RC=y
DSNJ202I csect_name INSUFFICIENT VIRTUAL STORAGE
      AVAILABLE TO CONTINUE WITH UTILITY.
```

No CSECT name appears if only one CSECT can issue this message.

2. Determine if the message contains any variables, such as return or reason codes.
3. If no CSECT name appears, build a keyword similar to MSGxxxxxxx, replacing xxxxxxxx with the message number. Add this to the keyword string. If the message contains any variables, turn to [“Message modifier keyword” on page 49](#). Otherwise, turn to [Chapter 2, “Searching for known problems and solutions for Db2 for z/OS,” on page 3](#).

```
(free format):
5740XYR00 R131 MSGDSNE004E
(structured format):
PIDS/5740XYR00 LVLS/131 MS/DSNE004E
```

4. If a CSECT name does appear, build a keyword similar to MSGxxxxxxxxyyyyyyyyy. Replace xxxxxxxx with the message number, and replace yyyyyyy with the CSECT name. Add this to the keyword string. If the message contains any variables, turn to [“Message modifier keyword” on page 49](#). Otherwise, turn to [Chapter 2, “Searching for known problems and solutions for Db2 for z/OS,” on page 3](#).

```
(free format):
5740XYR00 R131 MSGDSNJ202I DSNJU0001
(structured format):
PIDS/5740XYR00 LVLS/131 MS/DSNJ2021
RIDS/DSNJU0001
```

Procedure for IRLM messages

1. Determine if the message contains any variables, such as return or reason codes.
2. Build a keyword similar to **MSGxxxxxxx**, replacing **xxxxxxx** with the message number. Add this to the keyword string. If the message contains variables, turn to [“Message modifier keyword” on page](#)

49. Otherwise, turn to [Chapter 2, “Searching for known problems and solutions for Db2 for z/OS,”](#) on page 3.

```
(free format):  
569516401 AR220 MSGDXR124E  
(structured format):  
PIDS/569516401 LVLS/220 MS/DXR124E
```

PERFM keyword

Most performance problems can be resolved through system tuning and should be handled by the Db2 system administrator.

About this task

Before following this procedure, use the checklist below to verify that the performance problem cannot be resolved through other means.

- Try to tune performance.
- Verify that the performance problem is not related to a wait or loop. Refer to [“Initial procedure for the WAIT/LOOP keywords”](#) on page 21.
- If performance degraded after someone tuned Db2, verify that the tuning options selected were appropriate. Most likely, the problem can be resolved by choosing other options.

Procedure

1. Record the actual performance, expected performance, and source of expected performance criteria.
2. Add PERFM to the keyword string, and turn to [Chapter 2, “Searching for known problems and solutions for Db2 for z/OS,”](#) on page 3.

```
(free format):  
5740XYR00 R131 PERFM  
(structured format):  
PIDS/5740XYR00 LVLS/131 PERFM
```

INCORROUT keyword

Use this procedure when output was expected but not received or when output was different from expected.

Procedure

Add INCORROUT to the existing keyword string, and use [“INCORROUT modifier keyword”](#) on page 42 to specify the problem.

```
(free format):  
5740XYR00 R131 INCORROUT (for DB2)  
569516401 R220 INCORROUT (for IRLM 2.2)  
(structured format):  
PIDS/5740XYR00 LVLS/131 INCORROUT  
(for DB2)  
  
PIDS/569516401 LVLS/220 INCORROUT  
(for IRLM 2.2)
```

CSECT keyword

To find the name of the failing CSECT, use the SVC dump title. If the dump title is not available, use the SYS1.LOGREC entry or the first page of an SVC dump.

Any CSECT name that is located following these procedures should begin with:

DSN for Db2

DXR for IRLM

DSNC or DSN2 for CICS attachment

DSNM for IMS attachment

DSN3, DSNA, DSNV, or DSNZ for the subsystem initialization component, or the primary Db2 component. Some DSNA CSECTs belong to the call attachment facility.

- If the CSECT prefix is DSN3, DSNA, DSNV, or DSNZ follow the steps below.
 1. Use SMP to list the CSECT. Issue the command 'LIST CDS MOD(*csect-name*)'. Locate the FMID from this.
 2. If the FMID is HIZ2220, use the subsystem initialization component identifier (5740XYR01) in the keyword string. If the FMID is HDB2220, use the primary Db2 component identifier (5740XYR00). Consult the z/OS diagnostic techniques publication to identify the failing product.
- If the prefix denotes an attachment facility, use its component identifier in the keyword string: 5740IX100 for CICS (old attachment facility), 565501800 for CICS (new attachment facility), and 5740IY100 for IMS.

If a CSECT name with a different prefix is found, the problem probably is not in Db2. Isolate the failing product using the z/OS diagnostic techniques publication.

Other ways of finding the CSECT name are explained elsewhere in this section. For example:

- If you received a Db2-generated message and if the type-of-failure keyword is MSGx, follow the procedures for [“MSGx keyword”](#) on page 33.
- If using SQL and access is available to an SQL Communication Area (SQLCA) for the problem, use the procedure for [“SQLCODE modifier keyword”](#) on page 48. You have access to the SQLCA if the application program displays it or if a dump was generated for this specific problem.

If the CSECT cannot be found, turn to [“Recovery routine modifier keyword”](#) on page 40.

Procedure using SVC dump title

To find the name of the failing CSECT using the SVC dump title:

1. Locate the second word following the label LOC=. This is the CSECT name (a number follows it).
 - For an example, see [“SVC dump titles that are issued by Db2”](#) on page 199.
 - If the dump title does not contain this label, follow either [“Procedure using a SYS1.LOGREC entry”](#) on page 37 or [“Procedure using first page of an SVC dump”](#) on page 37.
2. If the reason code found when following [“ABENDx keyword”](#) on page 14 is X'00C90101', X'00C90102', X'00C90105', or in the range of X'00C902xx', locate the four-character reason code qualifier in the dump title. See [Figure 37](#) on page 201.

Build a keyword similar to xxxxxxxx VRACE yyyy or xxxxxxxx ERQUAL yyyy. Replace xxxxxxxx with the CSECT name and yyyy with the reason code qualifier. Add this to the keyword string. To find the load module name and any appropriate VRA data, see [“Modifier keyword”](#) on page 37.

```
(free format):
5740XYR00 R131 ABEND04E RC00C90105 DSNKDLEV
VRACE0D01
5740XYR00 R131 ABEND04E RC00C90105 DSNKDLEV
ERQUAL 0D01
(structured format):
PIDS/5740XYR00 LVLS/131 PRCS/00C90105
RIDS/DSNKDLEV FLDS/VRACE VALU/H0D01
PIDS/5740XYR00 LVLS/131 PRCS/00C90105
RIDS/DSNKDLEV FLDS/ERQUAL VALU/H0D01
```

3. If the reason code found when following [“ABENDx keyword”](#) on page 14 begins with X'00E7', locate the P or M sign in the dump title and the three-digit decimal reason code qualifier that follows it. For an example, see [Figure 39](#) on page 201.

Append the qualifier to SQLERRD. Add this and the CSECT name to the keyword string. To find the load module name and any appropriate VRA data, see [“Modifier keyword” on page 37](#).

```
(free format):
5740XYR00 R131 ABEND04E RC00E70005 DSNXAMCT
SQLERRD101
(structured format):
PIDS/5740XYR00 LVLS/131 AB/S004E
PRCS/00E7005 RIDS/DSNXAMCT FLDS/SQLERRD101
```

4. If the reason code found when following [“ABENDx keyword” on page 14](#) begins with X'00E2', the name of the CSECT calling the failing Storage Manager CSECT is included in the VRA. Search the database using the information in the dump title. The search can be narrowed by including the CSECT name, turn to [“VRA data modifier keyword” on page 41](#).

```
(free format):
5740XYR00 R131 ABEND04E RC00E2000B
DSNSFBK
(structured format):
PIDS/5740XYR00 LVLS/131 AB/S004E
PRCS/00E2000B RIDS/DSNSFBK
```

Procedure using a SYS1.LOGREC entry

To find the name of the failing CSECT by using a SYS1.LOGREC entry:

1. Locate the SYS1.LOGREC entry related to the error. Compare the date, time, ERRORID string, and program name in the dump with those in the SYS1.LOGREC entry.
2. Locate the CSECT name in the SYS1.LOGREC entry. It follows the words NAME OF CSECT INVOLVED on the top left corner of the first page. For an example, turn to [“SYS1.LOGREC” on page 226](#).
3. Add the CSECT name to the keyword string. The component identifier must correspond to the CSECT prefix. To determine the load module name and any appropriate VRA data, see [“Modifier keyword” on page 37](#).

```
(free format):
5740XYR00 R131 ABEND0C4 DSNVATRM
(structured format):
PIDS/5740XYR00 LVLS/131 AB/S00C4
RIDS/DSNVATRM
```

Procedure using first page of an SVC dump

To find the name of the failing CSECT using the first page of an SVC dump:

1. Locate the SYMPTOM DATA column and note the name listed as the second item. This is the CSECT name, as the EXPLANATION column indicates. For an example, see [Figure 33 on page 197](#).
2. Add the CSECT name to the keyword string and turn to [“Modifier keyword” on page 37](#) if you want to narrow the search further.

```
(free format):
5740XYR00 R131 ABEND0C4 DSNVATRM
(structured format):
PIDS/5740XYR00 LVLS/131 AB/S00C4
RIDS/DSNCATRM
```

Modifier keyword

Modifier keywords identify problems more precisely and help to narrow the search. Most of these keywords change only certain failures.

The following table lists several modifier keywords. See the column "Conditions for Using This Keyword" to determine which keyword to use; follow the corresponding procedure.

Table 4. Types of modifier keywords

| Keyword | Conditions for Using This Keyword | Description | Procedure to Follow |
|---------------------------|---|--|--|
| Load Module modifier | You have a related SYS1.LOGREC entry or SVC dump. | Identifies the name of the load module involved. | “Load module modifier keyword” on page 38 |
| RECOVERY ROUTINE modifier | Both conditions are true: <ul style="list-style-type: none"> The load module and CSECT names cannot be found in the SVC dump or in the SYS1.LOGREC entry, and You have an SVC dump title. | Identifies the name of the functional recovery routine (FRR) or the extended specify task abnormal exit (ESTAE) that handled the program failure. | “Recovery routine modifier keyword” on page 40 |
| VRA Data modifier | You have access to a variable recording area (VRA) for the abend. The VRA is in the system diagnostic work area (SDWA), which when available is in the SYS1.DUMP data set and the SYS1.LOGREC data set. | Identifies information stored in the VRA that describes the abend. | “VRA data modifier keyword” on page 41 |
| SQLCODE modifiers | All these conditions are true: <ul style="list-style-type: none"> The type-of-failure keyword was INCORROUT, The function keyword was SQL, You are fairly certain the problem was not caused by a user error, and You have access to the SQL communication area (SQLCA). <p>(The SQLCA can be displayed by the application program or found in a dump.)</p> | Identifies the additional information to describe the problem, such as: <ul style="list-style-type: none"> SQLCODE (SQL return code) Name of the CSECT that issued the return code SQL subcode. | “SQLCODE modifier keyword” on page 48 |
| INCORROUT modifiers | The type-of-failure keyword is INCORROUT | Identifies various primary and secondary modifier keywords. | “INCORROUT modifier keyword” on page 42 |
| MESSAGE modifier | Both conditions are true: <ul style="list-style-type: none"> The type-of-failure keyword is MSGx, and The Db2-issued message you received contains variables | Identifies additional items of information provided when the message is issued. | “Message modifier keyword” on page 49 |

Load module modifier keyword

Use this keyword if the search was unsuccessful when you used the CSECT keyword or the search yielded too many possible matches when you used the CSECT keyword.

If the search was unsuccessful, replace the CSECT name with the load module name and try again. If the search yielded too many possible matches, add the load module name to the string to further narrow the search.

All Db2 load module names begin with DSN; all IRLM load module names begin with DXR. If you follow these instructions and find a load module name with a different prefix, the problem is probably in another product. Refer to [z/OS MVS Diagnosis: Tools and Service Aids](#) to identify the failing product.

To locate the load module, use the SVC dump title. If the dump title is not available, use the SYS1.LOGREC entry or the first page of an SVC dump.

Procedure using the SVC dump title

1. Locate the first word following the label LOC=. This is the load module name, and it precedes the CSECT name.

For an example, see [“SVC dump titles that are issued by Db2” on page 199](#).

If the dump title does not contain this word, follow either [“Procedure using a SYS1.LOGREC entry” on page 39](#) or [“Procedure using first page of an SVC dump” on page 40](#) on this page.

2. Add the load module name to the keyword string, or substitute it for the CSECT name as appropriate. If you have access to the variable recording area (VRA) for the abend, turn to [“VRA data modifier keyword” on page 41](#). If not, search the database again using the revised keyword string. Turn to Chapter 2, [“Searching for known problems and solutions for Db2 for z/OS,” on page 3](#).

```
(free format):
5740XYR00 R131 ABEND04E RC00E50013 DSNSLD1
  DSNSVSTK
  (with load module name and then CSECT name)
5740XYR00 R131 ABEND04E RC00E50013
  DSNSLD1
  (with load module name only)
(structured format):
PIDS/5740XYR00 LVLS/131 AB/S004E
  PRCS/00E50013 RIDS/DSNSLD1#L RIDS/DSNSVSTK
  (with load module name and then CSECT name)
PIDS/5740XYR00 LVLS/131 AB/S004E
  PRCS/00E50013 RIDS/DSNSLD1#L
  (with load module name only)
```

Procedure using a SYS1.LOGREC entry

To find the name of the failing CSECT using the first page of an SVC dump:

1. Locate the SYS1.LOGREC entry related to the error. Compare the date, time, program name, and ERRORID string in the dump with those in the SYS1.LOGREC entry.
2. Locate the load module name in the SYS1.LOGREC entry. It follows the words NAME OF MODULE INVOLVED in the top left corner of the first page. For an example, turn to [Figure 66 on page 227](#).

If UNKNOWN appears instead of the load module name, try to determine the load module by following [“Procedure using first page of an SVC dump” on page 40](#).

3. Add the load module name to the keyword string, or substitute it for the CSECT name as appropriate. If you have access to the variable recording area (VRA) for the abend, turn to [“VRA data modifier keyword” on page 41](#). If not, search the database again using the revised keyword string. turn to Chapter 2, [“Searching for known problems and solutions for Db2 for z/OS,” on page 3](#).

```
(free format):
5740XYR00 R131 ABEND04E RC00F9000C DSN9PREP
  DSN9SCNP
  (with load module name and then CSECT name)
5740XYR00 R131 ABEND04E RC00F9000C DSN9PREP
  (with load module name only)
(structured format):
PIDS/5740XYR00 LVLS/131 AB/S004E RIDS/DSN9PREP#L
  RIDS/DSN9SCNP
  (with load module name and then CSECT name)
PIDS/5740XYR00 LVLS/131 AB/S004E
  RIDS/DSN9PREP#L
  (with load module name only)
```

Procedure using first page of an SVC dump

To find the name of the failing CSECT using the first page of an SVC dump:

1. Locate the column titled SYMPTOM DATA and the name listed as the first item. This is the load module, as the EXPLANATION column indicates.
2. For an example, see [Figure 33 on page 197](#).

If UNKNOWN appears instead of the load module name, try to determine the load module by following [“Procedure using a SYS1.LOGREC entry” on page 39](#). If the SYS1.LOGREC entry also has UNKNOWN for the load module name, try to improve the search argument by turning to [“Recovery routine modifier keyword” on page 40](#).

3. Add the load module name to the keyword string, or substitute it for the CSECT name as appropriate. If you have access to the variable recording area (VRA) for the abend, turn to [“VRA data modifier keyword” on page 41](#). If not, search the database again using the revised keyword string. Turn to [Chapter 2, “Searching for known problems and solutions for Db2 for z/OS,” on page 3](#).

```
(free format):
5740XYR00 R131 ABEND04E RC00E50013 DSNSLD1
  DSNSVSTK
(with load module name and then CSECT name)
5740XYR00 R131 ABEND04E RC00E50013
  DSNSLD1
(with load module name only)
(structured format):
PIDS/5740XYR00 LVLS/131 AB/S004E
  PRCS/00E50013 RIDS/DSNSLD1#L RIDS/DSNSVSTK
(with load module name and then CSECT name)
PIDS/5740XYR00 LVLS/131 AB/S004E
  PRCS/00E50013 RIDS/DSNSLD1#L
(with load module name only)
```

Recovery routine modifier keyword

Include the name of the recovery routine only when an SVC dump is available and the names of the CSECT and load module that is involved at the time of failure cannot be determined after you look in both the SVC dump and the SYS1.LOGREC entry.

To obtain the recovery routine name, use the SVC dump title. If the dump title is not available, use the SYS1.LOGREC entry or the first page of an SVC dump.

Procedure using the SVC dump title

1. Locate the area of the dump title containing the symbol M=. The word following this identifies the FRR (functional recovery routine) or the ESTAE (extended specify task abnormal exit). For an example, turn to [“SVC dump titles that are issued by Db2” on page 199](#).
2. Add that word to the keyword string. If you have access to the variable recording area (VRA) for the abend, turn to [“VRA data modifier keyword” on page 41](#). If not, search the database, following the instructions in [Chapter 2, “Searching for known problems and solutions for Db2 for z/OS,” on page 3](#).

```
(free format):
5740XYR00 R131 ABEND04E RC00E20015
  DSNTFRCV
(structured format):
PIDS/5740XYR00 LVLS/131 AB/S004E
  PRCS/00E20015 RIDS/DSNTFRCV#R
```

Procedure using a SYS1.LOGREC entry

1. Locate the SYS1.LOGREC entry related to the error. Compare the date, time, program name, and ERRORID string in the dump with those in the SYS1.LOGREC entry.
2. Locate the recovery routine name in the SYS1.LOGREC entry. It follows the words RECOVERY ROUTINE in the top left corner of the first page. For an example, see [Figure 66 on page 227](#).

3. Add this name to the keyword string. If you have access to the variable recording area (VRA) for the abend, turn to [“VRA data modifier keyword” on page 41](#). If not, search the database, following the instructions in [Chapter 2, “Searching for known problems and solutions for Db2 for z/OS,” on page 3](#).

```
(free format):
5740XYR00 R131 ABEND04E RC00E20015
DSNTFRCV
(structured format):
PIDS/5740XYR00 LVLS/131 AB/S004E
PRCS/00E20015 RIDS/DSNTD4FRCV#R
```

Procedure using first page of SVC dump

1. Locate the column titled SYMPTOM DATA and the name listed as the fourth item. This is the recovery routine name, as the EXPLANATION column indicates. For an example, see [Figure 33 on page 197](#).
2. Add this to the keyword string. If you have access to the variable recording area (VRA) for the abend, turn to [“VRA data modifier keyword” on page 41](#). If not, search the database, following the instructions in [Chapter 2, “Searching for known problems and solutions for Db2 for z/OS,” on page 3](#).

```
(free format):
5740XYR00 R131 ABEND04E RC00E20015
DSNTFRCV
(structured format):
PIDS/5740XYR00 LVLS/131 AB/S004E
PRCS/00E20015 RIDS/DSNTFRCV#R
```

VRA data modifier keyword

Use this keyword procedure if the type of failure was ABEND and you have access to a variable recording area (VRA), in the system diagnostic work area (SDWA) of an SVC dump or a SYS1.LOGREC entry.

About this task

Some information in the VRA is common to all Db2 subcomponents; however, a given subcomponent can add specialized information at failure. In the examples that are shown here, the abend reason code X'00E2000B' is listed as one of the reason codes for the storage manager subcomponent for which significant data is stored in the VRA.

Procedure

To use the VRA data modifier keyword:

1. Determine whether the reason code received for the abend indicates that significant information is recorded in the VRA.
2. If significant information is recorded, locate the VRA Diagnostic Information Report. See [Figure 45 on page 205](#) for an example. It usually appears in the beginning of the formatted dump. For more information about the VRA report and details on locating it, see [“The variable recording area \(VRA\)” on page 205](#).
3. Scan the KEY column until the key that contains the significant data is found.
4. Determine the appropriate key name that is based on the hexadecimal key value. For example, a key of 3D for reason code X'00E2000B' indicates the key name "VRACAN". See [“SYS1.LOGREC” on page 226](#).
5. Build a keyword similar to xxxxxx yyyyyy. Replace xxxxxx with the VRA key name and yyyyyy with the data that appears in the corresponding VRA DATA FIELDS column. Add this information to the keyword string and search the database by using the CSECT name or the load module identifier (or both, if appropriate). Turn to [Chapter 2, “Searching for known problems and solutions for Db2 for z/OS,” on page 3](#).

Results

```
(free format):
5740XYR00 R131 ABEND04E RC00E2000B DSNSFBK
VRACAN DSNVASIM (with CSECT name)
5740XYR00 R131 ABEND04E RC00E2000B DSNSLD1
VRACAN DSNVASIM (with load module identifier)
5740XYR00 R131 ABEND04E RC00E2000B DSNSLD1
DSNSFBK VRACAN DSNVASIM
(with load module identifier and CSECT name)
(structured format):
PIDS/5740XYR00 LVLS/131 AB/S00E4
PRCS/00E2000B RIDS/DSNSFBK FLDS/VRACAN
RIDS/DSNVASIM (with CSECT name)
PIDS/5740XYR00 LVLS/131 AB/S00E4
PRCS/00E2000B RIDS/DSNSLD1#L FLDS/VRACAN
RIDS/DSNVASIM (with load module identifier)
PIDS/5740XYR00 LVLS/131 AB/S00E4
PRCS/00E2000B RIDS/DSNSLD1#L RIDS/DSNSFBK
FLDS/VRACAN RIDS/DSNVASIM
(with CSECT name and load module identifier)
```

Related concepts

The variable recording area (VRA)

More diagnostic information for Db2 abend reason codes is placed in the variable recording area (VRA) of the system diagnostic work area (SDWA) and is extracted and displayed in the VRA Diagnostic Information Report. This data can be produced by common recording routines and certain Db2 subcomponents.

INCORROUT modifier keyword

Use this modifier keyword if the type-of-failure keyword is INCORROUT.

Procedure

1. Determine the primary and secondary modifier keywords for the problem.
2. If the primary modifier keyword is not SQL, add that keyword and the secondary modifier keyword to the string. Refer to [Chapter 2, “Searching for known problems and solutions for Db2 for z/OS,” on page 3](#).

```
(free format):
5740XYR00 R131 INCORROUT UTILITY
MERGECOPY
(structured format):
PIDS/5740XYR00 LVLS/131 INCORROUT
```

3. If the primary modifier keyword is SQL, determine whether you have access to the SQLCA (structured query language communication area).

This area is available if the application program displays the SQLCA or if a dump was issued for this specific problem. Refer to [“Finding the SQLCA” on page 217](#) for examples.

[“The SQL communication area \(SQLCA\)” on page 216](#) describes how to locate the SQLCA in a dump.

4. If you have access to the SQLCA, add the primary and secondary modifier keywords to the string. Next, go to [“SQLCODE modifier keyword” on page 48](#).

```
(free format):
5740XYR00 R131 INCORROUT SQL INSERT
(structured format):
PIDS/5740XYR00 LVLS/131 PCSS/SQL
PCSS/INSERT INCORROUT
```

5. If you do not have access to the SQLCA, add the primary and secondary modifier keywords to the string. Next, turn to [Chapter 2, “Searching for known problems and solutions for Db2 for z/OS,” on page 3](#).

What to do next

Table 5. INCORROUT modifier keywords

| Primary keyword | Secondary keywords | Problem occurrence |
|---------------------|------------------------|--|
| CLIST | | While you use a CLIST |
| | INSTALL | While you use DSNTINST installation CLIST |
| | PREP | While you use DSNH program preparation CLIST |
| | UTILITIES | While you use DSNU utilities CLIST |
| DB2I | | During DB2I processing |
| | BIND | After you run BIND panel request |
| | DB2 COMMAND | After you run the DB2 COMMANDS panel request |
| | DCLGEN | After you run DCLGEN panel request |
| | FREE | After you run FREE panel request |
| | PREP | After you run Db2 PROGRAM PREPARATION panel request |
| | REBIND | After you run REBIND panel request |
| | SPUFI | After you run SPUFI panel request |
| IMSATTACH | UTILITY | After you run UTILITY panel request |
| | | With IMS attachment facility |
| | APPLICATION | While an IMS application was processing an SQL statement |
| | DB2 COMMAND | While you process a Db2 command by using the IMS attachment facility |
| IRLM | DISC | During IMS attachment facility disconnect connect processing |
| | | In IRLM |
| | REQLOCK | Error in lock processing |
| MIGRATION/ FALLBACK | REQPURGE | Error in PURGE processing |
| | | During migration to next release or fallback to previous release |
| | LOAD | During load phase of migration |
| | REMIGRATE | During migration after a fallback |
| | RESTORE | During restore phase of fallback |
| PRECOMPILER | UNLOAD | During unload phase of migration |
| | | During precompile |
| | ASM | During assembler processing |
| | C | During C processing |
| | COBOL | During COBOL processing |
| | FORTRAN | During FORTRAN processing |
| | PLI | During PL/I processing |
| REXX | During REXX processing | |

Table 5. INCORROUT modifier keywords (continued)

| Primary keyword | Secondary keywords | Problem occurrence |
|-----------------|--------------------|---|
| RECOVERY | | During recovery, but not associated with the RECOVER utility |
| | BACKOUT | At backout time |
| | CHECKPOINT | At checkpoint time |
| | COMMIT | At commit time |
| | LOGGING | During logging |
| | RECOVER | During attempt to recover indoubt threads |
| | RESTART | During restart processor |
| RRS ATTACH | | With Resource Recovery Services attachment facility (RRSAF) |
| | APPLICATION | While an application in the RRS environment was processing an SQL statement |
| | DB2 COMMAND | While you process a Db2 command by using RRSAF |
| | DISC | During RRSAF TERMINATE IDENTIFY processing |
| SERVICEAID | | While you run a service aid utility |
| | DSN1COPY | While you run DSN1COPY |
| | DSN1LOGP | While you run DSN1LOGP |
| | DSN1PRNT | While you run DSN1PRNT |
| | DSN1SDMP | While you run DSN1SDMP |

Table 5. INCORROUT modifier keywords (continued)

| Primary keyword | Secondary keywords | Problem occurrence |
|-----------------|--------------------|---|
| SQL | | After you issue an SQL statement |
| | ALLOCATE CURSOR | After you issue an ALLOCATE CURSOR statement |
| | ALTER | After you issue an ALTER statement |
| | ASSOCIATE LOCATORS | After you issue an ASSOCIATE LOCATORS statement |
| | CALL | After you issue a CALL statement |
| | CLOSE | After you issue a CLOSE statement |
| | COMMENT | After you issue a COMMENT statement |
| | COMMIT | After you issue a COMMIT statement |
| | CONNECT | After you issue a CONNECT statement |
| | CREATE | After you issue a CREATE statement |
| | DECLARE | After you issue a DECLARE statement |
| | DELETE | After you issue a DELETE statement |
| | DESCRIBE | After you issue a DESCRIBE statement |
| | DROP | After you issue a DROP statement |
| | EXECUTE | After you issue an EXECUTE statement |
| | EXPLAIN | After you issue an EXPLAIN statement |
| | FETCH | After you issue a FETCH statement |
| | FREE | After you issue a FREE statement |
| | GRANT | After you issue a GRANT statement |
| | HOLD | After you issue a HOLD statement |

Table 5. INCORROUT modifier keywords (continued)

| Primary keyword | Secondary keywords | Problem occurrence |
|------------------------|---|---|
| SQL continued | INCLUDE | After you issue an INCLUDE statement |
| | INSERT | After you issue an INSERT statement |
| | LOCK | After you issue a LOCK statement |
| | OPEN | After you issue an OPEN statement |
| | PREPARE | After you issue a PREPARE statement |
| | RELEASE | After you issue a RELEASE statement |
| | RELEASE SAVEPOINT | After you issue a RELEASE SAVEPOINT statement |
| | RENAME | After you issue a RENAME statement |
| | REVOKE | After you issue a REVOKE statement |
| | ROLLBACK | After you issue a ROLLBACK statement |
| | ROLLBACK TO SAVEPOINT | After you issue a ROLLBACK TO SAVEPOINT statement |
| | SAVEPOINT | After you issue a SAVEPOINT statement |
| | SELECT | After you issue a SELECT statement |
| | SELECT INTO | After you issue a SELECT INTO statement |
| | SET CONNECTION | After you issue a SET CONNECTION statement |
| | SET CURRENT APPLICATION ENCODING SCHEME | After you issue a SET CURRENT APPLICATION ENCODING SCHEME statement |
| | SET CURRENT DEGREE | After you issue a SET CURRENT DEGREE statement |
| | SET CURRENT LOCALE LC_CTYPE | After you issue a SET CURRENT LOCALE LC_CTYPE statement |
| | SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION | After you issue a SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION statement |
| | SET CURRENT OPTIMIZATION HINT | After you issue a SET CURRENT OPTIMIZATION HINT statement |

Table 5. INCORROUT modifier keywords (continued)

| Primary keyword | Secondary keywords | Problem occurrence |
|-----------------|--------------------------|--|
| SQL continued | SET CURRENT PACKAGE PATH | After you issue a SET CURRENT PACKAGE PATH statement |
| | SET CURRENT PACKAGESET | After you issue a SET CURRENT PACKAGESET statement |
| | SET CURRENT PRECISION | After you issue a SET CURRENT PRECISION statement |
| | SET CURRENT REFRESH AGE | After you issue a SET CURRENT REFRESH AGE statement |
| | SET CURRENT RULES | After you issue a SET CURRENT RULES statement |
| | SET CURRENT SQLID | After you issue a SET CURRENT SQLID statement |
| | SET host variable | After you issue a SET host variable statement |
| | SET transition variable | After you issue a SET transition variable statement |
| | UPDATE | After you issue an UPDATE statement |
| | VALUES | After you issue a VALUES statement |
| | VALUES INTO | After you issue a VALUES INTO statement |
| | WHENEVER | After you issue a WHENEVER statement |
| | STAND-ALONE | |
| CHANGE LOG | | While you use the Change Log Inventory Utility |
| LOG READ | | While you use the Log Read utility |
| TSOATTACH | PRINT LOGMAP | While you use the Print Log Map utility |
| | | With TSO/Batch attachment facility |
| | BIND | During BIND subcommand processing |
| | DB2 COMMAND | During Db2 command processing |
| | DCLGEN | During DCLGEN subcommand processing |
| | FREE | During FREE subcommand processing |
| | REBIND | During REBIND subcommand processing |
| | RUN | While you run an application program |
| | SPUFI | While you invoke SPUFI as a panel request |
| | TSOCOMMAND | During TSO command processing |

Table 5. INCORROUT modifier keywords (continued)

| Primary keyword | Secondary keywords | Problem occurrence |
|-----------------|--------------------|--|
| UTILITY | | While a utility was processing |
| | BACKUP | While you run BACKUP SYSTEM utility |
| | CHECK | While you run CHECK utility |
| | COPY | While you run COPY utility |
| | COPYTOCOPY | While you run COPYTOCOPY utility |
| | DIAGNOSE | While you run DIAGNOSE utility |
| | INDEXBUILD | While you run REORG or LOAD (BUILD phase) |
| | LOAD | While you run LOAD utility |
| | MERGECOPY | While you run MERGECOPY utility |
| | MODIFY | While you run MODIFY utility |
| | REBUILD | While you run REBUILD INDEX utility |
| | RECOVER | While you run RECOVER utility |
| | RELOAD | While you run REORG utility (RELOAD phase) |
| | REORG | While you run REORG utility |
| | REPAIR | While you run REPAIR utility |
| | REPORT | While you run REPORT utility |
| | RESTORE | While you run RESTORE SYSTEM utility |
| | RUNSTATS | While you run RUNSTATS utility |
| | QUIESCE | While you run QUIESCE utility |
| | STOSPACE | While you run STOSPACE utility |
| | UNLOAD | While you run REORG (UNLOAD phase) or UNLOAD utility |

If Db2 rolls back to the wrong savepoint: or if you encounter problems when you roll back to a savepoint:

1. Collect log records for the pertinent transaction.
2. Verify that the savepoint you intended was not accidentally replaced by another savepoint of the same name. (Savepoint log records have a record type of X'2200' and subtype X'0014'.)

If Db2 rolls back to the wrong savepoint, the data is probably incorrect. In that case, you must recover to a prior point in time.

3. If you suspect a Db2 error, then contact the IBM Support Center and report the symptoms and supply the pertinent log records.

SQLCODE modifier keyword

Use the SQLCODE modifier keyword when an incorrect output problem occurred in response to an SQL statement, the SQLCODE was not issued because of a user error, and the SQLCA is available.

Procedure

To use the SQLCODE modifier keyword:

1. Locate the SQLCA.

[“Finding the SQLCA” on page 217](#) describes how to locate the SQLCA in a dump.

An example appears in “Finding the SQLCA” on page 217.

2. Locate the three-digit decimal SQL code that follows the label SQLCODE. Find any variables in the SQLCODE message, such as the name of an unavailable resource.
3. Find any additional relevant information that can be present. The format and contents of the SQLCA varies from problem to problem.

Check for the following information:

- Any CSECT names that follow the label SQLERRP.
 - Any meaningful 4-byte hexadecimal subcodes that follow the label SQLERRD. There are six such subcode fields. Find only those fields whose values do not equal X'00000000' or X'FFFFFFFF'.
4. Build a keyword that is based on the relevant data that was collected. Include the SQL return code and any pertinent CSECT names, message variables, and subcodes. The SQL return code should appear similar to SQLCODExxx, with xxx indicating the three-digit code with the minus sign omitted. Add this information to the keyword string and turn to [Chapter 2, “Searching for known problems and solutions for Db2 for z/OS,”](#) on page 3.

```
(free format):
5740XYR00 R131 INCORROUT SQL CREATE SQLCODE901
DSNHPA RC0000C901
(structured format):
PIDS/5740XYR00 LVLS/131 INCORROUT
PCSS/SQL PCSS/CREATE FLDS/SQLCODE901
RIDS/DSNHPA PRCS/0000C901
```

Message modifier keyword

Use the message modifier keyword when the type-of-failure keyword is MSGx and the message you receive contains variables.

Procedure

To use the message modifier keyword:

1. If the message contains return or reason codes, usually indicated by "RC", build a keyword similar to RCx. Replace x with the return or reason code, and add the following information to the string. Turn to [Chapter 2, “Searching for known problems and solutions for Db2 for z/OS,”](#) on page 3.

```
(free format):
5740XYR00 R131 MSGDSNM002I RCE
(structured format):
PIDS/5740XYR00 LVLS/131 MS/DSNM002I
PRCS/0000000E
```

2. If the message contains any other types of variables, append them to the keyword string. Turn to [Chapter 2, “Searching for known problems and solutions for Db2 for z/OS,”](#) on page 3.

```
(free format):
5740XYR00 R131 MSGDSNJ104I OPEN
(structured format):
PIDS/5740XYR00 LVLS/131 MS/DSNJ104I
MS/OPEN
```

Dependency keywords

Program-dependent or machine-dependent symptom keywords qualify the problem description beyond the normal set that is given so far.

For example, a failure can occur only in a CICS environment. The dependency keyword, D/CICS, when added to the set of keywords, can help reduce the number of problem descriptions that need to be examined.

Generally, a dependency keyword should be used only at the direction of a person from IBM Support.

**Keyword
Environment**

D/BATCH
Batch

D/CICS
CICS

D/IMS
IMS

D/ISPF
Interactive System Productivity Facility

D/QMF
Db2 Query Management Facility (QMF)

D/TSO
TSO

D/VTAM
VTAM

D/ZOS
z/OS

Techniques for varying the search

If you cannot find an appropriate PTF or APAR after searching the IBM Support site, you might try varying the search argument.

Use the following guidelines to vary your search:

- If you used a complete set of keywords, and you were unable to find any problem descriptions to examine, drop one or more of these keywords and try again:
 - [“Dependency keywords” on page 49](#)
 - Release level keyword
 - Load Module modifier keyword
 - Recovery routine modifier keyword
 - SQLCODE modifier keyword
 - Performance modifier keyword
 - CSECT keyword
- If you tried to search with an incomplete set of keywords and found too many problem descriptions to examine, add keywords to narrow the search. For example, for storage manager abends (which begin with a reason code of X'00E2'), use the CSECT name that is recorded in the VRA as a means to narrow or vary the search.
- If you tried to search with a complete set of keywords and found too many matching descriptions, and if you received a 4-byte Db2 abend reason code, you might be able to make the set of keywords more precise. Look up the 4-byte abend reason code in [Db2 reason codes \(Db2 Codes\)](#).
- If the type-of-failure keyword is WAIT, LOOP, or PERFM, and a matching problem description is not found, try to replace whichever type-of-failure keyword that was used with one of the other two listed here. Sometimes a problem that appears to be a performance problem might actually be a wait or loop; likewise, a problem that seems to be a wait or a loop might actually be recorded as a performance problem.
- If the type-of-failure keyword is MSGx and you received more than one message near the time of the problem, replace the number of the message in the keyword with other related message numbers.
- If the type-of-failure keyword is MSGx, PERFM, or INCORROUT, and if the problem occurred immediately after you performed an action that a Db2 publication described, then the problem can

be recorded as a DOC type of failure. In this case, try searching with DOC as the type-of-failure keyword, rather than with MSGx, PERFM, or INCORROUT.

Related information

[IBM Support website](#)

Getting fixes

A product fix might be available to resolve your problem.

About this task

Instructions in this topic assume that you have a PTF or APAR number. If you do not know a PTF or APAR number that is likely to resolve your problem, see [Chapter 2, “Searching for known problems and solutions for Db2 for z/OS,”](#) on page 3.

Procedure

- If you have a PTF or APAR number but are unsure of whether it is applicable to your situation, search for the number on the [IBM Support website](#). Read the description of the fix and decide if it can resolve your problem. If so, obtain the fix as described below.
- Obtain fixes at the [IBM Shopz website](#) by searching for the PTF or APAR number. Follow the instructions to obtain the fix.
- Subscribe to receive weekly email notifications about fixes and other IBM Support information so that you learn of important fixes before you experience the problems with which they are associated. See [Using the IBM My Notifications subscription service](#) for more information.

Chapter 3. Recovering from different Db2 for z/OS problems

You can troubleshoot and recover from many Db2 problems on your own by using the provided recovery procedures.

Recovering from IRLM failure

You can recover from an IRLM failure, regardless of whether the failure results in a wait, loop, or abend.

Symptoms

The IRLM waits, loops, or abends. The following message might be issued:

```
DXR122E irlmm ABEND UNDER IRLM TCB/SRB IN MODULE xxxxxxxx  
ABEND CODE zzzz
```

Environment

If the IRLM abends, Db2 terminates. If the IRLM waits or loops, the IRLM terminates, and Db2 terminates automatically.

Resolving the problem

Operator response:

1. Start the IRLM if you did not set it for automatic start when you installed Db2.
2. Start Db2.
3. Connect IMS to Db2, by issuing the following command, where *ssid* is the subsystem ID:

```
/START SUBSYS ssid
```

4. Connect CICS to Db2 by issuing the following command:

```
DSNC STRT
```

Related tasks

[Connecting from CICS \(Db2 Administration Guide\)](#)

[Starting Db2 \(Db2 Administration Guide\)](#)

[Starting the IRLM \(Db2 Administration Guide\)](#)

Recovering from z/OS or power failure

You can recover from a situation in which z/OS or your processor power fails.

Symptoms

No processing is occurring.

Resolving the problem

Operator response:

- If the power failure or z/OS failure has occurred:
 1. IPL z/OS, and initialize the job entry subsystem (JES).
 2. If you normally run VTAM with Db2, start VTAM at this point.

3. Start the IRLM if it was not set for automatic start during Db2 installation.
 4. Start Db2.
 5. Use the **RECOVER POSTPONED** command if postponed-abort units of recovery were reported after restarting Db2, and if the AUTO or LIGHTAUTO option of the LIMIT BACKOUT field on installation panel DSNTIPL was not specified. If the LIGHTAUTO option is specified, postponed-abort units of recovery are processed during the next normal Db2 restart.
 6. Restart IMS or CICS.
 - IMS automatically connects and resynchronizes when it is restarted.
 - CICS automatically connects to Db2 if the CICS PLT contains an entry for the attachment facility module DSNTCCOM0. Alternatively, use the command **DSNC STRT** to connect the CICS attachment facility to Db2.
- If you know that a power failure is imminent, issue a **STOP DB2 MODE(FORCE)** command to allow Db2 to stop cleanly before the power is interrupted. If Db2 is unable to stop completely before the power failure, the situation is no worse than if Db2 were still operational.

Related concepts

[Connections to the IMS control region \(Db2 Administration Guide\)](#)

Related tasks

[Connecting from CICS \(Db2 Administration Guide\)](#)

[Starting Db2 \(Db2 Administration Guide\)](#)

[Starting the IRLM \(Db2 Administration Guide\)](#)

Recovering from disk failure

When a disk hardware failure occurs and an entire unit is lost, you can recover from this situation.

Symptoms

No I/O activity occurs for the affected disk address. Databases and tables that reside on the affected unit are unavailable.

Resolving the problem

Operator response:

1. Assure that no incomplete I/O requests exist for the failing device. One way to do this is to force the volume offline by issuing the following z/OS command, where xxx is the unit address:

```
VARY xxx, OFFLINE, FORCE
```

To check disk status, issue the following command:

```
D U, DASD, ONLINE
```

The following console message is displayed after you force a volume offline:

```
UNIT  TYPE  STATUS  VOLSER  VOLSTATE
4B1   3390  O-BOX  XTRA02  PRIV/RSDNT
```

The disk unit is now available for service.

If you previously set the I/O timing interval for the device class, the I/O timing facility terminates all requests that are incomplete at the end of the specified time interval, and you can proceed to the next step without varying the volume offline. You can set the I/O timing interval either through the IEIOSxx z/OS parameter library member or by issuing the following z/OS command:

```
SETIOS MIH, DEV=devnum, IOTIMING=mm:ss.
```

- Issue (or request that an authorized operator issue) the following Db2 command to stop all databases and table spaces that reside on the affected volume:

```
-STOP DATABASE(database-name) SPACENAM(space-name)
```

If the disk unit must be disconnected for repair, stop all databases and table spaces on all volumes in the disk unit.

- Select a spare disk pack, and use ICKDSF to initialize from scratch a disk unit with a different unit address (*yyy*) and the same volume serial number (VOLSER).

```
// Job
//ICKDSF EXEC PGM=ICKDSF
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
REVAL UNITADDRESS(yyy) VERIFY(volser)
```

If you initialize a 3380 or 3390 volume, use **REVAL** with the VERIFY parameter to ensure that you initialize the intended volume, or to revalidate the home address of the volume and record 0. Alternatively, use ISMF to initialize the disk unit.

- Issue the following z/OS console command, where *yyy* is the new unit address:

```
VARY yyy, ONLINE
```

- To check disk status, issue the following command:

```
D U, DASD, ONLINE
```

The following console message is displayed:

```
UNIT  TYPE  STATUS  VOLSER  VOLSTATE
7D4   3390   0       XTRA02  PRIV/RSDNT
```

- Delete all table spaces (VSAM linear data sets) from the ICF catalog by issuing the following access method services command for each one of them, where *y* is either I or J:

```
DELETE catnam.DSNDBC.dbname.tsname.y0001.Annn CLUSTER NOSCRATCH
```

where *nnn* is the data set or partition number, left padded by 0 (zero).

- For user-managed table spaces, define the VSAM cluster and data components for the new volume by issuing the access method services **DEFINE CLUSTER** command with the same data set name as in the previous step, in the following format:

```
catnam.DSNDBx.dbname.tsname.y0001.znnn
```

The *y* is I or J, the *x* is C (for VSAM clusters) or D (for VSAM data components), and *znnn* is the data set or partition number, left padded by 0 (zero). For more information, see [Data set naming conventions](#) (Db2 Administration Guide).

- For a user-defined table space, define the new data set before an attempt to recover it. You can recover table spaces that are defined in storage groups without prior definition.
- Issue the following Db2 command to start all the appropriate databases and table spaces that were previously stopped:

```
-START DATABASE(database-name) SPACENAM(space-name)
```

- Recover the table spaces by using the Db2 RECOVER utility.

Related reference

[RECOVER \(Db2 Utilities\)](#)

Related information

[DFSMS Access Method Services Commands](#)

[Device Support Facilities \(ICKDSF\) Device Support Facilities \(ICKDSF\) User's Guide and Reference](#)

Recovering from application errors

You can recover from a problem in which an application program placed a logically incorrect value in a table.

Symptoms

Unexpected data is returned from an SQL SELECT statement, even though the SQLCODE that is associated with the statement is 0.

Causes

An SQLCODE of 0 indicates that Db2 and SQL did not cause the problem, so the cause of the incorrect data in the table is the application.

Resolving the problem

System programmer response: You might be able to use the Db2 RECOVER utility with the TOLOGPOINT option to restore the database to a point before the error occurred. However, in many circumstances you must manually back out the changes that were introduced by the application. Among those circumstances are:

- Other applications changed the database after the error occurred. If you recover the table spaces that were modified by the bad application, all subsequent changes that were made by the other applications are lost.
- Db2 checkpoints were taken after the error occurred. In this case, you can use RECOVER TOLOGPOINT to restore the data up to the last checkpoint before the error occurred. However, all subsequent changes to the database are lost.

If you have a situation for which using RECOVER TOLOGPOINT is appropriate, you can use one of the following procedures as a basis for backing out the incorrect changes that were made by the application. The procedure that you use depends on whether you have established a quiesce point.

Backing out incorrect application changes (with a quiesce point)

If you have an established quiesce point, you can back out incorrect changes that your application made.

Procedure

To back out the incorrect changes:

1. Run the REPORT utility twice, once using the RECOVERY option and once using the TABLESPACESET option. On each run, specify the table space that contains the inaccurate data.
If you want to recover to the last quiesce point, specify the option CURRENT when running REPORT RECOVERY.
2. Examine the REPORT output to determine the RBA of the quiesce point.
3. Run RECOVER TOLOGPOINT with the RBA that you found, specifying the names of all related table spaces.

Results

Recovering all related table spaces to the same quiesce point prevents violations of referential constraints.

Backing out incorrect application changes (without a quiesce point)

Even if you do not have an established quiesce point, you can back out incorrect changes that your application made. Be aware, however, that if you use this procedure, you lose any updates to the database that occurred after the last checkpoint and before the application error occurred.

Procedure

To back out the incorrect changes:

1. Run the DSN1LOGP stand-alone utility on the log scope that is available at Db2 restart, using the SUMMARY(ONLY) option.
2. Determine the RBA of the most recent checkpoint before the first bad update occurred, from one of the following sources:
 - Message DSNR003I on the operator's console, which looks similar to this message:

```
DSNR003I RESTART      . . . . . PRIOR CHECKPOINT RBA=000007425468
```

The required RBA in this example is X'7425468'.

This technique works only if no checkpoints have been taken since the application introduced the bad updates.
 - Output from the print log map utility. You must know the time that the first bad update occurred. Find the last BEGIN CHECKPOINT RBA before that time.
3. Run DSN1LOGP again, using SUMMARY(ONLY), and specify the checkpoint RBA as the value of RBASTART.

The output lists the work in the recovery log, including information about the most recent complete checkpoint, a summary of all processing, and an identification of the databases that are affected by each active user.
4. Find the unit of recovery in which the error was made.

One of the messages in the output (identified as DSN1151I or DSN1162I) describes the unit of recovery in which the error was made. To find the unit of recovery, use your knowledge of the time that the program was run (START DATE= and TIME=), the connection ID (CONNID=), authorization ID (AUTHID=), and plan name (PLAN=). In that message, find the starting RBA as the value of START=.
5. Run the Db2 RECOVER utility with the TOLOGPOINT option, and specify the starting RBA that you found in the previous step.
6. Recover any related table spaces or indexes to the same point in time.

Related concepts

[DSN1LOGP summary report \(Db2 Administration Guide\)](#)

Related reference

[DSN1LOGP \(Db2 Utilities\)](#)

Recovering from IMS-related failures

When you work in a Db2-IMS environment and problems occur, you can recover from those problems.

Symptoms

Problems that occur in a Db2-IMS environment can result in a variety of symptoms:

- An IMS wait, loop, or abend is accompanied by a Db2 message that goes to the IMS console. This symptom indicates an IMS control region failure.
- When IMS connects to Db2, Db2 detects one or more units of recovery that are indoubt.
- When IMS connects to Db2, Db2 detects that it has committed one or more units of recovery that IMS indicates should be rolled back.

- Messages are issued to the IMS master terminal, to the logical terminal, or both to indicate that some sort of IMS or Db2 abend has occurred.

Environment

Db2 can be used in an XRF (Extended Recovery Facility) recovery environment with IMS.

To resolve IMS-related problems, follow the appropriate procedure.

Related concepts

[Plans for extended recovery facility toleration \(Db2 Administration Guide\)](#)

Recovering from IMS control region failure

You can recover from a problem in which the IMS control region fails.

Symptoms

- IMS waits, loops, or abends.
- Db2 attempts to send the following message to the IMS master terminal during an abend:

```
DSNM002 IMS/TM xxxx DISCONNECTED FROM SUBSYSTEM yyyy RC=RC
```

This message cannot be sent if the failure prevents messages from being displayed.
- Db2 does not send any messages for this problem to the z/OS console.

Environment

- Db2 detects that IMS has failed.
- Db2 either backs out or commits work that is in process.
- Db2 saves indoubt units of recovery, which need to be resolved at reconnection time.

Resolving the problem

Operator response: Use normal IMS restart procedures, which include starting IMS by issuing the z/OS **START IMS** command. The following results occur:

1. All DL/I and Db2 updates that have not been committed are backed out.
2. IMS is automatically reconnected to Db2.
3. IMS passes the recovery information for each entry to Db2 through the IMS attachment facility. (IMS indicates whether to commit or roll back.)
4. Db2 resolves the entries according to IMS instructions.

Recovering from IMS indoubt units of recovery

When IMS connects to Db2, and Db2 has indoubt units of recovery that have not been resolved, these units of recovery need to be resolved.

Symptoms

If Db2 has indoubt units of recovery that IMS did not resolve, the following message is issued at the IMS master terminal, where *xxxx* is the subsystem identifier:

```
DSNM004I RESOLVE INDOUBT ENTRY(S) ARE OUTSTANDING FOR SUBSYSTEM xxxx
```

Causes

When this message is issued, IMS was either cold started, or it was started with an incomplete log tape. Message DSNM004I might also be issued if Db2 or IMS abnormally terminated in response to a software error or other subsystem failure.

Environment

- The connection remains active.
- IMS applications can still access Db2 databases.
- Some Db2 resources remain locked out.

If the indoubt thread is not resolved, the IMS message queues might start to back up. If the IMS queues fill to capacity, IMS terminates. Be aware of this potential difficulty, and monitor IMS until the indoubt units of work are fully resolved.

Resolving the problem

System programmer response:

1. Force the IMS log closed by using the **/DBR FEOV** command.
2. Archive the IMS log.
3. Issue the command **DFSERA10** to print the records from the previous IMS log tape for the last transaction that was processed in each dependent region. Record the PSB and the commit status from the X'37' log that contains the recovery ID.
4. Run the DL/I batch job to back out each PSB that is involved that has not reached a commit point. The process might be time-consuming because transactions are still being processed. This process might also lock a number of records, which could affect the rest of the processing and the rest of the message queues.
5. Enter the Db2 command **DISPLAY THREAD** (*imsid*) TYPE (INDOUBT).
6. Compare the NIDs (IMSID + OASN in hexadecimal) that are displayed in the **DISPLAY THREAD** output with the OASNs (4 bytes decimal) as shown in the DFSERA10 output. Decide whether to commit or roll back.
7. Use **DFSERA10** to print the X'5501FE' records from the current IMS log tape. Every unit of recovery that undergoes indoubt resolution processing is recorded; each record with an 'IDBT' code is still indoubt. Note the correlation ID and the recovery ID, for use during the next step.
8. **GUPI** Enter the following Db2 command, choosing to commit or roll back, and specify the correlation ID:

```
-RECOVER INDOUBT (imsid) ACTION(COMMIT|ABORT) NID (nid)
```

If the command is rejected because of associated network IDs, use the same command again, substituting the recovery ID for the network ID.

GUPI

Related concepts

[Duplicate IMS correlation IDs \(Db2 Administration Guide\)](#)

Recovering IMS indoubt units of work that need to be rolled back

When units of recovery between IMS and Db2 are indoubt at restart time, Db2 and IMS sometimes handle the indoubt units of recovery differently. When this situation happens, you might need to roll back the changes.

Symptoms

The following messages are issued after a Db2 restart:

```
DSNM005I  IMS/TM RESOLVE INDOUBT PROTOCOL PROBLEM WITH SUBSYSTEM xxxx  
DFS3602I xxxx SUBSYSTEM RESOLVE-INDOUBT FAILURE,RC=yyyy
```

Causes

The reason that these messages are issued is that indoubt units of work exist for a Db2-IMS application, and the way that Db2 and IMS handle these units of work differs.

At restart time, Db2 attempts to resolve any units of work that are indoubt. Db2 might commit some units and roll back others. Db2 records the actions that it takes for the indoubt units of work. At the next connect time, Db2 verifies that the actions that it took are consistent with the IMS decisions. If the Db2 **RECOVER INDOUBT** command is issued prior to an IMS attempt to reconnect, Db2 might decide to commit the indoubt units of recovery, whereas IMS might decide to roll back the units of recovery. This inconsistency results in the DSNM005I message being issued. Because Db2 tells IMS to retain the inconsistent entries, the DFS3602I message is issued when the attempt to resolve the indoubt units of recovery ends.

Environment

- The connection between Db2 and IMS remains active.
- Db2 and IMS continue processing.
- No Db2 locks are held.
- No units of work are in an incomplete state.

Resolving the problem

System programmer response: Do not use the Db2 **RECOVER INDOUBT** command. The problem is that Db2 was not indoubt but should have been. Database updates have probably been committed on one side (IMS or Db2) and rolled back on the other side.

1. Enter the IMS command **/DISPLAY OASN SUBSYS DB2** to display the IMS list of units of recovery that need to be resolved. This command generates the list of OASNs in a decimal format, not in a hexadecimal format.
2. Issue the IMS command **/CHANGE SUBSYS DB2 RESET** to reset all the entries in the list. (No entries are passed to Db2.)
3. Use **DFSERA10** to print the log records that were recorded at the time of failure and during restart. Look at the X'37', X'56', and X'5501FE' records at reconnect time. Notify IBM Support about the problem.
4. Determine what the inconsistent unit of recovery was doing by examining the log information, and manually make the IMS and Db2 databases consistent.

Related concepts

[Duplicate IMS correlation IDs \(Db2 Administration Guide\)](#)

Recovering from IMS application failure

You can recover from a situation in which an IMS application abnormally terminates in a Db2 environment.

Symptoms

The following messages are issued at the IMS master terminal and at the LTERM that entered the transaction that is involved:

```
DFS555 - TRAN tttttttt ABEND (SYSIDssss);  
          MSG IN PROCESS: xxxx (up to 78 bytes of data) timestamp  
DFS555A - SUBSYSTEM xxxx OASN yyyyyyyyyyyyyyyyyy STATUS COMMIT|ABORT
```

Causes

The problem might be caused by a usage error in the application or by a Db2 problem.

Environment

- The failing unit of recovery is backed out by both DL/I and Db2.
- The connection between IMS and Db2 remains active.

Resolving the problem

Operator response:

- If you think that the problem was caused by a usage error, investigate and resolve the error.
- If you think that the problem is a Db2 problem, rather than a usage error, try to diagnose the problem using standard diagnostic procedures. You might need to contact IBM Support if you cannot resolve the problem yourself.

Related concepts

[Techniques for debugging programs in IMS \(Db2 Application programming and SQL\)](#)

Recovering from a Db2 failure in an IMS environment

When Db2 fails in a Db2-IMS environment, you can recover from this situation.

Symptoms

Db2 fails or is not running, and one of the following status situations exists:

- If you specified error option Q, the program terminates with a U3051 user abend completion code.
- If you specified error option A, the program terminates with a U3047 user abend completion code.

In either of these situations, the IMS master terminal receives IMS message DFS554, and the terminal that is involved in the problem receives IMS message DFS555.

Resolving the problem

Operator response:

1. Restart Db2.
2. Follow the standard IMS procedures for handling application abends.

Recovering from CICS-related failure

When you work in a Db2-CICS environment and problems occur, you can recover from those problems.

Symptoms

Problems that occur in a Db2-CICS environment can result in a variety of symptoms, such as:

- Messages that indicate an abend in CICS or the CICS attachment facility
- A CICS wait or a loop
- Indoubt units of recovery between CICS and Db2

Environment

Db2 can be used in an XRF (Extended Recovery Facility) recovery environment with CICS.

Resolving the problem

To resolve CICS-related problems, follow the appropriate procedure.

Related concepts

[Plans for extended recovery facility toleration \(Db2 Administration Guide\)](#)

Recovering from CICS application failures

You can recover from a CICS application abend in a Db2 environment.

Symptoms

The following message is issued at the user's terminal:

```
DFH2206 TRANSACTION tranid ABEND abcode BACKOUT SUCCESSFUL
```

In this message, *tranid* represents the transaction that abnormally terminated, and *abcode* represents the specific abend code.

Environment

- The failing unit of recovery is backed out in both CICS and Db2.
- The connection between CICS and Db2 remains active.

Resolving the problem

Operator response: Investigate the abend by reading about the abend code.

- For an AEY9 abend, start the CICS attachment facility.
- For an ASP7 abend, determine why the CICS **SYNCPPOINT** was unsuccessful.
- For other abends, follow appropriate diagnostic procedures.

Related concepts

[CICS Transaction Server for z/OS troubleshooting and support](#)

Recovering Db2 when CICS is not operational

You can recover Db2 from a situation in which CICS is not operational.

Symptoms

Any of the following symptoms might occur:

- CICS waits or loops.
- CICS abends, as indicated by messages or dump output.

Environment

Db2 performs each of the following actions:

- Detects the CICS failure.
- Backs out inflight work.
- Saves indoubt units of recovery that need to be resolved when CICS is reconnected.

Diagnosing the problem

If you think that CICS is in a wait or loop situation, find the origin of the wait or loop. The origin might be in CICS, in CICS applications, or in the CICS attachment facility.

If you receive messages that indicate a CICS abend, examine the messages and dump output for more information.

If threads are connected to Db2 when CICS terminates, Db2 issues message DSN3201I. The message indicates that Db2 end-of-task (EOT) routines have cleaned up and disconnected any connected threads.

Resolving the problem

Operator response:

1. Correct the problem that caused CICS to terminate abnormally.
2. Do an emergency restart of CICS. The emergency restart performs each of the following actions:
 - Backs out inflight transactions that changed CICS resources
 - Remembers the transactions with access to Db2 that might be indoubt
3. Start the CICS attachment facility by entering the appropriate command for your release of CICS. The CICS attachment facility performs the following actions:
 - Initializes and reconnects to Db2
 - Requests information from Db2 about the indoubt units of recovery and passes the information to CICS
 - Allows CICS to resolve the indoubt units of recovery

Related tasks

[Connecting from CICS \(Db2 Administration Guide\)](#)

Related information

[Troubleshooting for CICS Db2 \(CICS Db2 Guide\)](#)

Recovering Db2 when the CICS attachment facility cannot connect to Db2

You can recover Db2 when the CICS attachment facility cannot connect to Db2.

Symptoms

Any of the possible symptoms can occur:

- CICS remains operational, but the CICS attachment facility abends.
- The CICS attachment facility issues a message that indicates the reason for the connection failure, or it requests a X'04E' dump.
- The reason code in the X'04E' dump indicates the reason for failure.
- CICS issues message DFH2206 that indicates that the CICS attachment facility has terminated abnormally with the DSNB abend code.
- CICS application programs that try to access Db2 while the CICS attachment facility is inactive are abnormally terminated. The code AEY9 is issued.

Environment

CICS backs out the abnormally terminated transaction and treats it like an application abend.

Resolving the problem

Operator response: Start the CICS attachment facility by entering the appropriate command for your release of CICS. After you start the CICS attachment facility, the following events occur:

1. The CICS attachment facility initializes and reconnects to Db2.
2. The CICS attachment facility requests information about the indoubt units of recovery and passes the information to CICS.
3. CICS resolves the indoubt units of recovery.

Recovering CICS indoubt units of recovery

When the CICS attachment facility abends, CICS and Db2 build lists of indoubt units of work, either dynamically or during restart, depending on the failing subsystem. If any units of recovery are indoubt at connect time, you can recover from this situation.

Symptoms

One of the following messages is sent to the user-named CICS destination that is specified for the MSGQUEUE(*name*) attribute in the RDO (resource definition online): DSN2001I, DSN2034I, DSN2035I, or DSN2036I.

Causes

For CICS, a Db2 unit of recovery might be indoubt if the forget entry (X'FD59') of the task-related installation exit routine is absent from the CICS system journal. The indoubt condition applies only to the Db2 unit of recovery in this case because CICS already committed or backed out any changes to its resources.

A Db2 unit of recovery is indoubt for Db2 if an End Phase 1 is present and the Begin Phase 2 is absent.

Environment

The following table summarizes the situations that can exist when CICS units of recovery are indoubt.

GUPI

Table 6. Situations that involve CICS abnormal indoubt units of recovery

| Message ID | Meaning |
|------------|---|
| DSN2001I | The named unit of recovery cannot be resolved by CICS because CICS was cold started. The CICS attachment facility continues the startup process. |
| DSN2034I | The named unit of recovery is not indoubt for Db2, but it is indoubt according to CICS log information. The reason is probably a CICS restart with the wrong tape. The problem might also be caused by a Db2 restart to a prior point in time. |
| DSN2035I | The named unit of recovery is indoubt for Db2, but it is not in the CICS indoubt list. This is probably due to an incorrect CICS restart. The CICS attachment facility continues the startup process and provides a transaction dump. The problem might also be caused by a Db2 restart to a prior point in time. |
| DSN2036I | CICS indicates rollback for the named unit of recovery, but Db2 has already committed the unit of recovery. The CICS attachment facility continues the startup process. |

GUPI

CICS retains details of indoubt units of recovery that were not resolved during connection startup. An entry is purged when it no longer shows up on the list that is presented by Db2 or, when the entry is present in the list, when Db2 resolves it.

Resolving the problem

System programmer response: If CICS cannot resolve one or more indoubt units of recovery, resolve them manually by using Db2 commands. Using the steps in this procedure is rarely necessary because it is required only where operational errors or software problems have prevented automatic resolution.

1. **GUPI** Obtain a list of the indoubt units of recovery from Db2 by issuing the following command:

```
-DISPLAY THREAD (connection-name) TYPE (INDOUBT)
```

Messages like these are then issued:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS - DSNV406I - INDOUBT THREADS - COORDINATOR
          STATUS      RESET URID      AUTHID
  coordinator_name status yes/no urid authid
DISPLAY INDOUBT REPORT COMPLETE DSN9022I - DSNVDT '-DISPLAY THREAD' NORMAL
COMPLETION
```

The *corr_id* (correlation ID) for CICS Transaction Server for z/OS 1.2 and subsequent releases of CICS consists of:

Bytes 1–4

Thread type: COMD, POOL, or ENTR

Bytes 5–8

Transaction ID

Bytes 9–12

Unique thread number

GUPI

Two threads can sometimes have the same correlation ID when the connection has been broken several times and the indoubt units of recovery have not been resolved. In this case, use the network ID (NID) instead of the correlation ID to uniquely identify indoubt units of recovery.

The network ID consists of the CICS connection name and a unique number that is provided by CICS at the time that the syncpoint log entries are written. This unique number is an 8-byte store clock value that is stored in records that are written to both the CICS system log and to the Db2 log at syncpoint processing time. This value is referred to in CICS as the *recovery token*.

2. Scan the CICS log for entries that are related to a particular unit of recovery. Look for a PREPARE record (JCRSTRID X'F959'), for the task-related installation where the recovery token field (JCSRMTKN) equals the value that is obtained from the network-ID. The network ID is supplied by Db2 in the **DISPLAY THREAD** command output.

You can find the CICS task number by locating the prepare log record in the CICS log for indoubt units of recovery. Using the CICS task number, you can locate all other entries on the log for this CICS task.

You can use the CICS journal print utility DFHJUP to scan the log.

3. Use the change log inventory utility (DSNJU003) to scan the Db2 log for entries that are related to a particular unit of recovery. Locate the End Phase 1 record with the required network ID. Then use the URID from this record to obtain the rest of the log records for this unit of recovery.

When scanning the Db2 log, note that the Db2 startup message DSNJ099I provides the start log RBA for this session.

4. If needed, do indoubt resolution in Db2. **GUPI** To invoke Db2 to take the recovery action for an indoubt unit of recovery, issue the Db2 **RECOVER INDOUBT** command, where the *correlation_id* is unique:

```
DSNC -RECOVER INDOUBT (connection-name)
      ACTION (COMMIT/ABORT)
      ID (correlation_id)
```

If the transaction is a pool thread, use the value of the correlation ID (*corr_id*) that is returned by **DISPLAY THREAD** for *thread#.tranid* in the **RECOVER INDOUBT** command. In this case, the first letter of the correlation ID is P. The transaction ID is in characters five through eight of the correlation ID.

If the transaction is assigned to a group (group is a result of using an entry thread), use *thread#.groupname* instead of *thread#.tranid*. In this case, the first letter of the correlation ID is a G, and the group name is in characters five through eight of the correlation ID. The *groupname* is the first transaction that is listed in a group.

Where the correlation ID is not unique, use the following command:

```
DSNC -RECOVER INDOUBT (connection-name)  
      ACTION (COMMIT|ABORT)  
      NID (network-id)
```

When two threads have the same correlation ID, use the NID keyword instead of the ID keyword. The NID value uniquely identifies the work unit.

To recover all threads that are associated with *connection-name*, omit the ID option.

The command results that are in either of the following messages indicate whether the thread is committed or rolled back:

```
DSNV414I - THREAD thread#.tranid COMMIT SCHEDULED  
DSNV414I - THREAD thread#.tranid ABORT SCHEDULED
```

When you resolve indoubt units of work, note that CICS and the CICS attachment facility are not aware of the commands to Db2 to commit or abort indoubt units of recovery because only Db2 resources are affected. However, CICS keeps details about the indoubt threads that could not be resolved by Db2. This information is purged either when the presented list is empty or when the list does not include a unit of recovery that CICS remembers.

Investigate any inconsistencies that you found in the preceding steps. 

Related reference

[DSNJU003 \(change log inventory\) \(Db2 Utilities\)](#)

Related information

[Reading log streams using batch jobs \(for example, DFHJUP\) \(CICS Transaction Server for z/OS\)](#)

Recovering from CICS attachment facility failure

You can recover Db2 when the CICS attachment facility abends or when a CICS attachment thread subtask abends.

Symptoms

The symptoms depend on whether the CICS attachment facility or one of its thread subtasks terminated:

- If the main CICS attachment facility subtask abends, an abend dump is requested. The contents of the dump indicate the cause of the abend. When the dump is issued, shutdown of the CICS attachment facility begins.
- If a thread subtask terminates abnormally, a X'04E' dump is issued, and the CICS application abends with a DSNB dump code. The X'04E' dump generally indicates the cause of the abend. The CICS attachment facility remains active.

Resolving the problem

Operator response: Correct the problem that caused the abend by analyzing the CICS formatted transaction dump or subtask SNAP dump. If the CICS attachment facility shuts down, use CICS commands to stop the execution of any CICS-Db2 applications.

Recovering from a QMF query failure

Receipt of a -805 SQL code in a Db2 for z/OS environment that includes Db2 Query Management Facility (QMF) might occur after you start QMF or issue a QMF command. If the Db2 subsystem was recently migrated to a new release, you might need to rerun certain QMF installation jobs.

Symptoms

One of the following QMF messages is issued:

- DSQ10202
- DSQ10205
- DSQ11205
- DSQ12105
- DSQ13005
- DSQ14152
- DSQ14153
- DSQ14154
- DSQ15805
- DSQ16805
- DSQ17805
- DSQ22889
- DSQ30805
- DSQ31805
- DSQ32029
- DSQ35805
- DSQ36805

Causes

Key QMF installation jobs were not run.

Environment

The Db2 for z/OS subsystem was migrated to a new release or migration to a new release of QMF.

Diagnosing the problem

User response:

- If your Db2 for z/OS subsystem was recently migrated to a new release, continue with "Resolving the problem."
- If your Db2 for z/OS subsystem was not recently migrated, see other possible causes of the DSQxxxxx messages in [Troubleshooting and correcting QMF bind problems associated with a -805 SQL code](#).

Resolving the problem

User response: Rerun QMF installation jobs as described in [Tasks to perform when you upgrade Db2 for z/OS after you install QMF](#).

Recovering from subsystem termination

You can recover Db2 after Db2 or an operator-issued cancel causes the subsystem to terminate.

Symptoms

When a Db2 subsystem terminates, the specific failure is identified in one or messages. The following messages might be issued at the z/OS console:

```
DSNV086E - DB2 ABNORMAL TERMINATION REASON=XXXXXXXXX
DSN3104I - DSN3EC00 -TERMINATION COMPLETE
DSN3100I - DSN3EC00 - SUBSYSTEM ssnm READY FOR -START COMMAND
```

The following message might be issued to the IMS master terminal:

```
DSNM002I  IMS/TM xxxx DISCONNECTED FROM SUBSYSTEM
          yyyy RC=rc
```

The following message might be issued to the CICS transient data error destination, which is defined in the RDO:

```
DSNC2025I  - THE ATTACHMENT FACILITY IS INACTIVE
```

Environment

- IMS and CICS continue.
- In-process IMS and CICS applications receive SQLCODE -923 (SQLSTATE '57015') when accessing Db2.

In most cases, if an IMS or CICS application program is running when a -923 SQLCODE is returned, an abend occurs. This is because the application program generally terminates when it receives a -923 SQLCODE. To terminate, some synchronization processing occurs (such as a commit). If Db2 is not operational when synchronization processing is attempted by an application program, the application program abends. In-process applications can abend with an abend code X'04F'.

- IMS applications that begin to run after subsystem termination begins are handled according to the error options.
 - For option R, SQL return code -923 is sent to the application, and IMS pseudo abends.
 - For option Q, the message is enqueued again, and the transaction abends.
 - For option A, the message is discarded, and the transaction abends.
- CICS applications that begin to run after subsystem termination begins are handled as follows:
 - If the CICS attachment facility has not terminated, the application receives a -923 SQLCODE.
 - If the CICS attachment facility has terminated, the application abends (code AEY9).

Resolving the problem

Operator response:

1. Restart Db2 by issuing the command **START DB2**.
2. For IMS environments, re-establish the IMS connection by issuing the IMS command **/START SUBSYS DB2**.
3. For CICS environments, re-establish the CICS connection by issuing the CICS attachment facility command **DSNC STRT**.

Recovering from temporary resource failure

Db2 sometimes experiences a temporary problem when it accesses log data sets. In this case, you need to recover from the situation so that processing can continue as normal.

Symptoms

Db2 issues messages for the access failure for each log data set. These messages provide information that is needed to resolve the access error. For example:

```
DSNJ104I  ( DSNJR206 RECEIVED ERROR STATUS 00000004
          FROM DSNPCLC FOR DSNAME=DSNC710.ARCHLOG1.A0000049

*DSNJ153E ( DSNJR006 CRITICAL LOG READ ERROR
          CONNECTION-ID = TEST0001
          CORRELATION-ID = CTHDCORID001
          LUWID = V71A.SYEC1DB2.B3943707629D=10
          REASON-CODE = 00D10345
```

Causes

Db2 might experience a problem when it attempts to allocate or open archive log data sets during the rollback of a long-running unit of recovery. These temporary failures can be caused by:

- A temporary problem with DFHSM recall
- A temporary problem with the tape subsystem
- Uncataloged archive logs
- Archive tape mount requests being canceled

Resolving the problem

User response: You can attempt to recover from temporary failures by issuing a positive reply (Y) to the following message:

```
*26 DSNJ154I ( DSNJR126 REPLY Y TO RETRY LOG READ REQUEST, N TO ABEND
```

If the problem persists, quiesce other work in the system before replying N, which terminates Db2.

Recovering from active log failures

A variety of active log failures might occur, but you can recover from them.

Symptoms

Most active log failures are accompanied by or preceded by error messages to inform you of out-of-space conditions, write or read I/O errors, or loss of dual active logging.

If you receive message DSNJ103I at startup time, the active log is experiencing dynamic allocation problems. If you receive message DSNJ104I, the active log is experiencing open-close problems. In either case, you should follow procedures in [“Recovering from BSDS or log failures during restart”](#) on page 80.

Recovering from being out of space in active logs

The available space in the active log is finite, so the active log might fill to capacity for one of several reasons. For example, delays in offloading and excessive logging can fill the active log. You can recover from out-of-space conditions in the active log.

Symptoms

The following warning message is issued when the last available active log data set is 5% full:

```
DSNJ110E - LAST COPYn ACTIVE LOG DATA SET IS nnn PERCENT FULL
```

The most recent DSNJ110E for COPYn is highlighted on the console until the situation is corrected. The message is non-scrollable.

The Db2 subsystem reissues the message after each additional 5% of the data set space is filled. Each time the message is issued, the offload process is started. IFCID trace record 0330 is also issued if statistics class 3 is active.

If the active log fills to capacity, after having switched to single logging, Db2 issues the following message, and an offload is started.

```
DSNJ111E - OUT OF SPACE IN ACTIVE LOG DATA SETS
```

The Db2 subsystem then halts processing until an offload is completed.

Causes

The active log is out of space.

Environment

An out-of-space condition on the active log has very serious consequences. Corrective action is required before Db2 can continue processing. When the active log becomes full, the Db2 subsystem cannot do any work that requires writing to the log until an offload is completed. Until that offload is completed, Db2 waits for an available active log data set before resuming normal Db2 processing. Normal shutdown, with either a **QUIESCE** or **FORCE** command, is not possible because the shutdown sequence requires log space to record system events that are related to shutdown (for example, checkpoint records).

Resolving the problem

Operator response:

1. Ensure that the offload is not waiting for a tape drive. If it is, mount a tape. Db2 then processes the offload task.
2. If you are uncertain about what is causing the problem, enter the following command:

```
-ARCHIVE LOG CANCEL OFFLOAD
```

This command causes Db2 to restart the offload task. Issuing this command might solve the problem.

3. If issuing this command does not solve the problem, determine and resolve the cause of the problem, and then reissue the command. If the problem cannot be resolved quickly, have the system programmer define additional active logs until you can resolve the problem.

System programmer response: Define additional active log data sets so that Db2 can continue its normal operation while the problem that is causing the offload failures is corrected. You can add active log data sets while Db2 remains online by issuing a **-SET LOG COMMAND** with the **NEWLOG** option, or you can use the following procedure for an offline change:

1. Use the z/OS command **CANCEL** to stop Db2.
2. Use the access method services **DEFINE** command to define new active log data sets.
3. Run utility DSNJLOGF to initialize the new active log data sets.
4. Define the new active log data sets in the BSDS by using the change log inventory utility (DSNJU003).
5. Restart Db2. Offload is started automatically during startup, and restart processing occurs.

Recommendation: To minimize the number of offloads that are taken per day in your installation, consider increasing the size of the active log data sets.

Related concepts

[Making changes for active logs \(Db2 Utilities\)](#)

Related tasks

[Adding an active log data set to the active log inventory with the SET LOG command \(Db2 Administration Guide\)](#)

Related reference

[-SET LOG command \(Db2\) \(Db2 Commands\)](#)

[DSNJLOGF \(preformat active log\) \(Db2 Utilities\)](#)

[DSNJU003 \(change log inventory\) \(Db2 Utilities\)](#)

Recovering from a write I/O error on an active log data set

You can recover from a situation in which a write error occurs on an active log data set.

Symptoms

The following message is issued:

```
DSNJ105I - csect-name LOG WRITE ERROR DSNAME=..., LOGRBA=...,  
          ERROR STATUS= ccccfss
```

Causes

Although this problem can be caused by several problems, one possible cause is a CATUPDT failure.

Environment

When a write error occurs on an active log data set, the following characteristics apply:

- Db2 marks the failing Db2 log data set TRUNCATED in the BSDS.
- Db2 goes on to the next available data set.
- If dual active logging is used, Db2 truncates the other copy at the same point.
- The data in the truncated data set is offloaded later, as usual.
- The data set is not stopped; it is reused on the next cycle. However, if a DSNJ104 message indicates a CATUPDT failure, the data set is marked STOPPED.

Resolving the problem

System programmer response: If the DSNJ104 message indicates a CATUPDT failure, use access method services and the change log inventory utility (DSNJU003) to add a replacement data set. In this case, you need to stop Db2. The timing of when you should take this action depends on how widespread the problem is.

- If the additional problem is localized and does not affect your ability to recover from any other problems, you can wait until the earliest convenient time.
- If the problem is widespread (perhaps affecting an entire set of active log data sets), stop Db2 after the next offload.

Related reference

[DSNJU003 \(change log inventory\) \(Db2 Utilities\)](#)

Recovering from a loss of dual active logging

If you use dual active logs, which is generally recommended, and one of the active log fails, Db2 reverts to use of a single active log. You can recover from this situation and return to dual active-log mode.

Symptoms

The following message is issued:

```
DSNJ004I - ACTIVE LOG COPY n INACTIVE, LOG IN SINGLE MODE,  
          ENDRBA=...
```

Causes

This problem occurs when Db2 completes one active log data set and then finds that the subsequent copy (COPY *n*) data sets have not been offloaded and are marked STOPPED.

Environment

Db2 continues in single mode until offloading completes and then returns to dual mode. If the data set is marked STOPPED, however, intervention is required.

Resolving the problem

System programmer response:

1. Verify that offload is proceeding and is not waiting for a tape mount. You might need to run the Db2 print log map utility (DSNJU004) to determine the status of all data sets.
2. If any data sets are marked STOPPED, use IDCAMS to delete the data sets, and then re-add them by using the Db2 change log inventory utility (DSNJU003).

Related reference

[DSNJU003 \(change log inventory\) \(Db2 Utilities\)](#)

Recovering from I/O errors while reading the active log

You can recover from situations in which an I/O error occurs when Db2 is reading the active log.

Symptoms

The following message is issued:

```
DSNJ106I - LOG READ ERROR DSNAME=..., LOGRBA=...,  
          ERROR STATUS=ccccffss
```

Environment

- If the error occurs during offload, offload tries to identify the RBA range from a second copy of the active log.
 - If no second copy of the active log exists, the data set is stopped.
 - If the second copy of the active log also has an error, only the original data set that triggered the offload is stopped. Then the archive log data set is terminated, leaving a discontinuity in the archived log RBA range.
 - The following message is issued:

```
DSNJ124I - OFFLOAD OF ACTIVE LOG SUSPENDED FROM RBA xxxxxx  
          TO RBA xxxxxx DUE TO I/O ERROR
```
 - If the second copy of the active log is satisfactory, the first copy is not stopped.
- If the error occurs during recovery, Db2 provides data from specific log RBAs that are requested from another copy or archive. If this is unsuccessful, recovery fails and the transaction cannot complete, but no log data sets are stopped. However, the table space that is being recovered is not accessible.

Resolving the problem

System programmer response:

- If the problem occurred during offload, determine which databases are affected by the active log problem, and take image copies of those. Then proceed with a new log data set.
- You can use the IDCAMS **REPRO** command to archive as much of the stopped active log data set as possible. Then run the change log inventory utility to notify the BSDS of the new archive log and its log RBA range. Repairing the active log does not solve the problem because offload does not go back to unload it.
- If the active log data set has been stopped, it is not used for logging. The data set is not deallocated; it is still used for reading.
- If the data set is not stopped, an active log data set should nevertheless be replaced if persistent errors occur. The operator is not told explicitly whether the data set has been stopped. To determine the status of the active log data set, run the print log map utility (DSNJU004).
- If you need to replace the data set:
 1. Ensure that the data is saved.

If you have dual active logs, the data is saved on the other active log, which becomes your new data set. Skip to step [“4”](#) on page 73.

If you are not using dual active logs, take the following steps to determine whether the data set with the error has been offloaded:

- a. Use the print log map utility (DSNJU004) to list information about the archive log data sets from the BSDS.

- b. Search the list for a data set whose RBA range includes the range of the data set with the error.
2. If the data set with the error has been offloaded (that is, if the value for high RBA offloaded in the print log map utility output is greater than the RBA range of the data set with the error), manually add a new archive log to the BSDS by using the change log inventory utility (DSNJU003). Use IDCAMS to define a new log that has the same LRECL and BLKSIZE values as defined in DSNZPxxx. You can use the access method services **REPRO** command to copy a data set with the error to the new archive log. If the archive log is not cataloged, Db2 can locate it from the UNIT and VOLSER values in the BSDS.
3. If an active log data set has been stopped, an RBA range has not been offloaded; copy from the data set with the error to a new data set. If additional I/O errors prevent you from copying the entire data set, a gap occurs in the log and restart might fail, although the data still exists and is not overlaid. If this occurs, see [“Recovering from BSDS or log failures during restart” on page 80](#).
4. Stop Db2, and use the change log inventory utility to update information in the BSDS about the data set with the error.
 - a. Use **DELETE** to remove information about the bad data set.
 - b. Use **NEWLOG** to name the new data set as the new active log data set and to give it the RBA range that was successfully copied.

The **DELETE** and **NEWLOG** operations can be performed by the same job step; put **DELETE** before **NEWLOG** in the SYSIN input data set. This step clears the stopped status, and Db2 eventually archives it.
 - c. Delete the data set that is in error by using access method services.
 - d. Redefine the data set so that you can write to it. Use the access method services **DEFINE** command to define the active log data sets. If you use dual logs and have a good copy of the log, use the **REPRO** command to copy the contents to the new data set that you just created. If you do not use dual logs, initialize the new data set by using the DSNJLOGF utility.

Related reference

[PRIMARY QUANTITY field \(PRIQTY subsystem parameter\) \(Db2 Installation and Migration\)](#)
[DSNJU004 \(print log map\) \(Db2 Utilities\)](#)

Recovering from archive log failures

You can recover from situations in which archive logging fails.

Symptoms

Archive log failures can result in a variety of Db2 and z/OS messages that identify problems with archive log data sets.

One specific symptom that might occur is message DSNJ104I, which indicates an open-close problem on the archive log.

Recovering from allocation problems with the archive log

You can recover from situations in which allocation problems occur for the archive log.

Symptoms

The following message is issued:

```
DSNJ103I - csect-name LOG ALLOCATION ERROR DSNAME=dsname,
ERROR STATUS=eeeeiii, SMS REASON CODE=sssssss
```

z/OS dynamic allocation provides the ERROR STATUS information. If the allocation is for offload processing, the following message is also issued:

DSNJ115I - OFFLOAD FAILED, COULD NOT ALLOCATE AN ARCHIVE DATA SET

Causes

Archive log allocation problems can occur when various Db2 operations fail; for example:

- The RECOVER utility executes and requires an archive log. If neither archive log can be found or used, recovery fails.
- The active log becomes full, and an offload is scheduled. Offload tries again the next time it is triggered. The active log does not wrap around; therefore, if no more active logs are available, the offload fails, but data is not lost.
- The input is needed for restart, which fails. If this is the situation that you are experiencing, see [Recovering from BSDS or log failures during restart \(Db2 Administration Guide\)](#)

Resolving the problem

Operator response: Check the allocation error code for the cause of the problem, and correct it. Ensure that drives are available, and run the recovery job again. If a DFSMSdftp ACS user-exit filter exists for an archive log data set, be careful because this can cause the Db2 subsystem to fail on a device allocation error when Db2 attempts to read the archive log data set.

Recovering from write I/O errors during archive log offload

You can recover from write I/O errors that occur during the offload of an archive log.

Symptoms

No specific Db2 message is issued for write I/O errors. Only a z/OS error recovery program message is issued.

If Db2 message DSNJ128I is issued, an abend in the offload task occurred, in which case you should follow the instructions for this message.

Environment

- Offload abandons that output data set (no entry in BSDS).
- Offload dynamically allocates a new archive and restarts offloading from the point at which it was previously triggered. For dual archiving, the second copy waits.
- If an error occurs on the new data set, these additional actions occur:
 - For dual archive mode, the following DSNJ114I message is generated, and the offload processing changes to single mode.

```
DSNJ114I - ERROR ON ARCHIVE DATA SET, OFFLOAD CONTINUING  
          WITH ONLY ONE ARCHIVE DATA SET BEING GENERATED
```
 - For single mode, the offload process abandons the output data set. Another attempt to offload this RBA range is made the next time offload is triggered.
 - The active log does not wrap around; if no more active logs are available, data is not lost.

Resolving the problem

Operator response: Ensure that offload activity is allocated on a drive and control unit that are operational.

Related information

[DSNJ128I \(Db2 Messages\)](#)

Recovering from read I/O errors on an archive data set during recovery

You can recover from read I/O errors that occur on an archive log during recovery.

Symptoms

No specific Db2 message is issued; only the z/OS error recovery program message is issued.

Environment

- If a second copy of the archive log exists, the second copy is allocated and used.
- If a second copy of the archive log does not exist, recovery fails.

Resolving the problem

Operator response: If you recover from tape, try recovering by using a different drive. If this approach does not work, contact the system programmer.

System programmer response: Recover to the last image copy or to the RBA of the last quiesce point.

Related reference

[RECOVER \(Db2 Utilities\)](#)

Recovering from insufficient disk space for offload processing

If offload processing terminates unexpectedly when Db2 is offloading the active log data sets to disk, you can recover from this situation.

Symptoms

Prior to the failure, z/OS issues abend message IEC030I, IEC031I, or IEC032I. Offload processing terminates unexpectedly. Db2 issues the following message:

```
DSNJ128I - LOG OFFLOAD TASK FAILED FOR ACTIVE LOG nnnnn
```

Additional z/OS abend messages might accompany message DSNJ128I.

Causes

The following situations can cause problems with insufficient disk space during Db2 offload processing:

- The size of the archive log data set is too small to contain the data from the active log data sets during offload processing. All secondary space allocations have been used.
- All available space on the disk volumes to which the archive data set is being written has been exhausted.
- The primary space allocation for the archive log data set (as specified in the load module for subsystem parameters) is too large to allocate to any available online disk device.

Environment

The archive data sets that are allocated to the offload task in which the error occurred are deallocated and deleted. Another attempt to offload the RBA range of the active log data sets is made the next time offload is invoked.

Resolving the problem

System programmer response: The actions that you take depend on what caused Db2 message DSNJ128I to be issued:

- If z/OS abend message IEC030I precedes Db2 message DSNJ128I, increase the primary or secondary allocations (or both) for the archive log data set in DSNZPxxx. Another option is to reduce the size of the

active log data set. Modifications to DSNZPxxx require that you stop and start Db2 for the changes to take effect. If the data that is to be offloaded is particularly large, you can mount another online storage volume or make one available to Db2.

- If z/OS abend message IEC032I precedes message DSNJ128I, make space available on the disk volumes, or make another online storage volume available for Db2. After you make additional space available, issue the Db2 command **ARCHIVE LOG CANCEL OFFLOAD**. Db2 then retries the offload.
- If z/OS abend message IEC032I precedes Db2 message DSNJ128I, make space available on the disk volumes, or make available another online storage volume for Db2. If this approach is not possible, adjust the value of PRIQTY in the DSNZPxxx module to reduce the primary allocation. If the primary allocation is reduced, you might need to increase the size of the secondary space allocation to avoid future abends.

Related reference

[PRIMARY QUANTITY field \(PRIQTY subsystem parameter\) \(Db2 Installation and Migration\)](#)

Recovering from BSDS failures

When the bootstrap data set (BSDS) is damaged, you need to recover that BSDS, regardless of whether you are running Db2 in dual-BSDS or single-BSDS mode.

Symptoms

If a BSDS is damaged, Db2 issues one of the following message numbers: DSNJ126I, DSNJ100I, or DSNJ120I.

Related concepts

[Management of the bootstrap data set \(Db2 Administration Guide\)](#)

Recovering from an I/O error on the BSDS

When an I/O error occurs on the only copy of the BSDS, you need to recover the BSDS before Db2 can operate normally. If an I/O error occurs on one copy of the BSDS in a dual-BSDS mode environment, you need to recover that copy of the BSDS before the next restart.

Symptoms

The following message is issued:

```
DSNJ126I - BSDS ERROR FORCED SINGLE BSDS MODE
```

The following messages are then issued:

```
DSNJ107I - READ ERROR ON BSDS
           DSNAME=... ERROR STATUS=...
DSNJ108I - WRITE ERROR ON BSDS
           DSNAME=... ERROR STATUS=...
```

Causes

A write I/O error occurred on a BSDS.

Environment

If Db2 is in a dual-BSDS mode and one copy of the BSDS is damaged by an I/O error, the BSDS mode changes from dual-BSDS mode to single-BSDS mode. If Db2 is in a single-BSDS mode when the BSDS is damaged by an I/O error, Db2 terminates until the BSDS is recovered.

Resolving the problem

System programmer response:

1. Use access method services to rename or delete the damaged BSDS and to define a new BSDS with the same name as the failing BSDS. You can find control statements in job DSNTIJIN.
2. Issue the Db2 command **RECOVER BSDS** to make a copy of the good BSDS in the newly allocated data set and to reinstate dual-BSDS mode.

Related tasks

[Recovering the BSDS from a backup copy \(Db2 Administration Guide\)](#)

Recovering from an error that occurs while opening the BSDS

You need to recover the bootstrap data set (BSDS) if an error occurs when Db2 opens the BSDS.

Symptoms

The following message is issued:

```
DSNJ100I - ERROR OPENING BSDS n DSNAME=..., ERROR STATUS=eeii
```

Resolving the problem

System programmer response:

1. Use access method services to delete or rename the damaged data set, to define a replacement data set, and to copy (with the **REPRO** command) the remaining BSDS to the replacement.
2. Use the **START DB2** command to start the Db2 subsystem.

Related tasks

[Recovering the BSDS from a backup copy \(Db2 Administration Guide\)](#)

Recovering from unequal timestamps on BSDSs

When timestamps on different copies of the bootstrap data set (BSDS) differ, Db2 attempts to resynchronize the BSDSs and restore dual BSDS mode. If this attempt succeeds, Db2 restart continues automatically. If this attempt fails, you need to recover from the situation.

Symptoms

The following message is issued:

```
DSNJ120I - DUAL BSDS DATA SETS HAVE UNEQUAL TIMESTAMPS,  
          BSDS1 SYSTEM=..., UTILITY=..., BSDS2 SYSTEM=..., UTILITY=...
```

Causes

Unequal timestamps can occur for the following reasons:

- One of the volumes that contains the BSDS has been restored. All information of the restored volume is outdated. If the volume contains any active log data sets or Db2 data, their contents are also outdated. The outdated volume has the lower timestamp.
- Dual BSDS mode has degraded to single BSDS mode, and you are trying to start without recovering the bad copy of the BSDS.
- The Db2 subsystem abended after updating one copy of the BSDS, but prior to updating the second copy.

Resolving the problem

Operator response: If Db2 restart fails, notify the system programmer.

System programmer response:

If Db2 fails to automatically resynchronize the BSDS data sets:

1. Run the print log map utility (**DSNJU004**) on both copies of the BSDS; compare the lists to determine which copy is accurate or current.
2. Rename the outdated data set, and define a replacement for it.
3. Copy the good data set to the replacement data set, using the **REPRO** command of access method services.
4. Use the access method services **REPRO** command to copy the current version of the active log to the outdated data set if all the following conditions are true:
 - The problem was caused by a restored outdated BSDS volume.
 - The restored volume contains active log data.
 - You were using dual active logs on separate volumes.

If you were not using dual active logs, cold start the subsystem.

If the restored volume contains database data, use the RECOVER utility to recover that data after successful restart.

Recovering the BSDS from a backup copy

In some situations, the bootstrap data set (BSDS) becomes damaged, and you need to recover the BSDS from a backup copy.

About this task

Db2 stops and does not restart until dual-BSDS mode is restored in the following situations:

- Db2 is operating in single-BSDS mode, and the BSDS is damaged.
- Db2 is operating in dual-BSDS mode, and both BSDSs are damaged.

Procedure

To recover the BSDS from a backup copy:

1. Locate the BSDS that is associated with the most recent archive log data set.

The data set name of the most recent archive log is displayed on the z/OS console in the last occurrence of message DSNJ003I, which indicates that offloading has successfully completed. In preparation for the rest of this procedure, keep a log of all successful archives that are noted by that message.

 - If archive logs are on disk, the BSDS is allocated on any available disk. The BSDS name is like the corresponding archive log data set name; change only the first letter of the last qualifier, from A to B, as in the following example:
Archive log name
DSN.ARCHLOG1.A0000001
BSDS copy name
DSN.ARCHLOG1.B0000001
 - If archive logs are on tape, the BSDS is the first data set of the first archive log volume. The BSDS is not repeated on later volumes.
2. If the most recent archive log data set has no copy of the BSDS (presumably because an error occurred during its offload), locate an earlier copy of the BSDS from an earlier offload.
3. Rename or delete any damaged BSDS.
 - To rename a damaged BSDS, use the access method services **ALTER** command with the NEWNAME option.
 - To delete a damaged BSDS, use the access method services **DELETE** command.

For each damaged BSDS, use access method services to define a new BSDS as a replacement data set. Job DSNTIJJIN contains access method services control statements to define a new BSDS. The BSDS is a VSAM key-sequenced data set (KSDS) that has three components: cluster, index, and data. You must rename all components of the data set. Avoid changing the high-level qualifier.

4. Use the access method services **REPRO** command to copy the BSDS from the archive log to one of the replacement BSDSs that you defined in the prior step. Do not copy any data to the second replacement BSDS; data is placed in the second replacement BSDS in a later step in this procedure.

- a) Use the print log map utility (DSNJU004) to print the contents of the replacement BSDS.

You can then review the contents of the replacement BSDS before continuing your recovery work.

- b) Update the archive log data set inventory in the replacement BSDS.

Examine the print log map output, and note that the replacement BSDS does not obtain a record of the archive log from which the BSDS was copied. If the replacement BSDS is a particularly old copy, it is missing all archive log data sets that were created later than the BSDS backup copy. Therefore, you need to update the BSDS inventory of the archive log data sets to reflect the current subsystem inventory.

Use the change log inventory utility (DSNJU003) NEWLOG statement to update the replacement BSDS, adding a record of the archive log from which the BSDS was copied. Ensure that the CATALOG option of the NEWLOG statement is properly set to CATALOG = YES if the archive log data set is cataloged. Also, use the NEWLOG statement to add any additional archive log data sets that were created later than the BSDS copy.

- c) Update DDF information in the replacement BSDS.

If the Db2 subsystem for your installation is part of a distributed network, the BSDS contains the DDF control record. You must review the contents of this record in the output of the print log map utility. If changes are required, use the change log inventory DDF statement to update the BSDS DDF record.

- d) Update the active log data set inventory in the replacement BSDS.

In unusual circumstances, your installation might have added, deleted, or renamed active log data sets since the BSDS was copied. In this case, the replacement BSDS does not reflect the actual number or names of the active log data sets that your installation has currently in use.

If you must delete an active log data set from the replacement BSDS log inventory, use the change log inventory utility DELETE statement.

If you need to add an active log data set to the replacement BSDS log inventory, use the change log inventory utility NEWLOG statement. Ensure that the RBA range is specified correctly on the NEWLOG statement.

If you must rename an active log data set in the replacement BSDS log inventory, use the change log inventory utility DELETE statement, followed by the NEWLOG statement. Ensure that the RBA range is specified correctly on the NEWLOG statement.

- e) Update the active log RBA ranges in the replacement BSDS.

Later, when a restart is performed, Db2 compares the RBAs of the active log data sets that are listed in the BSDS with the RBAs that are found in the actual active log data sets. If the RBAs do not agree, Db2 does not restart. The problem is magnified when a particularly old copy of the BSDS is used. To resolve this problem, use the change log inventory utility to change the RBAs that are found in the BSDS to the RBAs in the actual active log data sets. Take the appropriate action, described below, to change RBAs in the BSDS:

- If you are not certain of the RBA range of a particular active log data set, use DSN1LOGP to print the contents of the active log data set. Obtain the logical starting and ending RBA values for the active log data set from the DSN1LOGP output. The STARTRBA value that you use in the change log inventory utility must be at the beginning of a control interval. Similarly, the ENDRBA value that you use must be at the end of a control interval. To get these values, round the starting RBA value from the DSN1LOGP output down so that it ends in X'000'. Round the ending RBA value up so that it ends in X'FFF'.

- When the RBAs of all active log data sets are known, compare the actual RBA ranges with the RBA ranges that are found in the BSDS (listed in the print log map utility output).

If the RBA ranges are equal for all active log data sets, you can proceed to step [“4.f” on page 80](#) without any additional work.

If the RBA ranges are not equal, adjust the values in the BSDS to reflect the actual values. For each active log data set for which you need to adjust the RBA range, use the change log inventory utility DELETE statement to delete the active log data set from the inventory in the replacement BSDS. Then use the NEWLOG statement to redefine the active log data set to the BSDS.

- f) If only two active log data sets are specified in the replacement BSDS, add a new active log data set for each copy of the active log, and define each new active log data set of the replacement BSDS log inventory.

If only two active log data sets are specified for each copy of the active log, Db2 might have difficulty during restart. The difficulty can arise when one of the active log data sets is full and has not been offloaded, whereas the second active log data set is close to filling. Adding a new active log data set for each copy of the active log can alleviate difficulties on restart in this situation.

To add a new active log data set for each copy of the active log, use the access method services **DEFINE** command. The control statements to accomplish this task can be found in job DSNTIJIN. After the active log data sets are physically defined and allocated, use the change log inventory utility NEWLOG statement to define the new active log data sets of the replacement BSDS. You do not need to specify the RBA ranges on the NEWLOG statement.

5. Copy the updated BSDS copy to the second new BSDS data set.

The dual bootstrap data sets are now identical.

6. Optional: Use the print log map utility (DSNJU004) to print the contents of the second replacement BSDS at this point.

7. If you have lost your current active log data set, refer to the following topics:

- [“Recovering from BSDS or log failures during restart” on page 80](#)
- [Task 4: Truncate the log at the point of error \(Db2 Administration Guide\)](#), which provides information about how to construct a conditional restart control record (CRCR).

8. Restart Db2, using the newly constructed BSDS.

Db2 determines the current RBA and what active logs need to be archived.

Related information

[DFSMS Access Method Services Commands](#)

Recovering from BSDS or log failures during restart

When the bootstrap data set (BSDS) or part of the recovery log for Db2 is damaged or lost and that damage prevents restart, you need to recover from that situation. What you do to recover varies based on the particular circumstances.

If the problem is discovered at restart, begin with one of the following recovery procedures:

- [“Recovering from active log failures ” on page 69](#)
- [“Recovering from archive log failures ” on page 73](#)
- [Recovering from BSDS failures \(Db2 Administration Guide\)](#)

If the problem persists, return to the procedures in this section.

When Db2 recovery log damage terminates restart processing, Db2 issues messages to the console to identify the damage and issue an abend reason code. (The SVC dump title includes a more specific abend reason code to assist in problem diagnosis.) If the explanations for the reason codes indicate that restart failed because of some problem that is not related to a log error, contact IBM Software Support.

To minimize log problems during restart, the system requires two copies of the BSDS. Dual logging is also recommended.

Basic approaches to recovery: The two basic approaches to recovery from problems with the log are:

- Restart Db2, bypassing the inaccessible portion of the log and rendering some data inconsistent. Then recover the inconsistent objects by using the RECOVER utility, or re-create the data by using REPAIR. Use the methods that are described following this procedure to recover the inconsistent data.
- Restore the entire Db2 subsystem to a prior point of consistency. The method requires that you have first prepared such a point; for suggestions, see [Preparing to recover to a prior point of consistency \(Db2 Administration Guide\)](#). Methods of recovery are described under [“Recovering from unresolvable BSDS or log data set problem during restart”](#) on page 101.

Bypassing the damaged log

Even if the log is damaged, and Db2 is started by circumventing the damaged portion, the log is the most important source for determining what work was lost and what data is inconsistent.

Bypassing a damaged portion of the log generally proceeds with the following steps:

1. Db2 restart fails. A problem exists on the log, and a message identifies the location of the error. The following abend reason codes, which appear only in the dump title, can be issued for this type of problem. This is not an exhaustive list; other codes might occur.

```
00D10261
00D10262
00D10263
00D10264
00D10265
00D10266
00D10267
00D10268
00D10329
00D1032A
00D1032B
00D1032C
00E80084
```

The following figure illustrates the general problem:

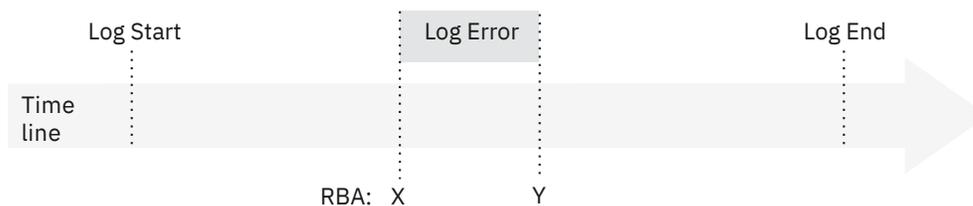


Figure 6. General problem of damaged Db2 log information

2. Db2 cannot skip over the damaged portion of the log and continue restart processing. Instead, you restrict processing to only a part of the log that is error free. For example, the damage shown in the preceding figure occurs in the log RBA range between X to Y. You can restrict restart to all of the log before X; then changes later than X are not made. Alternatively, you can restrict restart to all of the log after Y; then changes between X and Y are not made. In either case, some amount of data is inconsistent.
3. You identify the data that is made inconsistent by your restart decision. With the SUMMARY option, the DSN1LOGP utility scans the accessible portion of the log and identifies work that must be done at restart, namely, the units of recovery that are to be completed and the page sets that they modified.

Because a portion of the log is inaccessible, the summary information might not be complete. In some circumstances, your knowledge of work in progress is needed to identify potential inconsistencies.

4. You use the CHANGE LOG INVENTORY utility to identify the portion of the log to be used at restart, and to tell whether to bypass any phase of recovery. You can choose to do a cold start and bypass the entire log.
5. You restart Db2. Data that is unaffected by omitted portions of the log is available for immediate access.
6. Before you allow access to any data that is affected by the log damage, you resolve all data inconsistencies. That process is described under [“Resolving inconsistencies resulting from a conditional restart” on page 107](#).

Where to start

The specific procedure depends on the phase of restart that was in control when the log problem was detected. On completion, each phase of restart writes a message to the console. You must find the last of those messages in the console log. The next phase after the one that is identified is the one that was in control when the log problem was detected. Accordingly, start at:

- [“Recovering from failure during log initialization or current status rebuild” on page 83](#)
- [“Recovering from a failure during forward log recovery” on page 93](#)
- [“Recovering from a failure during backward log recovery” on page 98](#)

As an alternative, determine which, if any, of the following messages was last received and follow the procedure for that message. Other DSN messages can also be issued.

| Message ID | Procedure to use |
|------------|--|
| DSNJ001I | “Recovering from failure during log initialization or current status rebuild” on page 83 |
| DSNJ100I | “Recovering from unresolvable BSDS or log data set problem during restart” on page 101 |
| DSNJ107 | “Recovering from unresolvable BSDS or log data set problem during restart” on page 101 |
| DSNJ119I | “Recovering from unresolvable BSDS or log data set problem during restart” on page 101 |
| DSNR002I | None. Normal restart processing can be expected. |
| DSNR004I | “Recovering from a failure during forward log recovery” on page 93 |
| DSNR005I | “Recovering from a failure during backward log recovery” on page 98 |
| DSNR006I | None. Normal restart processing can be expected. |
| Other | “Recovering from failure during log initialization or current status rebuild” on page 83 |

Another procedure ([“Recovering from a failure resulting from total or excessive loss of log data” on page 103](#)) provides information to use if you determine (by using [“Recovering from failure during log initialization or current status rebuild” on page 83](#)) that an excessive amount (or all) of Db2 log information (BSDS, active, and archive logs) has been lost.

The last procedure, [“Resolving inconsistencies resulting from a conditional restart” on page 107](#), can be used to resolve inconsistencies introduced while using one of the restart procedures in this information. If you decide to use [“Recovering from unresolvable BSDS or log data set problem during restart” on page 101](#), you do not need to use [“Resolving inconsistencies resulting from a conditional restart” on page 107](#).

Because of the severity of the situations described, the procedures identify "Operations management action", rather than "Operator action". Operations management might not be performing all the steps in the procedures, but they must be involved in making the decisions about the steps to be performed.

Related reference

[DSN1LOGP \(Db2 Utilities\)](#)

Recovering from failure during log initialization or current status rebuild

When a failure occurs during the log initialization phase or the current status rebuild phase of restart, you need to recover from this situation.

Symptoms

An abend was issued, indicating that restart failed. In addition, either the last restart message that was received was a DSNJ001I message that indicates a failure during current status rebuild, or none of the following messages was issued:

- DSNJ001I
- DSNR004I
- DSNR005I

If none of the preceding messages was issued, the failure occurred during the log initialization phase of restart.

Environment

What happens in the environment depends on whether the failure occurred during log initialization or current status rebuild.

Failure during log initialization

Db2 terminates because a portion of the log is inaccessible, and Db2 cannot locate the end of the log during restart.

Failure during current status rebuild

Db2 terminates because a portion of the log is inaccessible, and Db2 cannot determine the state of the subsystem at the prior Db2 termination. Possible states include: outstanding units of recovery, outstanding database writes, and exception database conditions.

Resolving the problem

Operations management response:

To correct the problem, choose one of the following approaches:

- Correct the problem that has made the log inaccessible, and start Db2 again. To determine if this approach is possible, read the relevant information about the messages and codes that you received. The explanations for the messages and codes identify the corrective action that can be taken to resolve the problem.
- Restore the Db2 log and all data to a prior consistent point, and then start Db2. This procedure is described in [“Recovering from unresolvable BSDS or log data set problem during restart”](#) on page 101.
- Start Db2 without completing some database changes. Using a combination of Db2 services and your own knowledge, determine what work is likely to be lost if you truncate the log. The procedure for determining the page sets that contain incomplete changes is described in [“Restarting Db2 by truncating the log”](#) on page 86.

Failure during log initialization phase

When a failure occurs during the log initialization phase, certain characteristics of the situation are evident.

The following figure illustrates the timeline of events that exist when a failure occurs during the log initialization phase.

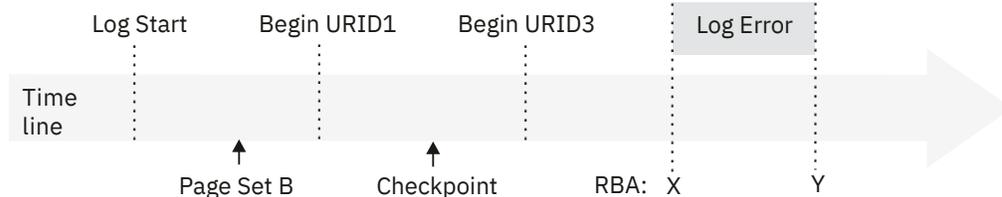


Figure 7. Failure during log initialization

The portion of the log between log RBAs X and Y is inaccessible. For failures that occur during the log initialization phase, the following activities occur:

1. Db2 allocates and opens each active log data set that is not in a stopped state.
2. Db2 reads the log until the last log record is located.
3. During this process, a problem with the log is encountered, preventing Db2 from locating the end of the log. Db2 terminates and issues an abend reason code. Some of the abend reason codes that might be issued include:
 - 00D10261
 - 00D10262
 - 00D10263
 - 00D10264
 - 00D10265
 - 00D10266
 - 00D10267
 - 00D10268
 - 00D10329
 - 00D1032A
 - 00D1032B
 - 00D1032C
 - 00E80084

During its operations, Db2 periodically records in the BSDS the RBA of the last log record that was written. This value is displayed in the print log map report as follows:

```
HIGHEST RBA WRITTEN: 00000742989E
```

Because this field is updated frequently in the BSDS, the "highest RBA written" can be interpreted as an approximation of the end of the log. The field is updated in the BSDS when any one of a variety of internal events occurs. In the absence of these internal events, the field is updated each time a complete cycle of log buffers is written. A complete cycle of log buffers occurs when the number of log buffers that are written equals the value of the OUTPUT BUFFER field of installation panel DSNTIPL. The value in the BSDS is, therefore, relatively close to the end of the log.

To find the actual end of the log at restart, Db2 reads the log forward sequentially, starting at the log RBA that approximates the end of the log and continuing until the actual end of the log is located.

Because the end of the log is inaccessible in this case, some information is lost:

- Units of recovery might have successfully committed or modified additional page sets past point X.
- Additional data might have been written, including those that are identified with writes that are pending in the accessible portion of the log.
- New units of recovery might have been created, and these might have modified data.

Because of the log error, Db2 cannot perceive these events.

A restart of Db2 in this situation requires truncation of the log.

Related tasks

[Restarting Db2 by truncating the log \(Db2 Administration Guide\)](#)

Description of failure during current status rebuild

When a failure occurs during current status rebuild, certain characteristics of the situation are evident.

The following figure illustrates the timeline of events that exist when a failure occurs during current status rebuild.

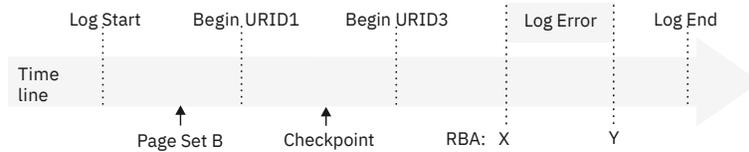


Figure 8. Failure during current status rebuild

The portion of the log between log RBAs X and Y is inaccessible. For failures that occur during the current status rebuild phase, the following activities occur:

1. Log initialization completes successfully.
2. Db2 locates the last checkpoint. (The BSDS contains a record of its location on the log.)
3. Db2 reads the log, beginning at the checkpoint and continuing to the end of the log.
4. Db2 reconstructs the status of the subsystem as it existed at the prior termination of Db2.
5. During this process, a problem with the log is encountered, preventing Db2 from reading all required log information. Db2 terminates and issues an abend reason code. Some of the abend reason codes that might be issued include:
 - 00D10261
 - 00D10262
 - 00D10263
 - 00D10264
 - 00D10265
 - 00D10266
 - 00D10267
 - 00D10268
 - 00D10329
 - 00D1032A
 - 00D1032B
 - 00D1032C
 - 00E80084

Because the end of the log is inaccessible in this case, some information is lost:

- Units of recovery might have successfully committed or modified additional page sets past point X.
- Additional data might have been written, including those that are identified with writes that are pending in the accessible portion of the log.
- New units of recovery might have been created, and these might have modified data.

Because of the log error, Db2 cannot perceive these events.

A restart of Db2 in this situation requires truncation of the log.

Related tasks

[Restarting Db2 by truncating the log \(Db2 Administration Guide\)](#)

Restarting Db2 by truncating the log

A portion of the log is inaccessible during the log initialization or current status rebuild phases of restart. When the log is inaccessible, Db2 cannot identify precisely what units of recovery failed to complete, what page sets had been modified, and what page sets have writes pending. You need to gather that information, and restart Db2.

Task 1: Find the log RBA after the inaccessible part of the log

The first task in restarting Db2 by truncating the log is to locate the log RBA after the inaccessible part of the log.

About this task

The range of the log between RBAs X and Y is inaccessible to all Db2 processes.

Procedure

To find the RBA after the inaccessible part of the log, take the action that is associated with the message number that you received (DSNJ007I, DSNJ012I, DSNJ103I, DSNJ104I, DSNJ106I, and DSNJ113E):

- **When message DSNJ007I is issued:**

The problem is that an operator canceled a request for archive mount. Reason code 00D1032B is associated with this situation and indicates that an entire data set is inaccessible.

For example, the following message indicates that the archive log data set DSNCAT.ARCHLOG1.A0000009 is not accessible. The operator canceled a request for archive mount, resulting in the following message:

```
DSNJ007I OPERATOR CANCELED MOUNT OF ARCHIVE
          DSNCAT.ARCHLOG1.A0000009 VOLSER=5B225.
```

To determine the value of X, run the print log map utility (DSNJU004) to list the log inventory information. The output of this utility provides each log data set name and its associated log RBA range, the values of X and Y.

- **When message DSNJ012I is issued:**

The problem is that a log record is logically damaged. Message DSNJ012I identifies the log RBA of the first inaccessible log record that Db2 detects. The following reason codes are associated with this situation:

- 00D10261
- 00D10262
- 00D10263
- 00D10264
- 00D10265
- 00D10266
- 00D10267
- 00D10268
- 00D10348

For example, the following message indicates a logical error in the log record at log RBA X'7429ABA'.

```
DSNJ012I ERROR D10265 READING RBA 000007429ABA
          IN DATA SET DSNCAT.LOGCOPY2.DS01
          CONNECTION-ID=DSN,
          CORRELATION-ID=DSN
```

A given physical log record is actually a set of logical log records (the log records that are generally spoken of) and the log control interval definition (LCID). Db2 stores logical records in blocks of physical

records to improve efficiency. When this type of an error on the log occurs during log initialization or current status rebuild, all log records within the physical log record are inaccessible. Therefore, the value of *X* is the log RBA that was reported in the message, rounded down to a 4-KB boundary. (For the example message above, the rounded 4-KB boundary value would be X'7429000'.)

- **When message DSNJ103I or DSNJ104I is issued:**

For message DSNJ103I, the underlying problem depends on the reason code that is issued:

- For reason code 00D1032B, an allocation error occurred for an archive log data set.
- For reason code 00E80084, an active log data set that is named in the BSDS could not be allocated during log initialization.

For message DSNJ104I, the underlying problem is that an open error occurred for an archive and active log data set.

In any of these cases, the message that accompanies the abend identifies an entire data set that is inaccessible. For example, the following DSNJ103I message indicates that the archive log data set DSNCAT.ARCHLOG1.A0000009 is not accessible. The STATUS field identifies the code that is associated with the reason for the data set being inaccessible.

```
DSNJ103I - csect-name LOG ALLOCATION ERROR
          DSNAME=DSNCAT . ARCHLOG1 . A0000009 , ERROR
          STATUS=04980004
          SMS REASON CODE=reasond-code
```

To determine the value of *X*, run the print log map utility (DSNJU004) to list the log inventory information. The output of the utility provides each log data set name and its associated log RBA range, the values of *X* and *Y*.

Verify the accuracy of the information in the print log map utility output for the active log data set with the lowest RBA range. For this active log data set only, the information in the BSDS is potentially inaccurate for the following reasons:

- When an active log data set is full, archiving is started. Db2 then selects another active log data set, usually the data set with the lowest RBA. This selection is made so that units of recovery do not need to wait for the archive operation to complete before logging can continue. However, if a data set has not been archived, nothing beyond it has been archived, and the procedure is ended.
- When logging begins on a reusable data set, Db2 updates the BSDS with the new log RBA range for the active log data set and marks it as "Not Reusable." The process of writing the new information to the BSDS might be delayed by other processing. Therefore, a possible outcome is for a failure to occur between the time that logging to a new active log data set begins and the time that the BSDS is updated. In this case, the BSDS information is not correct.

If the data set is marked "Not Reusable," the log RBA that appears for the active log data set with the lowest RBA range in the print log map utility output is valid. If the data set is marked "Reusable," you can assume for the purposes of this restart that the starting log RBA (*X*) for this data set is one greater than the highest log RBA that is listed in the BSDS for all other active log data sets.

- **When message DSNJ106I is issued:**

The problem is that an I/O error occurred while a log record was being read. The message identifies the log RBA of the first inaccessible log record that Db2 detects. Reason code 00D10329 is associated with this situation.

For example, the following message indicates an I/O error in the log at RBA X'7429ABA'.

```
DSNJ106I LOG READ ERROR DSNAME=DSNCAT.LOGCOPY2.DS01,
          LOGRBA=000007429ABA, ERROR STATUS=0108320C
```

A given physical log record is actually a set of logical log records (the log records that are generally spoken of) and the log control interval definition (LCID). When this type of an error on the log occurs during log initialization or current status rebuild, all log records within the physical log record, and beyond it to the end of the log data set, are inaccessible. This is due to the log initialization or current

status rebuild phase of restart. Therefore, the value of *X* is the log RBA that was reported in the message, rounded down to a 4-KB boundary. (For the example message above, the rounded 4-KB boundary value would be X'7429000'.)

- **When message DSNJ113E is issued:**

The problem is that the log RBA could not be found in the BSDS. Message DSNJ113E identifies the log RBA of the inaccessible log record. This log RBA is not registered in the BSDS. Reason code 00D1032B is associated with this situation.

For example, the following message indicates that the log RBA X'7429ABA' is not registered in the BSDS:

```
DSNJ113E RBA 000007429ABA NOT IN ANY ACTIVE OR ARCHIVE
LOG DATA SET. CONNECTION-ID=DSN, CORRELATION-ID=DSN
```

Use the print log map utility (DSNJU004) to list the contents of the BSDS.

A given physical log record is actually a set of logical log records (the log records that are generally spoken of) and the log control interval definition (LCID). When this type of an error on the log occurs during log initialization or current status rebuild, all log records within the physical log record are inaccessible.

Using the print log map output, locate the RBA that is closest to, but less than, X'7429ABA' for the value of *X*. If you do not find an RBA that is less than X'7429ABA', a considerable amount of log information has been lost. If this is the case, continue with [“Recovering from a failure resulting from total or excessive loss of log data” on page 103](#). Otherwise, continue with the next topic.

Related concepts

[Description of failure during current status rebuild \(Db2 Administration Guide\)](#)

[Failure during log initialization phase \(Db2 Administration Guide\)](#)

Related reference

[DSNJU004 \(print log map\) \(Db2 Utilities\)](#)

Related information

[DSNJ007I \(Db2 Messages\)](#)

[DSNJ012I \(Db2 Messages\)](#)

[DSNJ103I \(Db2 Messages\)](#)

[DSNJ104I \(Db2 Messages\)](#)

[DSNJ106I \(Db2 Messages\)](#)

[DSNJ113E \(Db2 Messages\)](#)

Task 2: Identify lost work and inconsistent data

In certain recovery situations (such as when you recover by truncating the log), you need to identify what work was lost and what data is inconsistent.

Procedure

To identify lost work and inconsistent data:

1. Obtain available information to help you determine the extent of the loss.

Db2 cannot determine what units of recovery are not completed, what database state information is lost, or what data is inconsistent in this situation. The log contains all such information, but the information is not available. The steps below explain what to do to obtain the information that is available within Db2 to help you determine the extent of the loss. The steps also explain how to start Db2 in this situation.

After restart, data is inconsistent. Results of queries and any other operations on such data vary from incorrect results to abends. Abends that occur either identify an inconsistency in the data or incorrectly assume the existence of a problem in the Db2 internal algorithms.



Attention: If the inconsistent page sets are not identified and the problems in them are not resolved after starting Db2, be aware that following this procedure and allowing access to inconsistent data involves some risk.

a) Run the print log map utility.

The report that the utility produces includes a description of the last 100 checkpoints and provides, for each checkpoint the following information:

- The location in the log of the checkpoint (begin and end RBA)
- The date and time of day that the checkpoint was performed

b) Locate the checkpoint on the log prior to the point of failure (X).

Do that by finding the first checkpoint with an end RBA that is less than X.

Continue with the step “2” on [page 89](#) unless one of the following conditions exists:

- You cannot find such a checkpoint. This means that a considerable amount of log has been lost.
- You find the checkpoint, but the checkpoint is several days old, and Db2 has been operational during the interim.

In these two cases, use one of the following procedures:

- “[Recovering from a failure resulting from total or excessive loss of log data](#)” on [page 103](#)
- “[Recovering from unresolvable BSDS or log data set problem during restart](#)” on [page 101](#)

2. Determine what work is lost and what data is inconsistent.

The portion of the log that represents activity that occurred before the failure provides information about work that was in progress at that point. From this information, you might be able to deduce what work was in progress within the inaccessible portion of the log. If use of Db2 was limited at the time or if Db2 was dedicated to a small number of activities (such as batch jobs that perform database loads or image copies), you might be able to accurately identify the page sets that were made inconsistent. To make the identification, extract a summary of the log activity up to the point of damage in the log by using the DSN1LOGP utility.

- Use the DSN1LOGP utility to specify the "BEGIN CHECKPOINT" RBA prior to the point of failure, which was determined in the previous task as the RBASTART. Terminate the DSN1LOGP scan prior to the point of failure on the log (X - 1) by using the RBAEND specification.
- Specify the last complete checkpoint. This is very important for ensuring that complete information is obtained from DSN1LOGP.
- Specify the SUMMARY(ONLY) option to produce a summary report.

The following figure is an example of a DSN1LOGP job that obtains summary information for the checkpoint that was described previously.

```
//ONE EXEC PGM=DSN1LOGP
//STEPLIB DD DSN=prefix.SDSNLOADSDSNLOAD,DISP=SHR
//SYSABEND DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//SYSSUMRY DD SYSOUT=A
//BSDS DD DSN=DSNCAT.BSDS01,DISP=SHR
//SYSIN DD *
RBASTART (7425468) RBAEND (7428FFF) SUMMARY (ONLY)
/*
```

Figure 9. Sample JCL for obtaining DSN1LOGP summary output for restart

3. Analyze the DSN1LOGP utility output.

Related reference

[DSN1LOGP \(Db2 Utilities\)](#)

DSN1LOGP summary report

The DSN1LOGP utility generates a summary report, which is placed in the SYSSUMRY file. The report includes a summary of completed events and a restart summary. You can use the information in this report to identify lost work and inconsistent data that needs to be resolved.

The following figure shows an excerpt from the restart summary in a sample **DSN1LOGP** summary report. The report is described after the figure.

```
DSN1157I RESTART SUMMARY
DSN1153I DSN1LSIT CHECKPOINT MEMBER=DB2A
          STARTRBA=0000000000002BBB8CAC ENDRBA=0000000000002BBC59E8
          STARTLRSN=00CA21F58479D042C000 ENDLRSN=00CA21F58480E67E4000
          DATE=12.250 TIME=14:20:29

DSN1162I DSN1LPRT MEMBER=DB2A UR CONNID=BATCH CORRID=ARCHIVE AUTHID=SYSADM
PLAN=ARCHIVE
          START DATE=00.161 TIME=11:27:30 DISP=INFLIGHT INFO=COMPLETE
          STARTRBA=0000000000002BBC888E STARTLRSN=00CA21F5849D6B88E000 NID=*
          LUWID=DSNCAT.SYEC1DB2.CA21F58084CF.0003 COORDINATOR=*
          PARTICIPANTS=*
          DATA MODIFIED:
            DATABASE=0119=JACKDB PAGE SET=0002=JACKTS
            DATABASE=0119=JACKDB PAGE SET=0005=TESTIX

DSN1160I DATABASE WRITES PENDING:
          DATABASE=0001=DSNDB01 PAGE SET=0008=DSNDB01X START=0000000000002BB8BC60
          DATABASE=0001=DSNDB01 PAGE SET=001F=DBD01 START=0000000000002BB8BED8
          DATABASE=0006=DSNDB06 PAGE SET=006C=DSNADX01 START=0000000000002BB8EE55
          DATABASE=0006=DSNDB06 PAGE SET=0787=DSNADH02 START=0000000000002BB8E858
          DATABASE=0006=DSNDB06 PAGE SET=0076=DSNUCX01
          . . . .
```

Figure 10. Partial sample of DSN1LOGP summary output

The following message acts as a heading, which is followed by messages that identify the units of recovery that have not yet completed and the page sets that they modified:

```
DSN1157I RESTART SUMMARY
```

Following the summary of outstanding units of recovery is a summary of page sets that have database writes that are pending.

In each case (units of recovery or databases with pending writes), the earliest required log record is identified by the START information. In this context, START information is the log RBA of the earliest log record that is required in order to complete outstanding writes for this page set.

Those units of recovery with a START log RBA equal to, or prior to, the point Y cannot be completed at restart. All page sets that were modified by these units of recovery are inconsistent after completion of restart when you attempt to identify lost work and inconsistent data.

All page sets that are identified in message DSN1160I with a START log RBA value equal to, or prior to, the point Y have database changes that cannot be written to disk. As in the previously described case, all of these page sets are inconsistent after completion of restart when you attempt to identify lost work and inconsistent data.

At this point, you need to identify only the page sets in preparation for restart. After restart, you need to resolve the problems in the page sets that are inconsistent.

Because the end of the log is inaccessible, some information is lost; therefore, the information is inaccurate. Some of the units of recovery that appear to be inflight might have successfully committed, or they might have modified additional page sets beyond point X. Additional data might have been written, including those page sets that are identified as having pending writes in the accessible portion of the log. New units of recovery might have been created, and these might have modified data. Db2 cannot detect that these events occurred.

From this and other information (such as system accounting information and console messages), you might be able to determine what work was actually outstanding and which page sets are likely to be inconsistent after you start Db2. This is because the record of each event contains the date and

time to help you determine how recent the information is. In addition, the information is displayed in chronological sequence.

Task 3: Determine what status information is lost

The third task in restarting Db2 by truncating the log is to determine what status information has been lost.

About this task

Depending on what was going on in your environment before the problem occurred, some amount of system status information might have been lost.

Procedure

To determine what system status information is lost:

1. If you already know what system status information is lost (such as in the case in which utilities are in progress), you do not need to do anything. Continue with the next topic.
2. If you do not already know what system status information is lost, examine all relevant messages that provide details about the loss of status information (such as in the cases of deferred restart pending or write error ranges).

If the messages provide adequate information about what information is lost, you do not need to do anything more. Continue with the next step.

3. If you find that all system status information is lost, try to reconstruct this information from recent console displays, messages, and abends that alerted you to these conditions.

These page sets contain inconsistencies that you must resolve.

Task 4: Truncate the log at the point of error

The fourth task in restarting Db2 by truncating the log is to truncate the log at the point of error.

About this task

No Db2 process, including the RECOVER utility, allows a gap in the log RBA sequence. You cannot process up to point X, skip over points X through Y, and continue after Y.

Procedure

Create a conditional restart control record (CRCR) in the BSDS by using the change log inventory utility. Specify the following options:

ENDRBA=*endrba*

The *endrba* value is the RBA at which Db2 begins writing new log records. If point X is X'7429000', specify ENDRBA=7429000 on the CRESTART control statement.

At restart, Db2 discards the portion of the log beyond X'7429000' before processing the log for completing work (such as units of recovery and database writes). Unless otherwise directed, Db2 performs normal restart processing within the scope of the log. Because log information is lost, Db2 errors might occur. For example, a unit of recovery that has actually been committed might be rolled back. Also, some changes that were made by that unit of recovery might not be rolled back because information about data changes is lost.

FORWARD=NO

Terminates forward-log recovery before log records are processed. This option and the BACKOUT=NO option minimize errors that might result from normal restart processing.

BACKOUT=NO

Terminates backward-log recovery before log records are processed. This option and the FORWARD=NO option minimize errors that might result from normal restart processing.

Results

Recovering and backing out units of recovery with lost information might introduce more inconsistencies than the incomplete units of recovery.

Example

The following example is a CRESTART control statement for the ENDRBA value of X'7429000':

```
CRESTART CREATE,ENDRBA=7429000,FORWARD=NO,BACKOUT=NO
```

Task 5: Start Db2 and resolve data inconsistencies

The final task in restarting Db2 by truncating the log is to restart Db2 and resolve inconsistencies.

Before you begin

You must have a CRESTART control statement with the correct ENDRBA value and the FORWARD and BACKOUT options set to NO.

Procedure

To start Db2 and resolve data inconsistencies:

1. Start Db2 with the following command:

```
-START DB2 ACCESS (MAINT)
```

In response to this command, Db2 performs the following actions:

- Discards from the checkpoint queue any entries with RBAs that are beyond the ENDRBA value in the CRCR (for example, X'7429000').
- Reconstructs the system status up to the point of log truncation.
- Performs pending database writes that the truncated log specifies and that have not already been applied to the data. You can use the DSN1LOGP utility to identify these writes. No forward recovery processing occurs for units of work in a FORWARD=NO conditional restart. All pending writes for in-commit and indoubt units of recovery are applied to the data. The standard forward-log recovery processing for the different unit of work states does not occur.
- Marks all units of recovery that have committed or are indoubt as complete on the log.
- Leaves inflight and in-abort units of recovery incomplete. Inconsistent data is left in tables that are modified by inflight or indoubt units of recovery. When you specify a BACKOUT=NO conditional restart, inflight and in-abort units of recovery are not backed out.

In a conditional restart that truncates the log, BACKOUT=NO minimizes Db2 errors for the following reasons:

- Inflight units of recovery might have been committed in the portion of the log that the conditional restart discarded. If these units of recovery are backed out (as would be normal during backward-log recovery) Db2 might back out database changes incompletely, which introduces additional errors.
- Data that is modified by in-abort units of recovery might have been modified again after the point of damage on the log. For in-abort units of recovery, Db2 might have written backout processing to disk after the point of log truncation. If these units of recovery are backed out (as would be normal during backward-log recovery), Db2 might introduce additional data inconsistencies by backing out units of recovery that are already partially or fully backed out.

At the end of restart, the conditional restart control record (CRCR) is marked "Deactivated" to prevent its use on a later restart. Until the restart completes successfully, the CRCR is in effect. Until data is consistent or page sets are stopped, start Db2 with the ACCESS (MAINT) option.

2. Resolve all data inconsistency problems.

Related concepts

[Phase 4: Backward log recovery \(Db2 Administration Guide\)](#)

[Phase 3: Forward log recovery \(Db2 Administration Guide\)](#)

Related tasks

[Resolving inconsistencies resulting from a conditional restart \(Db2 Administration Guide\)](#)

Recovering from a failure during forward log recovery

If a failure occurs during the forward-log recovery phase of restart, operations management can recover from this situation.

Symptoms

A Db2 abend occurred, indicating that restart had failed. In addition, the last restart message that was received was a DSNR004I message, which indicates that log initialization completed; therefore, the failure occurred during forward log recovery.

Environment

Db2 terminates because a portion of the log is inaccessible, and Db2 is therefore unable to guarantee the consistency of the data after restart.

Resolving the problem

Operations management response:

To start Db2 successfully, choose one of the following approaches:

- Read the information about relevant messages and codes that you received to determine if this approach is possible. The explanations of the messages and codes identify any corrective action that you can take to resolve the problem. If it is possible, correct the problem that made the log inaccessible, and start Db2 again.
- Restore the Db2 log and all data to a prior consistent point and start Db2. This procedure is described in [“Recovering from unresolvable BSDS or log data set problem during restart”](#) on page 101.
- Start Db2 without completing some database changes. Do this only if the exact changes cannot be identified; all that can be determined is which page sets might have incomplete changes and which units of recovery made modifications to those page sets. The procedure for determining which page sets contain incomplete changes and which units of recovery made the modifications is described in [“Recovering from BSDS or log failures during restart”](#) on page 80. Other topics might help you better understand the problem.

Forward-log recovery failure

When a failure occurs during the forward-log recovery phase of Db2 restart, certain characteristics of the situation are evident.

The following figure illustrates the events surrounding a failure during the forward-log recovery phase of Db2 restart.

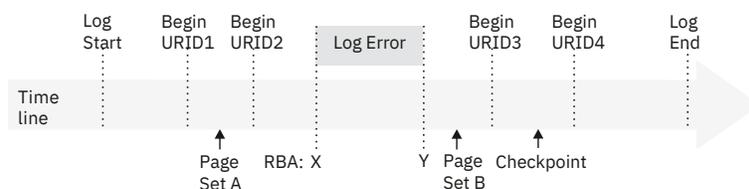


Figure 11. Illustration of failure during forward-log recovery

The portion of the log between log RBA X and Y is inaccessible. The log initialization and current status rebuild phases of restart completed successfully. Restart processing was reading the log in a forward direction, beginning at some point prior to X and continuing to the end of the log. Because of the

inaccessibility of log data (between points X and Y), restart processing cannot guarantee the completion of any work that was outstanding at restart prior to point Y.

Assume that the following work was outstanding at restart:

- The unit of recovery that is identified as URID1 was in-commit.
- The unit of recovery that is identified as URID2 was inflight.
- The unit of recovery that is identified as URID3 was in-commit.
- The unit of recovery that is identified as URID4 was inflight.
- Page set A had writes that were pending prior to the error on the log, continuing to the end of the log.
- Page set B had writes that were pending after the error on the log, continuing to the end of the log.

The earliest log record for each unit of recovery is identified on the log line in [Figure 11 on page 93](#). In order for Db2 to complete each unit of recovery, Db2 requires access to all log records from the beginning point for each unit of recovery to the end of the log.

The error on the log prevents Db2 from guaranteeing the completion of any outstanding work that began prior to point Y on the log. Consequently, database changes that are made by URID1 and URID2 might not be fully committed or backed out. Writes that were pending for page set A (from points in the log prior to Y) are lost.

Starting Db2 by limiting restart processing

When a portion of the log is inaccessible during forward recovery, starting Db2 is possible. You need to identify the units of recovery for which database changes cannot be fully guaranteed (either committed or backed out). You also need to identify the page sets that these units of recovery changed.

About this task

You must determine which page sets are involved because after this procedure is used, the page sets will contain inconsistencies that you must resolve. In addition, using this procedure results in the completion of all database writes that are pending.

Related concepts

[Write operations \(Db2 Performance\)](#)

Task 1: Find the log RBA after the inaccessible part of the log

The first task in restarting Db2 by limiting restart processing is to locate the log RBA that is after the inaccessible part of the log.

About this task

The range of the log between RBAs X and Y is inaccessible to all Db2 processes.

Procedure

To find the RBA after the inaccessible part of the log, take the action that is associated with the message number that you received (DSNJ007I, DSNJ012I, DSNJ103I, DSNJ104I, DSNJ106I, and DSNJ113E):

- **When message DSNJ007I is issued:**

The problem is that an operator canceled a request for archive mount. Reason code 00D1032B is associated with this situation and indicates that an entire data set is inaccessible.

For example, the following message indicates that the archive log data set DSNCAT.ARCHLOG1.A0000009 is not accessible. The operator canceled a request for archive mount, resulting in the following message:

```
DSNJ007I OPERATOR CANCELED MOUNT OF ARCHIVE
          DSNCAT.ARCHLOG1.A0000009 VOLSER=5B225.
```

To determine the value of *X*, run the print log map utility (DSNJU004) to list the log inventory information. The output of this utility provides each log data set name and its associated log RBA range, the values of *X* and *Y*.

- **When message DSNJ012I is issued:**

The problem is that a log record is logically damaged. Message DSNJ012I identifies the log RBA of the first inaccessible log record that Db2 detects. The following reason codes are associated with this situation:

- 00D10261
- 00D10262
- 00D10263
- 00D10264
- 00D10265
- 00D10266
- 00D10267
- 00D10268
- 00D10348

For example, the following message indicates a logical error in the log record at log RBA X'7429ABA'.

```
DSNJ012I ERROR D10265 READING RBA 000007429ABA
          IN DATA SET DSNCAT.LOGCOPY2.DS01
          CONNECTION-ID=DSN,
          CORRELATION-ID=DSN
```

A given physical log record is actually a set of logical log records (the log records that are generally spoken of) and the log control interval definition (LCID). Db2 stores logical records in blocks of physical records to improve efficiency. When this type of an error on the log occurs during forward log recovery, all log records within the physical log record are inaccessible. Therefore, the value of *X* is the log RBA that was reported in the message, rounded down to a 4-KB boundary. (For the example message above, the rounded 4-KB boundary value would be X'7429000'.)

- **When message DSNJ103I or DSNJ104I is issued:**

For message DSNJ103I, the underlying problem depends on the reason code that is issued:

- For reason code 00D1032B, an allocation error occurred for an archive log data set.
- For reason code 00E80084, an active log data set that is named in the BSDS could not be allocated during log initialization.

For message DSNJ104I, the underlying problem is that an open error occurred for an archive and active log data set.

In any of these cases, the message that accompanies the abend identifies an entire data set that is inaccessible. For example, the following DSNJ103I message indicates that the archive log data set DSNCAT.ARCHLOG1.A0000009 is not accessible. The STATUS field identifies the code that is associated with the reason for the data set being inaccessible.

```
DSNJ103I - csect-name LOG ALLOCATION ERROR
          DSNAME=DSNCAT.ARCHLOG1.A0000009,ERROR
          STATUS=04980004
          SMS REASON CODE=reason-code
```

To determine the value of *X*, run the print log map utility (DSNJU004) to list the log inventory information. The output of the utility provides each log data set name and its associated log RBA range, the values of *X* and *Y*.

- **When message DSNJ106I is issued:**

The problem is that an I/O error occurred while a log record was being read. The message identifies the log RBA of the first inaccessible log record that Db2 detects. Reason code 00D10329 is associated with this situation.

For example, the following message indicates an I/O error in the log at RBA X'7429ABA'.

```
DSNJ106I LOG READ ERROR DSNAME=DSNCAT.LOGCOPY2.DS01,  
LOGRBA=000007429ABA,ERROR STATUS=0108320C
```

A given physical log record is actually a set of logical log records (the log records that are generally spoken of) and the log control interval definition (LCID). When this type of an error on the log occurs during forward log recovery, all log records within the physical log record, and beyond it to the end of the log data set, are inaccessible to the forward log recovery phase of restart. This is due to the log initialization or current status rebuild phase of restart. Therefore, the value of X is the log RBA that was reported in the message, rounded down to a 4-KB boundary. (For the example message above, the rounded 4-KB boundary value would be X'7429000'.)

- **When message DSNJ113E is issued:**

The problem is that the log RBA could not be found in the BSDS. Message DSNJ113E identifies the log RBA of the inaccessible log record. This log RBA is not registered in the BSDS. Reason code 00D1032B is associated with this situation.

For example, the following message indicates that the log RBA X'7429ABA' is not registered in the BSDS:

```
DSNJ113E RBA 000007429ABA NOT IN ANY ACTIVE OR ARCHIVE  
LOG DATA SET. CONNECTION-ID=DSN, CORRELATION-ID=DSN
```

Use the print log map utility (DSNJU004) to list the contents of the BSDS.

A given physical log record is actually a set of logical log records (the log records that are generally spoken of) and the log control interval definition (LCID). When this type of an error on the log occurs during forward log recovery, all log records within the physical log record are inaccessible.

Using the print log map output, locate the RBA that is closest to, but less than, X'7429ABA' for the value of X. If you do not find an RBA that is less than X'7429ABA', the value of X is zero. Locate the RBA that is closest to, but greater than, X'7429ABA'. This is the value of Y.

Related concepts

[Forward-log recovery failure \(Db2 Administration Guide\)](#)

Related reference

[DSNJU004 \(print log map\) \(Db2 Utilities\)](#)

Related information

[DSNJ007I \(Db2 Messages\)](#)

[DSNJ012I \(Db2 Messages\)](#)

[DSNJ103I \(Db2 Messages\)](#)

[DSNJ104I \(Db2 Messages\)](#)

[DSNJ106I \(Db2 Messages\)](#)

[DSNJ113E \(Db2 Messages\)](#)

Task 2: Identify incomplete units of recovery and inconsistent page sets

The second task in restarting Db2 by limiting restart processing is to identify incomplete units of recovery and inconsistent page sets.

About this task

Units of recovery that cannot be fully processed are considered *incomplete units of recovery*. Page sets that will be inconsistent after completion of restart are considered *inconsistent page sets*.

Procedure

To identify incomplete units of recovery and inconsistent page sets:

1. Determine the location of the latest checkpoint on the log by looking at one of the following sources, whichever is more convenient:
 - The operator's console contains the following message, identifying the location of the start of the last checkpoint on the log at log RBA X'876B355'. For example:

```
DSNR003I RESTART ... PRIOR CHECKPOINT  
RBA=00007425468
```

- The print log map utility output identifies the last checkpoint, including its BEGIN CHECKPOINT RBA
2. Obtain a report of the outstanding work that is to be completed at the next restart of Db2 by running the DSN1LOGP utility.

When you run the DSN1LOGP utility, specify the checkpoint RBA as the STARTRBA and the SUMMARY(ONLY) option. In order to obtain complete information, be sure to include the last complete checkpoint from running DSN1LOGP.

3. Analyze the output of the DSN1LOGP utility.

The summary report that is placed in the SYSSUMRY file contains two sections of information: a complete summary of completed events and a restart summary.

Related concepts

[DSN1LOGP summary report \(Db2 Administration Guide\)](#)

Related reference

[DSN1LOGP \(Db2 Utilities\)](#)

Task 3: Restrict restart processing to the part of the log after the damage

The third task in restarting Db2 by limiting restart processing is to restrict restart processing to the part of the log that is after the damage.

Procedure

To restrict restart processing to the part of the log after the damage:

1. Create a conditional restart control record (CRCR) in the BSDS by using the change log inventory utility.
2. Identify the accessible portion of the log beyond the damage by using the STARTRBA specification, which will be used at the next restart.
3. Specify the value Y+1 (that is, if Y is X'7429FFF', specify STARTRBA=742A000).

Restart restricts its processing to the portion of the log beginning with the specified STARTRBA and continuing to the end of the log.

For example:

```
CRESTART CREATE, STARTRBA=742A000
```

Task 4: Start Db2 and resolve inconsistent data

The final task in restarting Db2 by limiting restart processing is to start Db2 and resolve problems with inconsistent data.

About this task

At the end of restart, the CRCR is marked DEACTIVATED to prevent its use on a subsequent restart. Until the restart is complete, the CRCR will be in effect. Use START DB2 ACCESS(MAINT) until data is consistent or page sets are stopped.

Procedure

To start Db2 and resolve data inconsistencies:

1. Start Db2 with the following command:

```
-START DB2 ACCESS (MAINT)
```

At the end of restart, the conditional restart control record (CRCR) is marked "Deactivated" to prevent its use on a later restart. Until the restart completes successfully, the CRCR is in effect. Until data is consistent or page sets are stopped, start Db2 with the ACCESS (MAINT) option.

2. Resolve all data inconsistency problems.

Related tasks

[Resolving inconsistencies resulting from a conditional restart \(Db2 Administration Guide\)](#)

Recovering from a failure during backward log recovery

When a failure occurs during backward log recovery, Db2 terminates because it cannot access a portion of the log that it needs. Operations management can recover from this situation.

Symptoms

An abend is issued to indicate that restart failed because of a log problem. In addition, the last restart message that is received is a DSNR005I message, indicating that forward log recovery completed and that the failure occurred during backward log recovery.

Environment

Because a portion of the log is inaccessible, Db2 needs to roll back some database changes during restart.

Resolving the problem

Operations management response:

To start Db2, choose one of the following approaches:

- Read the information about relevant messages and codes that you received to determine if this approach is possible. The explanations of the messages and codes identify any corrective action that you can take to resolve the problem. If it is possible, correct the problem that made the log inaccessible, and start Db2 again.
- Restore the Db2 log and all data to a prior consistent point and start Db2. This procedure is described in [“Recovering from unresolvable BSDS or log data set problem during restart”](#) on page 101.
- Start Db2 without completing some database changes. Do this only if the exact changes cannot be identified; all that can be determined is which page sets might have incomplete changes. The procedure for determining which page sets contain incomplete changes is described in [“Bypassing backout before restarting”](#) on page 99. Other related topics might help you better understand the problem.

Backward log recovery failure

If a failure occurs during the backward-log recovery phase of restart, operations management can recover from this situation.

When a failure occurs during the backward log recovery phase, certain characteristics of the situation are evident, as the following figure shows.

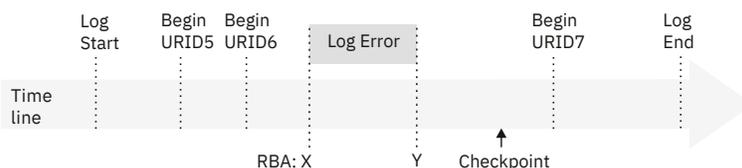


Figure 12. Illustration of failure during backward log recovery

The portion of the log between log RBA X and Y is inaccessible. The restart process was reading the log in a backward direction, beginning at the end of the log and continuing backward to the point marked by Begin URID5 in order to back out the changes that were made by URID5, URID6, and URID7. You can assume that Db2 determined that these units of recovery were inflight or in-abort. The portion of the log from point Y to the end of the log has been processed. However, the portion of the log from Begin URID5 to point Y has not been processed and cannot be processed by restart. Consequently, database changes that were made by URID5 and URID6 might not be fully backed out. All database changes made by URID7 have been fully backed out, but these database changes might not have been written to disk. A subsequent restart of Db2 causes these changes to be written to disk during forward recovery.

Related concepts

[Recommendations for changing the BSDS log inventory \(Db2 Administration Guide\)](#)

Bypassing backout before restarting

A portion of the log becomes inaccessible when a failure occurs during backward log recovery. Operations management can recover from this situation by starting Db2 in a certain way, and then identifying the page sets that are inconsistent because of the incomplete units of recovery.

Procedure

To bypass backout before recovery:

1. Determine the units of recovery that cannot be backed out and the page sets that will be inconsistent after the completion of restart.

- a) Determine the location of the latest checkpoint on the log by looking at one of the following sources, whichever is more convenient:

- The operator's console contains message DSNR003I, which identifies the location of the start of the last checkpoint on the log at log RBA X'7425468'.

```
DSNR003I RESTART ... PRIOR CHECKPOINT
RBA=00007425468
```

- The print log map utility output identifies the last checkpoint, including its BEGIN CHECKPOINT RBA.

- b) Obtain a report of the outstanding work that is to be completed at the next Db2 restart by running the DSN1LOGP.

When you run DSN1LOGP, specify the checkpoint RBA as the RBASTART and the SUMMARY(ONLY) option. Include the last complete checkpoint in the execution of DSN1LOGP in order to obtain complete information.

Analyze the output of the DSN1LOGP utility. The summary report that is placed in the SYSSUMRY file contains two sections of information. The heading of first section of the output is the following message:

```
DSN1150I SUMMARY OF COMPLETED EVENTS
```

That message is followed by other messages that identify completed events, such as completed units of recovery. That section of the output does not apply to this procedure.

The heading of the second section of the output is the following message:

```
DSN1157I RESTART SUMMARY
```

That message is followed by others that identify units of recovery that are not yet completed and the page sets that they modified. After the summary of outstanding units of recovery is a summary of page sets with database writes that are pending.

The restart processing that failed was able to complete all units of recovery processing within the accessible scope of the log after point Y. Database writes for these units of recovery are completed

during the forward recovery phase of restart on the next restart. Therefore, do not bypass the forward recovery phase. All units of recovery that can be backed out have been backed out.

All remaining units of recovery that are to be backed out (DISP=INFLIGHT or DISP=IN-ABORT) are bypassed on the next restart because their STARTRBA values are less than the RBA of point Y. Therefore, all page sets that were modified by those units of recovery are inconsistent after restart. This means that some changes to data might not be backed out. At this point, you only need to identify the page sets in preparation for restart.

2. Use the change log inventory utility to create a conditional restart control record (CRCR) in the BSDS, and direct restart to bypass backward recovery processing during the subsequent restart by using the BACKOUT specification.

At restart, all units of recovery that require backout are declared complete by Db2, and log records are generated to note the end of the unit of recovery.

The change log inventory utility control statement is:

```
CRESTART CREATE, BACKOUT=NO
```

3. Start Db2.

At the end of restart, the CRCR is marked "Deactivated" to prevent its use on a subsequent restart. Until the restart is complete, the CRCR is in effect. Use **START DB2 ACCESS(MAINT)** until data is consistent or page sets are stopped.

4. Resolve all inconsistent data problems. After the successful start of Db2, resolve all data inconsistency problems. [“Resolving inconsistencies resulting from a conditional restart” on page 107](#) describes how to do this. At this time, make all other data available for use.

Related concepts

[DSN1LOGP summary report \(Db2 Administration Guide\)](#)

Recovering from a failure during a log RBA read request

A failure might occur during a log RBA read request. For example, because of problems with the BSDS, the requested RBA, which contains the dropped log data set, cannot be read. You can recover from the situation.

Symptoms

Abend code 00D1032A is issued, and message DSNJ113E is displayed:

```
DSNJ113E RBA log-rba NOT IN ANY ACTIVE OR ARCHIVE  
LOG DATA SET. CONNECTION-ID=aaaaaaaa, CORRELATION-ID=aaaaaaaa
```

Causes

The BSDS is wrapping around too frequently when log RBA read requests are submitted; when the last archive log data sets were added to the BSDS, the maximum allowable number of log data sets in the BSDS was exceeded. This caused the earliest data sets in the BSDS to be displaced by the new entry. Subsequently, the requested RBA containing the dropped log data set cannot be read after the wrap occurs.

Resolving the problem

System programmer response:

1. Stop Db2 with the **STOP DB2** command, if it has not already been stopped automatically as a result of the problem.
2. Check any other messages and reason codes that are displayed, and correct the errors that are indicated. Locate the output from an old execution of the print log map utility, and identify the data set that contains the missing RBA. If the data set has not been reused, run the change log inventory utility to add this data set back into the inventory of log data sets.

3. Start Db2 with the **START DB2** command.

Related tasks

[Updating subsystem parameter and application default values \(Db2 Installation and Migration\)](#)

Related reference

[DSNJU003 \(change log inventory\) \(Db2 Utilities\)](#)

Recovering from unresolvable BSDS or log data set problem during restart

During a restart of Db2, serious problems with the bootstrap data set (BSDS) or log data sets might occur. However, operations management can recover from these problems. Use of dual logging (active logs, archive logs, and bootstrap data sets) can reduce your efforts in resolving these sorts of problems.

Symptoms

The following messages are issued:

- DSNJ100I
- DSNJ107I
- DSNJ119I

Causes

Any of the following problems might cause problems with the BSDS or log data sets during restart:

- A log data set is physically damaged.
- Both copies of a log data set are physically damaged in the case of dual logging mode.
- A log data set is lost.
- An archive log volume was reused even though it was still needed.
- A log data set contains records that are not recognized by Db2 because they are logically broken.

Environment

Db2 cannot be started until this procedure is performed.

Resolving the problem

Operations management response:

Serious cases such as this sometimes necessitate a fallback to a prior shutdown level.

- If you decide to fall back (because the amount of lost information is not excessive):
 1. See [Preparing to recover to a prior point of consistency \(Db2 Administration Guide\)](#).
 2. Follow the procedure in [Falling back to a prior shutdown point \(Db2 Administration Guide\)](#).

If you use do a fallback, all database changes between the shutdown point and the present are lost. However, all the data that is retained will be consistent within Db2.

- If you decide not to fall back (because too much log information has been lost), use the alternative approach that is described in [Recovering from a failure resulting from total or excessive loss of log data \(Db2 Administration Guide\)](#).

Falling back to a prior shutdown point

When a failure occurs in your environment, you might decide to fall back to a prior shutdown point.

Procedure

To fallback to a prior shutdown point:

1. Use the print log map utility on the most current copy of the BSDS.
Even if you are not able to do this, continue with the next step. (If you are unable to do this, an error message is issued.)
2. Use the access method services **IMPORT** command to restore the backed-up versions of the BSDS and active log data sets.
3. Use the print log map utility on the copy of the BSDS with which Db2 is to be restarted.
4. Determine whether any archive log data sets must be deleted.
 - If you have a copy of the most current BSDS, compare it to the BSDS with which Db2 is to be restarted. Delete and uncatalog any archive log data sets that are listed in the most current BSDS but are not listed in the previous one. These archive log data sets are normal physical sequential (SAM) data sets. If you are able to do this step, continue with step “5” on page 102.
 - If you were not able to print a copy of the most current BSDS and the archive logs are cataloged, use access method services LISTCAT to check for archive logs with a higher sequence number than the last archive log that is shown in the BSDS that is being used to restart Db2.
 - If no archive log data sets with a higher sequence number exist, you do not need to delete or uncatalog any data sets, and you can continue with step “5” on page 102.
 - Delete and uncatalog all archive log data sets that have a higher sequence number than the last archive log data set in the BSDS that is being used to restart Db2. These archive log data sets are SAM data sets. Continue with the next step.

If the archive logs are not cataloged, you do not need to uncatalog them.

5. Issue the **START DB2 ACCESS(MAINT)** command until data is consistent or page sets are stopped.
6. Determine what data needs to be recovered, what data needs to be dropped, what data can remain unchanged, and what data needs to be recovered to the prior shutdown point.
 - For table spaces and indexes that might have been changed after the shutdown point, use the Db2 RECOVER utility to recover these table spaces and indexes. They must be recovered in the proper order.
 - For data that has not been changed after the shutdown point (data used with RO access), you do not need to use RECOVER or DROP.
 - For table spaces that were deleted after the shutdown point, issue the DROP statement. These table spaces will not be recovered.
 - Re-create any objects that were created after the shutdown point.

You must recover all data that has potentially been modified after the shutdown point. If you do not use the RECOVER utility to recover modified data, serious problems might occur because of data inconsistency.

If you try to access inconsistent data, any of the following events can occur (and the list is not comprehensive):

- You can successfully access the correct data.
 - You can access data without Db2 recognizing any problem, but it might not be the data that you want (the index might be pointing to the wrong data).
 - Db2 might recognize that a page is logically incorrect and, as a result, abend the subsystem with an X'04E' abend completion code and an abend reason code of X'00C90102'.
 - Db2 might notice that a page was updated after the shutdown point and, as a result, abend the requester with an X'04E' abend completion code and an abend reason code of X'00C200C1'.
7. Analyze the CICS log and the IMS log to determine the work that must be redone (work that was lost because of shutdown at the previous point). Inform all users (TSO users, QMF users, and batch users for whom no transaction log tracking has been performed) about the decision to fall back to a previous point.
 8. When Db2 is started after being shut down, indoubt units of recovery might exist. This occurs if transactions are indoubt when the **STOP DB2 MODE (QUIESCE)** command is issued. When Db2

is started again, these transactions will still be indoubt to Db2. IMS and CICS cannot know the disposition of these units of recovery.

To resolve these indoubt units of recovery, use the **RECOVER INDOUBT** command.

9. If a table space was dropped and re-created after the shutdown point, drop and re-create it again after Db2 is restarted. To do this, use SQL DROP and SQL CREATE statements.

Do not use the RECOVER utility to accomplish this, because it will result in the old version (which might contain inconsistent data) that is being recovered.

10. If any table spaces and indexes were created after the shutdown point, re-create these after Db2 is restarted.

You can accomplish this in these ways:

- For data sets that are defined in Db2 storage groups, use the CREATE TABLESPACE statement, and specify the appropriate storage group names. Db2 automatically deletes the old data set and redefines a new one.
- For user-defined data sets, use the access method services **DELETE** command to delete the old data sets. After these data sets have been deleted, use the access method services **DEFINE** command to redefine them; then use the CREATE TABLESPACE statement.

Related reference

[RECOVER \(Db2 Utilities\)](#)

Recovering from a failure resulting from total or excessive loss of log data

If a situation occurs that causes the entire log or an excessive amount of log data to be lost or destroyed, operations management needs to recover from that situation.

Symptoms

This situation is generally accompanied by messages or abend reason codes that indicate that an excessive amount of log information, or the entire log, has been lost.

Diagnosing the problem

In this situation, you need to rely on your own sources to determine what data is inconsistent as a result of the failure; Db2 cannot provide any hints of inconsistencies. For example, you might know that Db2 was dedicated to a few processes (such as utilities) during the Db2 session, and you might be able to identify the page sets that they modified. If you cannot identify the page sets that are inconsistent, you must decide whether you are willing to assume the risk that is involved in restarting Db2 under those conditions.

Resolving the problem

Operations management response:

If you decide that a restart is needed, restart Db2 without any log data by using the appropriate procedure, depending on whether the log is totally or partially (but excessively) lost.

Recovering from a total loss of the log

If all system and user table spaces remain intact and you have a recent copy of the BSDS, you can recover from a total loss of the log. Db2 can still be restarted, and data that belongs to that Db2 subsystem can still be accessed.

Before you begin

All system and user table spaces must be intact, and you must have a recent copy of the BSDS. Other VSAM clusters on disk, such as the system databases DSNDB01, DSNDB04, and DSNB06, and also user databases are assumed to exist.

Procedure

To restart Db2 when the entire log is lost:

1. Define and initialize the BSDSs by recovering the BSDS from a backup copy.
2. Define the active log data sets by using the access method services **DEFINE** command. Run utility DSNJLOGF to initialize the new active log data sets.
3. Prepare to restart Db2 with no log data.

Each data and index page contains the log RBA of the last log record that was applied against the page. Safeguards within Db2 disallow a modification to a page that contains a log RBA that is higher than the current end of the log. You have two choices:

- Determine the highest possible log RBA of the prior log. From previous console logs that were written when Db2 was operational, locate the last DSNJ001I message. When Db2 switches to a new active log data set, this message is written to the console, identifying the data set name and the highest potential log RBA that can be written for that data set. Assume that this is the value X'8BFFF'. Add one to this value (X'8C000'), and create a conditional restart control record that specifies the following change log inventory control statement:

```
CRESTART CREATE,STARTRBA=8C000,ENDRBA=8C000
```

When Db2 starts, all phases of restart are bypassed, and logging begins at log RBA X'8C000'. If you choose this method, you do not need to use the RESET option of the DSN1COPY utility, and you can save a lot of time.

- Run the DSNJU003 utility, specifying the DELETE and NEWLOG options to delete and create new logs for all active log data sets.
- Run the DSN1COPY utility, specifying the RESET option to reset the log RBA in every data and index page. Depending on the amount of data in the subsystem, this process might take quite a long time. Because the BSDS has been redefined and reinitialized, logging begins at log RBA 0 when Db2 starts.

If the BSDS is not reinitialized, you can force logging to begin at log RBA 0 by constructing a conditional restart control record (CRCR) that specifies a STARTRBA and ENDRBA that are both equal to 0, as the following command shows:

```
CRESTART CREATE,STARTRBA=0,ENDRBA=0
```

4. Start Db2. Use the **START DB2 ACCESS(MAINT)** command until data is consistent or page sets are stopped.
5. After restart, resolve all inconsistent data as described in [“Resolving inconsistencies resulting from a conditional restart”](#) on page 107.

Related tasks

[Deferring restart processing \(Db2 Administration Guide\)](#)

[Recovering the BSDS from a backup copy \(Db2 Administration Guide\)](#)

Related reference

[DSNJLOGF \(preformat active log\) \(Db2 Utilities\)](#)

Recovering from an excessive loss of active log data

When your site experiences an excessive loss of active log data, you can develop a procedure for restarting in this situation. Do not redefine the BSDS.

About this task

You can recover from an excessive loss of active log data in one of two ways.

Recovering Db2 by creating a gap in the active log

If your site experiences an excessive loss of active log data, you can recover by creating a gap in the active log.

Procedure

To recover by creating a gap in the active log:

1. Use the print log map utility (DSNJU004) on the copy of the BSDS with which Db2 is to be restarted.
2. Use the print log map output to obtain the data set names of all active log data sets. Use the access method services **LISTCAT** command to determine which active log data sets are no longer available or usable.
3. Use the access method services **DELETE** command to delete all active log data sets that are no longer usable.
4. Use the access method services **DEFINE** command to define new active log data sets. Run the DSNJLOGF utility to initialize the new active log data sets. Define one active log data set for each one that is found to be no longer available or usable in step “2” on page 105. Use the active log data set name that is found in the BSDS as the data set name for the access method services **DEFINE** command.
5. Refer to the print log map utility (DSNJU004) output, and note whether an archive log data set exists that contains the RBA range of the redefined active log data set.

To do this, note the starting and ending RBA values for the active log data set that was recently redefined, and look for an archive log data set with the same starting and ending RBA values.

If no such archive log data sets exist:

- a) Use the change log inventory utility (DSNJU003) **DELETE** statement to delete the recently redefined active log data sets from the BSDS active log data set inventory.
- b) Use the change log inventory utility (DSNJU003) **NEWLOG** statement to add the active log data set to the BSDS active log data set inventory. Do not specify RBA ranges on the **NEWLOG** statement.

If the corresponding archive log data sets exist, two courses of action are available:

- If you want to minimize the number of potential read operations on the archive log data sets, use the access method services **REPRO** command to copy the data from each archive log data set into the corresponding active log data set. Ensure that you copy the proper RBA range into the active log data set.

Ensure that the active log data set is large enough to hold all the data from the archive log data set. When Db2 does an archive operation, it copies the log data from the active log data set to the archive log data set, and then pads the archive log data set with binary zeros to fill a block. In order for the access method services **REPRO** command to be able to copy all of the data from the archive log data set to a recently defined active log data set, the new active log data set might need to be larger than the original one.

For example, if the block size of the archive log data set is 28 KB, and the active log data set contains 80 KB of data, Db2 copies the 80 KB and pads the archive log data set with 4 KB of nulls to fill the last block. Thus, the archive log data set now contains 84 KB of data instead of 80 KB. In order for the access method services **REPRO** command to complete successfully, the active log data set must be able to hold 84 KB, rather than just 80 KB of data.

- If you are not concerned about read operations against the archive log data sets, complete the two steps that appear in the steps “5.a” on page 105 and “5.b” on page 105 (as though the archive data sets did not exist).
6. Choose the appropriate point for Db2 to start logging.

To do this, determine the highest possible log RBA of the prior log. From previous console logs that were written when Db2 was operational, locate the last DSNJ001I message. When Db2 switches to a new active log data set, this message is written to the console, identifying the data set name and the highest potential log RBA that can be written for that data set. Assume that this is the value X'8BFFF'.

Add one to this value (X'8C000'), and create a conditional restart control record that specifies the following change log inventory control statement:

```
CRESTART CREATE,STARTRBA=8C000,ENDRBA=8C000
```

When Db2 starts, all phases of restart are bypassed, and logging begins at log RBA X'8C000'. If you choose this method, you do not need to use the RESET option of the DSN1COPY utility, and you can save a lot of time.

7. To restart Db2 without using any log data, create a conditional restart control record for the change log inventory utility (DSNJU003).
8. Start Db2. Use the **START DB2 ACCESS(MAINT)** command until data is consistent or page sets are stopped.
9. After restart, resolve all inconsistent data as described in [“Resolving inconsistencies resulting from a conditional restart”](#) on page 107.

Results

This procedure causes all phases of restart to be bypassed and logging to begin at the point in the log RBA that you identified in step “6” on page 105 (X'8C000' in the example given in this procedure). This procedure creates a gap in the log between the highest RBA kept in the BSDS and, in this example, X'8C000', and that portion of the log is inaccessible.

What to do next

Because no Db2 process can tolerate a gap, including RECOVER, you need to take image copies of all data after a cold start, even data that you know is consistent.

Related reference

[DSNJU003 \(change log inventory\) \(Db2 Utilities\)](#)

Recovering Db2 without creating a gap in the active log

You can do a cold start without creating a gap in the log. Although this approach does eliminate the gap in the physical log record, you cannot use a cold start to resolve the logical inconsistencies.

Procedure

To recover without creating a gap in the active log:

1. Locate the last valid log record by using the DSN1LOGP utility to scan the log.
Message DSN1213I identifies the last valid log RBA.
2. Identify the last RBA that is known to be valid by examining message DSN1213I.
For example, if message DSN1213I indicates that the last valid log RBA is X'89158', round this value up to the next 4-KB boundary, which in this example is X'8A000'.
3. Create a conditional restart control record (CRCR).
For example:

```
CRESTART CREATE,STARTRBA=8A000,ENDRBA=8A000
```

4. Start Db2 with the **START DB2 ACCESS(MAINT)** command until data is consistent or page sets are stopped.
5. Take image copies of all data for which data modifications were recorded beyond the log RBA that was used in the CRESTART statement (in this example, X'8A000'). If you do not know what data was modified, take image copies of all data.

If you do not take image copies of data that has been modified beyond the log RBA that was used in the CRESTART statement, future RECOVER utility operations might fail or result in inconsistent data.

What to do next

After restart, resolve all inconsistent data as described in [“Resolving inconsistencies resulting from a conditional restart” on page 107](#).

Resolving inconsistencies resulting from a conditional restart

When a conditional restart of the Db2 subsystem is done, several problems might occur. Recovery from these problems is possible and varies based on the specific situation.

About this task

The following problems might occur after a conditional restart of Db2:

- Some amount of work is left incomplete.
- Some data is left inconsistent.
- Information about the status of objects within the Db2 subsystem is made unusable.

Inconsistencies in a distributed environment

In a distributed environment, when a Db2 subsystem restarts, Db2 indicates its restart status and the name of its recovery log to the systems that it communicates with. The two possible conditions for restart status are warm and cold.

A cold status is associated with a *cold start*, which is a process by which a Db2 subsystem restarts without processing any log records. Db2 has no memory of previous connections with its partner. The general process that occurs with a cold start in a distributed environment is as follows:

1. The partner (for example CICS) accepts the cold start connection and remembers the recovery log name of the Db2 subsystem that experienced the cold start.
2. If the partner has indoubt thread resolution requirements with the cold-starting Db2 subsystem, those requirements cannot be satisfied.
3. The partner terminates its indoubt resolution responsibility with the cold-starting Db2 subsystem. However, as a participant, the partner has indoubt logical units of work that must be resolved manually.
4. Because the Db2 subsystem has an incomplete record of resolution responsibilities, Db2 attempts to reconstruct as much resynchronization information as possible.
5. Db2 displays the information that it was able to reconstruct in one or more DSNL438 or DSNL439 messages.
6. Db2 then discards the synchronization information that it was able to reconstruct and removes any restrictive states that are maintained on the object.

Resolving inconsistencies

In some problem situations, you need to determine what you must do in order to resolve any data inconsistencies that exist.

Procedure

To resolve inconsistencies:

1. Determine the scope of any inconsistencies that are introduced by the situation.
 - a) If the situation is either a cold start that is beyond the current end of the log or a conditional restart that skips backout or forward log recovery, use the DSN1LOGP utility to determine what units of work have not been backed out and which objects are involved.
For a cold start that is beyond the end of the log, you can also use DSN1LOGP to help identify any restrictive object states that have been lost.

b) If a conditional restart truncates the log in a non-data sharing environment, recover all data and indexes to the new current point in time, and rebuild the data and indexes as needed.

You need to recover or rebuild (or both recover and rebuild) the data and indexes because data and index updates might exist without being reflected in the Db2 log. When this situation occurs, a variety of inconsistency errors might occur, including Db2 abends with reason code 00C200C1.

2. Decide what approach to take to resolve inconsistencies by reading related topics about the approaches:

- Recovery to a prior point of consistency
- Restoration of a table space
- Use of the REPAIR utility on the data

The first two approaches are less complex than, and therefore preferred over, the third approach.

3. If one or more of the following conditions are applicable, take image copies of all Db2 table spaces:

- You did a cold start.
- You did a conditional restart that altered or truncated the log.
- The log is damaged.
- Part of the log is no longer accessible.

When a portion of the Db2 recovery log becomes inaccessible, all Db2 recovery processes have difficulty operating successfully, including restart, RECOVER, and deferred restart processing. CONDITIONAL restart allows circumvention of the problem during the restart process. To ensure that RECOVER does not attempt to access the inaccessible portions of the log, secure a copy (either full or incremental) that does not require such access. A failure occurs any time a Db2 process (such as the RECOVER utility) attempts to access an inaccessible portion of the log. You cannot be sure which Db2 processes must use that portion of the recovery log. Therefore, you need to assume that all data recovery activity requires that portion of the log.

A cold start might cause down-level page set errors, which you can find out about in different ways:

- Message DSNB232I is sometimes displayed during Db2 restart, once for each down-level page set that Db2 detects. After you restart Db2, check the console log for down-level page set messages.
 - If a small number of those messages exist, run DSN1COPY with the RESET option to correct the errors to the data before you take image copies of the affected data sets.
 - If a large number of those messages exist, the actual problem is not that page sets are down-level but that the conditional restart inadvertently caused a high volume of DSNB232I messages. In this case, temporarily turn off down-level detection by turning off the DLDFREQ ZPARM.

In either case, continue with step [“4” on page 108](#).

- If you run the COPY utility with the SHRLEVEL REFERENCE option to make image copies, the COPY utility sometimes issues message DSNB232I about down-level page sets that Db2 does not detect during restart. If any of those messages were issued when you are making image copies, correct the errors, and continue making image copies of the affected data sets.
 - If you use some other method to make image copies, you will find out about down-level page set errors during normal operation. In this case, you need to correct the errors by using the information in [“Recovering from a down-level page set problem ” on page 116](#).
4. For any Db2 (catalog and directory) system table spaces that are inconsistent, recover them in the proper order. You might need to recover to a prior point in time, prior to the conditional restart.
5. For any objects that you suspect might be inconsistent, resolve the database inconsistencies before proceeding.
- First, resolve inconsistencies in Db2 system databases DSNDB01 and DSNDB06. Catalog and directory inconsistencies need to be resolved before inconsistencies in other databases because the subsystem databases describe all other databases, and access to other databases requires information from DSNDB01 and DSNDB06.

- If you determine that the existing inconsistencies involve indexes only (not data), use the REBUILD INDEX utility to rebuild the affected indexes. Alternatively, you can use the RECOVER utility to recover the index if rebuilding the indexes is not possible.
- For a table space that cannot be recovered (and is thus inconsistent), determine the importance of the data and whether it can be reloaded. If the data is not critical or can be reloaded, drop the table after you restart Db2, and reload the data rather than trying to resolve the inconsistencies.

Related concepts

[Recovery of data to a prior point in time \(Db2 Administration Guide\)](#)

Related tasks

[Recovering catalog and directory objects \(Db2 Utilities\)](#)

Related reference

[LEVELID UPDATE FREQ field \(DLDFREQ subsystem parameter\) \(Db2 Installation and Migration\)](#)

[DSN1COPY \(Db2 Utilities\)](#)

[RECOVER \(Db2 Utilities\)](#)

Restoring the table space

You can restore the table space by reloading data into it or by re-creating the table space, which requires advance planning. Either of these methods is easier than using REPAIR.

About this task

Reloading the table space is the preferred approach, when it is possible, because reloading is easier and requires less advance planning than re-creating a table space. Re-creating a table space involves dropping and then re-creating the table space and associated tables, indexes, authorities, and views, which are implicitly dropped when the table space is dropped. Therefore, re-creating the objects means that you need to plan ahead so that you will be prepared to re-establish indexes, views, authorizations, and the data content itself.

Restriction:

You cannot drop Db2 system tables, such as the catalog and directory. For these system tables, follow one of these procedures instead of this one:

- [Recovery of data to a prior point in time \(Db2 Administration Guide\)](#)
- [“Using the REPAIR utility on inconsistent data” on page 110](#)

Procedure

To restore the table space:

1. Decide whether you can reload the table space or must drop and re-create it.
 - If you can reload the table space, run the appropriate LOAD utility jobs to do so; specify the REPLACE option. After you load the content of the table space, skip to step [“6” on page 109](#).
 - If you cannot reload the table space, continue with step [“2” on page 109](#).
2. Issue an SQL DROP TABLESPACE statement for the table space that is suspected of being involved in the problem.
3. Re-create the table space, tables, indexes, synonyms, and views by using SQL CREATE statements.
4. Grant access to these objects the same way that access was granted prior to the time of the error.
5. Reconstruct the data in the tables.
6. Run the RUNSTATS utility on the data.
7. Use the COPY utility to acquire a full image copy of all data.
8. Use the **REBIND** command on all plans that use the tables or views that are involved in this activity.

Related concepts

[Recovery of data to a prior point in time \(Db2 Administration Guide\)](#)

Related tasks

[Using the REPAIR utility on inconsistent data \(Db2 Administration Guide\)](#)

Related reference

[LOAD \(Db2 Utilities\)](#)

[COPY \(Db2 Utilities\)](#)

Using the REPAIR utility on inconsistent data

You can resolve inconsistencies with the REPAIR utility. However, using REPAIR is not recommended unless the inconsistency is limited to a small number of data or index pages.

About this task

Db2 does not provide a mechanism to automatically inform users about data that is physically inconsistent or damaged. When you use SQL to access data that is physically damaged, Db2 issues messages to indicate that data is not available due to a physical inconsistency.

However, Db2 includes several utilities that can help you identify data that is physically inconsistent before you try to access it. These utilities are:

- CHECK DATA
- CHECK INDEX
- CHECK LOB
- COPY with the CHECKPAGE option
- DSN1COPY with the CHECK option



Attention: If you decide to use this method to resolve data inconsistencies, use extreme care. Use of the REPAIR utility to correct inconsistencies requires in-depth knowledge of Db2 data structures. Incorrect use of the REPAIR utility can cause further corruption and loss of data. Read this topic carefully because it contains information that is important to the successful resolution of the inconsistencies.

Recommendation: Avoid using this procedure if you are experiencing extensive data inconsistency because it is more time-consuming and complex (and therefore prone to error) than recovering to a point in time or re-creating the table spaces. If possible, use those alternative procedures instead.

Restrictions:

- Although the DSN1LOGP utility can identify page sets that contain inconsistencies, this utility cannot identify the specific data modifications that are involved in the inconsistencies within a given page set.
- Any pages that are on the logical page list (perhaps caused by this restart) cannot be accessed by using the REPAIR utility.

Procedure

To use the REPAIR utility to resolve the inconsistency:

1. Issue the following command to start Db2 and allow access to data:

```
START DATABASE (dbase) SPACENAM (space) ACCESS(FORCE)
```

In this command, *space* identifies the table space that is involved.

2. If any system data is inconsistent, use the REPAIR utility to resolve those inconsistencies.
Db2 system data (such as data that is in the catalog and directory) exists in interrelated tables and table spaces. Data in Db2 system databases cannot be modified with SQL, so use of the REPAIR utility is necessary to resolve the inconsistencies that are identified.
3. Determine if you have any structural violations in data pages.
Db2 stores data in data pages. The structure of data in a data page must conform to a set of rules for Db2 to be able to process the data accurately. Using a conditional restart process does not cause

violations to this set of rules; but, if violations existed prior to conditional restart, they continue to exist after conditional restart.

4. Use the DSN1COPY utility with the CHECK option to identify any violations that you detected in the previous step, and then resolve the problems, possibly by recovering or rebuilding the object or by dropping and re-creating it.
5. Examine the various types of pointers that Db2 uses to access data (indexes and hashes), and identify inconsistencies that need to be manually corrected.

Hash pointers exist in the Db2 directory database. Db2 uses these pointers to access data. During a conditional restart, data pages might be modified without update of the corresponding pointers. When this occurs, one of the following actions might occur:

- If a pointer addresses data that is nonexistent or incorrect, Db2 abends the request. If SQL is used to access the data, a message that identifies the condition, and the page in question is issued.
- If data exists but no pointer addresses it, that data is virtually invisible to all functions that attempt to access it by using the damaged hash pointer. The data might, however, be visible and accessible by some functions, such as SQL functions that use another pointer that was not damaged. This situation can result in inconsistencies.

If a row that contains a varying-length field is updated, it can increase in size. If the page in which the row is stored does not contain enough available space to store the additional data, the row is placed in another data page, and a pointer to the new data page is stored in the original data page. After a conditional restart, one of the following conditions might exist.

- The row of data exists, but the pointer to that row does not exist. In this case, the row is invisible, and the data cannot be accessed.
 - The pointer to the row exists, but the row itself no longer exists. Db2 abends the requester when any operation (for instance, a SELECT) attempts to access the data. If termination occurs, one or more messages are issued to identify the condition and the page that contains the pointer.
6. Use the REPAIR utility to resolve any inconsistencies that you detected in the previous step.
 7. Reset the log RBA in every data and index page set that are to be corrected with this procedure by using the DSN1COPY RESET option.

This step is necessary for the following reason. If the log was truncated, changing data by using the REPAIR utility can cause problems. Each data and index page contains the log RBA of the last recovery log record that was applied against the page. Db2 does not allow modification of a page that contains a log RBA that is higher than the current end of the log.

8. When all known inconsistencies have been resolved, take full image copies of all modified table spaces, in order to use the RECOVER utility to recover from any future problems.

This last step is imperative.

Related concepts

[Recovery of data to a prior point in time \(Db2 Administration Guide\)](#)

Related tasks

[Restoring the table space \(Db2 Administration Guide\)](#)

Related reference

[REPAIR \(Db2 Utilities\)](#)

Recovering from Db2 database failure

If a Db2 failure occurs because of an allocation or open problem, you can recover from this situation.

Symptoms

The symptoms vary based on whether the failure was an allocation or an open problem:

Allocation problem

The following message indicates an allocation problem:

DSNB207I - DYNAMIC ALLOCATION OF DATA SET FAILED.
REASON=*rrrr* DSNAME=*dsn*

The *rrrr* is a z/OS dynamic allocation reason code.

Open problem

The following messages indicate an open problem:

IEC161I *rc*[(*sfi*)] - *ccc, iii, sss, ddn,*
ddd, ser, xxx, dsn, cat

DSNB204I - OPEN OF DATA SET FAILED. DSNAME = *dsn*

In the IEC161I message:

rc

Is a return code.

sfi

Is subfunction information, which is displayed only with certain return codes.

ccc

Is a function code.

iii

Is a job name.

sss

Is a step name.

ddn

Is a DD name.

ddd

Is a device number (if the error is related to a specific device).

ser

Is a volume serial number (if the error is related to a specific volume).

xxx

Is a VSAM cluster name.

dsn

Is a data set name.

cat

Is a catalog name.

Environment

When this type of problem occurs:

- The table space is automatically stopped.
- Programs receive a -904 SQLCODE (SQLSTATE '57011').
- If the problem occurs during restart, the table space is marked for deferred restart, and restart continues. The changes are applied later when the table space is started.

Resolving the problem

Operator response:

1. Check reason code, and correct problems.
2. Ensure that drives are available for allocation.
3. Enter the command **START DATABASE**.

Related reference

[MVS Authorized Assembler Services Guide](#)

Recovering a Db2 subsystem to a prior point in time

You can recover a Db2 subsystem and data sharing group to a prior point in time by using the BACKUP SYSTEM and RESTORE SYSTEM utilities.

About this task

In this recovery procedure, you create and populate a table that contains data that is both valid and invalid. You need to restore your Db2 subsystem to a point in time before the invalid data was inserted into the table, but after the point in time when the valid data was inserted. Also, you create an additional table space and table that Db2 will re-create during the log-apply phase of the restore process.

Procedure

To insert data into a table, determine the point in time that you want to recover to, and then recover the Db2 subsystem to a prior point in time:

1. Issue the **START DB2** command to start Db2 and all quiesced members of the data sharing group.
Quiesced members are ones that you removed from the data sharing group either temporarily or permanently. Quiesced members remain dormant until you restart them.
2. Issue SQL statements to create a database, a table space, and two tables with one index for each table.
3. Issue the BACKUP SYSTEM DATA ONLY utility control statement to create a backup copy of only the database copy pool for a Db2 subsystem or data sharing group.
4. Issue an SQL statement to first insert rows into one of the tables, and then update some of the rows.
5. Use the LOAD utility with the LOG NO attribute to load the second table.
6. Issue SQL statements to create an additional table space, table, and index in an existing database.
Db2 will re-create the additional table space and table during the log-apply phase of the restore process.
7. Issue the SET LOG SUSPEND command or the SET LOG RESUME command to obtain a log truncation point, *logpoint1*, which is the point you want to recover to.
For a non-data sharing group, use the RBA value. For a data sharing group, use the lowest log record sequence number (LRSN) from the active members.

The following example shows sample output for the SET LOG SUSPEND command:

```
14.21.49          -db2aset log suspend
14.21.49 STC00059 DSN9022I  -DB2A DSNJC001 '-SET LOG' NORMAL COMPLETION
14.21.50 STC00059 *DSNJ372I  -DB2A DSNJC09A UPDATE ACTIVITY HAS BEEN
SUSPENDED FOR DB2A AT RBA 00000000000028B5588C, LRSN
00CA2981028F3D000000, PRIOR CHECKPOINT RBA 00000000000028B52667
```

8. Issue an SQL statement to first insert rows into one of the tables and then to update and delete some rows.
9. Issue the **STOP DB2** command to stop Db2 and all active members of the data sharing group.
10. Run the DSNJU003 change log inventory utility to create a SYSPITR CRCR record (CRESTART CREATE SYSPITR=*logpoint1*).
The log truncation point is the value that you obtained from issuing either the **SET LOG SUSPEND** command, or the **SET LOG RESUME** command.
11. For a data sharing group, delete all of the coupling facility structures.
12. Issue the **START DB2** command to restart Db2 and all members of the data sharing group.
13. Run the RESTORE SYSTEM utility.
For a data sharing group, this utility can be run only on one member. If the utility stops and you must restart it, you can restart the utility only on the member on which it was initially run.
14. After the RESTORE SYSTEM utility completes successfully, issue the **STOP DB2** command to stop Db2 and all active members of the data sharing group.

The Db2 subsystem resets to RECOVER-pending status.

15. Issue the **START DB2** command to restart Db2 and all members of the data sharing group.
16. Issue the **DISPLAY** command to identify the utilities that are active and the objects that are restricted.
For example:

```
-DIS UTIL(*)
```

```
-DIS DB(DSNDB01) SP(*)
```

```
-DIS DB(DSNDB06) SP(*) LIMIT(*)
```

```
-DIS DB(DSNDB06) SP(*) LIMIT(*)RESTRICT
```

17. Stop all of the active utilities that you identified in the previous step.
18. Recover any objects that are in RECOVER-pending status or REBUILD-pending status from the table that you created in step “6” on page 113.

Recovering the catalog and directory to a point in time before a CATMAINT or a function level upgrade in a data sharing environment

If you are in a data sharing environment, and you need to recover the catalog and directory to a point in time before CATMAINT was run or a function level upgrade was performed, you need to follow a special procedure.

Procedure

1. Create image copies of the Db2 catalog and directory.
2. Determine the point in time to which you need to recover, which is called xxxxxxxx in this procedure.
3. Run the following DIAGNOSE utility control statement to clear the catalog and function level information from an internal control structure:

```
DIAGNOSE TYPE(0) REPAIR SET DBID(8) PSID(0000) RESET
```

Important: This control statement should be issued only as part of this procedure. DBID(8) identifies an internal control structure for system database information. It is not a Db2 catalog database or any physical database provided by Db2.

4. Stop all members of the data sharing group, and issue the following SETXCF FORCE command to delete the coupling facility structures for the data sharing group:

```
SETXCF FORCE,STRUCTURE,STRNAME=XXXX_SCA(sca-structure-name)
```

5. If xxxxxxxx is prior to execution of the CATMAINT utility or activation of a function level, restore the BSDS from the most recent archived copy of the BSDS after LRSN xxxxxxxx, and before the CATMAINT or function level activation occurred. If that is not possible, take these actions:
 - Restore the most recent BSDS prior to xxxxxxxx.
 - Run DSNJU003 to update the archive log and active log information to include logs up to xxxxxxxx.
6. Create a conditional restart control record (CRCR) with an ENDLRSN value of xxxxxxxx.
7. For one member of the data sharing group, set the DEFER and ALL subsystem parameters, and start Db2 on that member with ACCESS(MAINT).
8. Recover the catalog and directory table spaces without the TOLOGPOINT option in the RECOVER utility control statements. Rebuild the catalog and directory indexes with the REBUILD INDEX utility. This step recovers the catalog and directory table spaces to the log truncation point of xxxxxxxx.

9. Recover all user objects.

This step keeps the definitions of user objects synchronized with the information on those objects in the catalog or directory after CATMAINT was run or a function level upgrade was performed.

10. Stop Db2 on the data sharing member on which you previously started Db2.
11. Set the RESTART subsystem parameter, and start Db2 without the ACCESS(MAINT) option on the data sharing member on which you stopped Db2 in the previous step.
12. Start Db2 on the other members of the data sharing group.

Related tasks

[Installation step 23: Back up the Db2 directory and catalog: DSNTIJIC \(Db2 Installation and Migration\)](#)

Related reference

[DSNJU003 \(change log inventory\) \(Db2 Utilities\)](#)

[RESTART OR DEFER field \(RESTART subsystem parameter\) \(Db2 Installation and Migration\)](#)

[START NAMES field \(ALL subsystem parameter\) \(Db2 Installation and Migration\)](#)

[-STOP DB2 command \(Db2\) \(Db2 Commands\)](#)

[-START DB2 command \(Db2\) \(Db2 Commands\)](#)

Recovering the catalog and directory to a point in time before a CATMAINT or a function level upgrade in a non-data sharing environment

If you are in a non-data sharing environment, and you need to recover the catalog and directory to a point in time before CATMAINT was run or a function level upgrade was performed, you need to follow a special procedure.

Procedure

1. Create image copies of the Db2 catalog and directory.
2. Determine the point in time to which you need to recover, which is called xxxxxxxx in this procedure.
3. Run the following DIAGNOSE utility control statement to clear the catalog and function level information from an internal control structure:

```
DIAGNOSE TYPE(0) REPAIR SET DBID(8) PSID(0000) RESET
```

Important: This control statement should be issued only as part of this procedure. DBID(8) identifies an internal control structure for system database information. It is not a Db2 catalog database or any physical database provided by Db2.

4. Stop Db2.
5. If xxxxxxxx is prior to execution of the CATMAINT utility or activation of a function level, restore the BSDS from the most recent archived copy of the BSDS after RBA xxxxxxxx, and before the CATMAINT or function level activation occurred. If that is not possible, take these actions:
 - Restore the most recent BSDS prior to xxxxxxxx.
 - Run DSNJU003 to update the archive log and active log information to include logs up to xxxxxxxx.
6. Create a conditional restart control record (CRCR) with an ENDRBA value of xxxxxxxx.
7. Set the DEFER and ALL subsystem parameters, and start Db2 with the ACCESS(MAINT) option.
8. Recover the catalog and directory table spaces without the TOLOGPOINT option in the RECOVER utility control statements. Rebuild the catalog and directory indexes with the REBUILD INDEX utility. This step recovers the catalog and directory table spaces to the log truncation point of xxxxxxxx.
9. Recover all user objects.

This step keeps the definitions of user objects synchronized with the information on those objects in the catalog or directory after CATMAINT was run or a function level upgrade was performed.

10. Stop Db2.

11. Set the RESTART subsystem parameter, and start Db2 without the ACCESS(MAINT) option.

Related tasks

[Installation step 23: Back up the Db2 directory and catalog: DSNTIJIC \(Db2 Installation and Migration\)](#)

Related reference

[DSNJU003 \(change log inventory\) \(Db2 Utilities\)](#)

[RESTART OR DEFER field \(RESTART subsystem parameter\) \(Db2 Installation and Migration\)](#)

[START NAMES field \(ALL subsystem parameter\) \(Db2 Installation and Migration\)](#)

[-STOP DB2 command \(Db2\) \(Db2 Commands\)](#)

[-START DB2 command \(Db2\) \(Db2 Commands\)](#)

Recovering from a down-level page set problem

When using a stand-alone utility or a non-Db2 utility, you might inadvertently replace a Db2 page set with an incorrect or outdated copy. This type of copy is called *down-level*. Using a down-level page set can cause complex problems; therefore, you need to recover from this situation.

Symptoms

The following message is issued:

```
DSNB232I csect-name - UNEXPECTED DATA SET LEVEL ID ENCOUNTERED
```

The message also contains the level ID of the data set, the level ID that Db2 expects, and the name of the data set.

Causes

A down-level page set can be caused by:

- A Db2 data set is inadvertently replaced by an incorrect or outdated copy. Usually this happens in conjunction with use of a stand-alone or non-Db2 utility, such as DSN1COPY or DFSMSHsm.
- A cold start of Db2 occurs.
- A VSAM high-used RBA of a table space becomes corrupted.

Db2 associates a level ID with every page set or partition. Most operations detect a down-level ID, and return an error condition, when the page set or partition is first opened for mainline or restart processing. The exceptions are the following operations, which do not use the level ID data:

- LOAD REPLACE
- RECOVER
- REBUILD INDEX
- DSN1COPY
- DSN1PRNT

Environment

- If the error was reported during mainline processing, Db2 sends a "resource unavailable" SQLCODE and a reason code to the application to explain the error.
- If the error was detected while a utility was processing, the utility generates a return code 8.

Diagnosing the problem

Determine whether the message was issued during restart or at some other time during normal operation. This information is important for determining what steps to take below

Resolving the problem

System programmer response: The actions that you need to do to recover depend on when the message was issued:

- If the message was issued during restart, take one of the following actions:
 - Replace the data set with one that is at the proper level, by using DSN1COPY, DFSMSHsm, or some equivalent method. To check the level ID of the new data set, run the stand-alone utility DSN1PRNT on it, with the options PRINT(0) (to print only the header page) and FORMAT. The formatted print output identifies the level ID.
 - Recover the data set to the current time, or to a prior time, by using the RECOVER utility.
 - Replace the contents of the data set, by using LOAD REPLACE.
- If the message was issued during normal operation (not during restart):
 1. Take one of the actions that are listed for situations when the message was issued during restart.
 2. Accept the down-level data set by changing its level ID. You can use the REPAIR utility with the LEVELID statement. (You cannot use the LEVELID option in the same job step with any other REPAIR utility control statement.)



Attention: If you accept a down-level data set or disable down-level detection, your data might be inconsistent.

Related system programmer actions:

Consider taking the following actions, which might help you minimize or deal with down-level page set problems in the future:

- To control how often the level ID of a page set or partition is updated, specify a value in the range 0–32767 on the LEVELID UPDATE FREQ field of panel DSNTIPL.
- To disable down-level detection, specify 0 in the LEVELID UPDATE FREQ field of panel DSNTIPL.
- To control how often level ID updates are taken, specify a value in the range 1–32767.

Related reference

[LEVELID UPDATE FREQ field \(DLDFREQ subsystem parameter\) \(Db2 Installation and Migration\)](#)

[DSN1COPY \(Db2 Utilities\)](#)

[DSN1PRNT \(Db2 Utilities\)](#)

[LOAD \(Db2 Utilities\)](#)

[RECOVER \(Db2 Utilities\)](#)

[REPAIR \(Db2 Utilities\)](#)

Recovering from a problem with invalid LOBs

If a LOB table space is defined with LOG NO and you need to recover that table space, you can recover the LOB data to the point at which you made your last image copy of the table space.

About this task

Unless your LOBs are fairly small, specifying LOG NO for LOB objects is recommended for the best performance. However, to maximize recoverability, specifying LOG YES is recommended. The performance cost of logging exceeds the benefits that you can receive from logging such large amounts of data. If no changes are made to LOB data, the logging cost is not an issue. However, you should make image copies of the LOB table space to prepare for failures. The frequency with which you make image copies is based on how often you update LOB data.

Procedure

To recover LOB data from a LOB table space that is defined with LOG NO:

1. Run the RECOVER utility as you do for other table spaces:

```
RECOVER TABLESPACE dbname.lobts
```

If changes were made after the image copy, Db2 puts the table space in auxiliary warning status, which indicates that some of your LOBs are invalid. Applications that try to retrieve the values of those LOBs receive SQLCODE -904. Applications can still access other LOBs in the LOB table space.

2. Get a report of the invalid LOBs by running CHECK LOB on the LOB table space:

```
CHECK LOB TABLESPACE dbname.lobts
```

Db2 generates the following messages:

```
LOB WITH ROWID = 'xxxxxxx' VERSION = n IS INVALID
```

3. **GUIP** Fix the invalid LOBs, by updating the LOBs or setting them to the null value. For example, suppose that you determine from the CHECK LOB utility that the row of the EMP_PHOTO_RESUME table with ROWID X'C1BDC4652940D40A81C201AA0A28' has an invalid value for column RESUME. If host variable *hvlob* contains the correct value for RESUME, you can use this statement to correct the value:

```
UPDATE DSN8D10. EMP_PHOTO_RESUME  
SET RESUME = :hvlob  
WHERE EMP_ROWID = ROWID(X'C1BDC4652940D40A81C201AA0A28');
```

GUIP

Recovering from table space I/O errors

You can recover a table space after I/O errors have occurred and caused the table space to fail.

Symptoms

The following message is issued, where *ddddddd* is a table space name:

```
DSNU086I DSNUCDA1 READ I/O ERRORS ON SPACE=ddddddd.  
DATA SET NUMBER=nnn.  
I/O ERROR PAGE RANGE=aaaaaa, bbbbbb.
```

Any table spaces that are identified in DSNU086I messages must be recovered. Follow the steps later in this topic.

Environment

Db2 remains active.

Resolving the problem

Operator response:

1. Fix the error range:
 - a. Use the command **STOP DATABASE** to stop the failing table space.
 - b. Use the command **START DATABASE ACCESS (UT)** to start the table space for utility-only access.
 - c. Start a RECOVER utility step to recover the error range by using the following statement:

```
DB2 RECOVER (ddddddd) ERROR RANGE
```

If you receive message DSNU086I again to indicate that the error range recovery cannot be performed, continue with step “2” on page 119.

- d. Issue the command **START DATABASE** to start the table space for RO or RW access, whichever is appropriate. If the table space is recovered, you do not need to continue with the following procedure.
2. If error range recovery fails because of a hardware problem:
 - a. Use the command **STOP DATABASE** to stop the table space or table space partition that contains the error range. As a result of this command, all in-storage data buffers that are associated with the data set are externalized to ensure data consistency during the subsequent steps.
 - b. Use the INSPECT function of the IBM Device Support Facility, ICKDSF, to check for track defects and to assign alternate tracks as necessary. Determine the physical location of the defects by analyzing the output of messages DSNB224I, DSNU086I, and IOS000I. These messages are displayed on the system operator's console at the time that the error range was created. If damaged storage media is suspected, request assistance from IBM Hardware Support before proceeding.
 - c. Use the command **START DATABASE** to start the table space with ACCESS(UT) or ACCESS(RW).
 - d. Run the RECOVER utility with the ERROR RANGE option. Specify an error range that, from image copies, locates, allocates, and applies the pages within the tracks that are affected by the error ranges.

Related information

[Device Support Facilities \(ICKDSF\) Device Support Facilities \(ICKDSF\) User's Guide and Reference](#)

Recovering from Db2 catalog or directory I/O errors

When the Db2 catalog or directory fails because of I/O errors, you need to recover from this situation so that processing can return to normal.

Symptoms

The following message is issued, where *ddddddd* is the name of the table space from the catalog or directory that failed (for example, SYSIBM.SYSCOPY):

```
DSNU086I DSNUCDA1 READ I/O ERRORS ON SPACE=ddddddd.
        DATA SET NUMBER=NNN.
        I/O ERROR PAGE RANGE=aaaaaa, bbbbbb
```

This message can indicate either read or write errors. You might also receive a DSNB224I or DSNB225I message, which indicates an input or output error for the catalog or directory.

Environment

Db2 remains active.

If the Db2 directory or any catalog table is damaged, only user IDs with the RECOVERDB privilege in DSNDB06, or an authority that includes that privilege, can perform the recovery. Furthermore, until the recovery takes place, only those IDs can do anything with the subsystem. If an ID without proper authorization attempts to recover the catalog or directory, message DSNU060I is displayed. If the authorization tables are unavailable, message DSNT500I is displayed to indicate that the resource is unavailable.

Resolving the problem

Operator response: Recover each table space in the failing Db2 catalog or directory. If multiple table spaces need to be recovered, recover them in the recommended order as defined in the information about the RECOVER utility.

1. Stop the failing table spaces.
2. Determine the name of the data set that failed by using one of the following methods:

- Check *prefix.SDSNSAMP* (DSNTIJJIN), which contains the JCL for installing Db2. Find the fully qualified name of the data set that failed by searching for the name of the table space that failed (the one that is identified in the message as `SPACE=dddddddd`).
- Construct the data set name by performing one of the following actions:
 - If the table space is in the Db2 catalog, the data set name format is:

```
DSNC112.DSNDBC.DSNDB06.ddddddd.I0001.A001
```

The *ddddddd* is the name of the table space that failed.

- If the table space is in the Db2 directory, the data set name format is:

```
DSNC112.DSNDBC.DSNDB01.ddddddd.I0001.A001
```

The *ddddddd* is the name of the table space that failed.

If you do not use the default (IBM-supplied) formats, the formats for data set names might be different.

3. Use the access method services **DELETE** command to delete the data set, specifying the fully qualified data set name.
4. After the data set is deleted, use the access method services **DEFINE** command with the `REUSE` parameter to redefine the same data set, again specifying the same fully qualified data set name. Use the JCL for installing Db2 to determine the appropriate parameters.
5. Issue the command **START DATABASE ACCESS(UT)**, naming the table space that is involved.
6. Use the `RECOVER` utility to recover the table space that failed.
7. Issue the command **START DATABASE**, specifying the table space name and `RO` or `RW` access, whichever is appropriate.

Related tasks

[Recovering catalog and directory objects \(Db2 Utilities\)](#)

Recovering from integrated catalog facility failure

Sometimes VSAM volume data sets might be out of space or destroyed. Also, you might experience problems with other VSAM data sets being out of space or unable to be extended any further.

Symptoms

The symptoms for integrated catalog facility problems vary according to the underlying problems.

Recovering VSAM volume data sets that are out of space or destroyed

If the VSAM volume data set (VVDS) is out of space or destroyed, you can recover from the situation. You can recover by using Db2 commands, Db2 utilities, and access method services.

Symptoms

Db2 sends the following message to the master console:

```
DSNP012I - DSNPCT0 - ERROR IN VSAM CATALOG LOCATE FUNCTION
          FOR data_set_name
          CTLGRC=50
          CTLGRSN=zzzzRRRR
          CONNECTION-ID=xxxxxxxx,
          CORRELATION-ID=yyyyyyyyyyyyy
          LUW-ID=logical-unit-of-work-id=token
```

VSAM might also issue the following message:

```
IDC3009I VSAM CATALOG RETURN CODE IS 50, REASON CODE IS
          IGGCLaa - yy
```

In this VSAM message, yy is 28, 30, or 32 for an out-of-space condition. Any other values for yy indicate a damaged VVDS.

Environment

Your program is terminated abnormally, and one or more messages are issued.

Resolving the problem

Operator response: Begin by determining whether the VSAM volume data set is out of space or has been destroyed. Then follow these steps:

1. Determine the names of all table spaces that reside on the same volume as the VVDS. To determine the table space names, look at the VTOC entries list for that volume, which indicates the names of all the data sets on that volume.
2. Use the Db2 COPY utility to take image copies of all table spaces of the volume. Taking image copies minimizes reliance on the Db2 recovery log and can speed up the processing of the Db2 RECOVER utility (to be mentioned in a subsequent step).

If you cannot use the COPY utility, continue with this procedure. Be aware that processing time increases because more information must be obtained from the Db2 recovery log.

3. Use the command **STOP DATABASE** for all the table spaces that reside on the volume, or use the command **STOP DB2** to stop the entire Db2 subsystem if an unusually large number or critical set of table spaces are involved.
4. If possible, use access method services to export all non-Db2 data sets that resides on that volume.
5. Use access method services to recover all non-Db2 data sets that resides on that volume.
6. Use access method services **DELETE** and **DEFINE** commands to delete and redefine the data sets for all user-defined table spaces and Db2-defined data sets when the physical data set has been destroyed. Db2 automatically deletes and redefines all other STOGROUP-defined table spaces.

You do not need to delete and redefine table spaces that are STOGROUP-defined because Db2 takes care of them automatically.
7. Issue the Db2 **START DATABASE** command to restart all the table spaces that were stopped in step “3” on page 121. If the entire Db2 subsystem was stopped, issue the **START DB2** command.
8. Use the Db2 RECOVER utility to recover any table spaces and indexes.

Related tasks

[Backing up and recovering your data \(Db2 Administration Guide\)](#)

Related information

[DFSMS Access Method Services Commands](#)

[DFSMS Managing Catalogs](#)

[DSNP012I \(Db2 Messages\)](#)

Recovering from out-of-disk-space or extent limit problems

When a volume on which a data set is stored has insufficient space, or when the data set reaches its maximum size or its maximum number of VSAM extents, you can recover from this situation.

Symptoms

The symptoms vary based on the specific situation. The following messages and codes might be issued:

- DSNP007I
- DSNP001I
- -904 SQL return code (SQLSTATE '57011')

Environment

For a demand request failure during restart, the object that is supported by the data set (an index space or a table space) is stopped with deferred restart pending. Otherwise, the state of the object remains unchanged.

Diagnosing the problem

Read the following descriptions of possible problems, and determine which problem you are experiencing.

- Extend request failures: When an insert or update requires additional space, but the space is not available in the current table space or index space, Db2 issues the following message:

```
DSNP007I - DSNPmmmm - EXTEND FAILED FOR
           data-set-name. RC=rrrrrrrr
           CONNECTION-ID=xxxxxxxx,
           CORRELATION-ID=yyyyyyyyyyyy
           LUWID-ID=logical-unit-of-work-id=token
```

- Look-ahead warning: A look-ahead warning occurs when enough space is available for a few inserts or updates, but the index space or table space is almost full. On an insert or update at the end of a page set, Db2 determines whether the data set has enough available space. Db2 uses the following values in this space calculation:
 - The primary space quantity from the integrated catalog facility (ICF) catalog
 - The secondary space quantity from the ICF catalog
 - The allocation unit size

If enough space does not exist, Db2 tries to extend the data set. If the extend request fails, Db2 issues the following message:

```
DSNP001I - DSNPmmmm - data-set-name IS WITHIN
           nK BYTES OF AVAILABLE SPACE.
           RC=rrrrrrrr
           CONNECTION-ID=xxxxxxxx,
           CORRELATION-ID=yyyyyyyyyyyy
           LUWID-ID=logical-unit-of-work-id=token
```

Resolving the problem

What you need to do depends on your particular circumstances.

- In most cases, if the data set has not reached its maximum size, you can enlarge it. For the maximum sizes of data sets, see [Limits in Db2 for z/OS \(Db2 SQL\)](#).
- If the data set has reached its maximum size, you need to follow the appropriate procedure, depending on the situation you face.

Extending a data set

If a user-defined data set reaches the maximum number of VSAM extents, you can extend the data set by adding volumes.

Procedure

To extend a user-defined data set:

1. If possible, delete unneeded data on the current volume.
2. If deleting data from the volume does not solve the problem, add volumes to the data set in one of the following ways:
 - If the data set is defined in a Db2 storage group, add more volumes to the storage group by using the SQL ALTER STOGROUP statement.

- If the data set is not defined in a Db2 storage group, add volumes to the data set by using the access method services **ALTER ADDVOLUMES** command.

Enlarging a fully extended user-managed data set

If a user-managed data set reaches the maximum number of VSAM extents, you can enlarge the data set.

Procedure

To enlarge a user-managed data set:

1. To allow for recovery in case of failure during this procedure, ensure that you have a recent full image copy of your table spaces and indexes.

Use the DSNUM option to identify the data set for table spaces or partitioning indexes.

2. Issue the command **STOP DATABASE SPACENAM** for the last data set of the supported object.
3. Delete the last data set by using access method services.
4. Redefine the data set, and enlarge it as necessary.

The object must be a user-defined linear data set. The limit is 32 data sets if the underlying table space is not defined as LARGE or with a DSSIZE parameter, and the limit is 4096 for objects with greater than 254 parts. For a nonpartitioned index on a table space that is defined as LARGE or with a DSSIZE parameter, the maximum is $\text{MIN}(4096, 2^{32} / (\text{index piece size}/\text{index page size}))$.

5. Issue the command **START DATABASE ACCESS (UT)** to start the object for utility-only access.
6. To recover the data set that was redefined, use the RECOVER utility on the table space or index, and identify the data set by the DSNUM option (specify this DSNUM option for table spaces or partitioning indexes only).

The RECOVER utility enables you to specify a single data set number for a table space. Therefore, you need to redefine and recover only the last data set (the one that needs extension). This approach can be better than using the REORG utility if the table space is very large and contains multiple data sets, and if the extension must be done quickly.

If you do not copy your indexes, use the REBUILD INDEX utility.

7. Issue the command **START DATABASE** to start the object for either RO or RW access, whichever is appropriate.

Related reference

[-START DATABASE command \(Db2\) \(Db2 Commands\)](#)

[-STOP DATABASE command \(Db2\) \(Db2 Commands\)](#)

[RECOVER \(Db2 Utilities\)](#)

[REBUILD INDEX \(Db2 Utilities\)](#)

Enlarging a fully extended Db2-managed data set

If a Db2-managed data set reaches the maximum number of VSAM extents, you can enlarge the data set.

Procedure

To enlarge a Db2-managed data set:

1. Use the SQL statement ALTER TABLESPACE or ALTER INDEX with a USING clause.
(You do not need to stop the table space before you use ALTER TABLESPACE.) You can give new values of PRIQTY and SECQTY in either the same or a new Db2 storage group.

2. Use one of the following procedures.

No movement of data occurs until this step is completed.

- For indexes: If you have taken full image copies of the index, run the RECOVER INDEX utility. Otherwise, run the REBUILD INDEX utility.

- For table spaces other than LOB table spaces: Run one of the following utilities on the table space: REORG, RECOVER, or LOAD REPLACE.
- For LOB table spaces that are defined with LOG YES: Run the RECOVER utility on the table space.
- For LOB table spaces that are defined with LOG NO:
 - a. Start the table space in read-only (RO) mode to ensure that no updates are made during this process.
 - b. Make an image copy of the table space.
 - c. Run the RECOVER utility on the table space.
 - d. Start the table space in read-write (RW) mode.

Adding a data set

If a user-defined simple data set reaches its maximum size, you can use access method services to define another data set.

Procedure

To add another data set:

1. Use access method services to define another data set.

The name of the new data set must follow the naming sequence of the existing data sets that support the object. The last four characters of each name are a relative data set number: If the last name ends with A001, the next name must end with A002, and so on. Also, be sure to add either the character "I" or the character "J" to the name of the data set.

If the object is defined in a Db2 storage group, Db2 automatically tries to create an additional data set. If that fails, access method services messages are sent to an operator to indicate the cause of the problem.

2. If necessary, correct the problem (identified in the access method services messages) to obtain additional space.

Redefining a partition (index-based partitioning)

Sometimes each partition in a partitioned object is restricted to a single data set. If the data set reaches its maximum size, you need to redefine the partitions. Redefining a partition in an index-based partitioning environment is different than in a table-based partitioning environment.

Procedure

To redefine the partitions in an index-based partitioning environment:

1. Use the ALTER INDEX ALTER PARTITION statement to alter the key range values of the partitioning index.
2. Use the REORG utility with inline statistics on the partitions that are affected by the change in key range.
3. Use the RUNSTATS utility on the nonpartitioned indexes.
4. Rebind the dependent packages and plans.

Redefining a partition (table-based partitioning)

Sometimes each partition in a partitioned object is restricted to a single data set. If the data set reaches its maximum size, you need to redefine the partitions. Redefining a partition in a table-based partitioning environment is different than in an index-based partitioning environment.

Procedure

To redefine the partitions in a table-based partitioning environment:

1. Use the SQL statement ALTER TABLE ALTER PARTITION to alter the partition boundaries.
2. Use the REORG utility with inline statistics on the partitions that are affected by the change in partition boundaries.
3. Use the RUNSTATS utility on the indexes.
4. Rebind the dependent packages and plans.

Enlarging a fully extended data set for the work file database

If you have an out-of-disk-space or extent limit problem with the work file database (DSNDB07), you need to add space to the data set.

Procedure

Add space to the Db2 storage group, choosing one of the following approaches:

- Use SQL to create more table spaces in database DSNDB07.
- Execute these steps:
 - a. Use the command **STOP DATABASE (DSNDB07)** to ensure that no users are accessing the database.
 - b. Use SQL to alter the storage group, adding volumes as necessary.
 - c. Use the command **START DATABASE (DSNDB07)** to allow access to the database.

Recovering from referential constraint violation

When a referential constraint is violated, the table space is available for some actions, but you cannot run certain utilities or use SQL to update the data in the table space until you recover from this situation.

Symptoms

One of the following messages is issued at the end of utility processing, depending on whether the table space is partitioned:

```
DSNU561I csect-name - TABLESPACE=tablespace-name PARTITION=partnum  
IS IN CHECK PENDING  
DSNU563I csect-name - TABLESPACE=tablespace-name IS IN CHECK PENDING
```

Causes

Db2 detected one or more referential constraint violations.

Environment

The table space is still generally available. However, it is not available to the COPY, REORG, and QUIESCE utilities, or to SQL select, insert, delete, or update operations that involve tables in the table space.

Resolving the problem

Operator response:

1. Use the **START DATABASE ACCESS (UT)** command to start the table space for utility-only access.
2. Run the CHECK DATA utility on the table space. Consider these recommendations:
 - If you do not believe that violations exist, specify DELETE NO. If violations do not exist, specifying DELETE NO resets the CHECK-pending status; however, if violations do exist, the status is not reset.
 - If you believe that violations exist, specify the DELETE YES option and an appropriate exception table. Specifying DELETE YES results in deletion of all rows that are in violation, copies them to an exception table, and resets the CHECK-pending status.

- If the CHECK-pending status was set during execution of the LOAD utility, specify the SCOPE PENDING option. This checks only those rows that are added to the table space by LOAD, rather than every row in the table space.
3. Correct the rows in the exception table, if necessary, and use the SQL INSERT statement to insert them into the original table.
 4. Issue the command **START DATABASE** to start the table space for RO or RW access, whichever is appropriate. The table space is no longer in CHECK-pending status and is available for use. If you use the ACCESS (FORCE) option of this command, the CHECK-pending status is reset. However, using ACCESS (FORCE) is not recommended because it does not correct the underlying violations of referential constraints.

Related reference

[CHECK DATA \(Db2 Utilities\)](#)

Recovering from distributed data facility failure

You can recover from various problems that occur for the distributed data facility (DDF).

Symptoms

The symptoms for DDF failures vary based on the precise problems. The symptoms include messages, SQL return codes, and apparent wait states.

Recovering from conversation failure

A VTAM APPC or TCP/IP conversation might fail during or after allocation. The conversation is not available for use until you recover from the situation.

Symptoms

VTAM or TCP/IP returns a resource-unavailable condition along with the appropriate diagnostic reason code and message. A DSNL500 or DSNL511 (conversation failed) message is sent to the console for the first failure to a location for a specific logical unit (LU) mode or TCP/IP address. All other threads that detect a failure from that LU mode or IP address are suppressed until communications to the LU that uses that mode are successful.

Db2 returns messages DSNL501I and DSNL502I. Message DSNL501I usually means that the other subsystem is not operational. When the error is detected, it is reported by a console message, and the application receives an SQL return code.

If you use application-directed access or DRDA as the database protocols, SQLCODE -30080 is returned to the application. The SQLCA contains the VTAM diagnostic information, which contains only the RCPRI and RCSEC codes. For SNA communications errors, SQLCODE -30080 is returned. For TCP/IP connections, SQLCODE -30081 is returned.

Environment

The application can choose to request rollback or commit, both of which deallocate all but the first conversation between the allied thread and the remote database access thread. A commit or rollback message is sent over this remaining conversation.

Errors during the deallocation process of the conversation are reported through messages, but they do not stop the commit or rollback processing. If the conversation that is used for the commit or rollback message fails, the error is reported. If the error occurred during a commit process and if the remote database access was read-only, the commit process continues. Otherwise the commit process is rolled back.

Diagnosing the problem

System programmer response: Review the VTAM or TCP/IP return codes, and possibly discuss the problem with a communications expert. Many VTAM or TCP/IP errors, besides the error of an inactive remote LU or TCP/IP errors, require a person who has a knowledge of VTAM or TCP/IP and the network configuration to diagnose them.

Resolving the problem

Operator response: Correct the cause of the unavailable-resource condition by taking the action that is required by the diagnostic messages that are displayed on the console.

Related concepts

[SQL codes \(Db2 Codes\)](#)

Related information

[z/OS Communications Server: SNA Messages](#)

Recovering from communications database failure

You need to recover the communications database (CDB) when a failure occurs during an attempt to access the CDB.

Symptoms

A DSNL700I message, which indicates that a resource-unavailable condition exists, is sent to the console. Other messages that describe the cause of the failure are also sent to the console.

Environment

If the distributed data facility (DDF) has already started when an individual CDB table becomes unavailable, DDF does not terminate. Depending on the severity of the failure, threads are affected as follows:

- The threads receive a -904 SQL return code (SQLSTATE '57011') with resource type 1004 (CDB).
- The threads continue using VTAM default values.

The only threads that receive a -904 SQL return code are those that access locations that have not had any prior threads. Db2 and DDF remain operational.

Resolving the problem

Operator response:

1. Examine the messages to determine the source of the error.
2. Correct the error, and then stop and restart DDF.

Recovering from a problem with a communications database that is incorrectly defined

You need to recover from a situation in which the communications database (CDB) is not correctly defined. This problem occurs when distributed data facility (DDF) is started and the Db2 catalog is accessed to verify the CDB definitions.

Symptoms

A DSNL701I, DSNL702I, DSNL703I, DSNL704I, or DSNL705I message is issued to identify the problem. Other messages that describe the cause of the failure are also sent to the console.

Environment

DDF fails to start. Db2 continues to run.

Resolving the problem

Operator response:

1. Examine the messages to determine the source of the error.
2. Correct the error, and then restart DDF.

Recovering from database access thread failure

When a database access thread is deallocated, a conversation failure occurs, and you need to recover from this situation.

Symptoms

In the event of a failure of a database access thread, the Db2 server terminates the database access thread only if a unit of recovery exists. The server deallocates the database access thread and then deallocates the conversation with an abnormal indication (a negative SQL code), which is subsequently returned to the requesting application. The returned SQL code depends on the type of remote access:

- DRDA access

For a database access thread or non-Db2 server, a DDM error message is sent to the requesting site, and the conversation is deallocated normally. The SQL error status code is a -30020 with a resource type 1232 (agent permanent error received from the server).

Environment

Normal Db2 error recovery mechanisms apply, with the following exceptions:

- Errors that are encountered in the functional recovery routine are automatically converted to rollback situations. The allied thread experiences conversation failures.
- Errors that occur during commit, rollback, and deallocate within the DDF function do not normally cause Db2 to abend. Conversations are deallocated, and the database access thread is terminated. The allied thread experiences conversation failures.

Diagnosing the problem

System programmer response: Collect all diagnostic information that is related to the failure at the serving site. For a Db2 database access thread (DBAT), a dump is produced at the server.

Resolving the problem

Operator response: Communicate with the operator at the other site to take the appropriate corrective action, regarding the messages that are displayed at both the requesting and responding sites. Ensure that you and operators at the other sites gather the appropriate diagnostic information and give it to the programmer for diagnosis.

Recovering from VTAM failure

When VTAM terminates or fails, you need to recover from the situation.

Symptoms

VTAM messages and Db2 messages are issued to indicate that distributed data facility (DDF) is terminating and to explain why.

Causes

Environment

DDF terminates. An abnormal VTAM failure or termination causes DDF to issue a **STOP DDF MODE(FORCE)** command. The VTAM commands **Z NET,QUICK** and **Z NET,CANCEL** cause an abnormal VTAM termination. A **Z NET,HALT** causes a **STOP DDF MODE(QUIESCE)** to be issued by DDF.

Resolving the problem

Operator response: Correct the condition that is described in the messages that are received at the console, and restart VTAM and DDF.

Recovering from VTAM ACB OPEN problems

The Db2 distributed data facility (DDF) might terminate after startup due to a VTAM ACB OPEN failure. You can diagnose the situation to determine how to resolve the problem.

Symptoms

Db2 messages, such as DSNL013I and DSNL004I, are issued to indicate the problem.

DSNL013I

Contains the error field value that is returned from the VTAM ACB OPEN. For information about possible values, see [OPEN macroinstruction error fields\(z/OS Communications Server: IP and SNA Codes\)](#).

DSNL004I

Normally specifies a fully qualified LU name, *network-name.luname*. The absence of the *network-name* indicates a problem.

Resolving the problem

1. Determine whether VTAM is up:

- If VTAM is not up, start VTAM, and then start DDF.
- If VTAM did not start completely before DDF was started, start VTAM, and then start DDF.

If neither of these situations exists, continue with the next step.

2. If VTAM is up, determine the reason for the problem. One possible reason that the VTAM ACB OPEN failed is that one of the following problems exists for the LU name that was defined to Db2 through the Db2 change log inventory (DSNJU003) utility DDF statement:

- The LU name is not defined to VTAM.
- The LU name is defined, but the LU did not start automatically during VTAM startup.
- The LU name that is displayed in the DSNL004I message is not valid. In this case, stop Db2, run a DSNJU003 utility job, and restart Db2.

To diagnose the problem within VTAM:

a. Issue the following command: `DISPLAY NET ,ID=luname ,SCOPE=ALL`, where *luname* is displayed in message DSNL004I, and examine the output of the DISPLAY command.

- If the output suggests that the LU name does not exist, display each APPL node by issuing the DISPLAY APPLS command. For command details, see [DISPLAY APPLS command \(z/OS Communications Server: SNA Operation\)](#).
- If you do not find an active APPL node that contains the LU name or a model LU name (a wildcard LU name) that Db2 can use, take one of the following actions:
 - If no active APPL node exists and no APPL major node definition exists in any library that is referenced by the VTAM started procedure VTAMLST DD specification, define an APPL node to VTAM, start it, and then start DDF.
 - If no active APPL node exists, but an APPL major node definition exists in a library that is referenced by the VTAM started procedure VTAMLST DD specification, start the major node,

and then start DDF. The cause of this situation is that the major node was not automatically started during VTAM startup.

- b. Work with your VTAM administrator to ensure that the required APPL node or APPL major node is started automatically during VTAM startup in the future. For more information, see [The APPL statement \(Db2 Installation and Migration\)](#).
3. If the previous steps do not resolve the problem, examine the reason code from the DSNL013I message. If the reason code is X'24', the PRTCT value in the APPL definition does not match the password value that was defined to Db2 in the DSNJU003 DDF statement. If the password specification to Db2 is missing or incorrect, with Db2 stopped, run a DSNJU003 utility job, specifying a DDF statement with the correct password, and restart Db2.
4. If none of the previous steps resolves the issue, contact your VTAM administrator for additional help in resolving the problem.

Recovering from TCP/IP failure

When TCP/IP terminates or fails, you need to recover from this situation.

Symptoms

TCP/IP messages and Db2 messages are issued to indicate that TCP/IP is unavailable.

Environment

Distributed data facility (DDF) periodically attempts to reconnect to TCP/IP. If the TCP/IP listener fails, DDF automatically tries to re-establish the TCP/IP listener for the DRDA SQL port or the resync port every three minutes. TCP/IP connections cannot be established until the TCP/IP listener is re-established.

Resolving the problem

Operator response:

1. Examine the messages that are received at the console to determine the cause of the problem.
2. Correct the condition.
3. Restart TCP/IP. You do not need to restart DDF after a TCP/IP failure.

Recovering from remote logical unit failure

When a series of conversation or change number of sessions (CNOS) failures occur from a remote logical unit (LU), you need to recover from this situation.

Symptoms

Message DSNL501I is issued when a CNOS request to a remote LU fails. The CNOS request is the first attempt to connect to the remote site and must be negotiated before any conversations can be allocated. Consequently, if the remote LU is not active, message DSNL501I is displayed to indicate that the CNOS request cannot be negotiated. Message DSNL500I is issued only once for all the SQL conversations that fail as a result of a remote LU failure.

Message DSNL502I is issued for system conversations that are active to the remote LU at the time of the failure. This message contains the VTAM diagnostic information about the cause of the failure.

Environment

Any application communications with a failed LU receives a message to indicate a resource-unavailable condition. Any attempt to establish communication with such an LU fails.

Resolving the problem

System programmer response: Communicate with the other involved sites regarding the unavailable-resource condition, and request that appropriate corrective action be taken. If a DSNL502 message is received, activate the remote LU, or ask another operator to do so.

Recovering from an indefinite wait condition

If a distributed thread is hung, an application might be in an indefinite wait condition. For example, an allied thread might wait indefinitely for a response from a remote server location. Another example is a database access thread that waits for a new request from the remote requester location.

Symptoms

An application is in an indefinitely long wait condition. This can cause other Db2 threads to fail due to resources that are held by the waiting thread. Db2 sends an error message to the console, and the application program receives an SQL return code.

Environment

Db2 does not respond.

Diagnosing the problem

Operator response: To check for very long waits, look to see if the conversation timestamp is changing from the last time it was used. If it is changing, the conversation thread is not hung, but it is taking more time for a long query. Also, look for conversation state changes, and determine what they mean.

Resolving the problem

Operator response:

1. Use the **DISPLAY THREAD** command with the LOCATION and DETAIL options to identify the LUWID and the session allocation for the waiting thread.
2. Use the **CANCEL DDF THREAD** command to cancel the waiting thread.
3. If the **CANCEL DDF THREAD** command fails to break the wait (because the thread is not suspended in Db2), try using VTAM commands such as **VARY TERM, SID=xxx** or use the TCP/IP DROP command. For instructions on how to use the VTAM commands and TCP/IP commands, see [-CANCEL THREAD command \(Db2\) \(Db2 Commands\)](#).

Related tasks

[Canceling threads \(Db2 Administration Guide\)](#)

Recovering database access threads after security failure

During database access thread allocation, the remote site might not have the proper security to access Db2 through distributed data facility (DDF). When this happens, you can recover from the situation.

Symptoms

Message DSNL500I is issued at the requester for VTAM conversations (if it is a Db2 subsystem) with return codes RTNCD=0, FDBK2=B, RCPRI=4, and RCSEC=5. These return codes indicate that a security violation has occurred. The server has deallocated the conversation because the user is not allowed to access the server. For conversations that use DRDA access, LU 6.2 communications protocols present specific reasons for why the user access failed, and these reasons are communicated to the application. If the server is a Db2 database access thread, message DSNL030I is issued to describe what caused the user to be denied access into Db2 through DDF. No message is issued for TCP/IP connections.

If the server is a Db2 subsystem, message DSNL030I is issued. Otherwise, the system programmer needs to refer to the documentation of the server. If the application uses DRDA access, SQLCODE -30082 is returned.

Causes

This problem is caused by a remote user who attempts to access Db2 through DDF without the necessary security authority.

Resolving the problem

Operator response:

1. Read about the Db2 code 00D3103D.
2. Take the appropriate action:
 - If the security failure involves a Db2 database access thread, provide the DSNL030I message to the system programmer.
 - If the security failure does not involve a Db2 server, work with the operator or programmer at the server to get diagnostic information that is needed by the system programmer.

Related information

[00D3103D \(Db2 Codes\)](#)

Performing remote-site disaster recovery

When your local system experiences damage or disruption that prevents recovery from that site, you can recover by using a remote site that you have set up for this purpose.

Symptoms

The specific symptoms of a disaster that affects your local system hardware vary, but when this happens, the affected Db2 subsystem is not operational.

Causes

Your local system hardware has suffered physical damage.

Resolving the problem

System programmer response: Coordinate the activities that are detailed in [“Restoring data from image copies and archive logs”](#) on page 133.

Operator response: At the remote-site, the disaster-recovery procedures differ from other recovery procedures because you cannot use the hardware at your local Db2 site to recover data. Instead, you use hardware at a remote site to recover after a disaster by using one of a variety of methods.

Recovering from a disaster by using system-level backups

If you have recent system-level backups, you can use those backups along with one of several utilities to recover after a disaster.

Procedure

For a remote site recovery procedure where tape volumes that contain system data are sent from the production site, specify the dump class that is available at the remote site by using the following installation options on installation panel DSNTIP6:

- RESTORE FROM DUMP or RECOVER FROM DUMP
- DUMP CLASS NAME

Restoring data from image copies and archive logs

Follow the appropriate procedure for restoring from image copies and archive logs, depending on whether you are in a data sharing environment. Both procedures assume that all logs, copies, and reports are available at the recovery site.

Related information

[DFSMS Access Method Services Commands](#)

Restoring data in a non-data sharing environment

If you are in a non-data sharing environment, you might need to recover from a disaster by restoring data from image copies and logs. The procedure that you follow assumes that all logs, image copies, and reports are available at the recovery site.

Procedure

To recover from a disaster in a non-data sharing environment by using image copies and archive logs:

1. If an integrated catalog facility catalog does not already exist, run job DSNTIJCA to create a user catalog.
2. Use the access method services **IMPORT** command to import the integrated catalog facility catalog.
3. Restore Db2 libraries.
Some examples of libraries that you might need to restore include:
 - Db2 SMP/E libraries
 - User program libraries
 - User DBRM libraries
 - Db2 CLIST libraries
 - Db2 libraries that contain customized installation jobs
 - JCL for creating user-defined table spaces
4. Use IDCAMS **DELETE NOSCRATCH** to delete all catalog and user objects.
(Because step “2” on page 133 imports a user ICF catalog, the catalog reflects data sets that do not exist on disk.)
5. Obtain a copy of installation job DSNTIJIN, which creates Db2 VSAM and non-VSAM data sets. Change the volume serial numbers in the job to volume serial numbers that exist at the recovery site. Comment out the steps that create Db2 non-VSAM data sets, if these data sets already exist. Run DSNTIJIN. However, do not run DSNTIJID.
6. Recover the BSDS:
 - a) Use the access method services **REPRO** command to restore the contents of one BSDS data set (allocated in step “5” on page 133).
You can find the most recent BSDS image in the last file (archive log with the highest number) on the latest archive log tape.
 - b) Determine the RBA range for this archive log by using the print log map utility (DSNJU004) to list the current BSDS contents. Find the most recent archive log in the BSDS listing, and add 1 to its ENDRBA value. Use this as the STARTRBA. Find the active log in the BSDS listing that starts with this RBA, and use its ENDRBA as the ENDRBA.
 - c) Delete the oldest archive log from the BSDS.
 - d) Register this latest archive log tape data set in the archive log inventory of the BSDS that you just restored by using the change log inventory utility (DSNJU003).
This step is necessary because the BSDS image on an archive log tape does not reflect the archive log data set that resides on that tape. After these archive logs are registered, use the print log map utility (DSNJU004) to list the contents of the BSDS.

- e) Adjust the active logs in the BSDS by using the change log inventory utility (DSNJU003), as necessary:
- i) To delete all active logs in the BSDS, use the DELETE option of DSNJU003. Use the BSDS listing that is produced in step “6.d” on page 133 to determine the active log data set names.
 - ii) To add the active log data sets to the BSDS, use the NEWLOG statement of DSNJU003. Do not specify a STARTRBA or ENDRBA in the NEWLOG statement. This specification indicates to Db2 that the new active logs are empty.
- f) If you are using the Db2 distributed data facility, update the LOCATION and the LUNAME values in the BSDS by running the change log inventory utility with the DDF statement.
- g) List the new BSDS contents by using the print log map utility (DSNJU004). Ensure that the BSDS correctly reflects the active and archive log data set inventories.
In particular, ensure that:
- All active logs show a status of NEW and REUSABLE.
 - The archive log inventory is complete and correct (for example, the start and end RBAs are correct).
- h) If you are using dual BSDSs, make a copy of the newly restored BSDS data set to the second BSDS data set.
7. Optional: Restore archive logs to disk.

Archive logs are typically stored on tape, but restoring them to disk might speed later steps. If you elect this option, and the archive log data sets are not cataloged in the primary integrated catalog facility catalog, use the change log inventory utility to update the BSDS. If the archive logs are listed as cataloged in the BSDS, Db2 allocates them by using the integrated catalog and not the unit or VOLSER that is specified in the BSDS. If you are using dual BSDSs, remember to update both copies.

8. Use the DSN1LOGP utility to determine which transactions were in process at the end of the last archive log. Use the following job control language where *yyyyyyyyyyyy* is the STARTRBA of the last complete checkpoint within the RBA range on the last archive log from the previous print log map:

```
//SAMP EXEC PGM=DSN1LOGP
//SYSPRINT DD SYSOUT=*
//SYSSUMRY DD SYSOUT=*
//ARCHIVE DD DSN=last-archive, DISP=(OLD,KEEP),UNIT=TAPE,
           LABEL=(2,SL),VOL=SER=volser1
           (NOTE FILE 1 is BSDS COPY)
//SYSIN DD *
STARTRBA(yyyyyyyyyyyy) SUMMARY(ONLY)
/*
```

DSN1LOGP generates a report.

9. Examine the DSN1LOGP output, and identify any utilities that were executing at the end of the last archive log. Determine the appropriate recovery action to take on each table space that is involved in a utility job.

If DSN1LOGP output showed that utilities are inflight (PLAN=DSNUTIL), examine SYSUTILX to identify the utility status and determine the appropriate recovery approach.

10. Modify DSNZPxxx parameters:

- a) Run the DSNTINST CLIST in UPDATE mode.

For more information, see [Generating tailored Db2 13 installation, migration, or function level activation jobs \(Db2 Installation and Migration\)](#)

- b) To defer processing of all databases, select DATABASES TO START AUTOMATICALLY from panel DSNTIPB.

Panel DSNTIPS opens. On panel DSNTIPS, type DEFER in the first field and ALL in the second field; then press **Enter**.

You are returned to panel DSNTIPB.

- c) To specify where you are recovering, select OPERATOR FUNCTIONS from panel DSNTIPB.

Panel DSNTIPO opens. From panel DSNTIPO, type RECOVERYSITE in the SITE TYPE field. Press **Enter** to continue.

- d) To prevent format conversions during Disaster Recovery, select SQL OBJECT DEFAULTS PANEL 1 from panel DSNTIPB.

Panel DSNTIP7 opens. From panel DSNTIP7, set the UTILITY_OBJECT_CONVERSION value to NONE. Press **Enter** to continue. Format conversions complicate the recovery process and can lead to failures. Reset this parameter to its original value after the Disaster Recovery completes.

- e) Optional: Specify which archive log to use by selecting OPERATOR FUNCTIONS from panel DSNTIPB.

Panel DSNTIPO opens. From panel DSNTIPO, type YES in the READ COPY2 ARCHIVE field if you are using dual archive logging and want to use the second copy of the archive logs. Press **Enter** to continue.

- f) Reassemble DSNZPxxx by using job DSNTIJUZ (produced by the CLIST started in the first step of this procedure).

At this point, you have the log, but the table spaces have not been recovered. With DEFER ALL, Db2 assumes that the table spaces are unavailable but does the necessary processing to the log. This step also handles the units of recovery that are in process.

11. Create a conditional restart control record by using the change log inventory utility with one of the following forms of the CRESTART statement:

- `CRESTART CREATE,ENDRBA=nnnnnnnnn000`

The *nnnnnnnnn000* equals a value that is one more than the ENDRBA of the latest archive log.

- `CRESTART CREATE,ENDTIME=nnnnnnnnnnnn`

The *nnnnnnnnnnnn* is the end time of the log record. Log records with a timestamp later than *nnnnnnnnnnnn* are truncated.

12. Enter the command **START DB2 ACCESS(MAINT)**.

You must enter this command, because real-time statistics are active and enabled; otherwise, errors or abends could occur during Db2 restart processing and recovery processing (for example, GRECP recovery, LPL recovery, or the RECOVER utility).

Even though Db2 marks all table spaces for deferred restart, log records are written so that in-abort and inflight units of recovery are backed out. In-commit units of recovery are completed, but no additional log records are written at restart to cause this. This happens when the original redo log records are applied by the RECOVER utility.

At the primary site, Db2 probably committed or aborted the inflight units of recovery, but you have no way of knowing.

During restart, Db2 accesses two table spaces that result in **DSNT501I**, **DSNT500I**, and **DSNL700I** resource unavailable messages, regardless of DEFER status. The messages are normal and expected, and you can ignore them.

The following return codes can accompany the message. Other codes are also possible.

00C90081

This return code is issued for activity against the object that occurs during restart as a result of a unit of recovery or pending writes. In this case, the status that is shown as a result of **DISPLAY** is STOP, DEFER.

00C90094

Because the table space is currently only a defined VSAM data set, it is in a state that Db2 does not expect.

00C90095

Db2 cannot access the page, because the table space or index space has not been recovered yet.

00C900A9

An attempt was made to allocate a deferred resource.

13. Resolve the indoubt units of recovery.

The RECOVER utility, which you run in a subsequent step, fails on any table space that has indoubt units of recovery. Because of this, you must resolve them first. Determine the proper action to take (commit or abort) for each unit of recovery. To resolve indoubt units of recovery, see [Resolving indoubt units of recovery \(Db2 Administration Guide\)](#). From an installation SYSADM authorization ID, enter the **RECOVER INDOUBT** command for all affected transactions.

14. Recover the catalog and directory.

The RECOVER function includes: RECOVER TABLESPACE, RECOVER INDEX, or REBUILD INDEX. If you have an image copy of an index, use RECOVER INDEX. If you do not have an image copy of an index, use REBUILD INDEX to reconstruct the index from the recovered table space.

a) Recover DSNDB01.SYSUTILX.

This must be a separate job step.

b) Recover all indexes on SYSUTILX.

This must be a separate job step.

c) Determine whether a utility was running at the time the latest archive log was created by entering the **DISPLAY UTILITY(*)** command, and record the name and current phase of any utility that is running.

(You cannot restart a utility at the recovery site that was interrupted at the disaster site. You must use the **TERM UTILITY** command to terminate it. Use the **TERM UTILITY** command on a utility that is operating on any object except DSNDB01.SYSUTILX.)

d) Run the DIAGNOSE utility with the DISPLAY SYSUTIL option.

The output consists of information about each active utility, including the table space name (in most cases). This is the only way to correlate the object name with the utility. Message DSNU866I gives information about the utility, and DSNU867I gives the database and table space name in USUDBNAM and USUSPNAM, respectively.

e) Use the **TERM UTILITY** command to terminate any utilities that are in progress on catalog or directory table spaces.

f) Recover the rest of the catalog and directory objects, starting with DBD01, in the order shown in the description of the RECOVER utility.

15. Define and initialize the work file database:

a) Define temporary work files. Use installation job DSNTIJTM as a model.

b) Issue the command **START DATABASE** (*work-file-database*) to start the work file database.

16. Use any method that you want to verify the integrity of the Db2 catalog and directory.

Use the catalog queries in member DSNTESQ of data set DSN1310.SDSNSAMP after the work file database is defined and initialized.

17. If you use data definition control support, recover the objects in the data definition control support database.

18. If you use the resource limit facility, recover the objects in the resource limit control facility database.

19. Modify DSNZPxxx to restart all databases:

a) Run the DSNTINST CLIST in UPDATE mode.

b) From panel DSNTIPB, select DATABASES TO START AUTOMATICALLY.

Panel DSNTIPS opens. Type RESTART in the first field and ALL in the second field, and press **Enter**.

You are returned to DSNTIPB.

c) Reassemble DSNZPxxx by using job DSNTIJUZ (produced by the CLIST started in step “3” on page [133](#)).

20. Stop Db2.

21. Start Db2.
22. Make a full image copy of the catalog and directory.
23. Recover user table spaces and index spaces.

If utilities were running on any table spaces or index spaces, see [“What to do about utilities that were in progress at time of failure”](#) on page 145. You cannot restart a utility at the recovery site that was interrupted at the disaster site. Use the **TERM UTILITY** command to terminate any utilities that are running against user table spaces or index spaces.

- a) To determine which, if any, of your table spaces or index spaces are user-managed, perform the following queries for table spaces and index spaces.

- Table spaces:

```
SELECT * FROM SYSIBM.SYSTABLEPART WHERE STORTYPE='E';
```

- Index spaces:

```
SELECT * FROM SYSIBM.SYSINDEXPART WHERE STORTYPE='E';
```

To allocate user-managed table spaces or index spaces, use the access method services **DEFINE CLUSTER** command. To find the correct IPREFIX for the **DEFINE CLUSTER** command, perform the following queries for table spaces and index spaces.

- Table spaces:

```
SELECT DBNAME, TSNAME, PARTITION, IPREFIX FROM SYSIBM.SYSTABLEPART
WHERE DBNAME=dbname AND TSNAME=tsname
ORDER BY PARTITION;
```

- Index spaces:

```
SELECT IXNAME, PARTITION, IPREFIX FROM SYSIBM.SYSINDEXPART
WHERE IXCREATOR=ixcreator AND IXNAME=ixname
ORDER BY PARTITION;
```

Now you can perform the **DEFINE CLUSTER** command with the correct IPREFIX (I or J) in the data set name:

```
catname.DSNDBx.dbname.psname.y0001.znnn
```

The *y* can be either I or J, *x* is C (for VSAM clusters) or D (for VSAM data components), and *psname* is either the table space or index space name.

- b) If your user table spaces or index spaces are STOGROUP-defined, and if the volume serial numbers at the recovery site are different from those at the local site, use the SQL statement ALTER STOGROUP to change them in the Db2 catalog.
 - c) Recover all user table spaces and index spaces from the appropriate image copies.
 - If you do not copy your indexes, use the REBUILD INDEX utility to reconstruct the indexes.
 - d) Start all user table spaces and index spaces for read-write processing by issuing the command **START DATABASE** with the ACCESS(RW) option.
 - e) Resolve any remaining CHECK-pending states that would prevent COPY execution.
 - f) Run queries for which the results are known.
24. Make full image copies of all table spaces and indexes with the COPY YES attribute.
 25. Compensate for work that was lost since the last archive was created by rerunning online transactions and batch jobs.
 26. If subsystem parameter CDDS_MODE is set to SOURCE_ONLY, populate the compression dictionary data set (CDDS) so that Db2 can use the expansion dictionaries in the CDDS to decompress log records for IFI calls for IFCID 306. Follow these steps:

- a) Define the CDDS if it is not already defined. See [Storing the expansion dictionary for compressed log records in the compression dictionary data set \(Db2 Administration Guide\)](#) for an example of the CDDS definition.
- b) Issue the **-START CDDS** command to direct the Db2 subsystem or all data sharing members to allocate and open the CDDS.
- c) To populate the CDDS, run REORG TABLESPACE with the INITCDDS option on each compressed table space or partition that contains a table that is defined with DATA CAPTURE CHANGES. You can specify the SEARCHTIME option with the INITCDDS option to allow REORG to populate the CDDS with an earlier dictionary than the dictionary that currently resides in the table space.

What to do next

Determine what to do about any utilities that were in progress at the time of failure.

Related concepts

[Preparations for disaster recovery \(Db2 Administration Guide\)](#)

[What to do about utilities that were in progress at time of failure \(Db2 Administration Guide\)](#)

Related tasks

[Defining your own user-managed data sets \(Db2 Administration Guide\)](#)

[Migration step 1: Complete premigration tasks \(Db2 Installation and Migration\)](#)

[Recovering catalog and directory objects \(Db2 Utilities\)](#)

Related reference

[DSN1LOGP \(Db2 Utilities\)](#)

Restoring data in a data sharing environment

If you are in a data sharing environment, you might need to recover from a disaster by restoring data from image copies and logs. The procedure that you follow assumes that all logs, image copies, and reports are available at the recovery site.

About this task

Additional recovery procedures for data sharing environments are also available.

Procedure

To recover from a disaster by using image copies and archive logs:

1. If you have information in your coupling facility from practice startups, remove old information from the coupling facility.

If you do not have old information in your coupling facility, continue with the step “2” on page 139.

- a) Enter the following z/OS command to display the structures for this data sharing group:

```
D XCF,STRUCTURE,STRNAME=grpname*
```

- b) For group buffer pools, enter the following command to force off the connection of those structures:

```
SETXCF FORCE,CONNECTION,STRNAME=strname,CONNAME=ALL
```

Connections for the SCA are not held at termination; therefore you do not need to force off any SCA connections.

- c) Delete all the Db2 coupling facility structures that have a STATUS of ALLOCATED by using the following command for each structure:

```
SETXCF FORCE,STRUCTURE,STRNAME=strname
```

This step is necessary to remove old information that exists in the coupling facility from your practice startup when you installed the group.

2. If an integrated catalog facility catalog does not already exist, run job DSNTIJCA to create a user catalog.
3. Use the access method services **IMPORT** command to import the integrated catalog facility catalog.
4. Restore Db2 libraries.
Some examples of libraries that you might need to restore include:
 - Db2 SMP/E libraries
 - User program libraries
 - User DBRM libraries
 - Db2 CLIST libraries
 - Db2 libraries that contain customized installation jobs
 - JCL for creating user-defined table spaces
5. Use IDCAMS **DELETE NOSCRATCH** to delete all catalog and user objects.
(Because step “3” on page 139 imports a user ICF catalog, the catalog reflects data sets that do not exist on disk.)
6. Obtain a copy of the installation job DSNTIJIN, which creates Db2 VSAM and non-VSAM data sets, for the first data sharing member. Change the volume serial numbers in the job to volume serial numbers that exist at the recovery site. Comment out the steps that create Db2 non-VSAM data sets, if these data sets already exist. Run DSNTIJIN on the first data sharing member.
However, do not run DSNTIJID.

For subsequent members of the data sharing group, run the DSNTIJIN that defines the BSDS and logs.

7. Recover the BSDS by following these steps for each member in the data sharing group:
 - a) Use the access method services **REPRO** command to restore the contents of one BSDS data set (allocated in step “6” on page 139) on each member.
You can find the most recent BSDS image in the last file (archive log with the highest number) on the latest archive log tape.
 - b) Determine the RBA and LRSN ranges for this archive log by using the print log map utility (DSNJU004) to list the current BSDS contents. Find the most recent archive log in the BSDS listing, and add 1 to its ENDRBA value. Use this as the STARTRBA. Find the active log in the BSDS listing that starts with this RBA, and use its ENDRBA as the ENDRBA. Use the STARTLRSN and ENDLRSN of this active log data set as the LRSN range (STARTLRSN and ENDLRSN) for this archive log.
 - c) Delete the oldest archive log from the BSDS.
 - d) Register this latest archive log tape data set in the archive log inventory of the BSDS that you just restored by using the change log inventory utility (DSNJU003).

This step is necessary because the BSDS image on an archive log tape does not reflect the archive log data set that resides on that tape.

Running DSNJU003 is critical for data sharing groups. Include the group buffer pool checkpoint information that is stored in the BSDS from the most recent archive log.

After these archive logs are registered, use the print log map utility (DSNJU004) with the GROUP option to list the contents of all BSDSs. You receive output that includes the start and end LRSN and RBA values for the latest active log data sets (shown as NOTREUSABLE). If you did not save the values from the DSNJ003I message, you can get those values by running DSNJU004, which creates output as shown below

The following sample DSNJU004 output shows the (partial) information for the archive log member DB1G.

```
ACTIVE LOG COPY 1 DATA SETS
START RBA/LRSN/TIME    END RBA/LRSN/TIME    DATE/LTIME DATA SET INFORMATION
-----
0000000007A5C5360000  0000000007A5DB31FFFF  2005.034  DSN=DSNT3LOG.DT31.LOGCOPY1.DS01
00CAC6509C994A000000  00CAC650C5EDD8000000  20:22    PASSWORD=(NULL) STATUS=REUSABLE
```

```

2013.015 14:41:16.4 2013.015 14:41:59.7
0000000007A5DB320000 0000000007A5F12DFFFF 2007.051 DSN=DSNT3LOG.DT31.LOGCOPY1.DS04
00CAC650C5EDD8000000 00CAC650EA3857000000 13:27 PASSWORD=(NULL) STATUS=REUSABLE
2013.015 14:41:59.7 2013.015 14:42:37.7

```

The following sample DSNJU004 output shows the (partial) information for the archive log member DB2G.

```

ACTIVE LOG COPY 1 DATA SETS
START RBA/LRSN/TIME      END RBA/LRSN/TIME      DATE      LTIME      DATA SET INFORMATION
-----
EMPTY DATA SET
00000000000000000000 00000000000000000000 1996.361 14:14 DSN=DSNDB0G.DB2G.LOGCOPY1.DS03
STATUS=NEW, REUSABLE
0000.000 00:00:00.0 0000.000 00:00:00.0
00000000000000000000 00000000D6FFF 1996.361 14:14 DSN=DSNDB0G.DB2G.LOGCOPY1.DS01
STATUS=TRUNCATED, NOTREUSABLE
ADFA00BB70FB AE3C45276DD7
1996.361 22:30:51.4 1997.048 15:28:23.7
0000000D7000 00000045AFF 1996.361 14:14 DSN=DSNDB0G.DB2G.LOGCOPY1.DS02
STATUS=NOTREUSABLE
AE3C45276DD8
1997.048 15:28:23.7 .....

```

- e) Adjust the active logs in the BSDS by using the change log inventory utility (DSNJU003), as necessary:
 - i) To delete all active logs in the BSDS, use the DELETE option of DSNJU003. Use the BSDS listing that is produced in step “7.d” on page 139 to determine the active log data set names.
 - ii) To add the active log data sets to the BSDS, use the NEWLOG statement of DSNJU003. Do not specify a STARTRBA or ENDRBA in the NEWLOG statement. This specification indicates to Db2 that the new active logs are empty.
 - f) If you are using the Db2 distributed data facility, update the LOCATION and the LUNAME values in the BSDS by running the change log inventory utility with the DDF statement.
 - g) List the new BSDS contents by using the print log map utility (DSNJU004). Ensure that the BSDS correctly reflects the active and archive log data set inventories.

In particular, ensure that:

 - All active logs show a status of NEW and REUSABLE.
 - The archive log inventory is complete and correct (for example, the start and end RBAs are correct).
 - h) If you are using dual BSDSs, make a copy of the newly restored BSDS data set to the second BSDS data set.
8. Optional: Restore archive logs to disk for each member.
- Archive logs are typically stored on tape, but restoring them to disk might speed later steps. If you elect this option, and the archive log data sets are not cataloged in the primary integrated catalog facility catalog, use the change log inventory utility to update the BSDS. If the archive logs are listed as cataloged in the BSDS, Db2 allocates them by using the integrated catalog and not the unit or VOLSER that is specified in the BSDS. If you are using dual BSDSs, remember to update both copies.
9. Use the DSN1LOGP utility to determine, for each member of the data sharing group, which transactions were in process at the end of the last archive log. Use the following job control language where yyyyyyyyyyyy is the STARTRBA of the last complete checkpoint within the RBA range on the last archive log from the previous print log map:

```

//SAMP EXEC PGM=DSN1LOGP
//SYSPRINT DD SYSOUT=*
//SYSSUMRY DD SYSOUT=*
//ARCHIVE DD DSN=last-archive, DISP=(OLD,KEEP),UNIT=TAPE,
            LABEL=(2,SL),VOL=SER=volsr1
            (NOTE FILE 1 is BSDS COPY)
//SYSIN DD *
STARTRBA(yyyyyyyyyyyy) SUMMARY(ONLY)
/*

```

DSN1LOGP generates a report.

10. Examine the DSN1LOGP output for each data sharing member, and identify any utilities that were executing at the end of the last archive log. Determine the appropriate recovery action to take on each table space that is involved in a utility job.

If DSN1LOGP output showed that utilities are inflight (PLAN=DSNUTIL), examine SYSUTILX to identify the utility status and determine the appropriate recovery approach.

11. Modify DSNZPxxx parameters for each member of the data sharing group:

- a) Run the DSNTINST CLIST in UPDATE mode.

- b) To defer processing of all databases, select DATABASES TO START AUTOMATICALLY from panel DSNTIPB.

Panel DSNTIPS opens. On panel DSNTIPS, type DEFER in the first field and ALL in the second field; then press **Enter**.

You are returned to panel DSNTIPB.

- c) To specify where you are recovering, select OPERATOR FUNCTIONS from panel DSNTIPB.

Panel DSNTIPO opens. From panel DSNTIPO, type RECOVERYSITE in the SITE TYPE field. Press **Enter** to continue.

- d) Optional: Specify which archive log to use by selecting OPERATOR FUNCTIONS from panel DSNTIPB.

Panel DSNTIPO opens. From panel DSNTIPO, type YES in the READ ARCHIVE COPY2 field if you are using dual archive logging and want to use the second copy of the archive logs. Press **Enter** to continue.

- e) Reassemble DSNZPxxx by using job DSNTIJUZ (produced by the CLIST started in the first step of this procedure).

At this point, you have the log, but the table spaces have not been recovered. With DEFER ALL, Db2 assumes that the table spaces are unavailable but does the necessary processing to the log. This step also handles the units of recovery that are in process.

12. Create a conditional restart control record for each data sharing member by using the change log inventory utility with one of the following forms of the CRESTART statement:

- ```
CRESTART CREATE,ENDLRSN=nnnnnnnnnnnn
```

The *nnnnnnnnnnnn* is the LRSN of the last log record that is to be used during restart.

- ```
CRESTART CREATE,ENDTIME=nnnnnnnnnnnn
```

The *nnnnnnnnnnnn* is the end time of the log record. Log records with a timestamp later than *nnnnnnnnnnnn* are truncated.

Use the same LRSN or system time-of-day clock timestamp value for all members in a data sharing group. Determine the ENDLRSN value by using one of the following methods:

- Use the DSN1LOGP summary utility. In the "Summary of Completed Events" section, find the lowest LRSN value that is listed in the DSN1213I message for the data sharing group. Use this value for the ENDLRSN in the CRESTART statement.
- Use the print log map utility (DSNJU004) to list the BSDS contents. Find the ENDLRSN of the last log record that is available for each active member of the data sharing group. Subtract 1 from the lowest ENDLRSN in the data sharing group. Use this value for the ENDLRSN in the CRESTART statement. (In the sample output that is shown in step "7.d" on page 139, the value is AE3C45273A77 - 1, which is AE3C45273A76.)
- If only the console logs are available, use the archive offload message (DSNJ003I) to obtain the ENDLRSN. Compare the ending LRSN values for the archive logs of all members. Subtract 1 from the lowest LRSN in the data sharing group. Use this value for the ENDLRSN in the CRESTART statement. (In the sample output that is shown in step "7.d" on page 139, the value is AE3C45273A77 - 1, which is AE3C45273A76.)

Db2 discards any log information in the bootstrap data set and the active logs with an RBA greater than or equal to *nnnnnnnnn000* or an LRSN greater than *nnnnnnnnnnnn* as listed in the preceding CRESTART statements.

Use the print log map utility to verify that the conditional restart control record that you created in the previous step is active.

13. Enter the command **START DB2 ACCESS(MAINT)**.

You must enter this command, because real-time statistics are active and enabled; otherwise, errors or abends could occur during Db2 restart processing and recovery processing (for example, GRECP recovery, LPL recovery, or the RECOVER utility).

If a discrepancy exists among the print log map reports as to the number of members in the group, which would be an unlikely occurrence, record the one that shows the highest number of members. Start this Db2 subsystem first using ACCESS(MAINT). Db2 prompts you to start each additional Db2 subsystem in the group.

After all additional members are successfully restarted, and if you are going to run single-system data sharing at the recovery site, stop all except one of the Db2 subsystems by using the **STOP DB2** command with MODE(QUIESCE).

If you planned to use the light mode when starting the Db2 group, add the LIGHT parameter to the **START** command. Start the members that run in LIGHT(NO) mode first, followed by the light mode members.

Even though Db2 marks all table spaces for deferred restart, log records are written so that in-abort and inflight units of recovery are backed out. In-commit units of recovery are completed, but no additional log records are written at restart to cause this. This happens when the original redo log records are applied by the RECOVER utility.

At the primary site, Db2 probably committed or canceled the inflight units of recovery, but you have no way of knowing.

During restart, Db2 accesses two table spaces that result in **DSNT501I**, **DSNT500I**, and **DSNL700I** resource unavailable messages, regardless of DEFER status. The messages are normal and expected, and you can ignore them.

The following return codes can accompany the message. Other codes are also possible.

00C90081

This return code is issued for activity against the object that occurs during restart as a result of a unit of recovery or pending writes. In this case, the status that is shown as a result of **DISPLAY** is STOP, DEFER.

00C90094

Because the table space is currently only a defined VSAM data set, it is in a state that Db2 does not expect.

00C900A9

An attempt was made to allocate a deferred resource.

14. Resolve the indoubt units of recovery.

The RECOVER utility, which you run in a subsequent step, fails on any table space that has indoubt units of recovery. Because of this, you must resolve them first. Determine the proper action to take (commit or abort) for each unit of recovery. To resolve indoubt units of recovery, see [Resolving indoubt units of recovery \(Db2 Administration Guide\)](#). From an installation SYSADM authorization ID, enter the **RECOVER INDOUBT** command for all affected transactions.

15. Recover the catalog and directory.

The RECOVER function includes: RECOVER TABLESPACE, RECOVER INDEX, or REBUILD INDEX. If you have an image copy of an index, use RECOVER INDEX. If you do not have an image copy of an index, use REBUILD INDEX to reconstruct the index from the recovered table space.

a) Recover DSNDB01.SYSUTILX.

This must be a separate job step.

- b) Recover all indexes on SYSUTILX.
This must be a separate job step.
 - c) Determine whether a utility was running at the time the latest archive log was created by entering the **DISPLAY UTILITY(*)** command, and record the name and current phase of any utility that is running.
(You cannot restart a utility at the recovery site that was interrupted at the disaster site. To terminate a utility at the recovery site that was interrupted at the disaster site, you must use the TERM UTILITY command.)
 - d) Run the DIAGNOSE utility with the DISPLAY SYSUTIL option.
The output consists of information about each active utility, including the table space name (in most cases). This is the only way to correlate the object name with the utility. Message DSNU866I gives information about the utility, and DSNU867I gives the database and table space name in USUDBNAM and USUSPNAM, respectively.
 - e) Use the **TERM UTILITY** command to terminate any utilities that are in progress on catalog or directory table spaces.
 - f) Recover the rest of the catalog and directory objects, starting with DBD01, in the order shown in the description of the RECOVER utility.
16. Define and initialize the work file database
 - a) Define temporary work files. Use installation job DSNTIJTM as a model.
 - b) Issue the command **START DATABASE** (*work-file-database*) to start the work file database.
 17. Use any method that you want to verify the integrity of the Db2 catalog and directory.
Use the catalog queries in member DSNTESQ of data set DSN1310.SDSNSAMP after the work file database is defined and initialized.
 18. If you use data definition control support, recover the objects in the data definition control support database.
 19. If you use the resource limit facility, recover the objects in the resource limit control facility database.
 20. Modify DSNZPxxx to restart all databases on each member of the data sharing group:
 - a) Run the DSNTINST CLIST in UPDATE mode.
For more information, see [Generating tailored Db2 13 installation, migration, or function level activation jobs \(Db2 Installation and Migration\)](#).
 - b) From panel DSNTIPB, select DATABASES TO START AUTOMATICALLY.
Panel DSNTIPS opens. Type RESTART in the first field and ALL in the second field, and press **Enter**.
You are returned to DSNTIPB.
 - c) Reassemble DSNZPxxx by using job DSNTIJUZ (produced by the CLIST started in step “4” on page 139).
 21. Stop Db2.
 22. Start Db2.
 23. Make a full image copy of the catalog and directory.
 24. Recover user table spaces and index spaces.
If utilities were running on any table spaces or index spaces, see [“What to do about utilities that were in progress at time of failure” on page 145](#). You cannot restart a utility at the recovery site that was interrupted at the disaster site. Use the **TERM UTILITY** command to terminate any utilities that are running against user table spaces or index spaces.
 - a) To determine which, if any, of your table spaces or index spaces are user-managed, perform the following queries for table spaces and index spaces.
 - Table spaces:

```
SELECT * FROM SYSIBM.SYSTABLEPART WHERE STORTYPE='E' ;
```

- Index spaces:

```
SELECT * FROM SYSIBM.SYSINDEXPART WHERE STORTYPE='E' ;
```

To allocate user-managed table spaces or index spaces, use the access method services **DEFINE CLUSTER** command. To find the correct IPREFIX for the **DEFINE CLUSTER** command, perform the following queries for table spaces and index spaces.

- Table spaces:

```
SELECT DBNAME, TSNAME, PARTITION, IPREFIX FROM SYSIBM.SYSTABLEPART
WHERE DBNAME=dbname AND TSNAME=tsname
ORDER BY PARTITION;
```

- Index spaces:

```
SELECT IXNAME, PARTITION, IPREFIX FROM SYSIBM.SYSINDEXPART
WHERE IXCREATOR=ixcreator AND IXNAME=ixname
ORDER BY PARTITION;
```

Now you can perform the **DEFINE CLUSTER** command with the correct IPREFIX (I or J) in the data set name:

```
catname.DSNDBx.dbname.psname.y0001.znnn
```

The *y* can be either I or J, *x* is C (for VSAM clusters) or D (for VSAM data components), and *psname* is either the table space or index space name.

- b) If your user table spaces or index spaces are STOGROUP-defined, and if the volume serial numbers at the recovery site are different from those at the local site, use the SQL statement ALTER STOGROUP to change them in the Db2 catalog.
 - c) Recover all user table spaces and index spaces from the appropriate image copies.
If you do not copy your indexes, use the REBUILD INDEX utility to reconstruct the indexes.
 - d) Start all user table spaces and index spaces for read-write processing by issuing the command **START DATABASE** with the ACCESS(RW) option.
 - e) Resolve any remaining CHECK-pending states that would prevent COPY execution.
 - f) Run queries for which the results are known.
25. Make full image copies of all table spaces and indexes with the COPY YES attribute.
 26. Compensate for work that was lost since the last archive was created by rerunning online transactions and batch jobs.
 27. If subsystem parameter CDDS_MODE is set to SOURCE_ONLY, populate the compression dictionary data set (CDDS) so that Db2 can use the expansion dictionaries in the CDDS to decompress log records for IFI calls for IFCID 306. Follow these steps:
 - a) Define the CDDS if it is not already defined. See [Storing the expansion dictionary for compressed log records in the compression dictionary data set \(Db2 Administration Guide\)](#) for an example of the CDDS definition.
 - b) Issue the **-START CDDS** command to direct the Db2 subsystem or all data sharing members to allocate and open the CDDS.
 - c) To populate the CDDS, run REORG TABLESPACE with the INITCDDS option on each compressed table space or partition that contains a table that is defined with DATA CAPTURE CHANGES. You can specify the SEARCHTIME option with the INITCDDS option to allow REORG to populate the CDDS with an earlier dictionary than the dictionary that currently resides in the table space.

What to do next

Determine what to do about any utilities that were in progress at the time of failure.

Related concepts

[Preparations for disaster recovery \(Db2 Administration Guide\)](#)

[What to do about utilities that were in progress at time of failure \(Db2 Administration Guide\)](#)

[Recovering data in data sharing \(Db2 Data Sharing Planning and Administration\)](#)

Related tasks

[Migration step 1: Complete premigration tasks \(Db2 Installation and Migration\)](#)

[Recovering catalog and directory objects \(Db2 Utilities\)](#)

Related reference

[DSN1LOGP \(Db2 Utilities\)](#)

What to do about utilities that were in progress at time of failure

After you restore data from image copies and archives, you might need to take some additional steps. For example, you need to determine what to do about any utilities that were in progress at the time of the failure.

You might need to take additional steps if any utility jobs were running after the last time that the log was offloaded before the disaster.

After restarting Db2, only certain utilities need to be terminated with the **TERM UTILITY** command.

Allowing the RECOVER utility to reset pending states is preferable. However, you might occasionally need to use the REPAIR utility to reset them. Do not start the table space with ACCESS(FORCE) because FORCE resets any page set exception conditions described in "Database page set controls."

For the following utility jobs, perform the indicated actions:

CHECK DATA

Terminate the utility, and run it again after recovery is complete.

COPY

After you enter the **TERM UTILITY** command, Db2 places a record in the SYSCOPY catalog table to indicate that the COPY utility job was terminated. This makes it necessary for you to make a full image copy. When you copy your environment at the completion of the disaster recovery scenario, you fulfill that requirement.

LOAD

Find the options that you specified in the following table, and perform the specified actions. For the SORTKEYS option, you must specify a value that is greater than zero for *integer*. If you specify zero for *integer*, the SORTKEYS option does not apply.

Table 7. Actions to take when a LOAD utility job is interrupted

| LOAD options specified | What to do |
|--|---|
| LOG YES | If the RELOAD phase completed, recover to the current time. Recover the indexes. If the RELOAD phase did not complete, recover to a prior point in time. The SYSCOPY record that is inserted at the beginning of the RELOAD phase contains the RBA or LRSN. |
| LOG NO and <i>copy-spec</i> | If the RELOAD phase completed, the table space is complete after you recover it to the current time. Recover the indexes. If the RELOAD phase did not complete, recover the table space to a prior point in time. Recover the indexes. |
| LOG NO, <i>copy-spec</i> , and SORTKEYS <i>integer</i> | If the BUILD or SORTBLD phase completed, recover to the current time, and recover the indexes. If the BUILD or SORTBLD phase did not complete, recover to a prior point in time. Recover the indexes. |

Table 7. Actions to take when a LOAD utility job is interrupted (continued)

| LOAD options specified | What to do |
|------------------------|--|
| LOG NO | Recover the table space to a prior point in time. You can use the TOCOPY option of the RECOVER utility to do this. |

To avoid extra loss of data in a future disaster situation, run the QUIESCE utility on table spaces before invoking the LOAD utility. This enables you to recover a table space by using the TOLOGPOINT option instead of TOCOPY.

REORG

For a user table space, find the options that you specified in the following table, and perform the specified actions.

Recommendation: Make full image copies of the catalog and directory before you run REORG on them.

Table 8. Actions to take when the REORG utility is interrupted

| REORG options specified | What to do |
|---------------------------------------|--|
| LOG YES | If the RELOAD phase completed, recover to the current time. Recover the indexes. If the RELOAD phase did not complete, recover to the current time to restore the table space to the point before the REORG job began. Recover the indexes. |
| LOG NO | If the build or SORTBLD phase completed, recover to the current time, and recover the indexes. If the build or SORTBLD phase did not complete, recover to the current time to restore the table space to the point before the REORG job began. Recover the indexes. |
| SHRLEVEL CHANGE or SHRLEVEL REFERENCE | If the SWITCH phase completed, terminate the utility. Recover the table space to the current time. Recover the indexes. If the SWITCH phase did not complete, recover the table space to the current time. Recover the indexes. |

For a catalog or directory table space, the instructions are somewhat different. For those table spaces that were using online REORG, find the options that you specified in the preceding table, and perform the specified actions.

If you have no image copies from immediately before REORG failed, use this procedure:

1. From your **DISPLAY UTILITY** command and DIAGNOSE utility output, determine what phase the REORG job was in and which table space it was reorganizing when the disaster occurred.
2. Run the RECOVER utility on the catalog and directory in the correct order. Recover all table spaces to the current time, except the table space that was being reorganized at the time of the disaster. If the RELOAD phase of the REORG job on that table space had not completed when the disaster occurred, recover the table space to the current time. Because REORG does not generate any log records prior to the RELOAD phase for catalog and directory objects, a recovery to the current time restores the data to the state that it was in before the REORG job. If the RELOAD phase completed, perform the following actions:
 - a. Run the DSN1LOGP utility against the archive log data sets from the disaster site.
 - b. Find the begin-UR log record for the REORG job that failed in the DSN1LOGP output.

- c. Run the RECOVER utility with the TOLOGPOINT option on the table space that was being reorganized. Use the URID of the begin-UR record as the TOLOGPOINT value.
3. Recover or rebuild all indexes.

If you have image copies from immediately before the REORG job failed, run the RECOVER utility with the TOCOPY option to recover the catalog and directory, in the correct order.

Related tasks

[Recovering catalog and directory objects \(Db2 Utilities\)](#)

Recovering from disasters by using a tracker site

You can use a tracker site for disaster recovery. A Db2 *tracker site* is a separate Db2 subsystem or data sharing group that exists solely to keep shadow copies of the data at your primary site.

About this task

Using a tracker site for disaster recovery is somewhat similar to other methods.

Recommendation: Test and document a disaster procedure that is customized for your location.

From the primary site, you transfer the BSDS and the archive logs, and that tracker site runs periodic LOGONLY recoveries to keep the shadow data up-to-date. If a disaster occurs at the primary site, the tracker site becomes the *takeover* site. Because the tracker site has been shadowing the activity on the primary site, you do not need to constantly ship image copies; the takeover time for the tracker site might be faster because Db2 recovery does not need to use image copies.

Characteristics of a tracker site

A *tracker site* is a separate Db2 subsystem or data sharing group that exists solely for the purpose of keeping shadow copies of the data at your primary site.

Because the tracker site must use only the primary site logs for recovery, you must not update the catalog and directory or the data at the tracker site. The Db2 subsystem at the tracker site disallows updates.

- The following SQL statements are not allowed at a tracker site:

- GRANT or REVOKE
- DROP, ALTER, or CREATE
- UPDATE, INSERT, or DELETE

Dynamic read-only SELECT statements are allowed, but not recommended. At the end of each tracker site recovery cycle, databases might contain uncommitted data, and indexes might be inconsistent with the data at the tracker site.

- The only online utilities that are allowed are REPORT, DIAGNOSE, RECOVER, REBUILD, and RESTORE SYSTEM LOGONLY. Recovery to a prior point in time is not allowed.
- **BIND** is not allowed.
- **TERM UTIL** is not allowed for LOAD, REORG, REPAIR, and COPY.
- The **START DATABASE** command is not allowed when LPL or GRECP status exists for the object of the command. Use of the **START DATABASE** command is not necessary to clear LPL or GRECP conditions because you are going to be running RECOVER jobs that clear the conditions.
- The **START DATABASE** command with ACCESS(FORCE) is not allowed.
- Down-level detection is disabled.
- Log archiving is disabled.
- Real-time statistics are disabled.

Setting up a tracker site

For disaster recovery purposes, you might want to set up a tracker site. To set up a tracker site, you create a mirror image of your primary Db2 subsystem, and then ensure that the tracker site is synchronized with the primary site.

Procedure

To set up the tracker site:

1. Create a mirror image of your primary Db2 subsystem or data sharing group.
This process is described in steps 1 through 4 of the normal disaster recovery procedure, which includes creating catalogs and restoring Db2 libraries.
2. Modify the subsystem parameters as follows:
 - Set the TRKRSITE subsystem parameter to YES.
 - Optionally, set the SITETYP parameter to RECOVERYSITE if the full image copies that this site is to receive are created as remote site copies.
3. Use the access method services **DEFINE CLUSTER** command to allocate data sets for all user-managed table spaces that you plan to send over from the primary site.
4. Optional: Allocate data sets for user-managed indexes that you want to rebuild during recovery cycles.
The main reason that you rebuild indexes during recovery cycles is for running efficient queries on the tracker site. If you do not require indexes, you do not need to rebuild them for recovery cycles. For nonpartitioning indexes on very large tables, you can include indexes for LOGONLY recovery during the recovery cycle, which can reduce the amount of time that it takes to bring up the disaster site. Be sure that you define data sets with the proper prefix (either I or J) for both indexes and table spaces.
5. Send full image copies of all Db2 data at the primary site to the tracker site. Optionally, you can use the BACKUP SYSTEM utility with the DATA ONLY option and send copies of the database copy pool to the tracker site.
If you send copies that the BACKUP SYSTEM utility creates, this step completes the tracker site setup procedure.
6. If you did not use the BACKUP SYSTEM utility in the prior, tailor installation job DSNTIJJIN to create Db2 catalog data sets.

What to do next

Important: Do not attempt to start the tracker site when you are setting it up. You must follow the procedure described in [“Establishing a recovery cycle by using RESTORE SYSTEM LOGONLY”](#) on page 148.

Related reference

[BACKUP SYSTEM \(Db2 Utilities\)](#)

Establishing a recovery cycle by using RESTORE SYSTEM LOGONLY

Each time that you restore the logs and the BSDS from the primary site at your tracker site, you establish a new recovery cycle. One way to establish a recovery cycle is to use the RESTORE SYSTEM utility with the LOGONLY option.

Before you begin

Full image copies of all the data at the primary site must be available at the tracker site.

About this task

Using the LOGONLY option of the RESTORE SYSTEM utility enables you to periodically apply the active log, archive logs, and the BSDS from the primary site at the tracker site.

Procedure

To establish a recovery cycle at your tracker site by using the RESTORE SYSTEM utility:

1. While your primary site continues its usual workload, send a copy of the primary site active log, archive logs, and BSDS to the tracker site.

Send full image copies for the following objects:

- Table spaces or partitions that are reorganized, loaded, or repaired with the LOG NO option after the latest recovery cycle
- Objects that, after the latest recovery cycle, have been recovered to a point in time

Recommendation: If you are taking incremental image copies, run the MERGECOPY utility at the primary site before sending the copy to the tracker site.

2. At the tracker site, restore the BSDS that was received from the primary site by following these steps:
 - a) Locate the BSDS in the latest archive log that is now at the tracker site.
 - b) Register this archive log in the archive log inventory of the new BSDS by using the change log inventory utility (DSNJU003).
 - c) Register the primary site active log in the new BSDS by using the change log inventory utility (DSNJU003).
3. Use the change log inventory utility (DSNJU003) with the following CRESTART control statement:

```
CRESTART CREATE, ENDRBA=nnnnnnnn000, FORWARD=NO, BACKOUT=NO
```

In this control statement, *nnnnnnnn* equals the RBA at which the latest archive log record ends +1. Do not specify the RBA at which the archive log begins because you cannot cold start or skip logs in tracker mode.

Data sharing

If you are recovering a data sharing group, you must use the following CRESTART control statement on all members of the data sharing group. The ENDLRSN value must be the same for all members.

```
CRESTART CREATE, ENDLRSN=nnnnnnnnnnnn, FORWARD=NO, BACKOUT=NO
```

In this control statement, *nnnnnnnnnnnn* is the lowest LRSN of all the members that are to be read during restart. Specify one of the following values for the ENDLRSN:

- If you receive the ENDLRSN from the output of the print log map utility (DSNJU004) or from the console logs using message DSNJ003I, you must use ENDLRSN -1 as the input to the conditional restart.
- If you receive the ENDLRSN from the output of the DSN1LOGP utility (message DSN1213I), you can use the displayed value.

The ENDLRSN or ENDRBA value indicates the end log point for data recovery and for truncating the archive log. With ENDLRSN, the missing log records between the lowest and highest ENDLRSN values for all the members are applied during the next recovery cycle.

4. If the tracker site is a data sharing group, delete all Db2 coupling facility structures before restarting the tracker members.
5. If you used the DSN1COPY utility to create a copy of SYSUTILX during the last tracker cycle, restore this copy with DSN1COPY.

Data sharing

For data sharing, restart every member of the data sharing group.

6. At the tracker site, stop and start Db2 to begin a tracker site recovery cycle.
7. At the tracker site, run the RESTORE SYSTEM utility with the LOGONLY option to apply the logs (both archive and active) to the data at the tracker site.

8. If the RESTORE SYSTEM utility issues a return code of 4, use the DSN1COPY utility to make a copy of SYSUTILX and of indexes that are associated with SYSUTILX before you recover or rebuild those objects.
DSN1COPY issues a return code of 4 if application of the log results in one or more Db2 objects being marked as RECP or RBDP.
9. Stop and start Db2 at the tracker site.
10. Issue the **DISPLAY DATABASE RESTRICT** command to display objects that are marked RECP, RBDP, or LPL and to identify which objects are in a utility progress state (such as UTUT or UTRO). Run the RECOVER or REBUILD INDEX utility on these objects, or record which objects are in an exception state so that you can recover them at a later time.
The exception states of these objects are not retained in the next recovery cycle.
11. After all recovery activity complete at the tracker site, shut down the Db2 tracker site.
12. Optional: Stop and start the Db2 tracker site several times before completing a recovery cycle.

Related concepts

[Media failures during LOGONLY recovery \(Db2 Administration Guide\)](#)

Related tasks

[Establishing a recovery cycle by using the RECOVER utility \(Db2 Administration Guide\)](#)

[Restoring data from image copies and archive logs \(Db2 Administration Guide\)](#)

Establishing a recovery cycle by using the RECOVER utility

Each time that you restore the logs and the BSDS from the primary site at your tracker site, you establish a new recovery cycle. One way to establish a recovery cycle is to use the RECOVER utility.

Procedure

To establish a recovery cycle by using the RECOVER utility:

1. While your primary site continues its usual workload, send a copy of the primary site active log, archive logs, and BSDS to the tracker site.

Send full image copies for the following objects:

- Table spaces or partitions that are reorganized, loaded, or repaired with the LOG NO option after the latest recovery cycle.
- Objects that, after the latest recovery cycle, have been recovered to a point in time.
- SYSUTILX. Send a full image copy to DSNDB01.SYSUTILX for normal (full image copy and log) recoveries. For LOGONLY recoveries, create a copy of DSNDB01.SYSUTILX by using the DSN1COPY utility.

Db2 does not write SYSLGRNX entries for DSNDB01.SYSUTILX, which can lead to long recovery times at the tracker site. In addition, SYSUTILX and its indexes are updated during the tracker cycle when you run your recoveries. Because SYSUTILX must remain consistent with the SYSUTILX at the primary site, discard the tracker cycle updates before the next tracker cycle.

Recommendation: If you are taking incremental image copies, run the MERGECOPY utility at the primary site before sending the copy to the tracker site.

2. At the tracker site, restore the BSDS that was received from the primary site by using one of the following methods:
 - Locate the BSDS in the latest archive log that is now at the tracker site.
 - Register this archive log in the archive log inventory of the new BSDS by using the change log inventory utility (DSNJU003).
 - Register the primary site active log in the new BSDS by using the change log inventory utility (DSNJU003).
3. Use the change log inventory utility (DSNJU003) with the following CRESTART control statement:

```
CRESTART CREATE, ENDRBA=nnnnnnnn000, FORWARD=NO, BACKOUT=NO
```

In this control statement, *nnnnnnnn000* equals the value of the ENDRBA of the latest archive log plus 1. Do not specify STARTRBA because you cannot cold start or skip logs in a tracker system.

Data sharing

If you are recovering a data sharing group, you must use the following CRESTART control statement on all members of the data sharing group. The ENDLRSN value must be the same for all members.

```
CRESTART CREATE, ENDLRSN=nnnnnnnnnnnn, FORWARD=NO, BACKOUT=NO
```

In this control statement, *nnnnnnnnnnnn* is the lowest ENDLRSN of all the members that are to be read during restart. Specify one of the following values for the ENDLRSN:

- If you receive the ENDLRSN from the output of the print log map utility (DSNJU004) or from message DSNJ003I at the console logs use ENDLRSN -1 as the input to the conditional restart.
- If you receive the ENDLRSN from the output of the DSN1LOGP utility (DSN1213I message), use the displayed value.

The ENDLRSN or ENDRBA value indicates the end log point for data recovery and for truncating the archive log. With ENDLRSN, the missing log records between the lowest and highest ENDLRSN values for all the members are applied during the next recovery cycle.

4. If the tracker site is a data sharing group, delete all Db2 coupling facility structures before restarting the tracker members.
5. At the tracker site, restart Db2 to begin a tracker site recovery cycle.

Data sharing

For data sharing, restart every member of the data sharing group.

6. At the tracker site, submit RECOVER utility jobs to recover database objects. Run the RECOVER utility with the LOGONLY option on all database objects that do not require recovery from an image copy.

You must recover database objects as the following procedure specifies:

- a) Restore the full image copy or DSN1COPY of SYSUTILX.

If you are doing a LOGONLY recovery on SYSUTILX from a previous DSN1COPY backup, make another DSN1COPY copy of that table space after the LOGONLY recovery is complete and before recovering any other catalog or directory objects.

After you recover SYSUTILX and either recover or rebuild its indexes, and before you recover other system and user table spaces, determine what utilities were running at the primary site.

- b) Recover the catalog and directory in the correct order, as described in [Recovering catalog and directory objects \(Db2 Utilities\)](#).

Important: For the first recovery cycle after job DSNTIJCV converts the catalog and directory to the extended RBA or LRSN format (or if the catalog and directory are converted back to basic format), stop and start all Db2 members after the indexes for DSNDB01.SYSDBDXA are rebuilt. Then continue recovering the rest of the catalog and directory in the correct order.

If you have user-defined catalog indexes, rebuilding them is optional until the tracker Db2 site becomes the takeover Db2 site. (You might want to rebuild them sooner if you require them for catalog query performance.) However, if you are recovering user-defined catalog indexes, do the recovery in this step.

Exception: If you have any user-defined, STOGROUP-managed indexes on the Db2 catalog and directory, you must rebuild IBM-defined indexes by name.

- c) If needed, recover other system data such as the data definition control support table spaces and the resource limit facility table spaces.
- d) Recover user data and, optionally, rebuild your indexes.

You do not need to rebuild indexes unless you intend to run dynamic queries on the data at the tracker site.

For a tracker site, Db2 stores the conditional restart ENDRBA or ENDLRSN in the page set after each recovery completes successfully. By storing the log truncation value in the page set, Db2 ensures that it does not skip any log records between recovery cycles.

7. Issue the **DISPLAY UTILITY(*)** command for a list of currently running utilities.
8. Run the DIAGNOSE utility with the DISPLAY SYSUTIL statement to determine the names of the object on which the utilities are running.

Installation SYSOPR authority is required.

9. Perform the following actions for objects at the tracker site on which utilities are pending.

Restrictions apply to these objects because Db2 prevents you from using the **TERM UTILITY** command to remove pending statuses at a tracker site.

- If a LOAD, REORG, REPAIR, or COPY utility job is in progress on any catalog or directory object at the primary site, shut down Db2 subsystem. You cannot continue recovering by using the list of catalog and directory objects. Therefore, you cannot recover any user data. At the next recovery cycle, send a full image copy of the object from the primary site. At the tracker site, use the RECOVER utility to restore the object.
- If a LOAD, REORG, REPAIR, or COPY utility job is in progress on any user data, at the next recovery cycle, send a full image copy of the object from the primary site. At the tracker site, use the RECOVER utility to restore the object.
- If an object is in the restart-pending state, use LOGONLY recovery to recover the object when that object is no longer in a restart-pending state.

Data sharing

If read/write shared data (GPB-dependent data) is in the advisory recovery pending state, the tracker Db2 site performs recovery processing. Because the tracker Db2 site always performs a conditional restart, the postponed indoubt units of recovery are not recognized after the tracker Db2 site restarts.

10. After all recovery has completed at the tracker site, shut down the tracker Db2 site.

This is the end of the tracker site recovery cycle.

11. Optional: Stop and start the tracker Db2 site several times before completing a recovery cycle.

Related concepts

[Media failures during LOGONLY recovery \(Db2 Administration Guide\)](#)

Related tasks

[Establishing a recovery cycle by using RESTORE SYSTEM LOGONLY \(Db2 Administration Guide\)](#)

[Restoring data from image copies and archive logs \(Db2 Administration Guide\)](#)

Media failures during LOGONLY recovery

If an I/O error occurs during a LOGONLY recovery, you can recover the object by using the image copies and logs after you correct the media failure.

If an entire volume is corrupted and you are using Db2 storage groups, you cannot use the ALTER STOGROUP statement to remove the corrupted volume and add another. (This is possible, however, for a non-tracker system.) Instead, you must remove the corrupted volume and re-initialize another volume with the same volume serial number before you invoke the RECOVER utility for all table spaces and indexes on that volume.

Maintaining a tracker site

If you want to have a tracker site for possible disaster recovery needs, you need to maintain it so that it can operate as required.

Procedure

To maintain a tracker site:

1. Keep the tracker site and primary site at the same maintenance level to avoid unexpected problems.
2. Between recovery cycles, apply maintenance as you normally do, by stopping and restarting the Db2 subsystem or a Db2 data sharing member.
3. If a tracker site fails, restart it as you normally do.
4. Save your complete tracker site prior to testing a takeover site.

This step is necessary because bringing up a tracker site as the takeover site destroys the tracker site environment. After testing the takeover site, you can restore the tracker site and resume the recovery cycles.

Results

When restarting a data sharing group, the first member that starts during a recovery cycle puts the ENDLRSN value in the shared communications area (SCA) of the coupling facility. If an SCA failure occurs during a recovery cycle, you must go through the recovery cycle again, using the same ENDLRSN value for your conditional restart.

Making the tracker site be the takeover site

If a disaster occurs at the primary site, the tracker site must become the takeover site.

Before you begin

Save your complete tracker site prior to testing a takeover site.

Procedure

To make the tracker site be the takeover site:

1. Restart the takeover site.
2. Apply log data or image copies that were en route when the disaster occurred.
3. Follow the appropriate procedure for making the tracker site a takeover site, depending on whether you use RESTORE SYSTEM LOGONLY or the RECOVER utility in your tracker site recovery cycles.

Related tasks

[Maintaining a tracker site \(Db2 Administration Guide\)](#)

Recovering at a tracker site that uses the RESTORE SYSTEM utility

One way that you can make the tracker site be the takeover site is by using the RESTORE SYSTEM utility with the LOGONLY option in the recovery cycles at the tracker site.

Procedure

To make the tracker site be the takeover site by using the RESTORE SYSTEM utility with the LOGONLY option:

1. If log data for a recovery cycle is en route or is available but has not yet been used in a recovery cycle, perform the procedure in [“Establishing a recovery cycle by using RESTORE SYSTEM LOGONLY” on page 148](#).
2. Ensure that the TRKSITE NO subsystem parameter is specified.
3. For scenarios other than data sharing, continue with step [“4” on page 154](#).

Data sharing

If this is a data sharing system, delete the coupling facility structures.

4. Start Db2 at the same RBA or ENDLRSN that you used in the most recent tracker site recovery cycle. Specify FORWARD=YES and BACKOUT=YES in the CRESTART statement; this takes care of uncommitted work.
5. Restart the objects that are in GRECP or LPL status by issuing the START DATABASE(*) SPACENAM(*) command.
6. If you used the DSN1COPY utility to create a copy of SYSUTILX in the last recovery cycle, use DSN1COPY to restore that copy.
7. Terminate any in-progress utilities by using the following procedure:
 - a) Enter the **DISPLAY UTILITY(*)** command .
 - b) Run the DIAGNOSE utility with DISPLAY SYSUTIL to get the names of objects on which utilities are being run.
 - c) Terminate in-progress utilities in the correct order by using the **TERM UTILITY(*)** command.
8. Rebuild indexes, including IBM and user-defined indexes on the Db2 catalog and user-defined indexes on table spaces.
9. If subsystem parameter CDDS_MODE is set to SOURCE_ONLY, populate the compression dictionary data set (CDDS) so that Db2 can use the expansion dictionaries in the CDDS to decompress log records for IFI calls for IFCID 306. Follow these steps:
 - a) Define the CDDS if it is not already defined. See [Storing the expansion dictionary for compressed log records in the compression dictionary data set \(Db2 Administration Guide\)](#) for an example of the CDDS definition.
 - b) Issue the **-START CDDS** command to direct the Db2 subsystem or all data sharing members to allocate and open the CDDS.
 - c) To populate the CDDS, run REORG TABLESPACE with the INITCDDS option on each compressed table space or partition that contains a table that is defined with DATA CAPTURE CHANGES. You can specify the SEARCHTIME option with the INITCDDS option to allow REORG to populate the CDDS with an earlier dictionary than the dictionary that currently resides in the table space.

Related tasks

[Restoring data from image copies and archive logs \(Db2 Administration Guide\)](#)

[Recovering at a tracker site that uses the RECOVER utility \(Db2 Administration Guide\)](#)

Recovering at a tracker site that uses the RECOVER utility

One way that you can make the tracker site be the takeover site is by using the RECOVER utility in the recovery cycles at your tracker site.

Procedure

To make the tracker site be the takeover site by using the RECOVER utility:

1. Restore the BSDS, and register the archive log from the last archive log that you received from the primary site.
2. For environments that do not use data sharing, continue with step “3” on [page 154](#).

Data sharing

If this is a data sharing system, delete the coupling facility structures.

3. Ensure that the DEFER ALL and TRKSITE NO subsystem parameters are specified.
4. Take the appropriate action, which depends on whether you received more logs from the primary site.

If this is a non-data-sharing Db2 subsystem, the log truncation point varies depending on whether you have received more logs from the primary site since the last recovery cycle:

- If you did not receive more logs from the primary site:

Start Db2 using the same ENDRBA that you used on the last tracker cycle. Specify FORWARD=YES and BACKOUT=YES; this takes care of uncommitted work. If you have fully recovered the objects

during the previous cycle, they are current except for any objects that had outstanding units of recovery during restart. Because the previous cycle specified NO for both FORWARD and BACKOUT and you have now specified YES, affected data sets are placed in the LPL. Restart the objects that are in LPL status by using the following command:

```
START DATABASE(*) SPACENAM(*)
```

After you issue the command, all table spaces and indexes that were previously recovered are now current. Remember to rebuild any indexes that were not recovered during the previous tracker cycle, including user-defined indexes on the Db2 catalog.

- If you received more logs from the primary site:

Start Db2 using the truncated RBA *nnnnnnnnn000*, which equals the value of the ENDRBA of the latest archive log plus 1. Specify FORWARD=YES and BACKOUT=YES. Run your recoveries as you did during recovery cycles.

Data sharing

You must restart every member of the data sharing group; use the following CRESTART statement:

```
CRESTART CREATE, ENDLRSN=nnnnnnnnnnnn, FORWARD=YES, BACKOUT=YES
```

In this statement, *nnnnnnnnnnnn* is the LRSN of the last log record that is to be used during restart. Specify one of the following values for the ENDLRSN:

- If you receive the ENDLRSN from the output of the print log map utility (DSNJU004) or from message DSNJ003I at the console logs use ENDLRSN -1 as the input to the conditional restart.
- If you receive the ENDLRSN from the output of the DSN1LOGP utility (DSN1213I message), use the displayed value.

The ENDLRSN or ENDRBA value indicates the end log point for data recovery and for truncating the archive log. With ENDLRSN, the missing log records between the lowest and highest ENDLRSN values for all the members are applied during the next recovery cycle.

The takeover Db2 sites must specify conditional restart with a common ENDLRSN value to allow all remote members to logically truncate the logs at a consistent point.

5. As described for a tracker recovery cycle, recover SYSUTILX from an image copy from the primary site, or from a previous DSN1COPY copy that was taken at the tracker site.
6. Terminate any in-progress utilities by using the following procedure:
 - a) Enter the command **DISPLAY UTILITY(*)**.
 - b) Run the DIAGNOSE utility with DISPLAY SYSUTIL to get the names of objects on which utilities are being run.
 - c) Terminate in-progress utilities by using the command **TERM UTILITY(*)**.
7. Continue with your recoveries either with the LOGONLY option or with image copies. Remember to rebuild indexes, including IBM and user-defined indexes on the Db2 catalog and user-defined indexes on table spaces.
8. If subsystem parameter CDDS_MODE is set to SOURCE_ONLY, populate the compression dictionary data set (CDDS) so that Db2 can use the expansion dictionaries in the CDDS to decompress log records for IFI calls for IFCID 306. Follow these steps:
 - a) Define the CDDS if it is not already defined. See [Storing the expansion dictionary for compressed log records in the compression dictionary data set \(Db2 Administration Guide\)](#) for an example of the CDDS definition.
 - b) Issue the **-START CDDS** command to direct the Db2 subsystem or all data sharing members to allocate and open the CDDS.
 - c) To populate the CDDS, run REORG TABLESPACE with the INITCDDS option on each compressed table space or partition that contains a table that is defined with DATA CAPTURE CHANGES. You can specify the SEARCHTIME option with the INITCDDS option to allow REORG to populate the CDDS with an earlier dictionary than the dictionary that currently resides in the table space.

Related tasks

[Restoring data from image copies and archive logs \(Db2 Administration Guide\)](#)

[Recovering at a tracker site that uses the RESTORE SYSTEM utility \(Db2 Administration Guide\)](#)

Using data mirroring for disaster recovery

Data mirroring is the automatic replication of current data from your primary site to a secondary site. To recover after a disaster, you can use this secondary site for your recovery site without the need to restore Db2 image copies. You also do not need to apply Db2 logs to bring Db2 data to the current point in time.

About this task

The procedures for data mirroring are intended for environments that mirror an entire Db2 subsystem or data sharing group, which includes the catalog, directory, user data, BSDS, and active logs. You must mirror all volumes in such a way that they terminate at exactly the same point. You can achieve this final condition by using consistency groups.

Follow the appropriate procedure for recovering from a disaster by using data mirroring.

Role of data mirroring in recovery from a rolling disaster

In a real disaster, your local site gradually and intermittently fails for a duration of several seconds. This kind of Db2 failure is known as a *rolling disaster*. You can recover from a rolling disaster by using data mirroring.

To use data mirroring for disaster recovery, you must mirror data from your local site with a method that does not reproduce a rolling disaster at your recovery site. To recover a Db2 subsystem and data with data integrity, you must use volumes that end at a consistent point in time for each Db2 subsystem or data sharing group. Mirroring a rolling disaster causes volumes at your recovery site to end over a span of time rather than at one single point.

The following figure shows how a rolling disaster can cause data to become inconsistent between two subsystems.

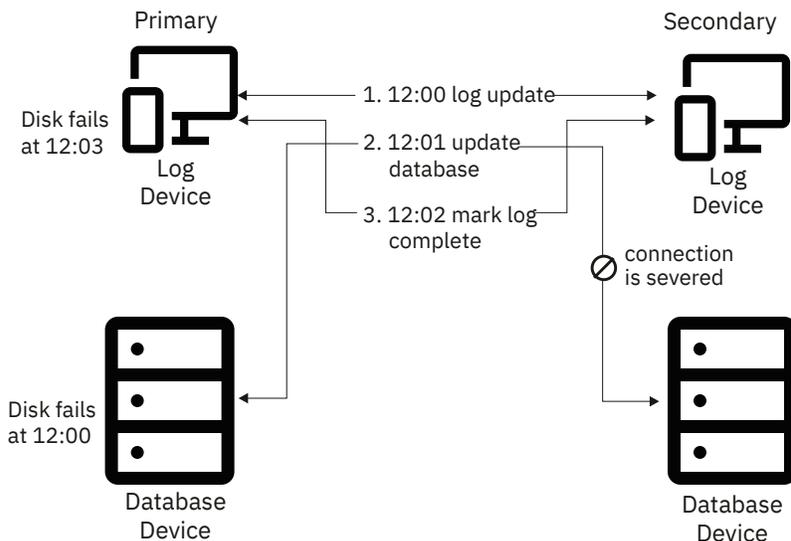


Figure 13. Data inconsistency caused by a rolling disaster

Example

In a rolling disaster, the following events at the primary site cause data inconsistency at your recovery site. This data inconsistency example follows the same scenario that the preceding figure depicts.

1. Some time prior to 12:00: A table space is updated in the buffer pool.

2. 12:00 The log record is written to disk on logical storage subsystem 1.
3. 12:01: Logical storage subsystem 2 fails.
4. 12:02: The update to the table space is externalized to logical storage subsystem 2 but is not written because subsystem 2 failed.
5. 12:03: The log record that indicates that the table space update was made is written to disk on logical storage subsystem 1.
6. 12:03: Logical storage subsystem 1 fails.

Because the logical storage subsystems do not fail at the same point in time, they contain inconsistent data. In this scenario, the log indicates that the update is applied to the table space, but the update is not applied to the data volume that holds this table space.

Important: Any disaster recovery solution that uses data mirroring must guarantee that all volumes at the recovery site contain data for the same point in time.

Role of consistency groups in recovery

Generally a *consistency group* is a collection of volumes that contain consistent, related data. Consistency groups play an important role in Db2 recovery.

A consistency group, which is a collection of related data, can span logical storage subsystems and disk subsystems. For Db2 specifically, a consistency group contains an entire Db2 subsystem or an entire Db2 data sharing group.

The following Db2 elements comprise a consistency group:

- Catalog tables
- Directory tables
- BSDS
- Logs
- All user data
- ICF catalogs

Additionally, all objects within a consistency group must represent the same point in time in at least one of the following situations:

- At the time of a backup
- After a normal Db2 restart

You can use the following methods to create consistency groups:

- XRC I/O timestamping and system data mover
- FlashCopy® consistency groups
- GDPSfreeze policies
- The Db2 SET LOG SUSPEND command

When a rolling disaster strikes your primary site, consistency groups guarantee that all volumes at the recovery site contain data for the same point in time. In a data mirroring environment, you must perform both of the following actions for each consistency group that you maintain:

- Mirror data to the secondary volumes in the same sequence that Db2 writes data to the primary volumes.

In many processing situations, Db2 must complete one write operation before it begins another write operation on a different disk group or a different storage server. A write operation that depends on a previous write operation is called a *dependent write*. Do not mirror a dependent write if you have not mirrored the write operation on which the dependent write depends. If you mirror data out of sequence, your recovery site will contain inconsistent data that you cannot use for disaster recovery.

- Temporarily suspend and queue write operations to create a group point of consistency when an error occurs between any pair of primary and secondary volumes.

When an error occurs that prevents the update of a secondary volume in a single-volume pair, this error might mark the beginning of a rolling disaster. To prevent your secondary site from mirroring a rolling disaster, you must suspend and queue data mirroring by taking the following steps after a write error between any pairs:

1. Suspend and queue all write operations in the volume pair that experiences a write error.
2. Invoke automation that temporarily suspends and queues data mirroring to all your secondary volumes.
3. Save data at the secondary site at a point of consistency.
4. If a rolling disaster does not strike your primary site, resume normal data mirroring after some amount of time that you define. If a rolling disaster does strike your primary site, follow the recovery procedure in [Recovering in a data mirroring environment \(Db2 Administration Guide\)](#).

Recovering in a data mirroring environment

In a data mirroring environment, you can recover data at your secondary site from a disaster at your primary site.

About this task

This procedure applies to all Db2 data mirroring scenarios except those that use Extended Remote Copy (XRC). This general procedure is valid only if you have established and maintained consistency groups before the disaster struck the primary site. If you use data mirroring to recover, you must recover your entire Db2 subsystem or data sharing group with data mirroring.

You do not need to restore Db2 image copies or apply Db2 logs to bring Db2 data to the current point in time when you use data mirroring. However, you might need image copies at the recovery site if the LOAD, UNLOAD, or RECOVER utility was active at the time of the disaster.

Procedure

To recover at the secondary site after a disaster:

1. At your recovery site, IPL all z/OS images that correspond to the z/OS images that you lost at your primary site.
2. For environments that do not use data sharing, continue with step “3” on page 158.

Data sharing

For data sharing groups, you must remove old information from the coupling facility.

- a. Enter the following z/OS command to display the structures for this data sharing group:

```
D XCF,STRUCTURE,STRNAME=grpname*
```

- b. For group buffer pools and the lock structure, enter the following command to force off the connections in those structures:

```
SETXCF FORCE,CONNECTION,STRNAME=strname,CONNAME=ALL
```

- c. Delete all the Db2 coupling facility structures by using the following command for each structure:

```
SETXCF FORCE,STRUCTURE,STRNAME=strname
```

3. If you are using the distributed data facility, set LOCATION and LUNAME in the BSDS to values that are specific to your new primary site.

To set LOCATION and LUNAME, run the stand-alone change log inventory utility (DSNJU003) with the following control statement:

```
DDF LOCATION=locname, LUNAME=luname
```

4. Start all Db2 members by using local DSNZPARM data sets and perform a normal restart.

Data sharing

For data sharing groups, Db2 performs group restart. Shared data sets are set to GRECP (group buffer pool RECOVER-pending) status, and pages are added to the LPL (logical page list).

5. For scenarios other than data sharing, continue to step “6” on page 159.

Data sharing

For objects that are in GRECP status, Db2 automatically recovers the objects during restart. Message DSNI049I is issued when the recovery for all objects that are in GRECP status is complete. A message is issued for each member, even if the member did not perform GRECP recovery.

After message DSNI049I is issued:

- a. Display all data sets with GRECP or LPL status by issuing the following Db2 command:

```
-DISPLAY DATABASE(*) SPACENAM(*) RESTRICT(GRECP, LPL) LIMIT(*)
```

Record the output that this command generates.

6. Use the following Db2 command to display all utilities that the failure interrupted:

```
-DISPLAY UTILITY(*)
```

If utilities are pending, record the output from this command, and continue to the next step. You cannot restart utilities at a recovery site. You will terminate these utilities in step “8” on page 159. If no utilities are pending, continue to step number “9” on page 159.

7. Use the DIAGNOSE utility to access the SYSUTIL directory table.

You cannot access this directory table by using normal SQL statements (as you can with most other directory tables). You can access SYSUTIL only by using the DIAGNOSE utility, which is normally intended to be used under the direction of IBM Software Support.

Use the following control statement to run the DIAGNOSE utility job:

```
DIAGNOSE DISPLAY SYSUTIL
```

To stop the utility, issue this control statement:

```
END DIAGNOSE
```

Examine the output. Record the phase in which each pending utility was interrupted, and record the object on which each utility was operating.

8. Terminate all pending utilities with the following command:

```
-TERM UTILITY(*)
```

9. For environments that do not use data sharing, continue to step “10” on page 160.

Data sharing

For data sharing groups, use the following **START DATABASE** command on each database that contains objects that are in LPL status:

```
-START DATABASE(database) SPACENAM(*)
```

When you use the **START DATABASE** command to recover objects, you do not need to provide Db2 with image copies.

Tip: Use up to 10 **START DATABASE** commands for each Db2 subsystem to increase the speed at which Db2 completes this operation. Multiple commands that run in parallel complete faster than a single command that specifies the same databases.

10. Start all remaining database objects with the following **START DATABASE** command:

```
START DATABASE(*) SPACENAM(*)
```

11. For each object that the LOAD utility places in a restrictive status, take one of the following actions:

- If the object was a target of a LOAD utility control statement that specified SHRLEVEL CHANGE, restart the LOAD utility on this object at your convenience. This object contains valid data.
- If the object was a target of a LOAD utility control statement that specified SHRLEVEL NONE and the LOAD job was interrupted before the RELOAD phase, rebuild the indexes on this object.
- If the object was a target of a LOAD utility control statement that specified SHRLEVEL NONE and the LOAD job was interrupted during or after the RELOAD phase, recover this object to a point in time that is before this utility ran.
- Otherwise, recover the object to a point in time that is before the LOAD job ran.

12. For each object that the REORG utility places in a restrictive status, take one of the following actions:

- When the object was a target of a REORG utility control statement that specified SHRLEVEL NONE:
 - If the REORG job was interrupted before the RELOAD phase, no further action is required. This object contains valid data, and the indexes on this object are valid.
 - If the REORG job was interrupted during the RELOAD phase, recover this object to a point in time that is before this utility ran.
 - If the REORG job was interrupted after the RELOAD phase, rebuild the indexes on the object.
- When the object was a target of a REORG utility control statement that does not specify SHRLEVEL NONE:
 - If the REORG job was interrupted before the SWITCH phase, no further action is required. This object contains valid data, and the indexes on this object are valid.
 - If the REORG job was interrupted during the SWITCH phase, no further action is required. This object contains valid data, and the indexes on this object are valid.
 - If the REORG job was interrupted after the SWITCH phase, you might need to rebuild non-partitioned secondary indexes.

13. If subsystem parameter CDDS_MODE is set to SOURCE_ONLY, populate the compression dictionary data set (CDDS) so that Db2 can use the expansion dictionaries in the CDDS to decompress log records for IFI calls for IFCID 306. Follow these steps:

a) Take one of the following actions, depending on whether the CDDS exists:

- If the CDDS exists:
 - i) Issue the **-STOP CDDS** command to direct the Db2 subsystem or all data sharing members to close and deallocate the CDDS.
 - ii) Delete and redefine the CDDS. See [Storing the expansion dictionary for compressed log records in the compression dictionary data set \(Db2 Administration Guide\)](#) for an example of the CDDS definition.
 - iii) Issue the **-START CDDS** command to direct the Db2 subsystem or all data sharing members to allocate and open the CDDS.
- If the CDDS does not exist:
 - i) Define the CDDS. See [Storing the expansion dictionary for compressed log records in the compression dictionary data set \(Db2 Administration Guide\)](#) for an example of the CDDS definition.
 - ii) Issue the **-START CDDS** command to direct the Db2 subsystem or all data sharing members to allocate and open the CDDS.

b) To populate the CDDS, run REORG TABLESPACE with the INITCDDS option on each compressed table space or partition that contains a table that is defined with DATA CAPTURE CHANGES. You

can specify the SEARCHTIME option with the INITCDDS option to allow REORG to populate the CDDS with an earlier dictionary than the dictionary that currently resides in the table space.

Managing DFSMSHsm default settings when using the BACKUP SYSTEM, RESTORE SYSTEM, and RECOVER utilities

In some data mirroring situations, you might need to set or override the DFSMSHsm default settings for the BACKUP SYSTEM, RESTORE SYSTEM, and RECOVER utilities.

About this task

For example, consider that the source volumes in the SMS storage groups for your database or log copy pools are mirrored, or that the target volumes in the SMS backup storage groups for your copy pools are mirrored. You can use IBM Remote Pair FlashCopy (Preserve Mirror) for Peer-to-Peer Remote Copy (PPRC). Also, you can allow FlashCopy to PPRC primary volumes. However, you might need to set or override the DFSMSHsm default settings for the BACKUP SYSTEM, RESTORE SYSTEM, and RECOVER utilities.

Procedure

Issue the DFSMSHsm FRBACKUP PREPARE command.

- To set the DFSMSHsm defaults for the BACKUP SYSTEM utility, the RESTORE SYSTEM utility, and the RECOVER utility, issue the following command:

```
FRBACKUP CP cp-name PREPARE ALLOWPPRCP (FRBACKUP (x) FRRECOV (x))
```

- To override the DFSMSHsm defaults for the RESTORE SYSTEM utility or the RECOVER utility, specify the FLASHCOPY_PPRCP utility option or issue the following command:

```
FRBACKUP CP cp-name PREPARE ALLOWPPRCP (FRRECOV (x))
```

Related concepts

[Considerations for using the BACKUP SYSTEM utility and DFSMSHsm \(Db2 Administration Guide\)](#)

Related reference

[FRBACKUP command: Requesting a fast replication backup or dump version DFSMSHsm Storage Administration Reference](#)

[Syntax and options of the RECOVER control statement \(Db2 Utilities\)](#)

[Syntax and options of the RESTORE SYSTEM control statement \(Db2 Utilities\)](#)

Recovering with Extended Remote Copy

One method that ensures that data volumes remain consistent at your recovery site involves Extended Remote Copy (XRC). In XRC remote mirroring, the DFSMS Advanced Copy services function automatically replicates current data from your primary site to a secondary site and establishes consistency groups.

Before you begin

This procedure assumes that you are familiar with basic use of XRC.

Procedure

To recover at an XRC secondary site after a disaster:

- Issue the TSO command **XEND XRC** to end the XRC session.
- Issue the TSO command **XRECOVER XRC**. This command changes your secondary site to your primary site and applies the XRC journals to recover data that was in transit when your primary site failed.
- Complete the procedure in [Recovering in a data mirroring environment \(Db2 Administration Guide\)](#).

Related information

[Extended Remote Copy \(DFSMS Advanced Copy Services\)](#)

Scenarios for resolving problems with indoubt threads

Indoubt threads can cause a variety of problems, but you can recover from these problems.

The recovery scenarios for indoubt threads are based on a sample environment, which this topic describes. System programmer, operator, and database administrator actions are indicated for the examples as appropriate. In these descriptions, the term "administrator" refers to the database administrator (DBA) if not otherwise specified.

Configuration

The configuration includes four systems at three geographic locations: Seattle (SEA), San Jose (SJ) and Los Angeles (LA). The system descriptions are as follows.

- Db2 subsystem at Seattle, Location name = IBMSEADB20001, Network name = IBM.SEADB21
- Db2 subsystem at San Jose, Location name = IBMSJ0DB20001, Network name = IBM.SJDB21
- Db2 subsystem at Los Angeles, Location name = IBMLA0DB20001, Network name = IBM.LADB21
- IMS subsystem at Seattle, Connection name = SEAIMS01

Applications

The following IMS and TSO applications run at Seattle and access both local and remote data.

- IMS application, IMSAPP01, at Seattle, accesses local data and remote data by DRDA access at San Jose, which accesses remote data on behalf of Seattle by Db2 private protocol access at Los Angeles.
- TSO application, TSOAPP01, at Seattle, accesses data by DRDA access at San Jose and at Los Angeles.

Threads

The following threads are described and keyed to [Figure 14 on page 163](#). Database access threads (DBAT) access data on behalf of a thread (either allied or DBAT) at a remote requester.

- Allied IMS thread A at Seattle accesses data at San Jose by DRDA access.
 - DBAT at San Jose accesses data for Seattle by DRDA access 1 and requests data at Los Angeles by Db2 private protocol access 2.
 - DBAT at Los Angeles accesses data for San Jose by Db2 private protocol access 2.
- Allied TSO thread B at Seattle accesses local data and remote data at San Jose and Los Angeles, by DRDA access.
 - DBAT at San Jose accesses data for Seattle by DRDA access 3.
 - DBAT at Los Angeles accesses data for Seattle by DRDA access 4.

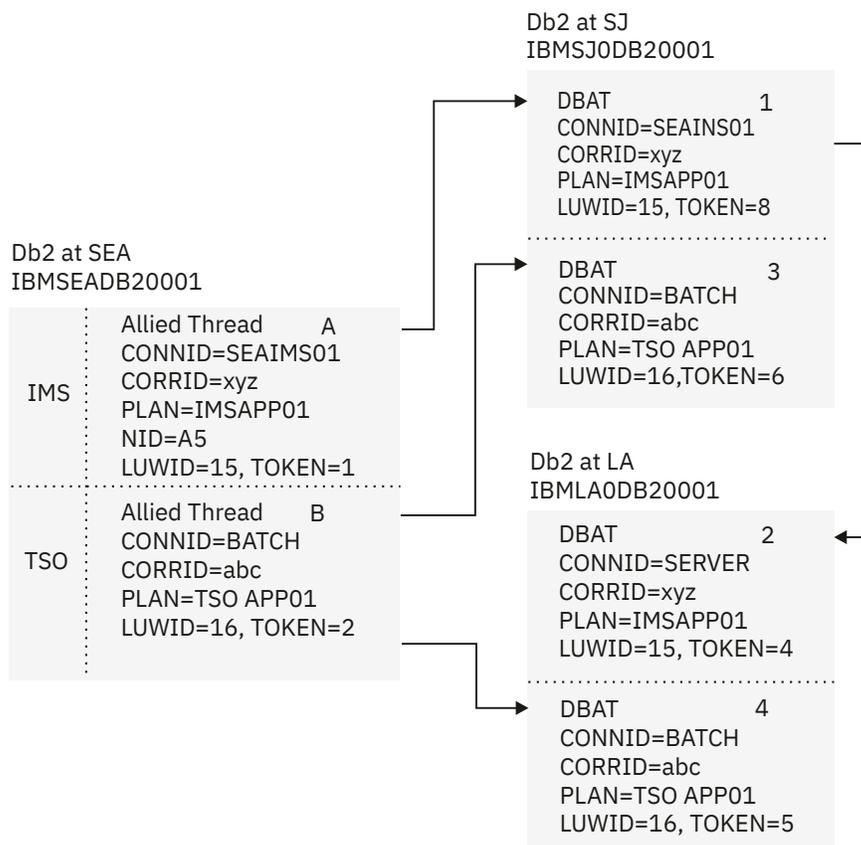


Figure 14. Resolution of indoubt threads

The results of issuing the **DISPLAY THREAD TYPE(ACTIVE)** command to display the status of threads at all Db2 locations are summarized in the boxes of the preceding figure. The logical unit of work IDs (LUWIDs) have been shortened for readability, as follows:

- LUWID=15 is IBM.SEADB21.15A86A876789.0010.
- LUWID=16 is IBM.SEADB21.16B57B954427.0003.

For the purposes of procedures that are based on this configuration, assume that both applications have updated data at all Db2 locations. In the following problem scenarios, the error occurs after the coordinator has recorded the commit decision, but before the affected participants have recorded the commit decision. These participants are therefore indoubt.

Read one or more of the scenarios to learn how best to handle problems with indoubt threads in your own environment.

Scenario: Recovering from communication failure

A communication failure can cause an indoubt thread.

Symptoms

A communication failure occurred between Seattle (SEA) and Los Angeles (LA) after the database access thread (DBAT) at LA completed phase 1 of commit processing. At SEA, the TSO thread, LUWID=16 and TOKEN=2 B, cannot complete the commit with the DBAT at LA4.

At SEA, NetView alert A006 is generated, and message DSNL406 is displayed, indicating that an indoubt thread at LA because of a communication failure. At LA, alert A006 is generated, and message DSNL405 is displayed, to indicate that a thread is in an indoubt state because of a communication failure with SEA.

Causes

A communication failure caused the indoubt thread.

Environment

The following figure illustrates the environment for this scenario.

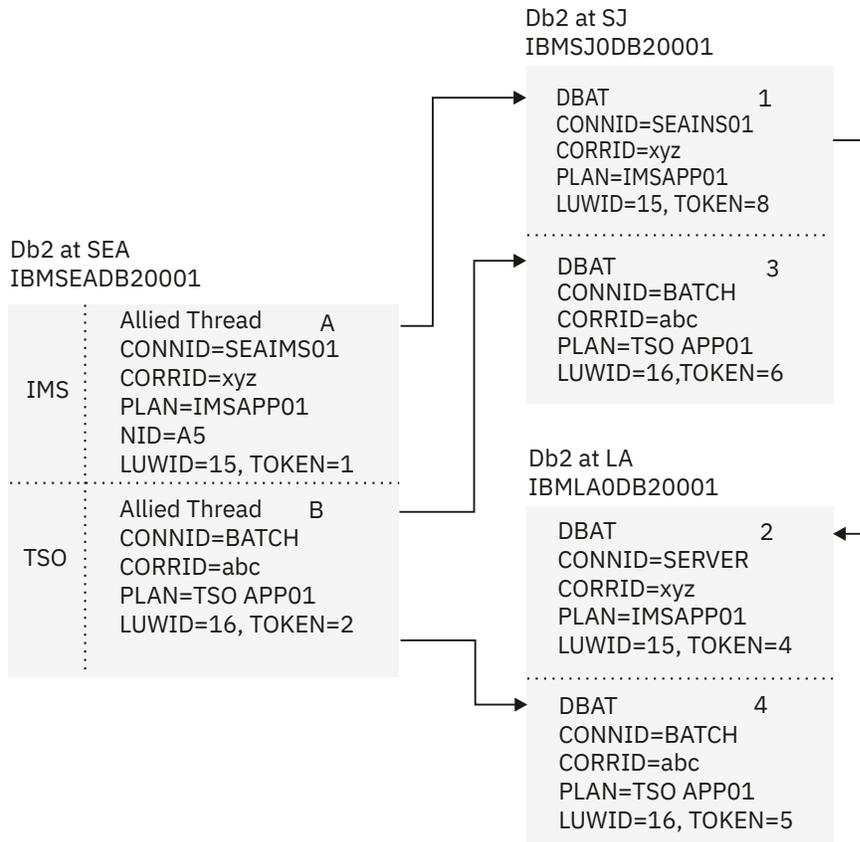


Figure 15. Resolution of indoubt threads

At SEA, an IFCID 209 trace record is written. After the alert is generated and the message is displayed, the thread completes the commit, which includes the DBAT at SJ 3. Concurrently, the thread is added to the list of threads for which the SEA Db2 subsystem has an indoubt resolution responsibility. The thread shows up in a **DISPLAY THREAD** report for indoubt threads. The thread also shows up in a **DISPLAY THREAD** report for active threads until the application terminates.

The TSO application is informed that the commit succeeded. If the application continues and processes another SQL request, it is rejected with an SQL code to indicate that it must roll back before any more SQL requests can be processed. This is to ensure that the application does not proceed with an assumption based on data that is retrieved from LA, or with the expectation that cursor positioning at LA is still intact.

At LA, an IFCID 209 trace record is written. After the alert is generated and the message displayed, the DBAT 4 is placed in the indoubt state. All locks remain held until resolution occurs. The thread shows up in a **DISPLAY THREAD** report for indoubt threads.

The Db2 subsystems, at both SEA and LA, periodically attempt to reconnect and automatically resolve the indoubt thread. If the communication failure affects only the session that is being used by the TSO application, and other sessions are available, automatic resolution occurs in a relatively short time. At this time, message DSNL407 is displayed by both Db2 subsystems.

Resolving the problem

Operator response: If message DSNL407 or DSNL415 for the thread that is identified in message DSNL405 is not issued in a reasonable period of time, contact the system programmer. A communication failure is making database resources unavailable.

System programmer response: Determine and correct the cause of the communication failure. When the problem is corrected, automatic resolution of the indoubt thread occurs within a short time. If the failure cannot be corrected for a long time, call the database administrator. The database administrator might want to make a heuristic decision to release the database resources that are held for the indoubt thread.

Scenario: Making a heuristic decision about whether to commit or abort an indoubt thread

An organization might need to make a heuristic decision about whether to commit or abort an indoubt thread.

Symptoms

In this scenario, an indoubt thread at Los Angeles (LA) holds database resources that are needed by other applications. The organization makes a heuristic decision about whether to commit or abort an indoubt thread.

Many symptoms are possible, including:

- Message DSNL405 to indicate a thread in the indoubt state
- A **DISPLAY THREAD** report of active threads showing a larger-than-normal number of threads
- A **DISPLAY THREAD** report of indoubt threads continuing to show the same thread
- A **DISPLAY DATABASE LOCKS** report that shows a large number of threads that are waiting for the locks that are held by the indoubt thread
- Some threads terminating due to timeout
- IMS and CICS transactions not completing

Environment

The following figure illustrates the environment for this scenario.

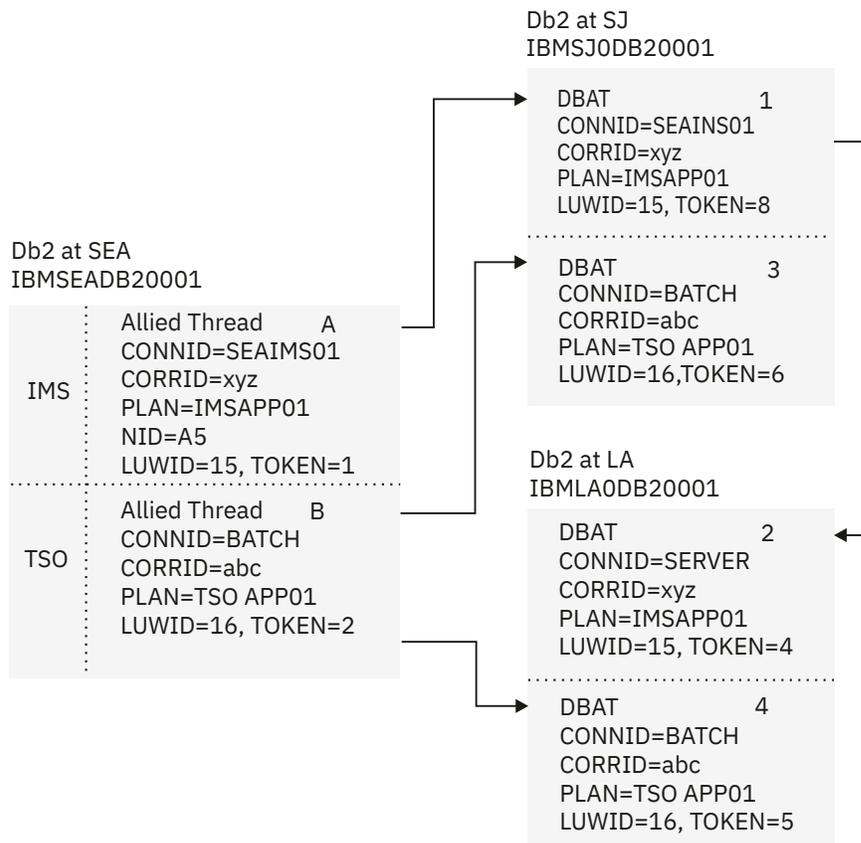


Figure 16. Resolution of indoubt threads

Resolving the problem

Database administrator response: Determine whether to commit or abort the indoubt thread. First, determine the name of the commit coordinator for the indoubt thread. This name matches the location name of the Db2 subsystem at SEA, and it is included in the Db2 indoubt thread **DISPLAY THREAD** report at LA. Then, have an authorized person at SEA perform one of the following actions:

- If the coordinator Db2 subsystem is active, or if it can be started, request a **DISPLAY THREAD** report for indoubt threads, specifying the LUWID of the thread. (Remember that the token that is used at LA is different than the token that is used at SEA). If no report entry exists for the LUWID, the proper action is to abort. If a report entry for the LUWID exists, it shows the proper action to take.
- If the coordinator Db2 subsystem is not active and cannot be started, and if statistics class 4 was active when Db2 was active, search the SEA SMF data for an IFCID 209 event entry that contains the indoubt LUWID. This entry indicates whether the commit decision was commit or abort.
- If statistics class 4 is not available, run the DSN1LOGP utility, and request a summary report. The volume of log data that is to be searched can be restricted if you can determine the approximate SEA log RBA value that was in effect at the time of the communication failure. A DSN1LOGP entry in the summary report for the indoubt LUWID indicates whether the decision was commit or abort.

After determining the correct action to take, issue the **RECOVER INDOUBT** command at the LA Db2 subsystem, specifying the LUWID and the correct action.

System action:

Issuing the **RECOVER INDOUBT** command at LA results in committing or aborting the indoubt thread. Locks are released. The thread does not disappear from the indoubt thread display until resolution with SEA is completed. The **RECOVER INDOUBT** report shows that the thread is either committed or aborted by heuristic decision. An IFCID 203 trace record is written, recording the heuristic action.

Scenario: Recovering from an IMS outage that results in an IMS cold start

An IMS outage can result in an IMS cold start. An organization that experiences this situation can recover.

Symptoms

When IMS is cold started and later reconnects with the SEA Db2 subsystem, IMS is not able to resolve the indoubt thread with Db2. Message DSNM004I is displayed at the IMS master terminal.

Environment

The following figure illustrates the environment for this scenario.

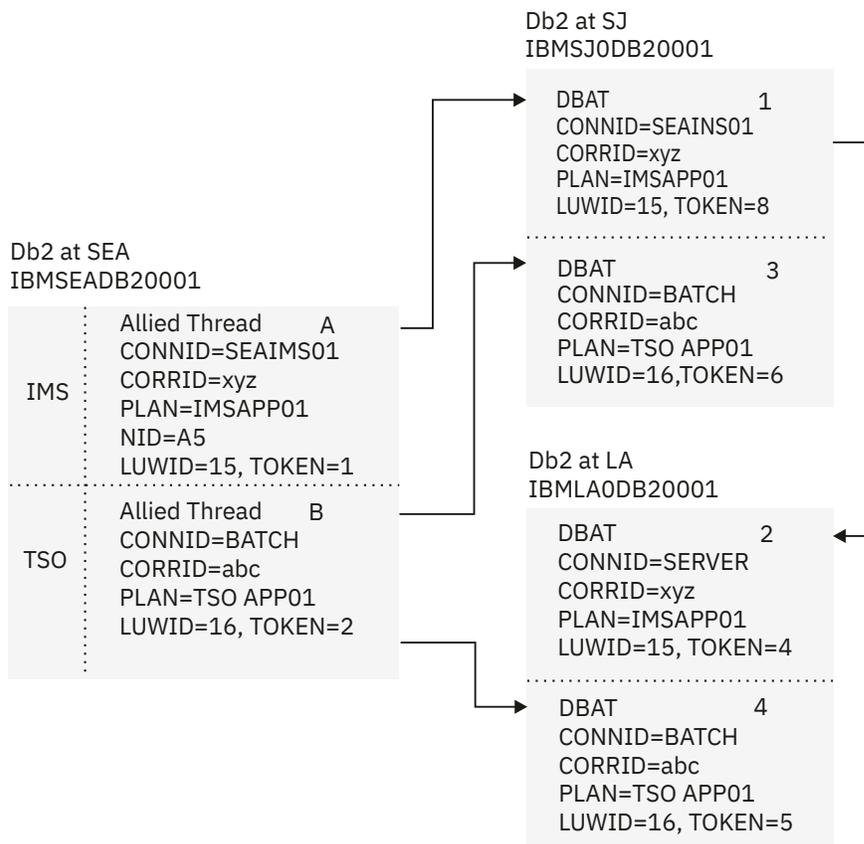


Figure 17. Resolution of indoubt threads

The abnormal termination of IMS has left one allied thread A at the SEA Db2 subsystem indoubt. This is the thread whose LUWID=15. Because the SEA Db2 subsystem still has effective communication with the Db2 subsystem at SJ, the LUWID=15 DBAT 1 at this subsystem is waiting for the SEA Db2 to communicate the final decision and is not aware that IMS has failed. Also, the LUWID=15 DBAT at LA 2, which is connected to SJ, is also waiting for SJ to communicate the final decision. This cannot be done until SEA communicates the decision to SJ.

- The connection remains active.
- IMS applications can still access Db2 databases.
- Some Db2 resources remain locked out.

If the indoubt thread is not resolved, the IMS message queues can start to back up. If the IMS queues fill to capacity, IMS terminates. Therefore, users must be aware of this potential difficulty and must monitor IMS until the indoubt units of work are fully resolved.

Resolving the problem

System programmer response: Issue the **RECOVER INDOUBT** command to resolve the indoubt thread at the SEA Db2 subsystem. This completes the two-phase commit process with the Db2 subsystems at SJ and LA, and the unit of work either commits or aborts.

1. Force the IMS log closed by using the **/DBR FEOV** command, and then archive the IMS log. Use the command **DFSERA10** to print the records from the previous IMS log tape for the last transaction that was processed in each dependent region. Record the PSB and the commit status from the X'37' log that contains the recovery ID.
2. Run the DL/I batch job to back out each PSB that is involved and that has not reached a commit point. The process might take some time because transactions are still being processed. The process might also lock up a number of records, which could affect the rest of the processing and the rest of the message queues.
3. Enter the Db2 command **DISPLAY THREAD** (*imsid*) TYPE (INDOUBT).
4. Compare the NIDs (IMSID + OASN in hexadecimal) that is displayed in the **DISPLAY THREAD** messages with the OASNs (4 bytes decimal) as shown in the **DFSERA10** output. Decide whether to commit or roll back.
5. Use **DFSERA10** to print the X'5501FE' records from the current IMS log tape. Every unit of recovery that undergoes indoubt resolution processing is recorded; each record with an 'IDBT' code is still indoubt. Note the correlation ID and the recovery ID, for use during the next step.
6. Enter the following Db2 command, choosing to commit or roll back, and specify the correlation ID:

```
-RECOVER INDOUBT (imsid) ACTION(COMMIT|ABORT) NID (nid)
```

If the command is rejected because network IDs are associated, use the same command again, substituting the recovery ID for the network ID.

Related concepts

[Duplicate IMS correlation IDs \(Db2 Administration Guide\)](#)

Scenario: Recovering from a Db2 outage at a requester that results in a Db2 cold start

When an outage at a Db2 requester results in a cold start, the organization that has this situation can recover.

Symptoms

The Db2 subsystem at SEA is started with a conditional restart record in the BSDS to indicate a cold start:

- When the IMS subsystem reconnects, it attempts to resolve the indoubt thread that is identified in IMS as NID=A5. IMS has a resource recovery element (RRE) for this thread. The SEA Db2 subsystem informs IMS that it has no knowledge of this thread. IMS does not delete the RRE, and the RRE can be displayed by using the IMS **DISPLAY OASN** command. The SEA Db2 subsystem also:
 - Generates message DSN3005 for each IMS RRE for which Db2 has no knowledge
 - Generates an IFCID 234 trace event
- When the Db2 subsystems at SJ and LA reconnect with SEA, each detects that the SEA Db2 subsystem has cold started. Both the SJ Db2 and the LA Db2 subsystem:
 - Display message DSNL411
 - Generate alert A001
 - Generate an IFCID 204 trace event
- A **DISPLAY THREAD** report of indoubt threads at both the SJ and LA Db2 subsystems shows the indoubt threads and indicates that the coordinator has cold started.

Causes

An abnormal termination of the SEA Db2 subsystem caused the outage.

Environment

The following figure illustrates the environment for this scenario.

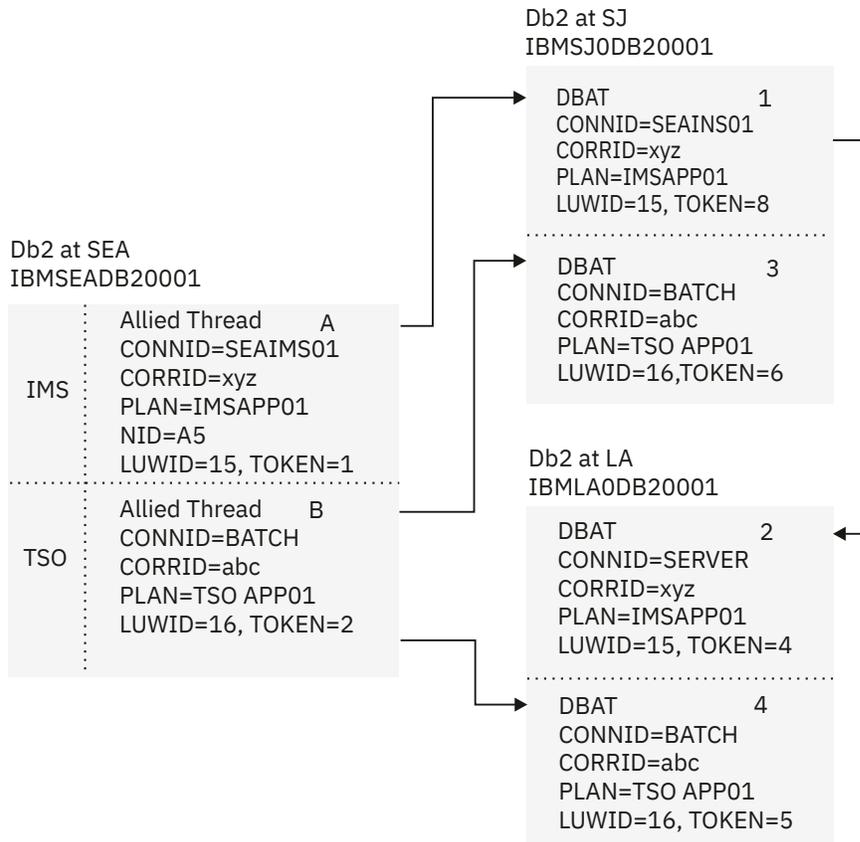


Figure 18. Resolution of indoubt threads

The abnormal termination of the SEA Db2 subsystem has left the two DBATs at SJ 1, 3, and the LUWID=16 DBAT at LA 4 indoubt. The LUWID=15 DBAT at LA 2, connected to SJ, is waiting for the SJ Db2 subsystem to communicate the final decision.

The IMS subsystem at SEA is operational and has the responsibility of resolving indoubt units with the SEA Db2 subsystem.

The Db2 subsystems at both SJ and LA accept the cold start connection from SEA. Processing continues, waiting for the heuristic decision to resolve the indoubt threads.

Resolving the problem

Database administrator response:

At this point:

- Neither the SJ nor the LA administrator know if the SEA coordinator was a participant of another coordinator. In this scenario, the SEA Db2 subsystem originated LUWID=16. However, the SEA Db2 subsystem was a participant for LUWID=15, which was being coordinated by IMS.
- The administrator at LA also does not know is the fact that SEA distributed the LUWID=16 thread to SJ, where it is also indoubt. Likewise, the administrator at SJ does not know that LA has an indoubt thread for the LUWID=16 thread. Both SJ and LA need to make the same heuristic decision. The administrators at SJ and LA also need to determine the originator of the two-phase commit.

- The recovery log of the originator indicates whether the decision was commit or abort. The originator might have more accessible functions to determine the decision. Even though the SEA Db2 subsystem cold started, you might be able to determine the decision from its recovery log. Alternatively, if the failure occurred before the decision was recorded, you might be able to determine the name of the coordinator, if the SEA Db2 subsystem was a participant. You can obtain a summary report of the SEA Db2 recovery log by running the DSN1LOGP utility.
- The LUWID contains the name of the logical unit (LU) where the distributed logical unit of work originated. This logical unit is most likely in the system that originated the two-phase commit.
- If an application is distributed, any distributed piece of the application can initiate the two-phase commit. In this type of application, the originator of two-phase commit can be at a different system than the site that is identified by the LUWID.
- The administrator must determine if the LU name that is contained in the LUWID is the same as the LU name of the SEA Db2 subsystem. If this is not the case (it is the case in this example), the SEA Db2 subsystem is a participant in the logical unit of work, and it is being coordinated by a remote system. The DBA must communicate with that system and request that facilities of that system be used to determine if the logical unit of work is to be committed or aborted.
- If the LUWID contains the LU name of the SEA Db2 subsystem, the logical unit of work originated at SEA and can be an IMS, CICS, TSO, or batch allied thread of the SEA Db2 subsystem. The **DISPLAY THREAD** report for indoubt threads at a Db2 participant includes message DSNV458 if the coordinator is remote. This line provides external information that is provided by the coordinator to assist in identifying the thread. A Db2 coordinator provides the following identifier:

```
connection-name.correlation-id
```

where *connection-name* is:

- SERVER: the thread represents a remote application to the Db2 coordinator and uses DRDA access.
- BATCH: the thread represents a local batch application to the Db2 coordinator.
- Anything else represents an IMS or CICS connection name. The thread represents a local application, and the commit coordinator is the IMS or CICS system by using this connection name.
- In this example, the administrator at SJ sees that both indoubt threads have an LUWID with the LU name that match the SEA Db2 subsystem LU name, and furthermore, that one thread (LUWID=15) is an IMS thread and the other thread (LUWID=16) is a batch thread. The LA administrator sees that the LA indoubt thread (LUWID=16) originates at the SEA Db2 subsystem and is a batch thread.
- The originator of a Db2 batch thread is Db2. To determine the commit or abort decision for the LUWID=16 indoubt threads, the SEA Db2 recovery log must be analyzed, if possible. Run the DSN1LOGP utility against the SEA Db2 recovery log, and look for the LUWID=16 entry. Three possibilities exist:
 1. No entry is found. That portion of the Db2 recovery log is not available.
 2. An entry is found but incomplete.
 3. An entry is found, and the status is committed or aborted.
- In the third case, the heuristic decision at SJ and LA for indoubt thread LUWID=16 is indicated by the status that is indicated in the SEA Db2 recovery log. In the other two cases, the recovery procedure that is used when cold starting Db2 is important. If recovery was to a previous point in time, the correct action is to abort. If recovery included repairing the SEA Db2 database, the SEA administrator might know what decision to make.
- The recovery logs at SJ and LA can help determine what activity took place. If the administrator determines that updates were performed at SJ, LA, or both (but not SEA), and if both SJ and LA make the same heuristic action, data inconsistency probably exists. If updates were also performed at SEA, the administrator can look at the SEA data to determine what action to take. In any case, both SJ and LA should make the same decision.
- For the indoubt thread with LUWID=15 (the IMS coordinator), several alternative paths to recovery are available. The SEA Db2 subsystem has been restarted. When it reconnects with IMS, message **DSN3005** is issued for each thread that IMS is trying to resolve with Db2. The message indicates that Db2 has no knowledge of the thread that is identified by the IMS-assigned NID. The outcome for the thread, either

commit or abort, is included in the message. Trace event IFCID=234 is also written to statistics class 4, which contains the same information.

- If only one such message exists, or if one such entry is in statistics class 4, the decision for indoubt thread LUWID=15 is known and can be communicated to the administrator at SJ. If multiple such messages exist, or if multiple such trace events exist, the administrator must match the IMS NID with the network LUWID. Again, the administrator should use **DSN1LOGP** to analyze the SEA Db2 recovery log if possible. Now four possibilities exist:
 1. No entry is found. That portion of the Db2 recovery log was not available.
 2. An entry is found but is incomplete because of lost recovery log data.
 3. An entry is found, and the status is indoubt.
 4. An entry is found, and the status is committed or aborted.
- In the fourth case, the heuristic decision at SJ for the indoubt thread LUWID=15 is determined by the status that is indicated in the SEA Db2 recovery log. If an entry is found and its status is indoubt, DSN1LOGP also reports the IMS NID value. The NID is the unique identifier for the logical unit of work in IMS and CICS. Knowing the NID enables correlation to the DSN3005 message, or to the 234 trace event, either of which provides the correct decision.
- If an incomplete entry is found, the NID might have been reported by DSN1LOGP. If it was reported, use it as previously discussed.
- Determine if any of the following conditions exists:
 - No NID is found.
 - The SEA Db2 subsystem has not been started.
 - Reconnecting to IMS has not occurred.

If any of these conditions exists, the administrator must use the *correlation-id* that is used by IMS to correlate the IMS logical unit of work to the Db2 thread in a search of the IMS recovery log. The SEA Db2 site provided this value to the SJ Db2 subsystem when distributing the thread to SJ. The SJ Db2 site displays this value in the report that is generated by the **DISPLAY THREAD TYPE(INDOUBT)** command.

- For IMS, the *correlation-id* is:

```
pst#.psbname
```

- In CICS, the *correlation-id* consists of four parts:

```
Byte 1 - Connection type - G=Group, P=Pool  
Byte 2 - Thread type - T=transaction, G=Group, C=Command  
Bytes 3-4 - Thread number  
Bytes 5-8 - Transaction-id
```

Related concepts

[Scenario: What happens when the wrong Db2 subsystem is cold started \(Db2 Administration Guide\)](#)

Scenario: What happens when the wrong Db2 subsystem is cold started

When one Db2 subsystem, instead of another Db2 subsystem, is cold started, threads are left indoubt. An organization that faces this situation can recover.

The following figure illustrates the environment for this scenario.

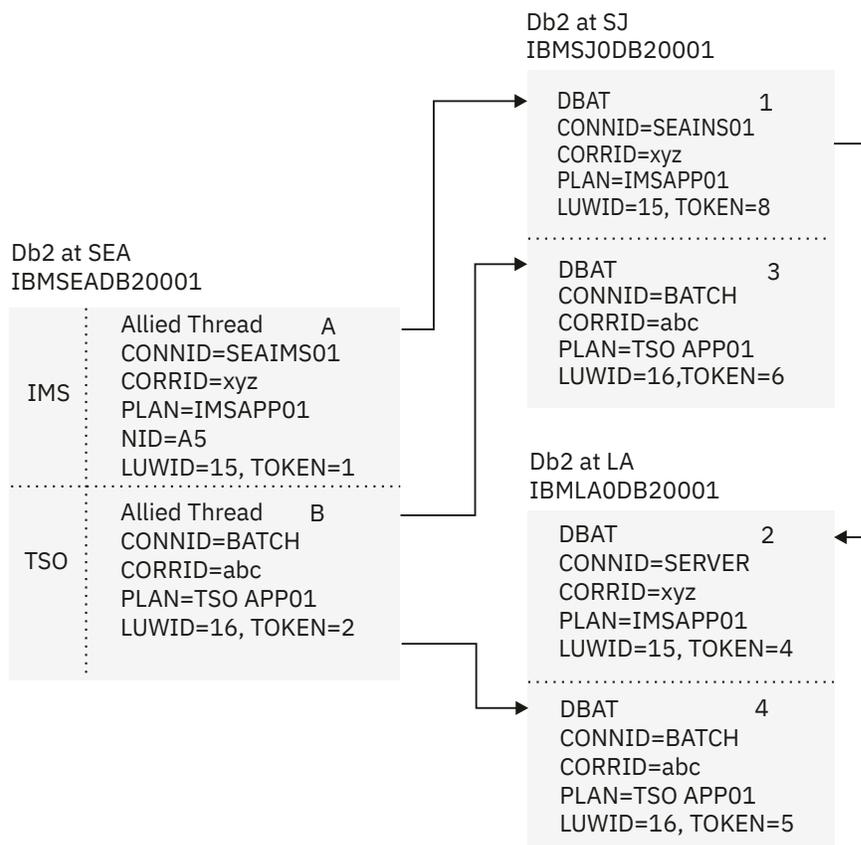


Figure 19. Resolution of indoubt threads

If the Db2 subsystem at SJ is cold started instead of the Db2 at SEA, the LA Db2 subsystem has the LUWID=15 2 thread indoubt. The administrator can see that this thread did not originate at SJ, but that it did originate at SEA. To determine the commit or abort action, the LA administrator requests that **DISPLAY THREAD TYPE (INDOUBT)** be issued at the SEA Db2 subsystem, specifying LUWID=15. IMS does not have any indoubt status for this thread because it completes the two-phase commit process with the SEA Db2 subsystem.

The Db2 subsystem at SEA tells the application that the commit succeeded.

When a participant cold starts, a Db2 coordinator continues to include in the display of information about indoubt threads all committed threads where the cold starting participant was believed to be indoubt. These entries must be explicitly purged by issuing the **RESET INDOUBT** command. If a participant has an indoubt thread that cannot be resolved because of coordinator cold start, the administrator can request a display of indoubt threads at the Db2 coordinator to determine the correct action.

Related information

Scenario: Recovering from communication failure (Db2 Administration Guide)

Scenario: Recovering from a Db2 outage at a requester that results in a Db2 cold start (Db2 Administration Guide)

Scenario: Correcting damage from an incorrect heuristic decision about an indoubt thread

If an incorrect heuristic decision is made regarding an indoubt thread, an organization can recover from this incorrect decision.

Symptoms

When the Db2 subsystem at SEA reconnects with the Db2 at LA, indoubt resolution occurs for LUWID=16. Both systems detect heuristic damage, and both generate alert A004; each writes an IFCID 207 trace record. Message DSNL400 is displayed at LA, and message DSNL403 is displayed at SEA.

Causes

This scenario is based on the conditions described in “Scenario: Recovering from communication failure ” on page 163.

The LA administrator is called to make a heuristic decision and decides to abort the indoubt thread with LUWID=16. The decision is made without communicating with SEA to determine the proper action. The thread at LA is aborted, whereas the threads at SEA and SJ are committed. Processing continues at all systems. The Db2 subsystem at SEA has indoubt resolution responsibility with LA for LUWID=16.

Environment

The following figure illustrates the environment for this scenario.

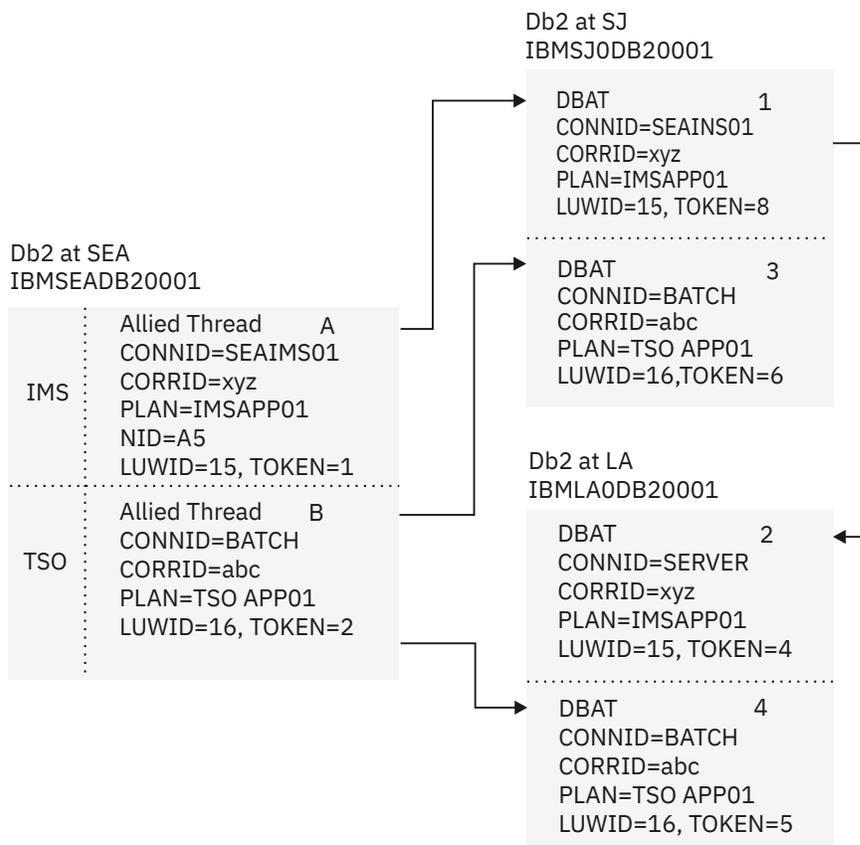


Figure 20. Resolution of indoubt threads

In this scenario, processing continues. Indoubt thread resolution responsibilities have been fulfilled, and the thread completes at both SJ and LA.

Resolving the problem

Database administrator response: Correct the damage. This is not an easy task. Since the time of the heuristic action, the data at LA might have been read or written by many applications. Correcting the damage can involve reversing the effects of these applications, also. The available tools are:

- DSN1LOGP utility, which generates a summary report that identifies the table spaces that were modified by the LUWID=16 thread.
- The statistics trace class 4, which contains an IFCID 207 entry. This entry identifies the recovery log RBA for the LUWID=16 thread.

Notify IBM Support about the problem.

Troubleshooting Db2 stored procedure problems

If you encounter problems setting up, calling, or running stored procedures, several troubleshooting techniques and tools are available in Db2 and z/OS.

Procedure

To troubleshoot Db2 stored procedures, perform one or more of the following actions:

- For general information about the available debugging tools and techniques, see [Debugging stored procedures \(Db2 Application programming and SQL\)](#).
- See [Db2 for z/OS Stored Procedures: Through the CALL and Beyond \(IBM Redbooks\)](#) if you are troubleshooting one of the following problems:
 - For problems with implementing RRS, see "RRS error samples."
 - For problems with calling a particular stored procedure, you might not have the required authorizations. See "Privileges to execute a stored procedure called statically."
 - For troubleshootingJava™ stored procedures, see "Common problems."
 - For invoking programs that receive SQLCODE -430, see "Classical debugging of stored procedures."

Identifying Db2 data inconsistency problems

Data inconsistency problems occur for various reasons, including internal Db2 problems, I/O errors, or system problems.

About this task

Data inconsistency problems can occur as a result of referential integrity constraint violations. Referential integrity describes the condition where all references to data in pages are valid. Db2 provides referential integrity by enforcing referential constraints on the data. Referential integrity constraint violations do not cause abends because Db2 does not depend on referential constraints to operate. Referential constraint violations do, however, cause incorrect results in processing.

Data inconsistency problems fall into one or more of the following categories:

- Unavailable data
- Internal inconsistency
- Inconsistent page
- Inconsistent page set.

These conditions can exist singly or in combination. They are evidenced by various symptoms, and can be identified by a number of messages and codes. Discovering an inconsistency in user or Db2 system data can be difficult, depending upon the complexity of the problem.

Procedure

To identify Db2 data inconsistency problems:

1. Identify the database *dbname* and table space *tsname* that contain the inconsistent data.
2. Enter **-START DATABASE(*dbname*) ACCESS(RO)** to limit access to the data before an attempt is made to resolve the inconsistency.

This action prevents subsequent updates to the inconsistent data from compounding the problem.

Inhibiting updates to the inconsistent data is particularly important if the Db2 catalog or directory is involved in the problem. If catalog or directory problems are suspected, issue **-START DATABASE(DSNDB06) ACCESS(RO)** to inhibit DDL, GRANT, REVOKE, and BIND activities.

3. Determine how to resolve the problem by following the guidelines in “Data inconsistency symptoms and actions” on page 175 or the detailed procedures in “Inconsistency resolution with RECOVER TABLESPACE and RECOVER INDEX” on page 343 or “Resolving inconsistencies manually” on page 335.
4. Enter **-START DATABASE(*dbname*) SPACENAM(*tsname*) ACCESS(UT)** to allow utility processing.
5. Invoke either RECOVER or REPAIR (depending on what the information in Table 9 on page 175 suggests).
6. Determine whether the damage is corrected.
7. After the data is made consistent, issue **-START DATABASE(*db*) SPACENAM(*ts*) ACCESS(RW)** to restore full access to the data.

Related concepts

[Data sharing problem diagnosis](#)

Problem diagnosis in the data sharing environment is similar to diagnosing problems that occur in a single Db2 subsystem. The key difference is that the actions required to identify and resolve problems must be applied to more than one Db2 subsystem.

[Referential constraints \(Introduction to Db2 for z/OS\)](#)

Data inconsistency symptoms and actions

There are common symptoms of each type of data inconsistency problem, and general recommendations to resolve these conditions.

Table 9. Types of data inconsistency

| Problem | Symptoms | Indicated By | Possible Causes | Suggested Actions |
|------------------|--|---|--|---|
| Unavailable data | Table or index space is marked STOP or LPL RESOURCE UNAVAILABLE condition indicated | Messages DSNT500I or DSNT501I | DM is unable to open a required VSAM data set during restart processing. | Start database that contains the table or index space to apply all log records that were deferred at Db2 restart. |
| | | Type codes 300 (page), 301 (index subpage), 302 (table space page), or 303 (index page) Abend reason code 00C9009F SQL return code -904 | Attempt to insert or update record with foreign key when index that enforces the primary key of the parent table is dropped. | Run RECOVER TABLESPACE on the table space, or RECOVER INDEX or REBUILD INDEX on the index. Resolve problem manually if it is a data inconsistency problem. Create an index that enforces the primary key, or drop the relationship. |

Table 9. Types of data inconsistency (continued)

| Problem | Symptoms | Indicated By | Possible Causes | Suggested Actions |
|------------------------|---|---|--|---|
| Internal inconsistency | Table or index space is marked STOP Many error messages or abends identifying the same set of pages | Messages DSNI013I or DSNI014I Abend reason codes 00C90101, 00C90107, 00C90108, or 00C90109 | Damaged recovery log is used during Db2 restart processing (00C90101 issued). Unavailable data in pages (usually not an inconsistency problem). | Refer to “Type-of-failure keywords” on page 12. Start the database that contains the table or index space to apply all log records that were deferred at Db2 restart. Run RECOVER TABLESPACE on the table space, or RECOVER INDEX or REBUILD INDEX on the index. Resolve problem manually if it is a data inconsistency problem. |
| Inconsistent page | Table or index space is marked STOPPED INCONSISTENT DATA or INDEX condition indicated | Messages DSNI011I or DSNI012I Abend reason codes 00C90105 or 00C90102 | DM tries to recover a page that contains inconsistent data by using the recovery log, but is unsuccessful. | Run DSN1COPY with CHECK option to obtain more information. Run RECOVER TABLESPACE on the table space, or RECOVER INDEX or REBUILD INDEX on the index. Resolve manually. Activate global trace to obtain more information. (See “Special considerations for suspected page inconsistencies” on page 178.) |
| Inconsistent page set | Table or index space is marked STOPPED. SQL statement is processing more rows or fewer rows than specified in the statement. | Messages DSNT500I or DSNT501I. Type codes 200 (table space) or 201 (index space). Abend reason codes 00C902xx or 00C9011x (DBDs). | Link or hash chain is damaged. Extra or missing index entry. Inconsistencies between two or more items of data. Inconsistencies in DBD. | Issue -DISPLAY DATABASE(*) RESTRICT to display STOPPED table or index spaces. Run CHECK utility on index space to verify validity of indexes. Run RECOVER TABLESPACE on the table space, or RECOVER INDEX or REBUILD INDEX on the index. |

Table 9. Types of data inconsistency (continued)

| Problem | Symptoms | Indicated By | Possible Causes | Suggested Actions |
|-----------------------|---|---|--|--|
| Inconsistent page set | In a segmented table space, results differ between index scan and table scan or total number of rows is less than expected. | For segmented table space inconsistencies, abend reason codes 00C90218 or 00C90219. | A non-empty segment in a segmented table space is not on any segment chain; a deallocated segment is on a segment chain; an allocated segment is on the wrong segment chain. | For segmented table space errors, analyze SDUMP and run DSN1PRNT or REPAIR with DUMP option to print pages that are covered by inconsistent segment; run REPAIR or reorganize table space by way of REORG. |
| Inconsistent page set | CHECK PENDING state indicated. | CHECK PENDING indicated by DISPLAY DATABASE command with table space or RESTRICT specified. | <p>CHECK PENDING set by CHECK, LOAD or RECOVER utilities, or ALTER statement. Indicates that associated page set can contain records that violate referential or check constraints.</p> <p>For indexes, indicates one of the following cases:</p> <ul style="list-style-type: none"> • The index and its table space were recovered to a previous point in time that is not a quiesce point. • The index and its table space were recovered with the TOCOPY option, but not in the same RECOVER statement. • The index and its table space were recovered with the TOCOPY option and in the same RECOVER statement, but the image copies were not taken with the same COPY statement. <p>Does not apply to databases.</p> | To reset CHECK PENDING on a table, rectify the violations and place deleted records into “exception” tables, run CHECK DATA utility with the DELETE YES option, DROP TABLE to alleviate the problem and LOAD REPLACE to replace the data that adheres to referential or check constraints or recover all members of table space that is set to 'quiesce' point. To reset CHECK PENDING on an index, run CHECK INDEX. |

Table 9. Types of data inconsistency (continued)

| Problem | Symptoms | Indicated By | Possible Causes | Suggested Actions |
|-----------------------|---|---|--|--|
| Inconsistent page set | RECOVER PENDING state indicated. LOAD or REORG does not complete normally. | RECOVER PENDING indicated by DISPLAY DATABASE command with table space, index space, or RESTRICT specified. | RECOVER PENDING set by utilities when data found to be inconsistent. Abnormal utility termination leaves the table space in RECOVER PENDING. Indicates that associated page set can contain inconsistent data, apart from referential constraint violations. Use REPAIR utility to manually turn off CHECK PENDING, and -START ACCESS (FORCE) to restart table space. This approach is not recommended. | To reset RECOVER PENDING, run RECOVER, the LOAD REPLACE, or REPAIR utilities. |
| Inconsistent page set | INCONSISTENT DATA or INDEX condition that is indicated by messages. | SQL return code -904. | Internal Db2 problems, I/O errors, system problems. | Resolve manually. |
| Inconsistent page set | Table space or index space is not at the latest level. | SQL return code -904. Reason code 00C2010D or 00C20110. Message DSNB232I. | Data set is restored to a level that is inconsistent with other data sets or with the Db2 log. VSAM high-used RBA value is corrupted. | Restore pageset or partition to correct level; or restore down-level pageset or partition to currency, run RECOVER, or RECOVER LOGONLY; or accept down-level, run REPAIR LEVELID; or replace contents, run LOAD REPLACE; or restore to prior point in time, run RECOVER. |

Special considerations for suspected page inconsistencies

During SQL processing of, for example, an INSERT, UPDATE, or DELETE statement, Db2 makes data consistency checks to verify that the processing did not produce any inconsistent data.

The number of checks that Db2 makes depends on whether the global trace is active. If the global trace is not active, only a subset of the data inconsistency checks is made. If the global trace is active, all the checks are made.

During normal operation, the subset of checks is sufficient. If you suspect data inconsistency problems are the result of an inconsistent page, activating the global trace is yet another method that can be used to help verify whether there is inconsistent data.

During utility operations, and operations, such as RESTART, that apply log records to a page, Db2 performs all the data inconsistency checks, whether the global trace is activated.

Related reference

Printing and analyzing global traces

A trace must be active on the Db2 subsystem that you want to trace. For data sharing, the trace commands have member or group scope.

Chapter 4. Diagnostic aids for single systems and data sharing

Use dumps, traces, error messages, SYS1 service aids, and -DISPLAY command output to diagnose problems that involve a single Db2 subsystem or a data sharing group.

One of the most useful diagnostic aids that Db2 provides is the SVC memory dump title. For most problems, this title provides enough information to build a keyword string.

For some problems, one or more memory dumps, SYS1.LOGREC entries, and trace tables might need to be reviewed.

For some failures, log records might be needed from before and at the time of failure. Always keep the Db2 archive log data sets until it is certain that no problems occurred for which they might be needed.

Printing and analyzing dumps

You might need to print and analyze a memory dump to help to diagnose a problem.

Types of Db2 for z/OS dumps

The following table gives the data set, output type, printing information, and cause of each Db2 dump type.

Table 10. Types of Db2 dumps

| Dump type | Data set | Output type | Printed by | Caused by |
|------------------------|--|------------------|--|--|
| Dynamic (z/OS console) | SYS1.DUMPxx | Machine readable | IPCS in conjunction with a Db2 verb exit | Operator entering z/OS 'DUMP' command |
| SNAP | Dynamic allocation of SNAP data set for SYSOUT | Formatted | Normally SYSOUT=A | An abend of a Db2 application |
| Stand-alone | Defined by installation (tape or disk) | Machine readable | IPCS in conjunction with a Db2 verb exit | Operator IPL of the stand-alone dump program |
| SVC | SYS1.DUMPxx | Machine readable | IPCS in conjunction with a Db2 verb exit | z/OS or Db2 functional recovery routine detecting error |
| SYSABEND | Defined by JCL (SYSOUT=A) | Formatted | Normally SYSOUT=A | An abend condition (and only taken if there is a SYSABEND DD statement for the step) |
| SYSUDUMP | Defined by JCL (SYSOUT=A) | Formatted | Normally SYSOUT=A | An abend condition (and only taken if there is a SYSUDUMP DD statement for the step) |

Related concepts

[Printing dumps](#)

Db2 supports IPCS under z/OS. It is recommended to use IPCS for versatility to print dumps.

Printing dumps

Db2 supports IPCS under z/OS. It is recommended to use IPCS for versatility to print dumps.

The Db2 dump formatter routine, DSNWDPRD, is invoked by IPCS with:

```
DB2 VERBEXIT DSNWDMP
```

The following sample contains JCL for printing and formatting a dump through IPCS. These JCL samples illustrate the appropriate parameters for formatting SVC, stand-alone, and dynamic dumps. The DSNWDMP control statement that is shown in the examples causes the Db2 dump formatter (DSNWDPRD) to be invoked and defines the Db2 control block structures to be formatted.

The JCL included here contains only the DD statements that are recommended for Db2. Your site might choose to include other DD statements as well.

Figure 21. Sample JCL for printing dumps through IPCS in the z/OS environment

```
//PRDMP00 JOB 1, 'PRINT SYS1.DUMP00'
//*****
//*   IPCS  INVOCATION AS 'VERB EXIT'   *
//*****
//DB2DMP EXEC PGM=IKJEFT01,REGION=5120K
//SYSTSPRT DD SYSOUT=*
//SYSABEND DD SYSOUT=*
//IPCSPRNT DD SYSOUT=*
//IPCSDDIR DD DSN=dump.directry-name,DISP=OLD
//DUMP00 DD DSN=SYS1.DUMP00,DISP=SHR
//SYSTSIN DD *
IPCS NOPARM TASKLIB('DB2 Load Library')
SETDEF PRINT TERMINAL DDNAME(DUMP00) NOCONFIRM
VERBEXIT DAEDATA
STATUS SYSTEM CPU REGS
VERBEXIT CPUDATA
VERBEXIT CVTMAP VERBEXIT LOGDATA
VERBEXIT TRACE
VERBEXIT GRSTRACE
SUMMARY JOBSUMMARY FORMAT ALL
VERBEXIT SUMDUMP
VERBEXIT DSNWDMP 'TT,ALL'
//*
//* For printing stand-alone and dynamic dumps,
//* the DSNWDMP statement should also include
//* SUBSYS=DB2_subsystem_name, for example:
//* VERBEXIT DSNWDMP 'TT,ALL,SUMDUMP=NO,
//* SUBSYS=DB2_subsystem_name'
//*
CLOSE ALL
END
The DSNWDMP statement can be changed to print SVC, stand-alone, or
dynamic dumps.
```

Format dumps by using IPCS options

There are a number of ways to format sections of a dump. One option is to use IPCS options.

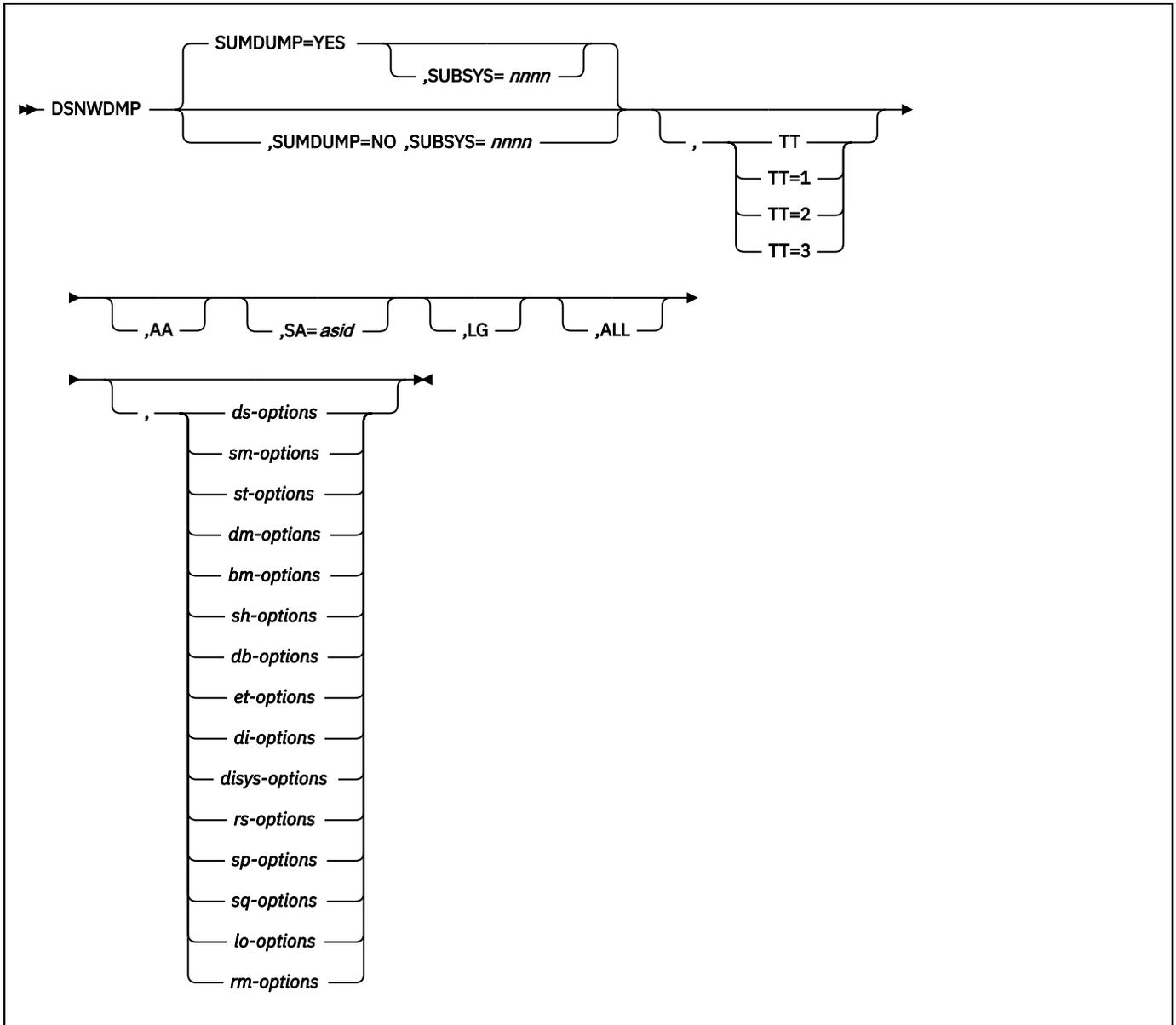
You can specify IPCS options to select portions of the dump to print:

- To request the TCB summary section, use the FORMAT statement.
- To indicate the type of data to be formatted from the dump, use the IPCS VERB keywords.
- To ensure that the dump contains formatted SYS1.LOGREC records, use the LOGDATA option.

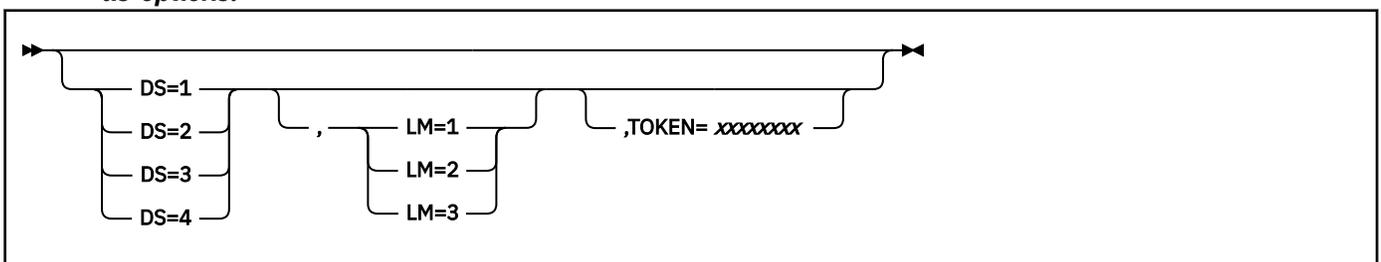
Format dumps by using the DSNWDMP statement

You can use the DSNWDMP statement to specify the dump records to be used as input, and causes the Db2 dump formatter (DSNWDPRD) to be invoked, which formats the specified Db2 control blocks.

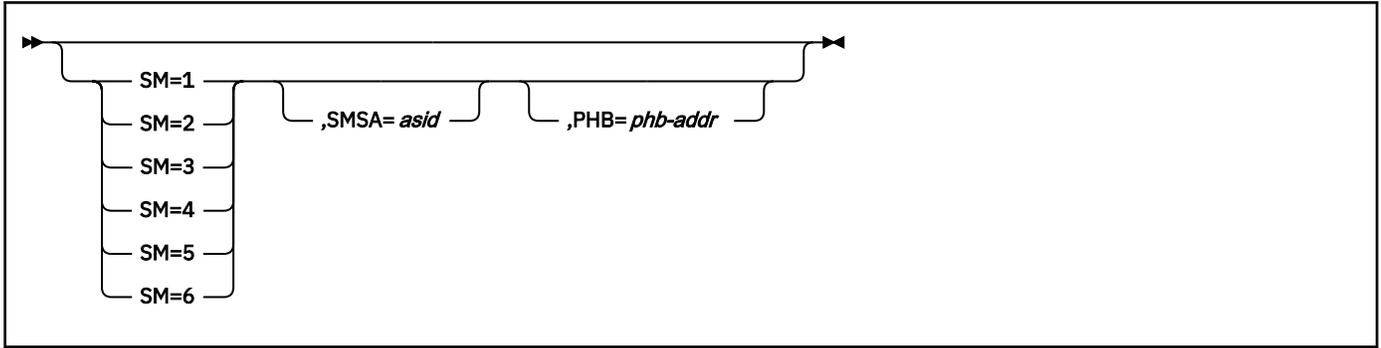
Format the Db2 control blocks in a dump by changing the parameters on the DSNWDMP control statement. The format of this control statement and the options that can be specified are shown in the following diagram.



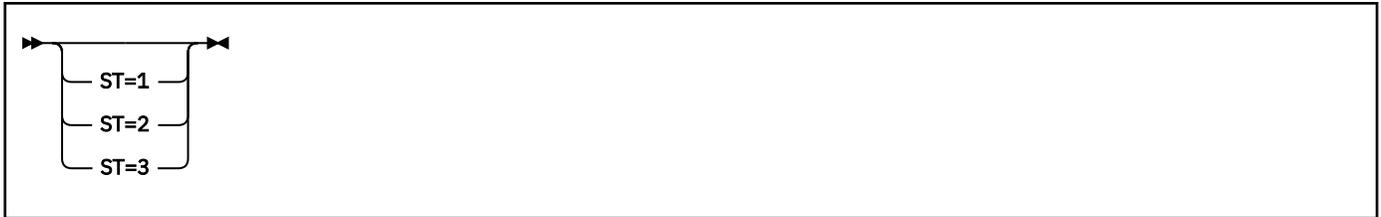
ds-options:



sm-options:



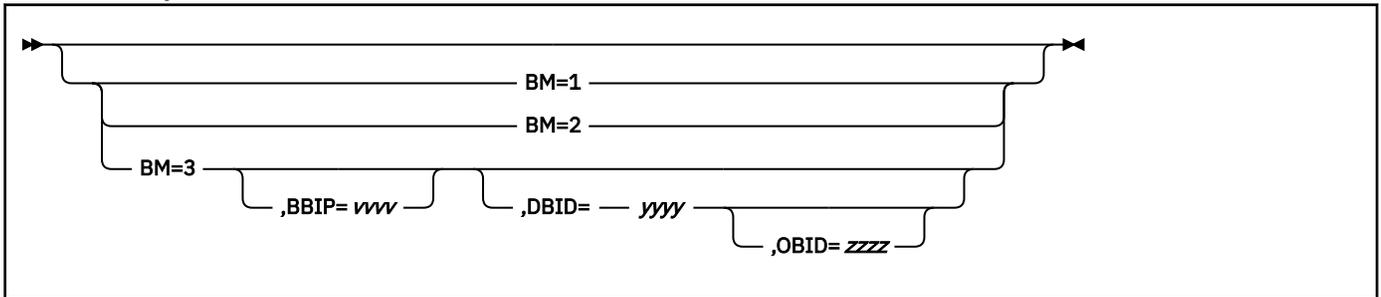
st-options:



dm-options:



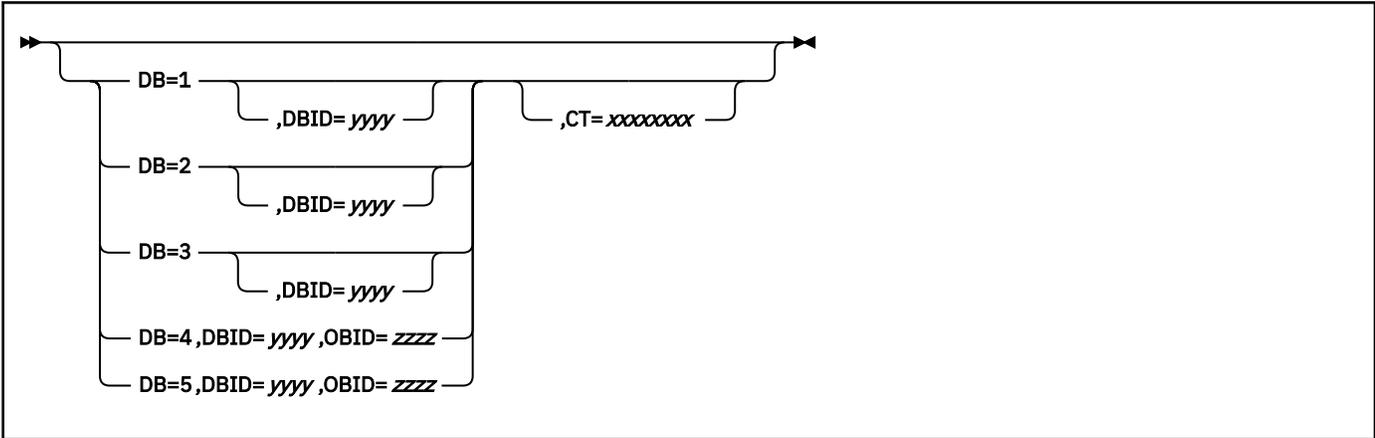
bm-options:



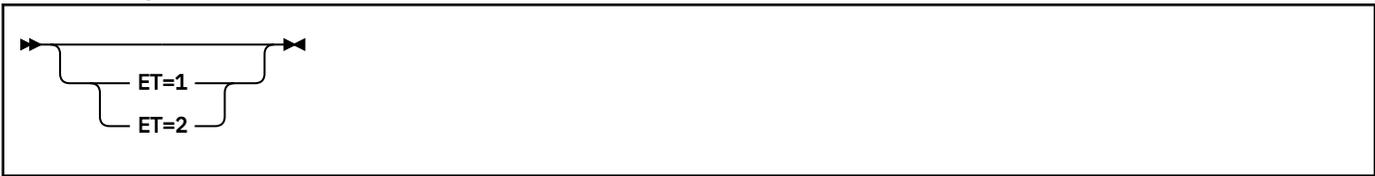
sh-options:



db-options:



et-options:



di-options:



disys-options:



rs-options:



sp-options:



sq-options:



lo-options:



rm-options:



Separate multiple operands by commas, not blanks. A blank that follows any operand in the control statement terminates the operand list, and any subsequent operands are ignored. The following table lists and explains each of the various keywords that can be specified in the Db2 control statement for formatting dumps.

Table 11. Keywords to format SVC dumps that are issued by Db2

| Keyword | Explanation |
|---------------------|---|
| SUBSYS= <i>nnnn</i> | Specifies the site-defined Db2 subsystem name (<i>nnnn</i>). Range is one to four characters. Optional if SUMDUMP=YES; required if SUMDUMP=NO. Refer to SUMDUMP Information for more information. |
| TT | Formats the Db2 global trace table. <ul style="list-style-type: none"> • If TT is specified without a suboption, format the global trace table. • If TT=1, display summary lines. • If TT=2, display the TT=1 information, as well as data items. • If TT=3, and DS is also specified, filter information by EB. |
| AA | Formats significant Db2 control blocks in all active address spaces for the subsystem. |
| SA | Formats significant Db2 control blocks that are associated with a single address space. The control statement must designate the ASID in hexadecimal (up to four digits). Apostrophes and leading zeros can be included or omitted (as in DE, 00DE, or '00DE'). |
| LG | Formats a long form of the significant Db2 global control block structures. Includes all resource manager-specific global information as well as the system-wide global control blocks. |

Table 11. Keywords to format SVC dumps that are issued by Db2 (continued)

| Keyword | Explanation |
|---------|--|
| ALL | <p>Formats all significant Db2 control blocks. Includes the trace table, control blocks in global storage, and control blocks in all active address spaces for the subsystem.</p> <p>Equivalent to specifying TT, AA, and LG.</p> |
| SUMDUMP | <p>Specifies the part of the dump data set to be used as input to the dump formatter.</p> <ul style="list-style-type: none"> • If SUMDUMP=YES, the input dump data is to be obtained from the summary portion of the dump data set. • If SUMDUMP=NO, the input dump data is to be obtained from the non-summary portion of the dump data set. <p>The DSNWDPRD dump formatter always tries to format the same control blocks and data structures, regardless of whether input dump data is obtained from the summary or non-summary portion of the dump data set.</p> |
| DS | <p>Specifies the level of detail about agents to report.</p> <ul style="list-style-type: none"> • If DS=1, display information about only those agents at thread level that are on the allied agent chain are reported. • If DS=2, display information about all agents on the allied agent chain are reported. • If DS=3, display information about the system agent chains, plus all agents on the allied chain are reported. The system agent chains are needed to obtain details of parallel tasks and DDF tasks. • If DS=4, in addition to the information for DS=3, the LUWID (Logical Unit of Work ID), XID (Global Transaction ID), user and accounting information to be displayed for each agent. |
| LM | <p>Specifies information about IRLM to report.</p> <ul style="list-style-type: none"> • If LM=1, report TRWA statistics. • If LM=2, if the agent is waiting for an IRLM request, report the status of the last request. • If LM=3, list the held locks for every agent. |
| SM | <p>Specifies information about Storage Manager to report.</p> <ul style="list-style-type: none"> • If SM=1, display the PHB chains from the GPVT. This information gives an idea of global storage usage. • If SM=2, in addition to the information for SM=1, display the SHB chains from each PHB. • If SM=3, display the information for SM=2 for every address space that is connected to Db2. • If SM=4, in addition to the information for SM=3, display statistics on the amounts of allocated and free storage in the variable pools. • If SM=5, in addition to the information for SM=4, display a sample of allocated elements and attempt to determine the module that GETMAINed the storage. • If SM=6, in addition to the information for SM=5, display free elements. |

Table 11. Keywords to format SVC dumps that are issued by Db2 (continued)

| Keyword | Explanation |
|----------------|---|
| PHB | Specifies a PHB address. If SM is specified, the storage manager information is displayed only for the PHB with this address. |
| ST | <p>Specifies information about stack storage.</p> <ul style="list-style-type: none"> • If ST=1, display the PSW, registers from the SDWA, and save area trace the abending task, except for a console dump. If the DS option is also specified, display the high-level SKB and stack storage usage. • If ST=2, in addition to the information for ST=1, display a formatted save area trace for each stack. The DS option must also be specified. • If ST=3, and the DS option is also specified, in addition to the information for ST=2, display a formatted display of stack storage. |
| SMSA | Specifies storage information for only the requested ASID and for global storage. |
| DM | <p>Specifies the Data Manager information to report:</p> <ul style="list-style-type: none"> • If DM=1, format agent DMTR and LKTR traces. • If DM=2, in addition to the information for DM=1, display parent lock and claim and drain information for agent. |
| BM | <p>Specifies Buffer Manager information to report:</p> <ul style="list-style-type: none"> • If BM=1, format agent BBTR trace. • If BM=2, in addition to the information for BM=1, display agent BBRA information. • If BM=3, format all PBs and PBO. To restrict the amount of information that is reported, you can filter by buffer pool, DBID, or OBID. |
| SH | <p>Specifies data sharing information to report:</p> <ul style="list-style-type: none"> • If SH=1, format agent BBTR trace. • If SH=2, in addition to the information for SH=1, display agent BBRA information. • If SH=3, in addition to the information for SH=2, display agent BBRA information. • If SH=4, in addition to the information for SH=3, display agent BBRA information. • If SH=5, in addition to the information for SH=4, display agent BBRA information. |

Table 11. Keywords to format SVC dumps that are issued by Db2 (continued)

| Keyword | Explanation |
|---------|---|
| DB | <p>Specifies database descriptor information to report:</p> <ul style="list-style-type: none"> • If DB=1, list the DBDs that are in memory. If DBID is also specified, list information only for that DBID. • If DB=2, in addition to the information for DB=1, display DBD section information. • If DB=3, in addition to the information for DB=2, display OBD information. • If DB=4, format information for the specified DBID and OBID. The DBID and OBID parameters are required. • If DB=5, in addition to the information for DB=4, display trigger and referential constraint information. The DBID and OBID parameters are required. |
| CT | <p>Specifies the CT address for a specific agent. Use this parameter with the DM or DB option to narrow the output to a single agent.</p> |
| ET | <p>Specifies information about exception states:</p> <ul style="list-style-type: none"> • If ET=1, list exception states. • If ET=2, in addition to the information for ET=1, list LPL, GRECP, and WEPR information. |
| DI | <p>Specifies information about DDF threads to report:</p> <ul style="list-style-type: none"> • If DI=1, DDF DTM (Distributed Transaction Manager) related information for the thread. • If DI=2, in addition to the information for DI=1, DDF DC (Data Communications) related information for the thread. |
| DISYS | <p>Specifies information about DDF system to report:</p> <ul style="list-style-type: none"> • If DISYS=1, general DDF system-related information. |
| TOKEN | <p>Specifies the TOKEN value for a specific thread to be displayed. Use this option with the DI option and the DS=3 or DS=4 option. A warning message is displayed if the TOKEN value is invalid or no threads are associated with the specified TOKEN.</p> |
| RS | <p>Specifies the Relational Data Services information to report:</p> <ul style="list-style-type: none"> • If RS=1, display SQL statement and high-level Relational Data Services information. • If RS=2, in addition to the information for RS=1, display more detailed information, including plan and package information and SQLCA information. |
| SP | <p>Specifies the stored procedure information to report:</p> <ul style="list-style-type: none"> • If SP=1, display stored procedure header information. • If SP=2, display run time information about WLM stored procedures. • If SP=3, display detailed information about the stored procedures, including trace information. |

Table 11. Keywords to format SVC dumps that are issued by Db2 (continued)

| Keyword | Explanation |
|---------|---|
| SQ | <p>Specifies the status information to report on Db2 internal service tasks in the system services address space (<i>ssnmMSTR</i>), database services address space (<i>ssnmDBM1</i>), and DDF address space (<i>ssnmDIST</i>):</p> <ul style="list-style-type: none"> • If SQ=1, display the current status of all service tasks. • If SQ=2, in addition to the SQ=1 information, display a list of waiters on the service task queue. • If SQ=3, in addition to the SQ=2 information, display latch information. |
| LO | <p>Specifies the log manager information to format and report:</p> <ul style="list-style-type: none"> • If LO=1, display BSDS information. • If LO=2, in addition to the LO=1 information, display active log data set information and active log pairs. • If LO=3, in addition to the LO=2 information, display archive log information. • If LO=4, in addition to the LO=3 information, display archive log reader information. • If LO=5, in addition to the LO=4 information, display log buffer information. • If LO=6, display physical log record information. • If LO=7, display logical log record information. • If LO=8, display help information. |
| RM | <p>Specifies the resource manager vector table (RMVT) and resource manager function table (RMFT) information to format and report:</p> <ul style="list-style-type: none"> • If RM=1, display RMVT information. • If RM=2, display RMFT information. • If RM=3, in addition to the RM=2 information, display RMFT information in compressed format. • If RM=4, in addition to the RM=2 information, display RMFT information in full format. |

Use the following keyword default values:

- If the SUBSYS keyword is specified with SUMDUMP=YES, the SUBSYS keyword is used only if no summary records are found in the dump data set. In this case, the SUBSYS keyword is used to locate significant Db2 control blocks within the non-summary portion of the dump data set.
- If none of the following operands are specified on a DSNWDMP control card, the default is ALL.

| | | |
|----|----|----|
| AA | SA | LG |
|----|----|----|

- For each set of the following operands, the first operand overrides the operands that follow it when all operands are used in the same control statement. This behavior is because the first operand generates formatted data that includes the formatted data that would be generated by the operands that follow it.

| | | |
|-----|----|----|
| ALL | TT | |
| ALL | LG | |
| ALL | AA | SA |

For example, ALL overrides LG if both operands are specified.

- If the DS option is not specified, the output does not contain formatted agent information.

SUMDUMP information

Because the formatted control blocks consist of both "volatile" and "static" data, input dump data selection should be based on the state of the control blocks at a time that is as close as possible to the time of the error.

SUMDUMP=YES

For SVC dumps, the default is to use summary data records as input to the dump formatter because the summary dump records contain "volatile" data that was captured at a time close to the time of the error. Specifying YES ensures that relevant Db2 diagnostic information is sent.

SUMDUMP=NO

Because non-summary data is obtained asynchronously to the current system activity when the dump is generated, "volatile" data might have been altered or lost by the time it was placed into the dump data set.

Use this option if errors occur when you gather dump data or formatting the summary portion of the dump data set. Appropriate error messages are issued in these circumstances. In addition, use this option to format stand-alone or dynamic (z/OS console) dumps.

DS information

Use the DS option to help diagnose hangs and waits. The DS option gives you a snapshot of the activity in Db2 at the time the dump was taken.

The output that was generated when you specify the DS=1 option looks like the following.

```
ACE: 0A85FD28 Status: T Req: 010B Allied Chain
Authid: FRNT Plan: FRNMFRT Corrid: GT00FRNT Corname: PLAC0A09 Token: 000004C0
EB Primary(Asid) Home(Asid) EBSPAWN TCB/SRB -Status-- R14
0A85FD98 ISCICQ09(007D) ISCICQ09(007D) 00000000 007ABD90 Running 8BA6D1FA
Jobname #T2QMBB ASID(0032)
ACE: 0D5C7990 Status: T Req: 0005 Allied Chain
CT: 67290030 Sh/Lg: 67248840 140K Vlong: 672488D0 56K
Authid: SOFPROD Plan: DSNREXX Corrid: #T2QMBB Corname: DB2CALL Token: 0001389C
EB Primary(Asid) Home(Asid) EBSPAWN TCB/SRB -Status-- R14
0D5C7A20 #T2QMBB(0032) #T2QMBB(0032) 00000000 007BF788 Running 8F869734 01/20/2004 15:27:02.850291
```

Figure 22. Sample output from DSNWDMP with DS=1 option

| Figure Label | Description |
|--------------|---|
| 1 | Address of the agent's ACE. |
| 2 | Status of the thread. This value is the same as the ST and A fields in output from the command -DISPLAY THREAD. |
| 3 | Request count. This value is the same as the REQ field in output from the command -DISPLAY THREAD. |
| 4 | The chain that the ACE is on. Possible values are System Chain, Allied Chain, or Stored Proc. |
| 5 | Authorization ID associated with a signed-on connection. This value is blank if no sign-on took place. |
| 6 | Plan name for the agent. This value is the same as the PLAN field in output from the command -DISPLAY THREAD. |
| 7 | Correlation ID for the agent. This value is the same as the ID field in output from the command -DISPLAY THREAD. |
| 8 | Correlation name for the agent. |
| 9 | Token that is associated with the thread. This value is the same as the TOKEN field in output from the command -DISPLAY THREAD. |
| 10 | Address of the execution block. |

| Figure Label | Description |
|--------------|--|
| 11 | ASID of the primary address space. |
| 12 | ASID of the home address space. |
| 13 | Address of a spawned execution block, if there is one. |
| 14 | The TCB address, if this is a TCB. This value is zero if this is an SRB. |
| 15 | Whether the agent was suspended by Db2 at the time of the abend. |
| 16 | Value in Register 14 the last time that suspend was requested. This value is valid only if the agent is suspended. |
| 17 | Timestamp when the suspend occurred. |

Run DSNWDMP with option DS=1 or DS=2 to view allied threads in Db2. Option DS=2 generates information about all allied threads, while DS=1 generates only active threads.

The following output is from DSNWDMP with option DS=2 after a Db2 hang at shutdown. The details of the first ACE show that a CICS thread was active at the time of the shutdown, which is the reason for the hang.

```
ACE: 0A85FD28 Status: T   Req: 010B   Allied Chain
Authid: FRNT   Plan: FRNMFRT Corrid: GT00FRNT   Corrname: PLACQA09 Token: 000004C0
EB   Primary(Asid) Home(Asid) EBSPAWN TCB/SRB -Status-- R14
0A85FD98 ISCICQ09(007D) ISCICQ09(007D) 00000000 007ABD90 Running 8BA6D1FA
ACE: 0A040318 Status: N   Req: 0003   Allied Chain
Authid: ISCICQ09 Plan:   Corrid:   Corrname: PLACQA09 Token: 00000000
EB   Primary(Asid) Home(Asid) EBSPAWN TCB/SRB -Status-- R14
0A040388 ISCICQ09(007D) ISCICQ09(007D) 00000000 007A53C8 Running 00000000
```

Figure 23. Output from DSNWDMP with DS=2 option after Db2 hangs at shutdown

Output from DS=3 shows both allied agents and system agents. Option DS=3 is helpful for diagnosing problems in such areas as CPU parallelism and the distributed data facility, where the tasks are related to allied work but appear on the system ACE chain.

The following output shows DSNWDMP with option DS=3 after a hang in a parallel task.

```
ACE: 04FA2D28 Status: PT* Req: 0000   System Chain
Authid: USRT001 Plan: MYPLAN Corrid: RUN01   Corrname: BATCH   Token: 00000000
EB   Primary(Asid) Home(Asid) EBSPAWN TCB/SRB -Status-- R14
04FA2D98 V42ADB1(0067) V42ADB1(0067) 80000000 00000000 Running 8BA6D1FA
ACE: 04FA2BB8 Status: PT* Req: 0000   System Chain
Authid: USRT001 Plan: MYPLAN Corrid: RUN01   Corrname: BATCH   Token: 00000000
EB   Primary(Asid) Home(Asid) EBSPAWN TCB/SRB -Status-- R14
04FA2C28 V42ADB1(0067) V42ADB1(0067) 80000000 00000000 Running 8BA6D1FA
ACE: 04FA2A48 Status: PT* Req: 0000   System Chain
Authid: USRT001 Plan: MYPLAN Corrid: RUN01   Corrname: BATCH   Token: 00000000
EB   Primary(Asid) Home(Asid) EBSPAWN TCB/SRB -Status-- R14
04FA2AB8 V42ADB1(0067) V42ADB1(0067) 80000000 00000000 Running 8BA6D1FA
```

Figure 24. Output from DSNWDMP with DS=3 for a hang involving CPU parallelism

LM information

Use the LM option to diagnose locking problems. The following output was obtained by running DSNWDMP with options DS=1 and LM=3 to display information about held locks for an agent at thread level:

```

ACE: 05419548 Status: T * Req: 0004 Allied Chain
Short: 7F298820
Authid: USRT001 Plan: TSTA85 Corrid: GT00XA85 Corrname: CICS41 Token: 00000003
EB Primary (Asid) Home (Asid) EBSPANND TCB/SRB - Status - R14
054195D8 VD1ADB1(0067) CICS41F (001C) 00000000 005A9388 Suspended 85539590
IRLM statistics for ACE: 05419548
Suspend count - latch conflict: 0
Suspend count - other conflict: 0
Lock request count: 2
Unlock request count: 1
Query request count: 0
Change request count: 0
Other IRLM requests count: 0
P-Lock Lock requests: 0
P-Lock Change requests: 0
P-Lock Unlock requests: 0
Lock requests sent to XES : 0
Change requests sent to XES : 0
Unlock requests sent to XES : 0
Suspends due to IRLM Global contention : 0
Suspends due to XES Global contention : 0
Suspends due to false contention : 0
Requests denied due to retained lock : 0
Notify messages sent : 0
WU requested a LOCK at 03/21/1997 20:00:47.736272 for resource 0C0000020004001700000000 at state 3
with modify intent.
This is a local resource that is held on this subsystem.
This request is incompatible with a lock on this subsystem and the current states of this resource are:
The local resultant held state is 8
The local modify held state is 8
Resource 0C0000020001005F00000000 is held in state 02 and was granted at 03/21/1997 20:00:47.620534.
Resource 0C000004E3E2E3C1F8F54840 is held in state 04 and was granted at 03/21/1997 20:00:45.712241.
Work unit 0067009505419548 holds 00000002 resource locks.

```

Figure 25. Output from DSNWDMP with DS=1 and LM=3 for held lock information

Analyze Db2 storage by using the SM options

The Db2 dump formatter provides a way to analyze the usage of Db2 storage with the Storage Manager (SM) options. Each level of the SM option provides more detailed information than the previous level.

Table 11 on page 186 summarizes the information generated for each value of SM and for SMSA.

All storage manager reports show information about local storage, global storage (z/OS CSA/ECSA), and EDM pool statistics.

To obtain the simplest local storage report, run the DSNWDMP statement with SM=1. The local storage report is divided into two sections: one for fixed storage and one for variable storage. Fixed storage is storage that has a predetermined length, and each element has the same length.

The following example shows a local storage report that is generated for the following DSNWDMP statement:

```
DSNWDMP 'SUBSYS=VD1A,SM=1'
```

```

==Storage (Local) VARIABLE POOLS
PHB      Subpool name          F/V          CL SP
---      -
 1      2      3      4 5 6
7E9CB630  ADMF AGL 31 .pA.          Var          136K 12 229
66D109C0  ADMF AGL 31 .\..          Var           0K 12 229
50C22EC0  ADMF AGL VL .Z..          Var          28K 12 229
589B7D90  ADMF AGL VL ....          Var         180K 12 229
7E9BF7F0  ADMF AGL VL '...          Var          56K 12 229
7E983AB0  ADMF AGL VL '!.          Var          28K 12 229
7E97CB20  ADMF AGL 31 'm..          Var          992K 12 229
7E968C60  ADMF AGL 31 '.N.          Var         1204K 12 229

```

Figure 26. Local storage report for DSNWDMP with SM=1 option

| Figure Label | Description |
|--------------|-------------|
|--------------|-------------|

- | | |
|---|---------------------------------------|
| 1 | Pool header block address |
| 2 | Pool name |
| 3 | Fixed (fix) or variable (var) storage |
| 4 | Size of pool |
| 5 | Db2 storage class |
| 6 | z/OS subpool |

When you specify SM=1, DSNWDMP also generates storage statistics. The following shows an example of a storage statistics report.

```
Storage statistics
-----
QSSTGPLF Get Fixed length pool in dataspace      149  QSSTGPLV Variable pool getmains      1081508
QSSTFPLF Free Fixed length pool in dataspace     72  QSSTFPLV Variable pool freemains    1076402
QSSTFREF Fixed pools                             6   QSSTFREV Variable pools             570252
QSSTEXPF Fixed pool extensions                   5153 QSSTEXPV Variable pool extensions    924971
QSSTCONF Fixed contracted blocks                 748  QSSTCONV Variable contracted blocks  333901
QSSTGETM Getmains(GETM)                        190487 QSSTFREM Freemains (GETM)         188142
QSSTRCNZ Non RC=0 Getmains                      0    QSSTCRIT Short on storage           10812
QSSTCONT Contractions initiated                  11238 QSSTABND Abends due to insufficient storage  51
```

Figure 27. Storage statistics report for DSNWDMP with SM=1 option

To obtain a local storage report with SHB control blocks, run the DSNWDMP statement with SM=2. In a fixed storage pool, the storage always follows the SHB, so only the SHB address is listed in the report. Each block of variable storage has a varying length. With variable storage, the SHB is not in the same place as the storage, so the address of the storage is listed after the address of the SHB.

The following shows an example of a local storage report that is generated for the following DSNWDMP statement:

```
DSNWDMP 'SUBSYS=VD1A,SM=2,SMSA=2D'
```

```
==Storage (Local) VARIABLE POOLS
PHB      Subpool name      F/V      CL SP
-----
7E9CB630  ADMF AGL 31 .pA.      Var      136K  12 229
          SHB: 65BC2D88 52A32000
66D109C0  ADMF AGL 31 .\..      Var      0K    12 229
50C22EC0  ADMF AGL VL .Z..      Var      28K   12 229
          1          2          3
          SHB: 50CB8B08 2CA5A000      16K
          SHB: 50CB8AB8 5089F000      8K
          SHB: 50CB8A18 50A3A000      4K
```

Figure 28. Local storage report for DSNWDMP with SM=2 option

| Figure Label | Description |
|--------------|-------------|
|--------------|-------------|

- | | |
|---|--|
| 1 | SHB address |
| 2 | Address of storage that SHB relates to |
| 3 | Length of storage |

Within a storage pool, the sum of the segment (SHB) lengths should always equal the length that is described by the pool header block (PHB). In the previous example, the PHB length, 32 KB, is equal to the sum of the SHB lengths: 12 KB+8 KB+8 KB+4 KB.

Use SM=4 and SM=5 to check for fragmentation within blocks and storage use within individual variable storage pool blocks. The report includes the total length of allocated and free elements for each segment of storage and the total for the storage pool. The total for the storage pool appears after the last SHB for the pool.

The following shows an example of a local storage report that is generated for the following DSNWDMP statement:

```
DSNWDMP 'SUBSYS=VD1A,SM=4,SMSA=2D'
```

```

Jobname VD1ADBM1 ASID(002D)
==Storage (Local) FIXED POOLS
PHB      Subpool name          F/V          CL SP
---      -
7F7594B8 EDM FIXED LENGTH POOL   Fix          4K 12 229
          SHB: 7F2DE000
7F759518 EDM FIXED LENGTH POOL   Fix          4K 12 229
          SHB: 7F0AF000
7F759810 RDS STPM LOCAL POOL        Fix          0K 12 229
7F75B290 SHBS FOR LOCAL V-POOLS    Fix          4K 12 229
          SHB: 7F705000

==Storage (Local) VARIABLE POOLS
PHB      Subpool name          F/V          CL SP
---      -
7F2C4510 ADMF AGENT LOCAL POOL   Var          32K 12 229
          SHB: 7F705AD8 7EF50000    12K  Free: 00000578    1K Alloc: 00002A48    10K
          SHB: 7F705AB8 7EF53000     8K  Free: 00001FD0    7K Alloc: 00000000     0K
          SHB: 7F7059D8 7EF6B000     8K  Free: 000003E0    0K Alloc: 00001BD0     6K
          SHB: 7F7059B8 7EF80000     4K  Free: 00000160    0K Alloc: 00000E20     3K
          Total:      Free: 00002A88    10K Alloc: 00005438    21K

```

Figure 29. Local storage report for DSNWDMP with SM=4 option

Use SM=5 to display the first 16 bytes, with their hexadecimal equivalents, of each allocated element. For the module that made a request to Db2 storage manager for storage, DSNWDMP attempts to display the following extra information:

- The owner token for the storage
- The CSECT and offset within the CSECT of the storage allocation request
- The maintenance level of the module

It is not always possible to determine the CSECT name. Therefore, this name might be incorrect or absent in some cases.

The following shows an example of a local storage report that is generated for the following DSNWDMP statement:

```
DSNWDMP 'SUBSYS=VD1A,SM=5,SMSA=2D'
```

```

==Storage (Local) VARIABLE POOLS
PHB      Subpool name          F/V          CL SP
---      -
7E9CB630 ADMF AGL 31 .pA.     Var          136K 12 229
          SHB: 55822088 52A52080    136K  Free: 00021F60    135K Alloc: 00000060     0K
          Total:      Free: 00021F60    135K Alloc: 00000060     0K
66D39FC0 ADMF AGL 31 .)....     Var          8K 12 229
59C32E30 ADMF AGL V1 .)....     Var          28K 12 229
2CA5A880: 00000048 261C083C E2E3C1C3 00000000 00000000 ...STAC..... 2FE99C30 DSNIALLI +2387 15.28
2CA5A808: 00000038 001C0830 C908E4C5 00000000 2FE99C30 ...IQE..... 2FE99C30 DSNIALLI +1A33 15.28
2CA5A120: 00003EC8 20FE3EC0 C4D4E309 2CA5A160 2CA5B860 ...DMTR.v... 2FE99C30 DSNIALLI +1740 15.28

```

Figure 30. Local storage report for DSNWDMP with SM=5 option

| Figure Label | Description |
|--------------|-------------|
|--------------|-------------|

- | | |
|---|--|
| 1 | Owner token for the storage |
| 2 | CSECT and offset of the storage allocation request |
| 3 | Maintenance level |

You can specify DS options with SM options to display storage information by thread. This information includes the short, long and vlong pool addresses. These addresses are addresses of storage pools within the database services address space (ssnmDBM1).

The following shows an example of a storage report that is generated for the following DSNWDMP statement:

```
DSNWDMP 'SUBSYS=VD1A,DS=1,SM=5,SMSA=2D'
```

This example shows that a CICS thread signed on as user USRT001, running plan TSTA85, on correlation ID PT01XA85 is using ADMF local pool 7F2C4510. The thread uses 32 KB of storage, of which 21 KB is currently allocated.

```

Jobname CICS41F ASID(0021)
ACE: 0624C008 Status: T Req: 0003 Allied Chain
CT: 7F2E5030 Short: 7F2C43C0 Long: 7F2C43C0 Vlong: 7F2C4350
Authid: USRT001 Plan: TSTA85 Corrid: PT03XA85 Corrname: CICS41 Token: 00000028
EB Primary(Asid) Home(Asid) EBSPAWN TCB/SRB -Status-- R14
0624C098 CICS41F (0021) CICS41F (0021) 00000000 006A67D0 Running 00000000
ACE: 0624BE38 Status: T Req: 0003 Allied Chain
CT: 7F2E3030 Short: 7F2C44A0 Long: 7F2C44A0 Vlong: 7F2C4430
Authid: USRT001 Plan: TSTA85 Corrid: PT02XA85 Corrname: CICS41 Token: 00000027
EB Primary(Asid) Home(Asid) EBSPAWN TCB/SRB -Status-- R14
0624BEC8 CICS41F (0021) CICS41F (0021) 00000000 006A6A60 Running 00000000
ACE: 0624BC78 Status: T Req: 0004 Allied Chain
CT: 7F2E1030 Short: 7F2C4580 Long: 7F2C4580 Vlong: 7F2C4510
Authid: USRT001 Plan: TSTA85 Corrid: PT01XA85 Corrname: CICS41 Token: 00000026
EB Primary(Asid) Home(Asid) EBSPAWN TCB/SRB -Status-- R14
0624BD08 CICS41F (0021) CICS41F (0021) 00000000 006A6CF0 Running 861EF578

```

Figure 31. Storage by thread report for DSNWDMP with DS=1 and SM=5 option

For all storage reports, DSNWDMP displays EDM pool statistics. Those statistics are the same ones that you obtain through Db2 statistics trace reports. The following shows an example report.

```

==EDM statistics
-----
Fails due to POOL full:                0
Pages in EDM POOL:                    225
REQ for CT sections:                  28
Load CT sections from DASD:            3
Pages used for CT:                     8
Free pages in free chain:              194
Pages used for DBDs:                   18
Pages used for SKCT:                   3
Requests for DBD:                      24
Loads of DBDS from DASD:               3
Requests for PT sections:              6
Loads of PT sections from DASD:        2
Pages used for PT:                     0
Pages used for SKPT:                   2
Inserts for dynamic cache:             0
Requests for dynamic cache:            0
Pages used for dynamic cache:          0

```

Figure 32. EDM pool statistics report for DSNWDMP with SM option

SVC dumps

For all abends, Db2 recovery routines request SVC dumps. SVC dumps that are issued by Db2 are the primary source of diagnostic information for Db2 problems.

When you review these dumps, keep in mind the following information:

- These dumps reflect the multiple address space environment in which Db2 was operating at the time of the failure.
- When Db2 requests the capture of pertinent Db2 storage areas as summary dump data (SUMLSTA), z/OS suspends Db2 and copies these areas into the z/OS dump services address space. This action preserves the status of these areas at time of error during subsequent Db2 recovery processing.

Processing of the z/OS dump services address space and capturing of more non-summary storage areas that are specified by Db2 are performed asynchronously to Db2 recovery processing once z/OS resumes Db2.

- The non-summary portion of the dump consists of the primary, secondary, and home address spaces.
- The dumps can be formatted or unformatted (this information refers to both). In most instances, use formatted dumps.
- z/OS provides two indexes in formatted dumps ("Print Dump Index" and "SUMDUMP Dump Index"), which help to locate particular storage areas.

Related concepts

[Program call linkages](#)

Db2 uses z/OS program calls as one form of program invocation.

Related reference

Dump indexes

The Print Dump Index and the SUMDUMP Dump Index in z/OS dumps make it much easier to find specific items and address ranges.

General contents of an SVC dump

When Db2 requests SVC dumps, it specifies several parameters to z/OS SDUMP services. For SDATA parameters, Db2 specifies SQA, ALLPSA, LSQA, and SUMDUMP. This action causes z/OS to collect such information as the selected common service area subpool, from which the global trace table is obtained, and the system queue area subpool.

Db2 also specifies address lists through the ASIDLST and SUMLSTA parameters, causing z/OS to include the system services address space (*ssnmMSTR*), database services address space (*ssnmDBM1*), allied address space of the failing allied task, and other pertinent diagnostic information.

Finally, Db2 specifies the BRANCH=YES and SUSPEND=YES parameters. Several areas are included as a result, such as the z/OS trace table and the home ASCB (address space control block).

For dumps requested by Db2 functional recovery routines (FRR and ESTAE), the SDWA (system diagnostic work area) is also included. The SDWA is described in:

- [“The system diagnostic work area \(SDWA\)” on page 204](#)
- [“SYS1.LOGREC” on page 226.](#)

Part of the SDWA includes the variable recording area (VRA). Db2 formats this area in the VRA Diagnostic Information Report. Each entry within the VRA is recorded during functional recovery processing to provide more diagnostic information that is related to the initial abend reason code.

For dumps requested by Db2 ESTAE routines, the RTM2WA is also included.

For details on the SDUMP parameters, see the z/OS system macros and facilities publications. For more information about SVC dumps, see the general set of the z/OS diagnostic techniques and debugging handbook publications.

Figure 33. First pages of an SVC dump that is issued by Db2

```
ERRORID FOR THIS DUMP = SEQ00013 CPU00 ASID0007 TIME13.11.29.2
ACTIVE CPU'S AT TIME OF DUMP
  ADDR  VERS.  SERIAL  MODEL
  0000   FF   120762  3084
***** DUMP ANALYSIS AND ELIMINATION (DAE) *****
THIS DUMP WAS NOT SUPPRESSED BECAUSE
DAE WAS NOT CHECKING FOR PREVIOUS OCCURRENCES.
CRITERIA FOR USE AS A UNIQUE DUMP IDENTIFIER BY DAE:
  MINIMUM NUMBER OF SYMPTOMS: 08 FOUND: 10
  MINIMUM TOTAL STRING LENGTH: 025 FOUND: 147
  SYMPTOMS REQUIRED TO BE PRESENT:
  MOD/CSECT/
  SYMPTOMS THAT ARE TO BE USED IF AVAILABLE, BUT ARE NOT REQUIRED:
  PIDS/ AB/S AB/U REXN/ FI/ REGS/ HRC1/ SUB1/
MVS SYMPTOM STRING:
MOD/DSNSLD1 CSECT/DSNSFBK PIDS/5740XYR00 AB/S004E REXN/DSNYEATE
FI/8910000C0A0D5980C5774770 REGS/0E0BE REGS/0946A HRC1/00E2000B
SUB1/IPC##/DSNYAGCS
RETAIN SEARCH ARGUMENT:
RIDS/DSNSLD1#L RIDS/DSNSFBK PIDS/5740XYR00 AB/S004E RIDS/DSNYEATE#R
VALU/HC5774770 REGS/0E0BE REGS/0946A PRCS/00E2000B VALU/CDSNYAGCS
SYMPTOMS PRESENT FOR USE AS A UNIQUE DUMP IDENTIFIER BY DAE:
  RETAIN
MVS KEY   KEY           SYMPTOM DATA           EXPLANATION
-----
MOD/      RIDS/         DSNSLD1                 LOAD MODULE NAME
CSECT/    RIDS/         DSNSFBK                 ASSEMBLY MODULE CSECT NAME
PIDS/     PIDS/         5740XYR00              PRODUCT/COMPONENT IDENTIFIER
AB/S      AB/S          S004E                  ABEND-CODE SYSTEM
REXN/     RIDS/         DSNYEATE               RECOVERY ROUTINE CSECT NAME
FI/       VALU/H        8910000C0A0D5980C5774770 FAILING INSTRUCTION AREA
REGS/     REGS/         0E0BE                  REG/PSW DIFFERENCE
```

| | | | |
|-------|--------|---------------|-----------------------|
| REGS/ | REGS/ | 0946A | REG/PSW DIFFERENCE |
| HRC1/ | PRCS/ | 00E2000B | REASON CODE |
| SUB1/ | VALU/C | IPC##DSNYAGCS | COMPONENT SUBFUNCTION |

:

:

ADDITIONAL SYMPTOM DATA NOT USED BY DAE TO IDENTIFY THIS DUMP:

| MVS KEY | RETAIN KEY | SYMPTOM DATA | EXPLANATION |
|-------------|---------------|-----------------------------|--------------------------------|
| VARC/ | PRCS/ | 00E2000B | ABEND REASON CODE |
| VCBA/ | ADRS/ | 01C31500 | CONTROL BLOCK ADDRESS |
| VAID/ | VALU/H | 0010 | CALLERS ASID |
| VAID/ | VALU/H | 000F | CALLERS ASID |
| VAID/ | VALU/H | 0007 | CALLERS ASID |
| VCAN/ | RIDS/ | AE##DSNVASIM03#10#8621#23## | MODULE NAME OF CALLER |
| VIM0/ | ADRS/ | FFFF048881C95F8016B00007 | OFFSET IN ASSEMBLY MODULE |
| CID1/ | VALU/C | XYR00 | COMPONENT IDENTIFIER |
| AMD1/ | VALU/C | 02#06#87 | MODULE ASSEMBLY DATE |
| VRS1/ | VALU/C | 11#40 | VERSION-PRODUCT/PTF IDENTIFIER |
| CDB1/ | VALU/C | 5740 | BASE COMPONENT IDENTIFIER |
| ASID1/ | VALU/H | 0007 | TASK RELATED ASID |
| ORCC1/ | PRCS/ | 04E000 | ORIGINAL COMPLETION CODE |
| atsign.202/ | VALU/C | SSID/DS GRP NAME | DEVELOPER ASSIGNED SYMPTOM KEY |
| atsign.204/ | VALU/C | SSUR | DEVELOPER ASSIGNED SYMPTOM KEY |
| atsign.219/ | VALU/C | ##W##N## | DEVELOPER ASSIGNED SYMPTOM KEY |

:

Related concepts

[The variable recording area \(VRA\)](#)

More diagnostic information for Db2 abend reason codes is placed in the variable recording area (VRA) of the system diagnostic work area (SDWA) and is extracted and displayed in the VRA Diagnostic Information Report. This data can be produced by common recording routines and certain Db2 subcomponents.

[The recovery termination manager work area \(RTM2WA\)](#)

The RTM2WA is a z/OS work area that is requested by Db2 ESTAE routines. z/OS formats and includes this block when a summary dump is printed or when a formatted address space dump is requested.

Contents of an SVC dump unique to Db2

SVC dumps contain extra information specific to Db2 in the summary portion of the dump.

Summary portion of the dump

The information that Db2 passes to the z/OS SDUMP service aid through the SUMLSTA parameter causes z/OS to capture volatile areas at failure before other system activity can change their contents.

- Table of Contents at the beginning of dumps that are formatted through IPCS
- SUMDUMP Dump Index (located by a reference in Print Dump Index)

Related concepts

[The module entry point list \(MEPL\)](#)

The Db2 module entry point list (MEPL) is found in all SVC dumps that are issued by Db2. It identifies the names of the load modules and CSECTs that are loaded into the subsystem at startup and remain in storage for the life of the subsystem.

[The SQL communication area \(SQLCA\)](#)

The SQL communication area is one of the most important blocks. It contains information about the status of one SQL statement. For diagnostic purposes, up to 4 KB of the SQL statement are included in both the formatted and unformatted sections of SVC dumps.

Related tasks

[Finding the SQL statement](#)

Up to 4 KB of the SQL statement are included in the sections of the SVC dump that pertain to SQL problems. To find the SQL statement in the formatted section of the SVC dump, skim through the pages for the statement on the right. It is displayed in the space block, which contains the "SPA" eye-catcher.

Related reference

Dump indexes

The Print Dump Index and the SUMDUMP Dump Index in z/OS dumps make it much easier to find specific items and address ranges.

SVC dump titles that are issued by Db2

The dump title at the beginning of an SVC dump that is issued by Db2 includes the abend completion and reason codes, the failing load module and CSECT names, and the release identifier.

The formats of SVC dump titles vary slightly, depending on the type of error. The following pages first illustrate and describe the dump title format for most abend reason codes. Then, variations are illustrated and described.

SVC dumps can be displayed with two messages: DSNW050I and DSNW051I. DSNW050I indicates no SDWA was available; DSNW051I indicates an error occurred during dump processing, which might render the dump inaccurate or incomplete.

Common dump titles for abends

Format for threads with no remote activity

The following sample illustrates the SVC dump title format for threads with no remote activity. Each field in the title is described after the figure.

```
ssid,ABND=compltn-reason,U=authid,  
C=compid.release.comp-function,  
M=module, LOC=loadmod.csect+csect_offset
```

Figure 34. Sample SVC dump title for threads with no remote activity

ssid

The four character subsystem name.

compltn

The three character abend completion code (often X'04E' or X'04F'). In structured database format, the completion code is preceded by "AB/S".

reason

The 4-byte hexadecimal reason code (such as X'00E20015'). In structured database format, the reason code is preceded by "PRCS".

authid

The primary authorization ID of the user (such as 'SYSOPR'). No structured database keyword equivalent is displayed in the dump.

compid

The last five characters of the component identification keyword, explained in “Component identifier keyword” on page 10. The value XYR00 uniquely identifies Db2. XYR01 also identifies Db2, but refers to the subsystem initialization, or ERLY, code in particular. In structured database format, the component identifier is preceded by "PIDS/".

release

A three-digit code that indicates the version, release, and modification level of Db2. In structured database format, the release identifier has no equivalent field.

comp

Often is an acronym for the subcomponent in control at the time of the abend (such as 'BMC'). This field is internally defined and varies depending on the circumstances under which the dump was generated. No structured database keyword equivalents is displayed in the dump.

function

Often is the name of a function, macro, or routine in control at the time of abend (such as 'DSNB1CMS'). This field is internally defined and varies depending on the circumstances under which the dump was generated. No structured database keyword equivalent is displayed in the dump.

module

The name of the FRR or ESTAE recovery routine (such as 'DSNTFRCV'). In structured database format, this information is preceded by "RIDS/".

loadmod

The name of the load module in control at the time of the abend (such as 'DSNSLD1'). In structured database format, the load module name is preceded by "RIDS/".

csect

The name of the CSECT in control at the time of abend (such as 'DSNSVSTK'). In structured database format, the CSECT name is preceded by "RIDS/".

csect_offset

Usually is the offset within the failing CSECT at time of abend (such as '001CE'). This data is sometimes replaced by an abend reason code qualifier. In structured database format, the CSECT offset has no equivalent field.

Format for Allied Threads with Remote Activity

The following sample illustrates the SVC dump title format for allied threads with remote activity (requesting location threads). The fields that are specific to this title format are described after the figure.

```
ssid,ABND=compltn-reason,U=authid,
C=compid.release.comp-function,
DISTRIBUTED,LOC=loadmod.csect+csect_offset
```

Figure 35. Sample SVC dump title for allied threads with remote activity

DISTRBUTED

A flag that indicates that the thread had a remote connection. (SDWA VRA fields contain further information.)

Format for database access threads

The following sample illustrates the SVC dump title format for database access threads (responding location threads). The fields that are specific to this title format are described after the figure.

```
ssid,ABND=compltn-reason,U=authid,
C=compid.release,LOCN=16_char_loc_name,
LOCN=16_char_loc_name,
LOC=loadmod.csect+csect_offset
```

Figure 36. Sample SVC dump title for database access threads

LOCN

The Db2 location name of the remote system.

Related information

[DSNW050I \(Db2 Messages\)](#)

[DSNW051I \(Db2 Messages\)](#)

Dump title variations

The SVC dump title can appear in alternative formats.

Data manager variation

Certain data manager (DM) reason codes replace the CSECT offset with a reason code qualifier, as shown in the following figure.

```
ssid,ABND=compltn-00C9xxxx,U=authid,  
C=compid.release.comp-function,  
M=module, LOC=loadmod.csect:qualifier
```

Figure 37. Dump title with data manager reason code qualifier

The X'00C9' reason code prefix that follows the "-ABND" label indicates a data manager abend. The *qualifier* following the CSECT name is a four-digit hexadecimal reason code qualifier, often a trace code (such as 'OD01'). A colon precedes it. No structured database keyword equivalent is displayed in the dump.

Distributed title variations

In distributed data facility modules that contain several occurrences of the same reason code, an *ABNDID* replaces the CSECT offset in the SVC dump title as shown in the following figure.

```
ssid,ABND=compltn-00D3xxxx,U=authid,  
C=compid.release.comp-function,  
DISTRBUTED,LOC=loadmod.csect:abndid
```

Figure 38. Dump title with distributed data facility reason code *ABNDID*

The X'00D3' reason code prefix that follows the "sABND" label indicates a distributed data facility abend. The *ABNDID* following the CSECT name uniquely identifies the abend from more than one occurrence of the same reason code in the failing module. A colon precedes it. No structured database keyword equivalent is displayed in the dump.

Relational data system variation

All relational data system (RDS) reason codes replace the CSECT offset with an abend reason code qualifier, which is shown in the following figure.

```
ssid,ABND=compltn-00E7xxxx,U=authid,  
C=compid.release.comp-function,  
M=module,LOC=loadmod.csect:sign_qualifier
```

Figure 39. Dump title with relational data system reason code qualifier

The X'00E7' reason code prefix that follows the "-ABND" indicates an RDS abend. A subset of this reason code prefix, X'00E72' indicates an error in the SORT component. The sign that follows the CSECT name is a one-character field that indicates plus or minus (P or M). The qualifier is a three-digit decimal reason code qualifier (such as 101). No structured database keyword equivalents are displayed in the dump.

Variation with PSW and ASID

Some dump titles replace the load module name, CSECT name, and CSECT offset (or abend reason code qualifier) with the PSW and ASID (address space identifier). The following figure illustrates this format.

```
ssid,ABND=compltn-reason,U=authid,  
C=compid.release.comp-function,  
M=module,PSW=psw_contents,A=address_spaceid
```

Figure 40. Dump title with PSW and ASID

psw_contents

Contains the PSW at time of error (such as X'077C100000729F9C'). No structured database keyword equivalent is displayed in the dump.

address_spaceid

Identifies the address space in control at time of abend (such as X'0011'). In structured database format, this item is preceded by "VALU/H".

Error qualifier

An error qualifier exists to provide more information about what was going on.

The error qualifier can be found in the LOC keyword of the SVC dump title or in the field of the CT named CTERQUAL. The first digit of this error qualifier identifies the Db2 resource manager that detected the problem. The following lists possible values for this error qualifier:

Code

Resource Manager that is involved at time of abend

X'1xxx'

Internal Resource Lock Manager (IRLM)

An error return code was returned by the IRLM on a lock request by the data manager (DM). This situation usually does not indicate an inconsistency problem. An example of an X'1xxx' error qualifier is a situation that occurs when IRLM runs out of virtual storage that it uses to hold locks.

X'2xxx'

Buffer Manager (BM)

An error return code was returned by the BM on a BM request by the DM. This result might indicate inconsistent data. An example is a situation in which a RID contains a page number that is in error, causing the BM to detect an error when it attempts to access the page with the invalid number.

X'3xxx'

Recovery Log Manager (RLM)

An error return code was returned by RLM on a log write request by the DM. This result usually does not indicate an inconsistency problem. An example is a situation in which the RLM attempts to write a log record whose length is greater than 32 KB; this error is a DM internal error.

X'5xxx'

Data Manager (DM)

When the DM returns an error qualifier, it detected some kind of internal inconsistency. In most cases, these situations do not involve inconsistent data, but rather, inconsistent internal parameters or control blocks. For a list of error qualifiers that are associated with DSN1COPY misuse, see [ABEND codes associated with DSN1COPY misuse \(Db2 Codes\)](#).

X'0Cxx'

Data Manager (DM)

This error qualifier from the DM indicates that data inconsistency was detected. Check DSNWEIDS in SDSNSAMP for descriptions of the error.

The low-order three digits of these error qualifiers represent a unique sequence number that identifies the place within the CSECT where the abend was issued. This information can be helpful to IBM Support if they get involved in resolving the inconsistency problem.

SVC dump titles that are issued by IRLM

IRLM issues SVC dump titles.

The following sample illustrates the format of an SVC dump title that is issued by IRLM.

```
DXR7 ESTAE ENTERED. ABEND U2025 MODULE DXRRL732+0162
APAR BASE 95/135 23:
```

Figure 41. Sample SVC dump title that is issued by IRLM

| SUMDUMP DUMP INDEX | | | | | |
|--------------------|------------|-----|-----------|--|-------------|
| DATA AREA | | | | | PAGE NUMBER |
| SDWA | | | | | 00000006 |
| PSA/PCCA/ICCA | | | | | 00000006 |
| ASCB | | | | | 00000011 |
| TCB | | | | | 00000011 |
| INT HANDLER SA | | | | | 00000011 |
| SUSPEND REGS SA | | | | | 00000014 |
| SUMLIST/SUMLISTA | - 00AE0000 | --- | ASID FFFF | | 00000014 |
| SUMLIST/SUMLISTA | - 00B2E000 | --- | ASID FFFF | | 00000014 |
| SUMLIST/SUMLISTA | - 00B2E2C4 | --- | ASID FFFF | | 00000017 |
| SUMLIST/SUMLISTA | - 00B2E2C8 | --- | ASID FFFF | | 00000017 |
| SUMLIST/SUMLISTA | - 00B2EA5C | --- | ASID FFFF | | 00000017 |
| SUMLIST/SUMLISTA | - 00B2EB34 | --- | ASID FFFF | | 00000017 |
| SUMLIST/SUMLISTA | - 00B2EC0C | --- | ASID FFFF | | 00000018 |
| SUMLIST/SUMLISTA | - 00B32000 | --- | ASID FFFF | | 00000018 |
| SUMLIST/SUMLISTA | - 00B4BED8 | --- | ASID FFFF | | 00000018 |
| SUMLIST/SUMLISTA | - 00B4BF40 | --- | ASID FFFF | | 00000018 |
| SUMLIST/SUMLISTA | - 00B4BF98 | --- | ASID FFFF | | 00000019 |
| SUMLIST/SUMLISTA | - 00B78198 | --- | ASID FFFF | | 00000019 |
| SUMLIST/SUMLISTA | - 01A75000 | --- | ASID FFFF | | 00000019 |
| SUMLIST/SUMLISTA | - 01A76000 | --- | ASID FFFF | | 00000021 |
| SUMLIST/SUMLISTA | - 01A77000 | --- | ASID FFFF | | 00000023 |
| SUMLIST/SUMLISTA | - 01A78000 | --- | ASID FFFF | | 00000025 |
| SUMLIST/SUMLISTA | - 01A79000 | --- | ASID FFFF | | 00000028 |
| SUMLIST/SUMLISTA | - 01A7A000 | --- | ASID FFFF | | 00000030 |
| SUMLIST/SUMLISTA | - 01A7B000 | --- | ASID FFFF | | 00000033 |
| SUMLIST/SUMLISTA | - 01A7C000 | --- | ASID FFFF | | 00000033 |
| SUMLIST/SUMLISTA | - 01A7DA40 | --- | ASID FFFF | | 00000034 |
| SUMLIST/SUMLISTA | - 01A7DC40 | --- | ASID FFFF | | 00000034 |
| SUMLIST/SUMLISTA | - 01A7F0E8 | --- | ASID FFFF | | 00000035 |
| SUMLIST/SUMLISTA | - 01A7F170 | --- | ASID FFFF | | 00000035 |
| SUMLIST/SUMLISTA | - 01A7F1C8 | --- | ASID FFFF | | 00000035 |
| SUMLIST/SUMLISTA | - 01A7F230 | --- | ASID FFFF | | 00000035 |

Figure 43. Sample SUMDUMP Dump Index

To locate this index, review the Print Dump Index or Table of Contents, and look for "SUMDUMP Index". The SUMDUMP index can be used to quickly locate addresses in the unformatted sections of the SVC dump.

The system diagnostic work area (SDWA)

Each SVC dump that is requested by a Db2 functional recovery routine usually contains an SDWA with information about the status of the subsystem at the time of the error. Typically, the SDWA is a starting point for diagnosis.

Finding the SDWA

Use the SUMDUMP Dump Index to locate the SDWA. For more information about the contents of the SDWA:

- [“SYS1.LOGREC” on page 226](#)
- [Using the SDWA \(MVS Programming Authorized Assembler Services Reference\)](#)

Using the SDWA during the secondary error recovery processing

The z/OS SDUMP macro uses the SDWA at time of invocation to gather the storage areas around the registers (SDWAGRSV) and PSW (SDWAEC1) at the time of error to include in the dump data set. If secondary errors occur during Db2 dump processing, the SDWA used during SDUMP processing would therefore reflect the registers and PSW of the secondary error; the original SDWA error information would be lost.

To avoid losing the original error information and to retain as much secondary error information as possible, Db2 saves selected fields from both the original and secondary SDWA in the DMPW data area. Before you start SDUMP, Db2 dump services then uses fields from the original SDWA to overlay the corresponding fields (restored) in the secondary SDWA.

The following list describes in greater detail the services that are performed when errors occur during dump processing.

- These fields from both the original and secondary SDWA are saved in the DMPW data area:

| FIELD | LENGTH | FUNCTION |
|------------|--------|-------------------------------|
| Eyecatcher | 4 | 'OERR' and 'SERR' |
| SDWAFLGS | 4 | SDWA flags (saved only) |
| SDWAABCC | 4 | Completion code (restored) |
| SDWAGRSV | 64 | Registers at error (restored) |
| SDWAEC1 | 8 | Ext/CTL PSW (restored) |
| SDWAAEC1 | 8 | EC mode data (restored) |
| SDWACOMU | 8 | SDWA communication (restored) |

Figure 44. DMPW data area

- The DMPW fields from the SDWA reflecting the original error 'OERR' are used to overlay the same fields in the secondary SDWA 'SERR'. The summary storage areas that are provided by SDUMP around the registers and PSW at the time of the error now reflect the original error.
- Both the original and secondary SDWA error fields are kept in the DMPW for later examination during problem analysis.
- This error message is inserted into the header portion of the DMPW to show that a secondary error occurred.

The variable recording area (VRA)

More diagnostic information for Db2 abend reason codes is placed in the variable recording area (VRA) of the system diagnostic work area (SDWA) and is extracted and displayed in the VRA Diagnostic Information Report. This data can be produced by common recording routines and certain Db2 subcomponents.

VRA entries begin at offset X'190' in the SDWA. Each entry within the VRA is recorded during functional recovery processing to provide more diagnostic information that is related to the initial abend reason code (SDWACRC).

Finding the VRA diagnostic information report

The VRA Diagnostic Information Report is displayed at the front of the section that is formatted by Db2 of the SVC dump and can be identified by its message number, DSNW053I. The Print Dump Index cites the location of the formatted dump under the entry "Output from DSNWDMP Verb".

The report cites the VRA hexadecimal key codes, the length of the VRA data, and the data itself. The following figure shows an example of the report.

```

DSN0053I - VRA DIAGNOSTIC INFORMATION REPORT
SDWA=1 (00FFA9C1) SDWA=1 (00B4070C340) OFFSET=0190
KEY: LEN: VRA DATA FIELDS: *S... *
00 04 00000000 *C... *
L4 04 01C31500 *... *
CA 04 03000000 *S310/OS GRP NAME *
SA 02 0010 *... *
SA 02 0007 *... *
CA 08 00000000 00000000 *... *
CC 0C E2E2E209 40404040 40404040 *S0UR *
00 08 027A2A0A WAGC0000 *... *
TC 0C FFFF040B 81199F0B 18800007 *... *
00 04 81000000 040200E5 C1E20004 F0F361F1 F061F0F4 F2F14BF2 F3404040 00000000 *E: DSNW053I/03/10/0621.23 ... *
00 00 00000000 *... *
S2 02 0000 *... *
DSN0054I - VRA DIAGNOSTIC INFORMATION REPORT COMPLETE

```

Figure 45. Sample VRA diagnostic information report

Related reference

[SYS1.LOGREC](#)

The SYS1.LOGREC data set records various errors that different components of the operating system encounter.

Related information

[DSNW053I \(Db2 Messages\)](#)

The recovery termination manager work area (RTM2WA)

The RTM2WA is a z/OS work area that is requested by Db2 ESTAE routines. z/OS formats and includes this block when a summary dump is printed or when a formatted address space dump is requested.

The RTM2WA block contains the most recent information about the failing task at the time the dump was produced. Use this information to relate the problem in a dump to a specific SYS1.LOGREC entry by comparing the completion code, PSW, and abend reason code (if present) in register 15. This control block is used by the RTM to control processing of abends. It includes registers, the PSW, the abend completion code, and other useful information about the status of the subsystem at the time of the failure. The RTM2WA maps a description of the errors and control flags for the functions of task termination or storage termination within the RTM (recovery termination manager).

Finding the RTM2WA

Multiple RTM2WAs are listed in order, beginning with the most recent failure. To find the unformatted RTM2WA control block or the RTM2WA summary (near its unformatted source), follow these steps:

1. Locate the TCB summary.

The Print Dump Index or Table of Contents contains an entry and page number for "Task Control Block Summary".

2. Find the TCB involved in the failure, looking for a TCB with a nonzero completion code.

Scan the second column of the TCB summary, looking for a nonzero CMP value. Notice the address of the TCB (in the first column) and the page number on which the TCB is located (in the last column). This TCB was involved in the failure.

3. Locate the RTM2WA "eye-catcher" in this TCB. This is the unformatted RTM2WA control block.

From the beginning of the TCB, turn forward a few pages. The eye-catcher is in the left margin.

If you have access to a tool that allows you to examine dumps in machine-readable form, then search for RECORD ID = X'0039' to find the unformatted RTM2WA control block.

In most error situations, a summary of the most important information in the RTM2WA block is formatted and can be found near the unformatted RTM2WA control block in the dump.

The following figures show the formatted and unformatted versions of the RTM2WA.

```
+1C COMPLETION CODE          0004E000
+8C ABENDING PROGRAM NAME   N/A
+94 ABENDING PROGRAM ADDR   00000000
+3C REGS AT TIME OF ERROR   00000000 0004E000 001957A0 005A1DDC 009B5CE0 005A1C2C 009B8C78 005A9BA8 (0-7)
+5C                          0099F550 0025B15A 005A1F58 4025AF24 005A1F58 4025B168 00C90001 (8-F)
+7C EC PSW AT TIME OF ERROR 0025B176 0002000D 0024F5A0
+DC SDWACOMP                 00000000
+E8 RETURN CODE FROM RECOVERY ROUTINE-00,CONTINUE WITH TERMINATION-IMPLIES PERCOLATION
+E0 RETRY ADDR RETURNED FROM RECOVERY EXIT 009BE2A0
+E4                          00000000
+C  CVT  ADDR                00035768
+38  RTCT ADDR               00FDAE78
+C8  SCB  ADDR               005FC7D0
+D4  SDWA ADDR               001477B8
+14  SVRB ADDR               005FD158
+16C PREV RTM2WA FOR THE TASK 00000000
+170 PREV RTM2WA FOR RECURSION 00000000
+B8  ASID OF ERROR IF CROSS MEMORY ABTERM 0000
+36C ERROR ASID              000E
+37C CURRENT TRACE ENTRY FOR SAVED TRACE TABLE 00000000
+380 FIRST TRACE ENTRY FOR SAVED TRACE TABLE 00000000
+384 LAST TRACE ENTRY FOR SAVED TRACE TABLE 00000000
```

Figure 46. Sample formatted RTM2WA summary (RTM1)

```

-RTM2WA-----AT LOCATION 005BF9A0
-----
+0    D9E3D4F2 005BF9A0 FF0004D4 00035768 005CAEA8 005FD158 00FC85F0 0004E000
+20   80808001 005FC7E8 80000000 00000000 005CAEA8 005FD158 00FDAE78
-EED TYPE1 REGS AND PSW
+3C   00000000 0004E000 001957A0 005A1DDC 009B5CE0 005A1C2C 009B8C78 005A9BA8
+5C   00000010 0099F550 0025B15A 005A1F58 4025AF24 005A1F58 4025B168 00C90001
+7C   077C2000 0025B176 0002000D 0024F5A0
-----
+8C   005FD040 00000000 00000000
-EED TYPE 3 MACHINE CHECK
+98   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
-----
+B4   04041041 00001000 00000000 005CA058 00000000 005FC7D0 005ED820 00000000
+D4   001477B8 FA000418 00000000 009BE2A0 00000000 00000000 FF74000D 6025B176
+F4   FF8500DF A07DF74A 00000000
-SNPPARMS
+100  00000000 00000000 00000000 00000000 005BFD84

```

Figure 47. Sample unformatted RTM2WA control block (RTM2)

Related concepts

The task control block (TCB) summary

When included in a dump, the TCB summary identifies the address spaces and their associated tasks.

The failing execution block (EB)

It is recommended that you use the save area trace report to find the failing EB and associated agent EBs.

The failing execution block (EB) can help in locating the cursor table (CT), stack storage, and other pertinent information. Knowing the address of the failing EB can also help you interpret the Db2 trace table.

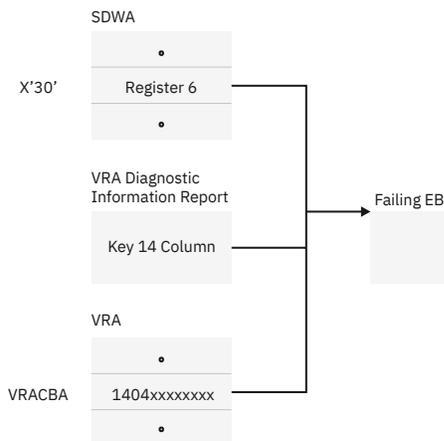


Figure 48. Finding the failing EB

If you are not able to use the save area trace report to find the failing EB, use [Figure 48 on page 207](#) to locate the address of the failing EB:

- SDWA field at offset X'30', which contains register 6.
- Key 14 column of the "VRA Diagnostic Information Report", which appears at the front of the formatted section of the SVC dump.
- VRACBA field (Key 14) in the variable recording area (VRA) of the SDWA. VRA entries begin at offset X'190' in the SDWA.
- SUMLSTA range in the unformatted section of the SVC dump.

Related concepts

Using global trace output

Trace output is based on the parameters specified for the -START TRACE(GLOBAL) command. Each record identifies one or more significant Db2 events.

The system diagnostic work area (SDWA)

Each SVC dump that is requested by a Db2 functional recovery routine usually contains an SDWA with information about the status of the subsystem at the time of the error. Typically, the SDWA is a starting point for diagnosis.

The variable recording area (VRA)

More diagnostic information for Db2 abend reason codes is placed in the variable recording area (VRA) of the system diagnostic work area (SDWA) and is extracted and displayed in the VRA Diagnostic Information Report. This data can be produced by common recording routines and certain Db2 subcomponents.

The save area trace report

The save area trace report is displayed in the first few pages of the formatted section of an SVC dump, immediately following the VRA diagnostic information report. This report is identified by the string ==Save Area trace.

The following sample shows a save area trace report.

The information in this report includes the register contents, module invocation sequence, and execution environments that lead up to the point of error. The save areas for the current failing agent execution block (EB) and all associated agent EBs are traced from the point of error and are displayed in order of invocation.

The address and content of each save area is displayed and identified by the name of the invoking module, as follows:

```
Save Area:  module_name  service_level_identifier
           .... WD1 ... HSA ... LSA ...
           RET ... EPA ... R0 ...
           R1 ... R2 ... R3 ...
           R4 ... R5 ... R6 ...
           R7 ... R8 ... R9 ...
           R10 ... R11 ... R12 ...
```

The *module_name* identifies the module responsible for obtaining the save area as reflected in the module entry point list (MEPL) at the time of error. The *service_level_identifier* is also obtained from the corresponding MEPL entry, and consists of the compilation date and the PTF number, which reflects the latest maintenance applied. As a first step in diagnosing errors, be sure that the PTF number is up-to-date. If no maintenance has been applied against this module, this field defaults to the function modification identifier (FMID). The SA identifies the address and contents of each word of the save area and is displayed by using the same format and register abbreviation conventions as defined by z/OS SNAP. All save area data references reflect the primary address space at time of execution.

The report displays the current execution environment each time the current agent EB changed during processing. This information consists of the address of the associated EB, the z/OS job name that is associated with the home address space, the home address space identifier (HASID), the primary address space identifier (PASID), the agent EB z/OS execution mode (task control mode (TCB), or service request mode (SRB)).

The report also displays the contents of the register save area chains for the current failing agent EB, and all suspended agent EBs at time or error. The name of the control block from which the registers for the current agent EB were obtained is displayed, along with the current execution status of the agent EB at time of error. The control block and registers might be displayed twice; once to list the registers at the time the agent was suspended, and once to list the registers in the save area at the last time they were saved.

All register data references refer to the primary address space (PASID) associated with the currently active EB. Each time the primary address space changes during execution, the following information is displayed:

```
CHANGE of ADDRESS SPACE, NEW PASID=pasid
```

The save area trace can be terminated when a control block or data page is not found in the dump.

```
--Save Area Trace
*****
Note: Save Area may be broken at 00000000,7EDC390, starting format right after this point
Save Area: 00000000,7E1969D2 14 21 + 012C EPA: 029A1380 0512103 2a 15 + 0000
00000000,7EDC450 M01 00000000,7EDC114 HSA 00000000,7EDC390 LSA 00000000,7EDC808
R01 00000000,00000000 EPA 00000000,00000000 R02 00000000,7EDC808
R3 00000000,00000000 R4 00000000,00000000 R5 00000000,00000000
R6 00000000,00000000 R7 00000000,00000000 R8 00000000,00000000
R9 00000000,00000000 R10 00000000,00000000 R11 00000000,00000000
R12 00000000,00000000
Extension: EB 00000000,0A8B820 CT 00000000,74C2730 MS1B 00000000,00000000 00000
0000000,00000000 .....
Service: 7EDC340 7EDC078 00000000 0A6A4F8 | ..E.....?..|
Save Area: 029A1380 0512103 16 15 + 0140 EPA: 029A1380 0312103 15 27 + 0000
Note: This save area may be broken at 00000000,7EDC390
00000000,7EDC390 M01 00000000,00000000 HSA 00000000,7EDC450 LSA 00000000,7EDC808
R01 00000000,00000000 EPA 00000000,00000000 R02 00000000,00000000
R3 00000000,00000000 R4 00000000,00000000 R5 00000000,00000000
R6 00000000,00000000 R7 00000000,00000000 R8 00000000,00000000
R9 00000000,00000000 R10 00000000,00000000 R11 00000000,00000000
R12 00000000,00000000
Extension: EB 00000000,0A8B820 CT 00000000,74C2730 MS1B 00000000,00000000 00000
0000000,00000000 .....
Service: 00000000 00000000 00000000 00000000 7EDC248 | .....K..|
```

Figure 49. Save area trace report

The module entry point list (MEPL)

The Db2 module entry point list (MEPL) is found in all SVC dumps that are issued by Db2. It identifies the names of the load modules and CSECTs that are loaded into the subsystem at startup and remain in storage for the life of the subsystem.

With the DISPLAY parameter of the DIAGNOSE utility you can, without forcing an SVC dump, dump either the Db2 MEPL or the DSNUTILB MEPL to SYSPRINT.

Contents of the MEPL

Header

One 32-byte entry per MEPL, containing:

- 2-byte hexadecimal control block identifier (X'00BA')
- 2 bytes of reserved space
- 4-byte EBCDIC control block identifier and "eye-catcher" (MEPL)
- 2-byte value for the total number of load module entries in the control block
- 2-byte value for the total number of entries (both load module and CSECT) in the control block
- 4-byte value for the length of the control block
- 16 bytes of reserved space

Load Module

One 32-byte entry for each load module in the MEPL, containing:

- 8-byte load module name
- 4-byte load module that starts address (if the high-order bit is 1, the load module runs in 31 bit addressing mode under z/OS)
- 4-byte load module ending address
- 2-byte load module storage residency indicated by one of the following items:
 - Address space identifier (ASID) if the load module is stored in local (private) storage
 - Zero (0000) if the load module is stored in global (common system area)
- 2-byte value for the total number of CSECT entries in the load module
- 4-byte program call (PC) linkage index (LX) value that is used to start a CSECT in this load module or zero (00000000) if PC is not used
- 8 bytes of reserved space

CSECT

One 32-byte entry for each CSECT in a load module in the MEPL, containing:

- 4-byte CSECT entry point address
- 8-byte CSECT name
- 8-byte CSECT assembly date
- 8-byte number of last APAR applied to this CSECT
- 1-byte program call (PC) entry table index (EX) value that is used to start this entry point or zero (00) if PC is not used
- 1-byte initialization entry point list (IEPL) flag bits:
 - X'80': Entry point that is started by PC linkage
 - X'40': PC with space switch (dual address space) permitted
 - X'20': Entry point that is started from application process environment
 - X'10': Entry point that is started by CALL linkage
 - X'08': Primary CSECT entry point
 - X'04': Reserved (unused)
 - X'02': Reserved (unused)
 - X'01': Reserved (unused)
- 2 bytes of reserved storage

Related reference

[DIAGNOSE \(Db2 Utilities\)](#)

Finding the MEPL in the SVC dump

The MEPL can be found in a formatted section of the SVC dump or in an unformatted section of the SVC dump.

About this task

To locate the MEPL in the formatted section of the SVC dump, scan each page, looking for the "MEPL" label at right or left. A formatted MEPL is located near the beginning of the section of the dump that is formatted by Db2.

Procedure

To locate the MEPL in the unformatted section of the SVC dump:

1. Referring to the following example, locate the EBHASCE (at offset X'24') or the EBPASCE field (at offset X'28') in the failing EB. Both fields contain the address of the ASCE (address space control element).
2. Locate the ASCESCOM field (at offset X'0C') in the ASCE. This field contains the address of the SCOM (subsystem communications block).
3. Locate the SCOMMEPL field (at offset X'94') in the SCOM. This field contains the address of the MEPL.

Example

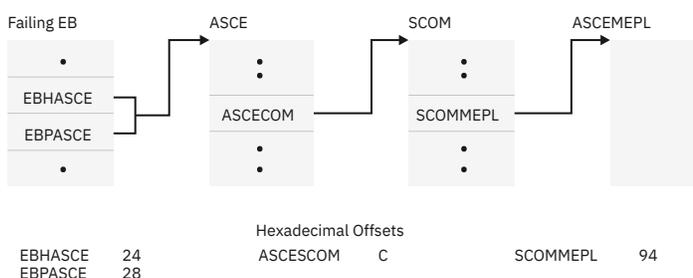


Figure 50. Finding the MEPL in the unformatted section of an SVC dump

Related concepts

[The failing execution block \(EB\)](#)

It is recommended that you use the save area trace report to find the failing EB and associated agent EBs.

Finding the MEPL in the IFCID trace

IFCID 186 is intended to be used only under the direction of IBM Support personnel.

Procedure

Specify IFCID 186 in the IFCID() parameter of the -START TRACE command.

The -START TRACE command writes the MEPL to a series of IFCID 186 records. Each record contains a 4096-byte section of the MEPL, except for the last. The number of records is equal to (Length of MEPL / 4096) + 1.

The following fields in IFCID 186 contain important information:

QW0186SC

The number that indicates which section of the MEPL is in this record.

QW0186TS

The total number of IFCID 186 records that are required to display the complete MEPL.

QW0186LN

The length of the MEPL data in field QW0186MP + 2 for this field. This number is 4098, unless this number is the last record. The length of the last record is (Length of MEPL - (4096 * (QW0186TS - 1))) + 2.

If the DEST() parameter is not specified for the -START TRACE command, IFCID 186 is written to the default destination for the type of trace specified. For example, if a performance trace is specified, IFCID 186 is written to the z/OS generalized trace facility (GTF). For accounting, statistics, and audit traces, the default destination is the z/OS System Management Facilities (SMF).

What to do next

For more information about IFCID 186, refer to *prefix.SDSNIVPD(DSNWMSG)*.

Finding the name of the failing load module in a MEPL

The name of a failing load module can be found in a MEPL.

Procedure

To find the name of the failing load module in a MEPL:

1. Locate the MEPL in the dump.
2. Locate the appropriate PSW and use the address in its second fullword as an index into the list.
Use the PSW and ASID listed in the PRINT DUMP ABSTRACT INFORMATION section, which is located just before the system diagnostic work area (SDWA). Refer to the data listed under INFORMATION AT TIME OF ERROR.
3. Review the beginning and ending addresses for each load module; the beginning address is the third word, and the ending address is the fourth word of each entry.
4. If the address in the PSW falls between the beginning and ending addresses of a particular load module, that can be the failing load module.

In the following example, assume that the address in the PSW at failure is 005A144A. Looking in the MEPL, you find that load module DSNVGEPL begins at 005A0A20 and ends at 005AB000. Because 005A144A falls between 005A0A20 and 005AB000, the name of the load module that contains the failing CSECT is DSNVGEPL.

```
09A0 00145830 C4E2D5E5 C4D9D940 F0F661F0 F/ 000 *.DSNVDRR.  
09C0 C4E2D5E5 C7C5D7D3 005A0A20 005AB000/ 00000 *.DSNVGEPL  
09E0 005A0A20 C4E2D5E5 C1E2C9D4 F0F661F/0080000 *.DSNVASIM  
0A00 005A1268 C4E2D5E5 E3D9E3C8 F0F66/ 00080000 *.DSNVTRTH  
0A20 005A1748 C4E2D5E5 C5E4E2F1 F0F/40 00080000 *.DSNVVEUS1
```

Figure 51. Finding the name of the failing load module in a MEPL

- Verify that the suspected load module was involved in the failure.

Review the ASID field in the MEPL. This field is the first halfword immediately following the ending address of the load module.

The load module is the one that was involved in the failure if one of these conditions is true:

- ASID field = 0000
- ASID field = xxxx, where xxxx is the ASID identified in the dump as primary and is the ASID of system services or database services.

For example, if the primary address space identifier (the identifier that follows the "eye-catcher" PASID in the beginning of the dump) is 0012, the ASID field of the MEPL for the load module that is involved in the failure must either be 0000 or 0012.

- If the suspected load module was not involved in the failure, continue searching the MEPL until a load module is found that contains the address in the PSW and whose ASID is equal to 0000 or to the ASID of the dump.

If the failure is not a loop, the name of the failing load module should begin with DSN. If it begins with anything else, the problem might be in another product. Use the z/OS diagnostic techniques publication for help in diagnosing the failure.

- If the search is unsuccessful, the name of the load module that is involved cannot be determined from the MEPL.

Example

```
MEPL (00014) ADDR=7F71A000 ASID=0010 VVV=00 FROM: SCOM (0006) OFFSET
0000 00BA0000 D4C5D7D3 00850594 0000B2A0 00000000 00000000 00000
0020 C4E2D5E8 C1E2C3D7 81D008C8 81D04000 00100004 00000000 00000
0040 81D00950 C4E2D5E8 C1E2E3D7 F0F361F1 F061F8F6 F1F84BF1 F3404
0060 81D010A8 C4E2D5E8 C1E2E3D9 F0F361F1 F061F8F6 F1F84BF1 F4404
0080 81D00950 C4E2D5E8 C5C3E3C5 F0F361F1 F061F8F6 F1F84BF2 F0404
00A0 81D02AE0 C4E2D5E8 C1E2C3D7 F0F261F1 F061F8F7 F1F64BF0 F5404
00C0 C4E2D5E8 C1C7C3E2 81CB7340 81CBA000 00000001 00000000 00000
00E0 81CB9638 C4E2D5E8 C1C7C3E2 F0F361F1 F061F8F6 F2F24BF4 F6404
0100 C4E2D5E8 C1D3D3C9 81CB7340 81CB9000 00000003 00000000 00000
0120 81CB73C8 C4E2D5E8 C5C1E3C5 F0F161F1 F961F8F7 F1F14BF4 F2404
0140 81CB73C8 C4E2D5E8 C5C1E3F2 F0F361F1 F061F8F6 F1F84BF1 F7404
0160 81CB8500 C4E2D5E8 C1D3D3C9 F0F161F1 F961F8F7 F1F84BF3 F7404
:
```

Figure 52. Sample MEPL (module entry point list)

Finding the name of the failing CSECT in the MEPL

Use this procedure only if the name of the load module that failed in the MEPL was found. If it is not, the CSECT name cannot be located either.

Procedure

To find the name of the failing CSECT in the MEPL:

- Locate the list of CSECTs that are contained in the load module. This list follows the load module in the dump.
- Locate the CSECT entry point address that is closest to the address in the PSW. This CSECT is the failing CSECT.

The first word of each CSECT entry contains its entry point address. Scan the column that contains the CSECT entry point addresses to find the appropriate one.

Assume the address that is shown in the PSW is 005A144A. This falls within the range of load module DSNVGEPL, whose entry point address is 005A0A20 and whose ending address is 005AB000. To determine the CSECT name, scan the first column for the entry point addresses of the CSECTs contained within that load module. The second CSECT in the load module DSNVGEPL has the last entry point address less than the address in the PSW. The CSECT name is DSNVTRTH.

- Message DSNU016I and load module DSNUTILB indicate that execution that is stopped in batch memory.
- Message DSNU017I and load module DSNUTILA indicate that execution that is stopped in database services memory.

The name of the failing load module is published in the dump title.

2. Determine whether the execution stopped in batch memory or database services memory.

- If execution stopped in batch memory:
 - Referring to the following figure, locate register 6 at offset X'30' in the SDWA. This location contains the address of the UJB (utilities job block).
 - Locate the UJBBEPL field at offset X'1C0'. This field contains the address of the batch utilities MEPL.

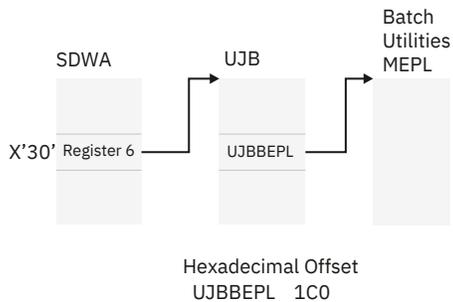


Figure 54. Finding the batch utilities MEPL (batch memory failures)

- If execution stopped in database services memory:
 - Referring to the following figure, locate the CTUTP field at offset X'38' in the CT (cursor table). CTUTP contains the address of the UCA (utility block).
 - Locate the UCAUJB field at offset X'24' in the UCA. This field contains the address of the UJB (utility job block) in batch memory.
 - Locate the UJBBEPL field at offset X'1C0' in the UJB. This field contains the address of the batch utilities MEPL.

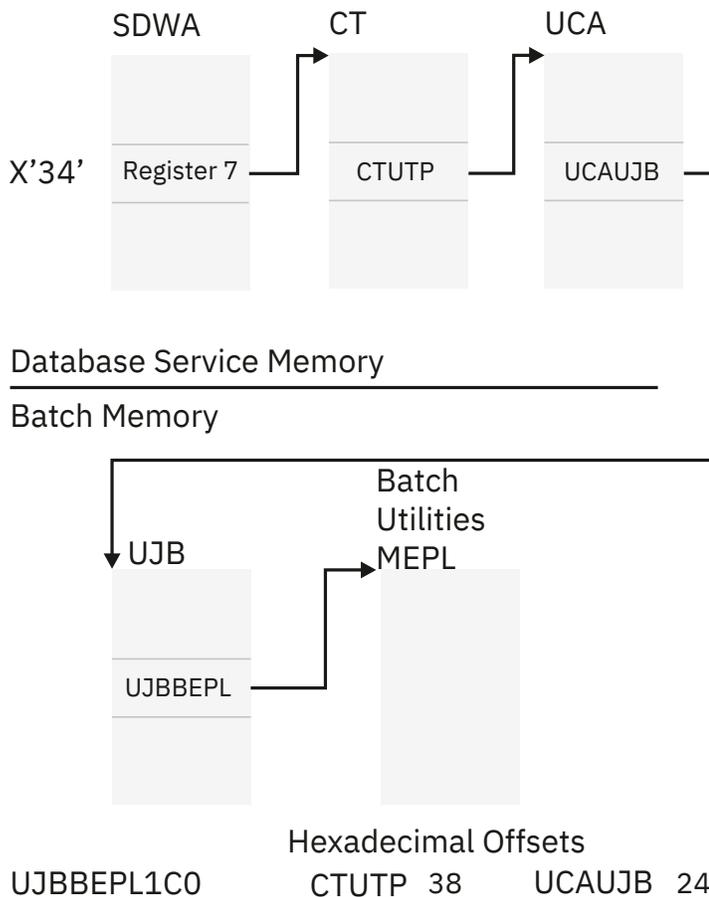


Figure 55. Finding the batch utilities MEPL (database service memory failures)

Related concepts

The cursor table (CT)

The cursor table (CT) is one of the most important blocks. It contains information about an application plan and the access that is being made through it to databases.

The cursor table (CT)

The cursor table (CT) is one of the most important blocks. It contains information about an application plan and the access that is being made through it to databases.

The procedures listed here cite methods for locating the CT in both the unformatted and the formatted sections of the dump.

Finding the CT in the unformatted section of the SVC dump

You might need to find the CT in an unformatted section of the SVC dump.

Procedure

To find the CT in the unformatted section of the SVC dump:

1. Locate register 13 at offset X'4C' from the beginning of the SDWA and examine the data at offset X'F8' from the address in register 13. If the 2 bytes at that address contain the characters, CT, you have found the CT.

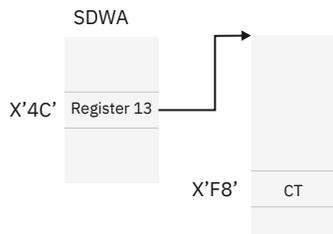


Figure 56. Finding the CT in the unformatted section of an SVC dump

2. Find the SUMLSTA range that contains the CT address.
3. Within this area, locate the beginning of the CT. The CT eye-catcher is displayed at the right.

Related concepts

The system diagnostic work area (SDWA)

Each SVC dump that is requested by a Db2 functional recovery routine usually contains an SDWA with information about the status of the subsystem at the time of the error. Typically, the SDWA is a starting point for diagnosis.

Finding the CT in the formatted section of the SVC dump

You might need to find the CT in a formatted section of the SVC dump.

Procedure

To find the CT in the formatted section of the SVC dump, choose one of the following approaches:

- Skim through the dump, looking for the "CT" eye-catcher on either the right or left side.
- Use a more precise method.
 - a) Determine the address of the failing execution block (EB).
 - b) Locate the EB in the formatted area of the dump. This EB is associated with other blocks related to the same agent by an z/OS-assigned control block number.
 - c) Use the z/OS control block number to find the associated agent control element (ACE). The ACECT field of the associated ACE contains the address of the cursor table.

Related concepts

The failing execution block (EB)

It is recommended that you use the save area trace report to find the failing EB and associated agent EBs.

The SQL communication area (SQLCA)

The SQL communication area is one of the most important blocks. It contains information about the status of one SQL statement. For diagnostic purposes, up to 4 KB of the SQL statement are included in both the formatted and unformatted sections of SVC dumps.

The SQLCA also contains the SQLSTATE, which is a five character return code for the outcome of the most recent execution of an SQL statement. The range of values is '00000' through '65535'. SQLSTATE provides application programs with common codes for common error conditions (the values of SQLSTATE are product-specific only if the error or warning is product-specific). Furthermore, SQLSTATE is designed so that application programs can test for specific errors or classes of errors. The coding scheme is the same for all database managers and is consistent with the proposed ISO/ANSI SQL92 standard.

Related reference

SQLSTATE (Db2 SQL)

Description of SQLCA fields (Db2 SQL)

Finding the SQLCA

The SQLCA is especially valuable for diagnosing abends with a completion code of X'04E' and an abend reason code that begins with X'00E7'.

About this task

When abends occur, the SQLERRP field contains the name of the CSECT for the code that issued the abend. The SQLERRD1 field contains an internal "subcode" that uniquely identifies the location within the CSECT where the error was detected. The SQLERRM field contains an optional error description.

Procedure

To find the SQLCA:

1. Locate the CTRDSP field at offset X'30' in the CT. This field contains the address of the RDA (relational data area).

If this field contains a fullword of zero, there is no SQLCA associated with this dump. Do not continue with this procedure.

2. Find the RDASQLCA field at offset X'158' in the RDA. This field contains the address of the SQLCA. At that address, the "SQLCA" eye-catcher is displayed.

Results

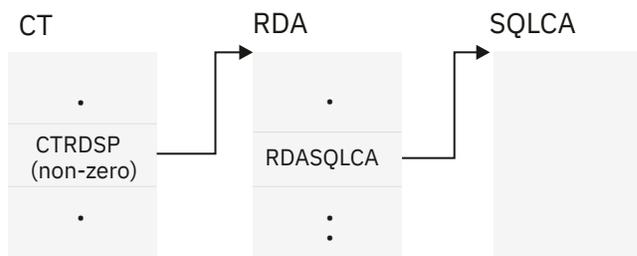


Figure 57. Finding the SQLCA

The SQLCA is especially valuable for abends with a completion code of X'04E' and an abend reason code that begins with X'00E7' (for example, 00E70005).

Sample SQLCA control blocks

The following figure shows a portion of an SQLCA in the formatted section of an SVC dump.

```

SQLC (00675) ADDR=7F6446A0 ASID=0016 VVV=00 FROM: RDA (00673) OFFSET=0000
0000 E2D8D3C3 C1404040 00000088 00000000 00004040 40404040 40404040 40404040
0020 40404040 40404040 40404040 40404040 40404040 40404040 40404040 *SQLCA ..... *
      0040 TO 005F SAME AS ABOVE
0060 00000000 00000000 00000000 FFFFFFFF 00000000 00000000 40404040 40404040 *..... *
0080 00000000 00000000
  
```

Figure 58. Sample SQLCA control block (formatted)

The following figure shows this portion of the SQLCA as it is displayed in the unformatted section of the SVC dump. Notice the eye-catcher, SQLCA.

```

7F6446A0 E2D8D3C3 C1404040 00000088 00000000 00004040 40404040 40404040 40404040
7F6446C0 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040
***** LINES FROM 7F6446E0 TO 7F644700 ARE THE SAME AS THE ABOVE LINE
7F644700 00000000 00000000 00000000 FFFFFFFF 00000000 00000000 40404040 40404040
7F644720 00000000 00000000 00280400 001EC4E2 D5E3C5D7 F2F2137E A1681D9C 3E200001
  
```

Figure 59. Sample SQLCA control block (unformatted)

Related concepts

[The cursor table \(CT\)](#)

The cursor table (CT) is one of the most important blocks. It contains information about an application plan and the access that is being made through it to databases.

Related reference

[Description of SQLCA fields \(Db2 SQL\)](#)

Finding the SQL statement

Up to 4 KB of the SQL statement are included in the sections of the SVC dump that pertain to SQL problems. To find the SQL statement in the formatted section of the SVC dump, skim through the pages for the statement on the right. It is displayed in the space block, which contains the "SPA" eye-catcher.

About this task

There are occasions when Db2 might not load the current space block (SPA). In this case, RDASPPT1 points to the previously run SQL statement. This situation causes the RDASPPT1 to point to the previously run SQL statement.

Procedure

To locate the SQL statement in the unformatted section of the SVC dump:

1. Locate the CTRDSP field at offset X'1C0' in the CT. This field contains the address of the RDA (relational data area).
2. Locate the RDASPPT1 field at offset X'108' in the RDA. This field contains the address of the SPA (space block). The SPA begins with a header section, followed by several space block entries.
3. Locate the SPASQLTX field at offset X'88' in the header section of the SPA. This field is a pointer to the SQL statement within the first space block entry.
4. You can scan the right of the dump to locate the SQL statement. Otherwise, use the contents of SPASQLTX to pinpoint a 2-byte length field, which precedes the SQL statement.

Example

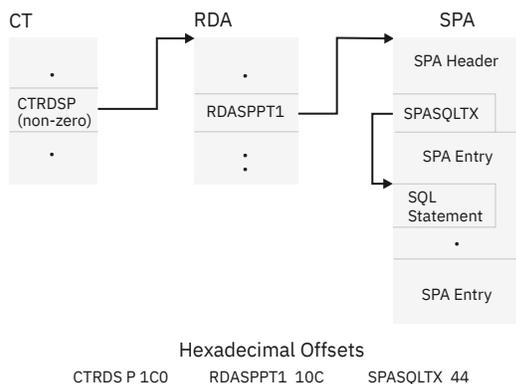


Figure 60. Finding the SQL statement

Related concepts

[The cursor table \(CT\)](#)

The cursor table (CT) is one of the most important blocks. It contains information about an application plan and the access that is being made through it to databases.

Related reference

[SQLSTATE \(Db2 SQL\)](#)

The global trace table

The Db2 trace table contains entries for significant Db2 activities and is useful for diagnosing WAIT or LOOP problems.

The formatted Db2 trace table usually appears at the end of the formatted section of an SVC dump. An unformatted table also appears in the unformatted section of the dump.

Related concepts

[Using global trace output](#)

Trace output is based on the parameters specified for the -START TRACE(GLOBAL) command. Each record identifies one or more significant Db2 events.

Formatted global trace table

Global trace data sent to the resident trace table usually appears at the end of the formatted section of an SVC dump that is issued by Db2.

To format the resident trace table, the DSNWDMP control card parameters must be either TT or ALL. The following figure illustrates a sample formatted trace table.

```
====Global trace
EB 06088628 RET 0052-8787240A DSNIDRCT 15.46 +01B2 EID 0E-0792 FUNC 00 ASID 0052 05/30/2003 06:59:58.152746
DATA 00000000 7EFC9A0 00000000 00000001
EB 06088628 RET 0052-8784798E DSNIDBRP 15.46 +013E EID 0E-05CC FUNC 00 ASID 0052 05/30/2003 06:59:58.152747
DATA 00000000 00000000
EB 06088628 RET 0052-866D4F28 DSNVSK 15.22 +19E8 EID 06-001C FUNC 4C ASID 0052 05/30/2003 06:59:58.152750
DATA 00000000 00000000 00000000 C4C2D907 00000000 7F4DA928 00000000 00000000
EB 06088628 RET 0052-878475F0 DSNIDBRP 15.46 +05A0 EID 0E-05CD FUNC 00 ASID 0052 05/30/2003 06:59:58.152752
DATA 00000000 00000000 00000000 7F42EC90
EB 06088628 RET 0052-8784798E DSNIDBRP 15.46 +013E EID 0E-05CC FUNC 00 ASID 0052 05/30/2003 06:59:58.152753
DATA 00000000 00000000
```

Figure 61. Example of a formatted global trace table

Related reference

[Format dumps by using the DSNWDMP statement](#)

You can use the DSNWDMP statement to specify the dump records to be used as input, and causes the Db2 dump formatter (DSNWDPRD) to be invoked, which formats the specified Db2 control blocks.

Finding the unformatted global trace table

Global trace data sent to the resident trace table is always included in the non-summary dump data set of SVC dumps that are issued by Db2.

About this task

The following information describes how to locate this unformatted table by using control blocks that are published in the unformatted summary dump section.

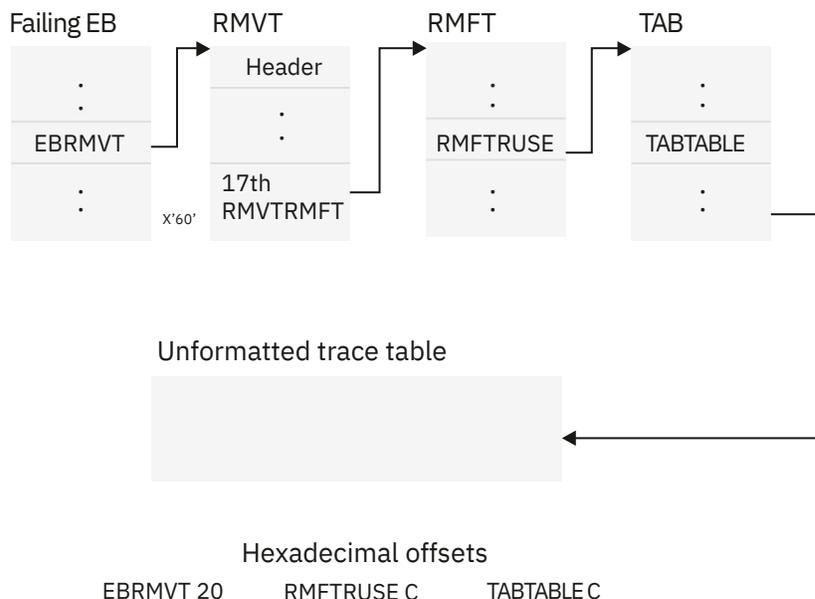


Figure 62. Finding the unformatted trace table

Procedure

To find the unformatted global trace table:

1. Locate the EBRMVT field at offset X'20' in the failing EB. This field contains the address of the RMVT.
The failing EB's address is at offset X'30' in the SDWA and is listed under Key 14 of the VRA Diagnostic Information Report.
2. Find the address of the RMFT (resource manager function table) at offset X'60' in the RMVT. The 17th RMVTRMFT element in the RMVT is at this offset.
3. Locate the RMFTRUSE field at offset X'0C' in the RMFT. This field contains the address of the trace anchor block (TAB). If the high-order bit of this word is on, then the trace is active.
4. Locate the TAB and these fields

TABTABLE

At offset X'0C' in the TAB and contains the address of the beginning of the trace table.

TABOTTOM

At offset X'14' in the TAB and contains the address of the end of the trace table.

TABSLLOT

At offset X'18' in the TAB contains the address of the oldest (next available) entry in the trace table. This address changes as the table wraps.

Results

Table 12. Global trace table header format

| Length in bytes | Trace table header field descriptions |
|-----------------|---------------------------------------|
| 12 | Eye-catcher 'TRACE TABLE' |
| 1 | Reserved |
| 8 | Command prefix |
| 1 | Reserved |
| 4 | Db2 subsystem name |
| 2 | Db2 system services ASID |

Table 12. Global trace table header format (continued)

| Length in bytes | Trace table header field descriptions |
|-----------------|---|
| 4 | Address of the Db2 TAB control block |
| 4 | Address of the Db2 RMFT control block for trace |
| 4 | Address of the Db2 SCOM control block |

Related concepts

The failing execution block (EB)

It is recommended that you use the save area trace report to find the failing EB and associated agent EBs.

The task control block (TCB) summary

When included in a dump, the TCB summary identifies the address spaces and their associated tasks.

By looking at the column that contains the completion codes (CMP) for each task, those columns with abends can easily be found because they have nonzero completion codes. A nonzero completion code might represent a problem from which Db2 has recovered.

Finding the TCB summary

The Table of Contents cites the location of the TCB summary under its entry on "Task Control Block Summary".

Stack storage blocks (SKBs)

SKBs consist of a fixed area that contains pointers to more storage segments.

The first 2 bytes of the fixed portion of all SKBs contain X'00AD'. In addition to the fixed portion, all SKBs contain a variable portion that is used differently by each Db2 resource manager. The contents of the variable area are not documented externally.

Finding the stack storage blocks

You might need to locate all stack storage blocks (SKBs) that are associated with the failing agent in any SVC dump that is issued by Db2, including the SKBs that are associated with all synchronously-created execution blocks (EBs) related to the failing agent.

About this task

Use this information at the request of IBM Support staff when you analyze the unformatted section of an SVC dump. Stack storage usually appears toward the end of the formatted section of the dump.

The following figure displays how to locate the stack storage for the failing agent.

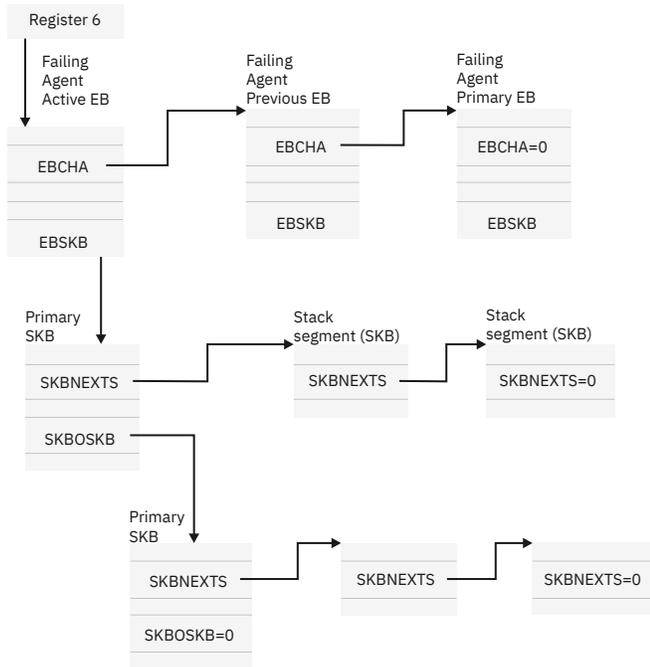


Figure 63. Locating stack storage for the failing agent

Procedure

To find the stack storage blocks:

1. Locate the EBSKB field in the failing EB. This field points to the primary SKB for that EB.
2. Determine the address space in which this primary SKB resides.
 - Use the EBPASCE field in the EB to obtain the address of the ASCE.
 - Find the address space in the ASCEASID field of the ASCE. This address space is the address space of the SKB.

In the SKB, two fields contain important addresses:

SKBNEXTS

This field contains the address of the next SKB within this stack. Multiple SKBs can be associated with this primary SKB. Each of these stack segments is considered part of this primary SKB.

When SKBNEXTS equals zero, all of the stack segments for that primary SKB are located.

SKBOSKB

This field contains the address of the previous primary SKB, which was in a previous address space (identified in the field named SKBPASCE). To determine which address space the previous primary SKB was in, see the SKBPASCE field in this SKB; this field points to the ASCE. The field in the ASCE named ASCEASID identifies the address space that contains the SKB.

Although this SKB is a different primary SKB than the one pointed to directly by the EB, it is still associated with that EB.

There can be a chain of primary SKBs associated with one EB. For each primary SKB in that chain, there can be a chain of stack segments (SKBs). When SKBOSKB equals zero, the end of the chain of primary SKBs associated with that EB is located.

3. After you locate all the stack storage for the failing EB, locate the stack storage for all other EBs related to the failing agent.

Review the EBCHA field in the failing EB. This field contains the address of another related EB (and likewise, that EB's EBCHA field points to another related EB). These other EBs also have chains of primary SKBs (and possibly of stack segments), as did the failing EB. When you locate an EB whose EBCHA field equals zero, all of the stack storage for all the EBs related to the failing agent is located.

Results

If the primary address space of an associated EB was not included in the SVC dump, the SKBs for that EB are not found in the dump.

Related concepts

[The failing execution block \(EB\)](#)

It is recommended that you use the save area trace report to find the failing EB and associated agent EBs.

Redacting buffer pool data in SVC dumps for data privacy

Db2 marks buffer pool data in SVC dumps as `sensitive=yes` for data privacy, and you can redact the data before sending the dump to IBM Support.

Before you begin

The dump must be captured on a IBM z15[®] or later processor.

About this task

Enterprises have requirements to prevent customer personal or other sensitive information from being exposed to those who have no need to see the data. In the course of data processing, various types of system and application errors can require you to send diagnostic data to IBM Support or other program vendors, for analysis and problem resolution. Among the types of diagnostic data usually collected for Db2 for z/OS, buffer pool data in dumps has the greatest exposure to containing sensitive data along with the system and or application data.

Procedure

To redact buffer pool data in SVC dumps for data privacy, complete the following steps:

1. Use the sample job `SYS1.SAMPLIB(BLSJDPFD)` to redact the buffer pool data.
2. Retain the original dump until the conclusion of all problem analysis, in case IBM Support requests specific information from the redacted data.

What to do next

To obtain a report about the pages which were marked as sensitive in a redacted dump, use `'SYS1.SBLSCLI0(BLSXREDR)'`, provide an input dump data set name, and optionally specify a filtering ASID.

Related information

[Data Privacy for Diagnostics \(DPfD\) \(MVS Diagnosis: Tools and Service Aids\)](#)

[OA57633: IN SUPPORT OF OA57570](#)

Suppression of SVC dumps by using z/OS DAE

SVC dumps that duplicate previous dumps can be suppressed.

One of the requirements for z/OS dump analysis and elimination (DAE) is that a new data set is defined.

To support DAE, Db2 defines two variable recording area (VRA) keys:

- KEY VRADAE (X'53') - no data is associated with this key
- KEY VRAMINSC (X'52') DATA (X'08')

Db2 provides the following data for the minimum symptom string in the system diagnostic work area (SDWA):

- LOAD MODULE NAME
- CSECT NAME

ABEND CODE
RECOVERY ROUTINE NAME
FAILING INST AREA
REG/PSW DIFFERENCE
REASON CODE
COMPONENT IDENTIFIER
COMPONENT SUBFUNCTION

Dumps are considered duplicates for purposes of duplicate dump suppression if 8 (the X'08' from the VRAMINSC key) of the nine symptoms are the same.

Related concepts

[Dump suppression \(MVS Diagnosis: Tools and Service Aids\)](#)

When SVC dumps are not produced

Occasionally, an SVC dump is not produced. Generally, dumps are suppressed for space, time, or security violations.

The following list summarizes other reasons why SVC dumps might not be produced:

- No empty dump data sets were available.
- The z/OS serviceability level indication processing (SLIP) commands suppressed the abends.

The z/OS initialization and tuning publication lists the defaults of these commands; however, sites can change them. More information can be found in the z/OS conversion notebook. Finally, refer to the z/OS diagnostic techniques publication for information about tailoring dumps with SLIP commands.

- One of the following abend reason codes was issued. These reason codes do not require a dump to determine the cause of abend.

```
00C90080 00C90096 00D10100 00E2002B  
00D99003 00E3000C 00E30085 00E30089  
00E30093 00E30301 00E30302 00E50013  
00E50070 00E7000C
```

- Db2 subsystem percolation occurred (SDWACOMU=DSN2).
- A dump was already provided (SDWAEAS=ON).
- The abend was '04F', '222', '33E', 'B37', 'D37', 'E37', or 'x13' (where x is any hexadecimal digit).
- The abend was '13E', and one or more of the following conditions was true:
 - Home address space of the failing agent (EB) was not in allied memory.
 - Termination of the execution unit TCB did not result from the z/OS CANCEL W/DUMP command.
 - Prior dumps were generated for the terminating TCB.
 - The Db2 dump work area (DMPW) is in use by another dump request:

```
VRA KEY VRARRK22 X'DE') DATA ('NODMPW')
```

SYSUDUMP dumps

Db2 SYSUDUMP dumps provide information useful for debugging application programs.

The following figure shows a sample SYSUDUMP dump.

```

JOB H443722      STEP TSUSER      TIME 102958  DATE 87152  ID = 000  CPUID = 632202333081
COMPLETION CODE  SYSTEM = 04E
PSW AT ENTRY TO ABEND 078D1000 000433FC      ILC 2  INTC 000D
PSW LOAD MODULE = DSNECP10  ADDRESS = 000433FC  OFFSET = 0000A7F4

ASCB: 00F56400
+0000 ASCB..... ASCB      FNDP..... 00F60180  BWDP..... 00F47800  CMSF..... 019D5A30  SVRB..... 008FE9E0
+0014 SYNC..... 00000D6F  IOSP..... 00000000  TNEW..... 008D18F0  CPUS..... 00000001  ASID..... 0066
+0026 R02B..... 0000      LLS..... 00      HLIH..... 01      DPHI..... 00      DP..... 9D
+002C TRQP..... 80F5D381  LDA..... 7F154E8  RSMF..... 00      R035..... 0000      TRQI..... 42
+0038 CSCB..... 00F4D048  TSB..... 00B61938  EJST..... 00000001  8C257E00
+0048 EWST..... 9CCDE747  76A09480  JSTL..... 000141A4  ECB..... 808FEF78  UBET..... 9CCDE740
+005C TLCH..... 00000000  DUMP..... 008FE520  AFFN..... FFFF      RCTF..... 00      FLG1..... 00
+0068 TMCH..... 00000000  ASXB..... 008FF118  SWCT..... 032E      DSP1..... 00      FLG2..... 00
+0074 RSV..... 0000      SRBS..... 0000      R078..... 00000000  RCTP..... 008FF338  LOCK..... 00000000
:
+0084 LSQH..... 00000000  QECB..... 00000000  MECB..... 40000000  OUCB..... 01947E90  OUXB..... 01988CB0
+0098 FMCT..... 008B      LEVL..... 02      R09B..... 00      XMPQ..... 00000000  IQEA..... 00000000
+00A4 RTMC..... 00000000  MCC..... 00000000  JBNJ..... 00000000  JBNS..... 00F4D050  SRQ1..... 00
+00B5 SRQ2..... 00      SRQ3..... 00      SRQ4..... 00      VGTI..... 00000000  PCTT..... 02035078
+00C0 SSRB..... 0000      SMCT..... 00      SRBM..... 07      SWTL..... 00002838
+00C8 SRBT..... 00000000  261ADA00  LSMQ..... 00000000  LSPL..... 00000000  TCBS..... 00000001
+00DC TCBL..... 00000000  WFRB..... 008FFA88  NDP..... 0D      TNDP..... FF      NTSG..... FF
+00E7 IODP..... 9D      LOCT..... 00000000  CMLH..... 00000000  CMLC..... 00000000  SS01..... 000000
+00F7 SS04..... 00      ASTE..... 00F7F660  LTOV..... 7FFF0000  ATOV..... 7FFFE610  ETC..... 0000
+0106 ETCN..... 0000      LXR..... 0000      AXR..... 0000      STKH..... 008FFA98  GQEL..... 038E0CF0
+0114 LQEL..... 02120A50  GSYN..... 00000000  XTCB..... 008CC608  CS1..... 00      R121..... 000000
+0124 GXL..... 00000000  EATT..... 00000000  97024600  INTS..... 9CCDE6A0  4572EE60
+0138 LL1..... 00      LL2..... 00      LL3..... 00      LL4..... 00      RCMS..... 00000000
+0140 IOSC..... 000001A5  PKML..... 0080      XCNT..... 01F4      NSQA..... 00000000  ASM..... 0195B860
+0150 ASSB..... 01946600  TCME..... 00000000  R15B..... 00000000  00000000  00000000
+0168 CREQ..... 0000002F  RSMC..... 0195B840  AVM1..... 00      AVM2..... 00      AGEN..... 0000
+0174 ARC..... 00000000  RSMR..... 0195B750  DCTI..... 00005735  AVTA..... 00000000  00000000
+0188 SACT..... 00000000  00000000  TCPT..... 00000001  9070B600
+0198 SCPT..... 00000000  261ADA00

ASSB: 01946600
+0000 ASSB..... ASSB      VAFN..... 00000000  EVST..... 00000000  00000000  00000000
+0010 VFAI..... 00000000  00000000  RSV..... 0000      XMCC..... 0000      XMCT..... 00000000
+0020 VSC..... 00000000  NVSC..... 0000004C  ASRR..... 00000000  R02C..... 00000000  00000000  00000000
+0038      00000000  00000000

*** ADDRESS SPACE SWITCH EVENT MASK OFF (ASTESSEM = 0) ***

TCB: 008D18F0
+0000 RBP..... 008FE7D8  PTE..... 00000000  DEB..... 008B1530  TIO..... 008D4000  CMP..... 8004E000
+0014 TRN..... 40000000  MSS..... 7FFF7418  PKF..... 80      FLGS..... 01000000  00
+0022 LMP..... FF      DSP..... FE      LLS..... 008D1A88  JLB..... 00011F18  JPQ..... 00000000
+0030 GPRO-3... 00001000  008A4000  00000000  00000000
+0040 GPR4-7... 00FDC730  008A50C8  00000002  80E73F04
+0050 GPR8-11.. 81CC4360  008A6754  008A67B4  00000008

```

Figure 64. Sample of the first page of a SYSUDUMP

Related reference

[SYSABEND, SYSMDUMP, and SYSUDUMP DD Statements \(MVS JCL Reference\)](#)

The RTM2WA summary in a SYSUDUMP

The recovery termination manager work area (RTM2WA) summary usually appears in the first few pages of a SYSUDUMP.

The following sample shows the RTM2WA summary.

```

RTM2WA SUMMARY
-----
COMPLETION CODE          8004E000
ABENDING PROGRAM NAME    DSNECP10
ABENDING PROGRAM ADDR    0003F4C0
REGS AT TIME OF ERROR    80000000 8004E000 000264F4 00031C74 00006FE8 00008BF4 000368B0 00036AD0
(0-7)                    00000000 000264F4 00031B98 600430AC 000440AB 00031B98 700433E0 00C50101
(8-F)
EC PSW AT TIME OF ERROR  078D1000 000433FC 0002000D 00000000
SOWACOMP                 00000000
RETURN CODE FROM RECOVERY ROUTINE-10,CONTINUE WITH TERMINATION-PREVENT FURTHER STAI/ESTAI PROCESSING
RETRY ADDR RETURNED FROM RECOVERY EXIT 80000010
RB ADDR FOR RETRY        00000000
CVT ADDR                 00FDC730
RTCT ADDR               00F71310
SCB                     00000000
SDWA ADDR               00000000
SVRB ADDR               008FE8F0
PREV RTM2WA FOR THE TASK 00000000
PREV RTM2WA FOR RECURSION 00000000
ASID OF ERROR IF CROSS MEMORY ABTERM          0000
ERROR ASID              0066

```

Figure 65. Sample RTM2WA summary from a SYSUDUMP

SYS1.LOGREC

The SYS1.LOGREC data set records various errors that different components of the operating system encounter.

Db2 recovery routines write information in the SDWA (system diagnostic work area) to the SYS1.LOGREC data set when retry is attempted, or when percolation to the next recovery routine occurs. Because two or more retries or percolations can occur for a single error, more than one SYS1.LOGREC entry can be recorded.

In distributed data processing, threads are assigned a logical unit of work identifier. This identifier is used to correlate errors across many instances of Db2 for the same global transaction. The logical unit of work identifier is included in the SYS1.LOGREC error recording entries.

The SYS1.LOGREC entries that are recorded near the time of abend can provide valuable historical information about the events that lead up to the abend.

Certain abends that occur as a result of errors in SQL statements are handled by Db2 recovery routines:

- An error in decimal arithmetic
- Overflow
- Underflow

The value of the SUPPRESS SOFT ERRORS field of installation panel DSNTIPM determines whether those errors are displayed in SYS1.LOGREC. The associated system parameter is SUPERRS. The default value is YES, which means that the abends are recorded in SYS1.LOGREC. If you do not want those abends to be recorded in SYS1.LOGREC, you can change the value to NO.

Finding the applicable SYS1.LOGREC information

To obtain a SYS1.LOGREC listing:

1. Use the IFCEREP1 service aid, described in the z/OS diagnostic techniques publication, to format records in the SYS1.LOGREC data set.
2. Specify the LOGDATA keyword when you print a formatted dump. Only records available in storage when the dump was requested are included. Each formatted record follows the heading *****LOGDATA*****.

Each SVC dump that is issued by Db2 contains one unformatted SDWA describing the subsystem status at the time of the error.

Correlate formatted SYS1.LOGREC records in a dump with the unformatted SDWA in the same dump by comparing the following information:

- Subsystem name. For data sharing, the data sharing group name and member name.
- ASID (address space identifier).
- Subcomponent involved (look at the fourth character in load module and CSECT names).
- Timestamp.

To find a formatted SYS1.LOGREC entry that corresponds to an SVC dump that is issued by Db2

1. Compare the ERRORID of the dump with the ERRORID of the formatted SYS1.LOGREC record. These values, which appear on the first pages of the dump and of the formatted SYS1.LOGREC record, should be the same.
2. Compare the time stamp of the dump and formatted SYS1.LOGREC record. These should also be about the same.

Interpreting SYS1.LOGREC Information

A formatted SYS1.LOGREC entry is fairly self-explanatory.

- The first page of a formatted SYS1.LOGREC entry
- The second page of a SYS1.LOGREC entry, which includes the unformatted SDWA, from which data is obtained and formatted.

The z/OS diagnostic techniques publication also includes information about formatted SYS1.LOGREC entries.

The unformatted section of a SYS1.LOGREC entry contains the SDWA at the time of the failure.

- The first part of the SDWA, which is standard for all components, is formatted in the SYS1.LOGREC record. Refer to the z/OS diagnostic techniques publication for information about the standard part of the SDWA.)
- The entire SDWA including the VRA is in the unformatted area of the SYS1.LOGREC record, following the standard section. (VRA entries begin at X'190' in the unformatted SDWA.)

The following figure shows the first two pages of a SYS1.LOGREC entry.

Figure 66. Sample formatted SYS1.LOGREC record (pages one and two)

```
TYPE:  SOFTWARE RECORD      REPORT:  SOFTWARE EDIT REPORT      DAY.YEAR
      (SVC 13)                REPORT DATE: 036.93
SCP:   VS 2 REL 4            MODEL:    9021                      HH MM SS.TH
                                SERIAL:   125784                   TIME: 11:05:49.50

JOBNAME: T032478
ERRORID: SEQ=00611 CPU=0041 ASID=002D TIME=11:05:37.5
SEARCH ARGUMENT ABSTRACT
PIDS/5740XYR00 RIDS/DSNIDM#L RIDS/DSNIMOST AB/S004E PRCS/00C90101 REGS/0C6EE
RIDS/DSNTFRCV#R
SYMPTOM          DESCRIPTION
-----          -
PIDS/5740XYR00   PROGRAM ID: 5740XYR00
RIDS/DSNIDM#L    LOAD MODULE NAME: DSNIDM
RIDS/DSNIMOST    CSECT NAME: DSNIMOST
AB/S004E         SYSTEM ABEND CODE: 004E
PRCS/00C90101   ABEND REASON CODE: 00C90101
REGS/0C6EE       REGISTER/PSW DIFFERENCE FOR R0C: 6EE
RIDS/DSNTFRCV#R RECOVERY ROUTINE CSECT NAME: DSNTFRCV
OTHER SERVICEABILITY INFORMATION
DATE ASSEMBLED:    12/15/92
MODULE LEVEL:      15.49
SUBFUNCTION:       DMC DSNISRTW
SERVICEABILITY INFORMATION NOT PROVIDED BY THE RECOVERY ROUTINE
RECOVERY ROUTINE LABEL
TIME OF ERROR INFORMATION
PSW: 077C0000 835D41B2 INSTRUCTION LENGTH: 02 INTERRUPT CODE: 000D
Failing INSTRUCTION TEXT: 00000000 00000000 00000000
REGISTERS 0-7
GR: 02EDB878 0004E000 7EFC03C0 7EFC0040 7EFC06D8 7F12E94B 02EDB878 7F3E4030
AR: 831B2716 00000000 00000000 00000000 00000000 00000000 00000000 00000000
REGISTERS 8-15
GR: 7EFCCEF8 836822E4 00000000 7F16E5E0 835D3AC4 7F16E5E0 7F16E654 00C90101
AR: 00000000 00000000 00000000 00000000 00000000 00000000 8325EDD4 02EDB878
HOME ASID: 002D PRIMARY ASID: 00CD SECONDARY ASID: 00CD
PKM: FFFF AX: 0035 EAX: 0000
RTM WAS ENTERED BECAUSE AN SVC WAS ISSUED IN AN IMPROPER MODE.
THE ERROR OCCURRED WHILE AN ENABLED RB WAS IN CONTROL.
NO LOCKS WERE HELD.
NO SUPER BITS WERE SET.
RECOVERY ENVIRONMENT
RECOVERY ROUTINE TYPE: FUNCTIONAL RECOVERY ROUTINE (FRR)
PSW AT ENTRY TO FRR: 070C0000 831C22A8
FRR PARAMETER AREA ON ENTRY TO FRR:
+00 7F16C71C C1C4D4C6 02EDB878 00000000 00000000 00000000

RECOVERY ROUTINE ACTION
THE RECOVERY ROUTINE REQUESTED THAT TERMINATION PROCESSING CONTINUE.
THE REQUESTED SVC DUMP WAS SUCCESSFULLY STARTED.
NO LOCKS WERE REQUESTED TO BE FREED.
HEXADECIMAL DUMP
```

| HEADER | | | | | |
|---|----------|------------|-----------|----------|-----------------|
| +000 | 40831820 | 00000000 | 0093028F | 11054950 | C.....L..... |
| +010 | 45125784 | 90210000 | | | ...D.... |
| JOBNAME | | | | | |
| +000 | E3F0F3F2 | F4F7F840 | | | T032478 |
| SDWA BASE | | | | | |
| +000 | 00000C60 | 0404E000 | 00000000 | 00000000 | ...-...\..... |
| +010 | 00000000 | 00000000 | 02EDB878 | 0004E000 |8...\.. |
| +020 | 7EFCDC30 | 7EFCDC040 | 7EFCDC6D8 | 7F12E94B | =.L{=.}=.0Q".Z. |
| : | | | | | |
| . | | | | | |
| +180 | 00000000 | C4E2D5F1 | 00000000 | 0006180F |DSN1..... |
| +190 | 00FFA0D6 | | | | ...0 |
| VARIABLE RECORDING AREA (SDWAVRA) | | | | | |
| +000 | KEY: 06 | LENGTH: 04 | | | |
| +002 | 00C90101 | | | | .I.. |
| +006 | KEY: 14 | LENGTH: 04 | | | |
| +008 | 02EDB878 | | | | ..8. |
| +00C | KEY: C9 | LENGTH: 04 | | | |
| +00E | 00000000 | | | | |
| +012 | KEY: CA | LENGTH: 04 | | | |
| +014 | C4E2D540 | | | | DSN |
| +018 | KEY: 3A | LENGTH: 02 | | | |
| +01A | 0039 | | | | .. |
| +01C | KEY: 3A | LENGTH: 02 | | | |
| +01E | 00CD | | | | .. |
| +020 | KEY: 3A | LENGTH: 02 | | | |
| +022 | 002D | | | | .. |
| +024 | KEY: CB | LENGTH: 08 | | | |
| +026 | C4C2F2C3 | C1D3D340 | | | DB2CALL |
| +02E | KEY: CC | LENGTH: 0C | | | |
| +030 | E3F0F3F2 | F4F7F840 | 40404040 | | T032478 |
| +03C | KEY: DF | LENGTH: 01 | | | |
| +03E | C1 | | | | A |
| +03F | KEY: DB | LENGTH: 08 | | | |
| +041 | A6F6003C | 62557F11 | | | W6....." |
| +049 | KEY: 7C | LENGTH: 0C | | | |
| +04B | FFFF070A | 835D3AA8 | 0CA800CD | |C).Y.Y.. |
| +057 | KEY: 06 | LENGTH: 04 | | | |
| +059 | 00000000 | | | | |
| : | | | | | |
| +05D | KEY: 53 | LENGTH: 00 | | | |
| +05F | KEY: 52 | LENGTH: 02 | | | |
| +061 | 0008 | | | | .. |
| +063 | KEY: CD | LENGTH: 05 | | | |
| +065 | C9D4D6E2 | E3 | | | IMOST |
| +06A | KEY: CE | LENGTH: 02 | | | |
| +06C | 5007 | | | | . |
| +06E | KEY: CF | LENGTH: 04 | | | |
| +070 | 00000004 | | | | |
| +074 | KEY: D0 | LENGTH: 04 | | | |
| +076 | 00000000 | | | | |
| +07A | KEY: D1 | LENGTH: 04 | | | |
| +07C | 00000000 | | | | |
| +080 | KEY: D2 | LENGTH: 04 | | | |
| +082 | 00000000 | | | | |
| +086 | KEY: D3 | LENGTH: 08 | | | |
| +088 | C4E2D540 | 40404040 | | | DSN |
| +090 | KEY: D4 | LENGTH: 04 | | | |
| +092 | 00000000 | | | | |
| +096 | KEY: D5 | LENGTH: 04 | | | |
| +098 | 00000000 | | | | |
| +09C | KEY: D6 | LENGTH: 04 | | | |
| +09E | 00000000 | | | | |
| +0A2 | KEY: D7 | LENGTH: 2C | | | |
| +0A4 | 00000000 | 00000000 | 00000000 | 00000000 | |
| +0B4 | 00000000 | 00000000 | 00000000 | 00000000 | |
| +0C4 | 00000000 | 00000000 | 00000000 | | |
| +0D0 | KEY: D8 | LENGTH: 04 | | | |
| +0D2 | 00000000 | | | | |
| : | | | | | |
| SDWA FIRST RECORDABLE EXTENSION (SDWARC1) | | | | | |
| +000 | E7E8D9F0 | F0C4D4C3 | 4040C4E2 | D5C9E2D9 | XYR00DMC DSNISR |
| +010 | E3E64040 | 40404040 | 40404040 | F1F261F1 | TW 12/1 |
| +020 | F561F9F2 | F1F54BF4 | F9404040 | 00C90101 | 5/9215.49 .I.. |
| : | | | | | |
| +180 | 00000000 | 00000000 | 00000000 | 00000000 | |
| +190 | 00000000 | 00000000 | 00000000 | 00000000 | |
| +1A0 | 00000000 | 7F7070B0 | 00000000 | 00000000 | ...".0..... |

```

SDWA SECOND RECORDABLE EXTENSION (SDWARC2)
+000 00000000 00000000 00000000 00000000 |.....|
SDWA THIRD RECORDABLE EXTENSION (SDWARC3)
+000 00000000 00000000 00000000 00000000 |.....|
+010 00000000 00000000 00000000 00000000 |.....|
ERRORID
+000 02630041 002D0006 180F |.....|

```

The second page of the SYS1.LOGREC entry includes the unformatted system diagnostic work area (SDWA). The formatted data on the first page is taken from this unformatted SDWA.

Two following fields are significant in the unformatted section of the SDWA:

SDWACMPC

The three-character completion code for the abend.

SDWAVRA

The beginning of the variable recording area (VRA). Selected Db2 recovery routines update the VRA with diagnostic information that concerns the abend. Dumps printed through IPCS format the VRA portion of the SDWA as shown in the example. The VRA is also formatted in the VRA diagnostic information report.

Related concepts

The system diagnostic work area (SDWA)

Each SVC dump that is requested by a Db2 functional recovery routine usually contains an SDWA with information about the status of the subsystem at the time of the error. Typically, the SDWA is a starting point for diagnosis.

The variable recording area (VRA)

More diagnostic information for Db2 abend reason codes is placed in the variable recording area (VRA) of the system diagnostic work area (SDWA) and is extracted and displayed in the VRA Diagnostic Information Report. This data can be produced by common recording routines and certain Db2 subcomponents.

Common VRA recording routines

Three common VRA recording routines exist: DSNWSDWA, DSNTFRCV, and DSN9SCN9. DSNWSDWA records significant information for all Db2 abends, regardless of the subcomponent involved. DSNTFRCV and DSN9SCN9 record significant information for only certain subcomponents.

The following tables identify the following information:

- The VRA key names that identify the VRA data
- The hexadecimal value of the VRA keys
- The length of the data that the VRA keys identify
- The information that the VRA keys identify.

The DSNWSDWA table also identifies if "DSN1" or "DSN2" is recorded in the SDWACOMU field of the SDWA. This information indicates whether percolation occurred and can help you interpret SYS1.LOGREC entries.

VRA data that is recorded by DSNWSDWA represents data that is stored on the first invocation of DSNWSDWA. If DSNWSDWA is invoked again because of an error that is percolated by the z/OS recovery termination manager (RTM), DSNWSDWA stores only the VRARC, VRACBA, VRARRK1, and VRARRK2 fields. The remaining fields, which do not change during RTM percolation, are not stored.

During Db2 recovery processing, the DSNWSDWA routine tries to gather information about the failing CSECT from the PSW address at the time of error (SDWAEC1) and the module entry point list (MEPL). This information is recorded in the VRA under the VRAIMO key and is then used to do the following tasks:

- Provide the failing load module, CSECT, and offset in the dump title that is associated with the SVC dump.
- Provide a summary storage entry for the failing CSECT to ensure that the entire failing CSECT is included in the SVC dump.

Table 13. VRA data recorded by DSNWSDWA

| VRAKEY key name | VRAKEY hex value | VRALEN data length | VRADAT data contents and format | Recorded in | | |
|-----------------|------------------|--------------------|---|-------------|------|-----|
| | | | | DSN1 | DSN2 | 04F |
| VRARC | 06 | 4 bytes | Abend reason code (first VRARC occurrence) | Yes | Yes | Yes |
| | | 4 bytes | DSNWSDWA return code (second VRARC occurrence) | | | |
| VRACBA | 14 | 4 bytes | EB address | Yes | Yes | |
| VRARRK1 | C9 | 4 bytes | EB latch hierarchy mask | Yes | Yes | Yes |
| VRARRK2 | CA | 4 bytes | Subsystem name (non-sharing Db2) | Yes | | |
| | | 16 bytes | Db2 data sharing group name (8 characters) and member name (8 characters) | | | |
| VRAAID | 3A | 2 bytes | Binary ASID of DSCF | Yes | | |
| | | 2 bytes | Binary ASID of ADMF | Yes | | |
| | | 2 bytes | Binary ASID of DDF address space (ssnmDIST) (if DDF is active) | Yes | | |
| | | 2 bytes | Binary ASID of Home address space | Yes | | |
| VRARRK3 | CB | 8 bytes | Connection-ID of home address space | Yes | | |
| VRARRK4 | CC | 12 bytes | Thread correlation token | Yes | | |
| VRARRK19 | DB | 8 bytes | Store clock (STCK) time of error | Yes | | |
| VRARRK21 | DD | 36 bytes | Additional thread information of the form: | Yes | Yes | Yes |
| | | 4 bytes | EBICE14: R14 (caller) of execution unit switch ¹ | | | |
| | | 4 bytes | EBSUS14: R14 (caller) of suspend ¹ | | | |
| | | 4 bytes | EBRES6: R6 (EB) of EB initiating resume | | | |
| | | 4 bytes | EBRES14: R14 (caller) of EB initiating resume ¹ | | | |
| | | 4 bytes | EBCAN6: R6 (EB) of EB initiating cancel | | | |
| | | 4 bytes | EBCAN14: R14 (caller) of EB cancel request ¹ | | | |

Table 13. VRA data recorded by DSNWSDWA (continued)

| VRAKEY key name | VRAKEY hex value | VRALEN data length | VRADAT data contents and format | Recorded in | | |
|-----------------|------------------|--------------------|---|-------------|------|-----|
| | | | | DSN1 | DSN2 | 04F |
| | | 4 bytes | EBOCAN14: R14 (caller) of original cancel request ¹ | | | |
| | | 4 bytes | EBSPAWN: Spawned EB | | | |
| | | 4 bytes | EBSC14: R14 (caller) of suspend when cancel is honored | | | |
| VRARRK23 | DF | 2 bytes | Thread type: <ul style="list-style-type: none"> 'S' Db2 system thread 'A' allied thread (no distributed activity) 'D' database access thread 'R' allied thread with remote activity | Yes | | |
| VRAIMO | 7C | 16 bytes | Failing CSECT information | Yes | | |
| | | 2 bytes | Possible values: <ul style="list-style-type: none"> X'FFFF'=CSECT data obtained X'0000'=CSECT not in MEPL | | | |
| | | 2 bytes | ASID of failing CSECT | | | |
| | | 4 bytes | Failing instruction offset | | | |
| | | 4 bytes | Failing CSECT starting address | | | |
| | | 4 bytes | CSECT length | | | |
| VRADAE | 53 | 0 bytes | Indicate use of DAE | Yes | | |
| VRAMINSC | 52 | 2 bytes | DAE Symptom string count | Yes | | |

Note:

1. A dump, or the Diagnose Display MEPL utility output is required to coordinate R14 addresses to specific Db2 CSECTs and offsets. A dump, or display MEPL output must be obtained for the same instance of Db2.

Table 14. VRA data recorded by DSNLFRCV. The Db2 distributed database facility uses this routine.

| VRAKEY key name | VRAKEY hex value | VRALEN data length | VRADAT data contents and format |
|-----------------|------------------|--------------------|--|
| VRARRK20 | DC | 2 bytes | Internal failure identifier for DDF |
| VRARRK24 | E0 | 16 bytes | Location name of the requesting location for this transaction (zero if this location is the requesting location) |
| VRARRK25 | E1 | 24 bytes | Logical unit of work identifier (LUWID) of the agent |
| VRARRK26 | E2 | 34 bytes | Distributed requester information: |
| | | 8 bytes | Requester product identifier, <i>pppvrrm</i> where: The product identifier (PRDID) value is an 8-byte character value in <i>pppvrrm</i> format, where: <i>ppp</i> is a 3-letter product code; <i>vv</i> is the version; <i>rr</i> is the release; and <i>m</i> is the modification level. For Db2 13 for z/OS, the modification level (0–9 or A–Z) indicates a specific function level. For example: DSN13012 for V13R1M501. DSN13011 for V13R1M500. DSN13010 for V13R1M100. For more information, see Product identifier (PRDID) values in Db2 for z/OS (Db2 Administration Guide) . |
| | | 8 bytes | Requester LU name or internal form of the IP address |
| | | 16 bytes | Requester location name |
| | | 2 bytes | Reserved |
| VRARRK27 | E3 | 17 bytes | Abend recovery retry routine identification information: |
| | | 4 bytes | Retry routine address |
| | | 4 bytes | Component ID of retry routine |
| | | 1 bytes | Not used, always blank |
| | | 8 bytes | Function ID of retry routine |

Table 15. VRA data recorded by DSNTFRCV. The Db2 database services subcomponents and the distributed data facility use this routine.

| VRAKEY key name | VRAKEY hex value | VRALEN data length | VRADAT data contents and format |
|-----------------|------------------|--------------------|--|
| VRARRK5 | CD | 5 bytes | The name of the data manager (DM) subcomponent CSECT (CTERPROC) that detected the error |
| VRARRK6 | CE | 2 bytes | An internal qualifier (CTERQUAL) of the data manager (DM) subcomponent |
| VRARRK7 | CF | 4 bytes | A data manager (DM) subcomponent return code (CTDMRETC) |
| VRARRK8 | D0 | 4 bytes | A reason code (CTSIRCOD) from the data manager (DM) subcomponent |
| VRARRK9 | D1 | 4 bytes | A return code from the last function that data manager (DM) subcomponent called (CTRMRC) |

Table 15. VRA data recorded by DSNTFRCV. The Db2 database services subcomponents and the distributed data facility use this routine. (continued)

| VRAKEY key name | VRAKEY hex value | VRALEN data length | VRADAT data contents and format |
|------------------------|-------------------------|---------------------------|---|
| VRARRK10 | D2 | 4 bytes | An SQL code (SQLCODE) from the relational data system (RDS) subcomponent |
| VRARRK11 | D3 | 8 bytes | An SQL error procedure name (SQLERRP) from the relational data system (RDS) subcomponent |
| VRARRK12 | D4 | 4 bytes | An internal code (SQLERRD1) returned by an SQL error procedure of the relational data system (RDS) subcomponent |
| VRARRK13 | D5 | Variable | A "resource unavailable" reason code (CTRURESN). If a reason code is available, length is 4 bytes. Otherwise, length is 0. |
| VRARRK14 | D6 | Variable | A "resource unavailable" resource type (CTRUTYPE). If a reason code is available, length is 4 bytes. Otherwise, length is 0. |
| VRARRK15 | D7 | Variable | A "resource unavailable" resource name (CTRUNAME). If a resource name is available, length is 1–44 bytes. Otherwise, length is 0. |
| VRARRK16 | D8 | 4 bytes | A reason code (CTUNERM) from the resource manager (RM) |
| VRARRK20 | DC | 2 bytes | Internal failure identifier for DDF |
| VRARRK27 | E3 | 17 bytes | Abend recovery retry routine identification information: |
| | | 4 bytes | Retry routine address |
| | | 4 bytes | Component ID of retry routine |
| | | 1 bytes | Not used, always blank |
| | | 8 bytes | Function ID of retry routine |
| VRARRK33 | E9 | 12 bytes | P-lock negotiation information |
| | | | 1 byte: p-lock held state |
| | | | 1 byte: p-lock cache state |
| | | | 1 byte: p-lock requested state |
| | | | 1 byte: unused |
| | | | 8 bytes: p-lock owning work unit |

Table 16. VRA data recorded by DSN9SCN9. Db2 system service subcomponents use this routine. These subcomponents include the message generator, recovery log manager, general command processor, and, in some cases, instrumentation facility.

| VRAKEY key name | VRAKEY hex value | VRALEN data length | VRADAT data contents and format |
|------------------------|-------------------------|---------------------------|---|
| VRACBM | 11 | 8 bytes | The name of the DSECT for DDT (DSNDDDT) |
| VRACB | 12 | Variable | The recordable portion of the diagnostic data table (DDT) control block that contains information such as the name of the module that invoked the recovery routine. |
| VRARC | 06 | 4 bytes | The abend reason code |

Table 17. VRA data recorded by DSNWSDM

| VRAKEY key name | VRAKEY hex value | VRALEN data length | VRADAT data contents and format |
|-----------------|------------------|--------------------|--|
| VRARRK5 | CD | 5 bytes | The name of the data manager (DM) subcomponent CSECT (CTERPROC) that detected the error |
| VRARRK6 | CE | 2 bytes | An internal qualifier (CTERQUAL) of the data manager (DM) subcomponent |
| VRARRK11 | D3 | 8 bytes | An SQL error procedure name (SQLERRP) from the relational data system (RDS) subcomponent |
| VRARRK12 | D4 | 4 bytes | An internal code |
| VRARRK20 | DC | 24 bytes | Internal failure identifier for DDF |
| VRARRK22 | DE | 6 bytes | A dump work area was not obtained. (NODMPW) |
| VRAIMO | 7C | 12 bytes | Failing CSECT information |
| | | 2 bytes | X'FFFF'=CSECT data obtained |
| | | | X'0000'=CSECT not in MEPL |
| | | 2 bytes | Failing instruction offset |
| | | 4 bytes | Failing CSECT starting address |
| | | 2 bytes | CSECT length |
| | | 2 bytes | ASID of failing CSECT |

Individual VRA recording subcomponents

Nine Db2 subcomponents provide additional VRA data: agent services manager, buffer manager, IMS attachment facility, initialization procedure, instrumentation facility, recovery log manager, recovery manager, subsystem support, and storage manager.

Unlike the common VRA recording routines, these subcomponents use different VRA keys for different abend reason codes. The following tables identify the VRA data that is provided by these individual Db2 subcomponents. For each of these subcomponents, the tables identify:

- The abend reason codes for which VRA data is produced
- The VRA key names for each abend reason code
- The hexadecimal value of the VRA key

Table 18. Agent services manager. The agent services manager supplies VRA data for the following abend reason codes.

| Code | Key names | Hex values |
|----------|----------------|------------|
| 00E50013 | VRAHEX, VRARRP | 38, 10 |
| 00E50014 | VRARRK18 | DA |
| 00E50015 | | |
| 00E50031 | VRAHEX | 38 |
| 00E50032 | | |
| 00E50044 | | |
| 00E50050 | VRARRK18 | DA |

Table 18. Agent services manager. The agent services manager supplies VRA data for the following abend reason codes. (continued)

| Code | Key names | Hex values |
|----------|------------------------------|------------|
| 00E50051 | | |
| 00E50052 | | |
| 00E50059 | | |
| 00E50070 | VRARRP | 10 |
| 00E50071 | | |
| 00E50072 | | |
| 00E50079 | VRARRK16, VRARRK17, VRARRK18 | D8, D9, DA |
| 00E50500 | VRARC, VRARRP | 06, 10 |
| 00E50501 | | |
| 00E50701 | VRARRP, VRARRK16, VRARRK17 | 10, D8, D9 |
| 00E50705 | VRARRP | 10 |
| 00E50707 | | |

Table 19. Buffer manager. The buffer manager supplies VRA data for the following abend reason codes:

| Code | Key name | Hex value |
|----------|--|---------------|
| 00C200D0 | VRARRP | 10 |
| 00C200D1 | | |
| 00C200D5 | | |
| 00C200D6 | | |
| 00C202A5 | VRARRK30, VRAKK31, VRAKK32 | E6, E7, E8 |
| 00C202AA | VRARRK30 | E6 |
| 00C202AB | VRARRK30 | E6 |
| 00C202AC | VRARRK13, VRARRK14, VRARRK15, VRARRK33 | D5, D6 D7, E9 |
| 00C202AE | VRARRK30 | E6 |
| 00C202AF | VRARRK13, VRARRK14, VRARRK15, VRARRK33 | D5, D6 D7, E9 |
| 00C202B0 | VRARRK13, VRARRK14, VRARRK15, VRARRK33 | D5, D6 D7, E9 |
| 00C202B1 | VRARRK30 | E6 |
| 00C202D0 | VRARRP | 10 |

Table 20. The IMS attach subcomponent VRA data. The IMS attach subcomponent supplies VRA data for the following abend reason code.

| Code | Key names | Hex values |
|----------|--------------|------------|
| 00D40008 | VRARC, VRAFP | 06, 23 |

Table 21. Initialization procedure. The initialization procedure subcomponent supplies VRA data 'E8' for all abend reason codes.

| Code | Key name | Hex value |
|---------------------------|-----------------|------------------|
| All E8 abend reason codes | VRARRP | 10 |

Table 22. Instrumentation facility. The instrumentation facility supplies VRA data for the following abend reason codes.

| Code | Key names | Hex values |
|-------------|------------------|-------------------|
| Trace super | VRAPLI, VRAPL | 20, 21 |
| 00E60887 | VRARC, VRAFP | 06, 23 |
| 00E60888 | | |
| 00E60889 | | |

Table 23. Recovery log manager. The recovery log manager supplies VRA data for the following abend reason codes.

| Code | Key names | Hex values |
|-------------|------------------|-------------------|
| 00D10250 | VRARC, VRARRP | 06, 10 |
| 00D10251 | | |
| 00D10252 | | |

Table 24. Recovery manager. The recovery manager supplies VRA data for the following abend reason codes.

| Code | Key name | Hex value |
|-------------|-----------------|------------------|
| 00D93011 | VRAHEX | 38 |
| 00D93012 | | |
| 00D94011 | | |
| 00D9AAAA | | |
| 00D9BBBB | | |
| 00D9CCCC | | |
| 00D9EEEE | | |

Table 25. Storage manager. The storage manager records the 32-byte MEPL entry of the calling module at abend. If the storage manager is unable to identify the invoking module, a VRACAN VRA entry is created with the following data: CALLER OF SMC UNKNOWN. The storage manager supplies VRA data for the following abend reason codes.

| Code | Key name | Hex value |
|-------------|-----------------|------------------|
| 00E20001 | VRACAN | 3D |
| 00E20002 | | |
| 00E20003 | | |
| 00E20004 | | |
| 00E20005 | | |

Table 25. Storage manager. The storage manager records the 32-byte MEPL entry of the calling module at abend. If the storage manager is unable to identify the invoking module, a VRACAN VRA entry is created with the following data: CALLER OF SMC UNKNOWN. The storage manager supplies VRA data for the following abend reason codes. (continued)

| Code | Key name | Hex value |
|--|----------|-----------|
| 00E20006 | | |
| 00E20009 | | |
| 00E2000B | | |
| 00E2000C | | |
| 00E2000D | | |
| 00E2000E | | |
| 00E2000F | | |
| 00E2001B | | |
| 00E2001F | | |
| 00E20022 | | |
| 00E20027 (only for DSNSVBK or DSNSFBK modules) | | |
| 00E20029 | | |

Subsystem support

The subsystem support subcomponent supplies the VRA data for all abend reason codes.

Table 26. Identify recovery. The Identify ESTAE recovery routine DSN3IDES generates the following VRADATA entries. The last entry, key VRAIMO, is generated only if the abend occurred within the Identify authorization exit.

| VRAKEY key name | VRAKEY hex value | VRALEN data length | VRADAT data contents and format |
|-----------------|------------------|--------------------|--|
| VRFPFI | 22 | 8 bytes | Constant 'IDESTRAK' |
| VRAFP | 23 | 24 bytes | 32-bit recovery tracking flag 32-bit integer AGNT block unique identifier AGNT block address AIDL block address Initial primary authorization ID |

Table 27. Sign-on recovery. The sign-on ESTAE recovery routine DSN3SIES generates the following VRADATA entries.

| VRAKEY key name | VRAKEY hex value | VRALEN data length | VRADAT data contents and format |
|-----------------|------------------|--------------------|---|
| VRFPFI | 22 | 8 bytes | Constant 'SIESTRAK' |
| VRAFP | 23 | 20 bytes | Primary authorization ID (CCBUSER) AGNT block address Identify-level CCB block address Signon-level CCB block address |

Table 28. VRA data recorded by DSN3AUFR. Authorization Services Functional Recovery. VRAIMO is included if the error was in an exit routine.

| VRAKEY key name | VRAKEY hex value | VRALEN data length | VRADAT data contents and format |
|-----------------|------------------|--------------------|---|
| VRFPFI | 22 | 8 bytes | AUFRTRAK |
| VRAFP | 23 | 12 bytes | Primary Auth Id (8 bytes) Address of CCB (4 bytes) |
| VRAIMO | 7C | 10 bytes | Load module (4 bytes) Entry point (4 bytes) Offset of failing instruction (2 bytes) |

Printing and analyzing global traces

A trace must be active on the Db2 subsystem that you want to trace. For data sharing, the trace commands have member or group scope.

PSPI

You can activate a trace at all members of a data sharing group by issuing the START TRACE command with the SCOPE(GROUP) parameter on one member of the data sharing group. Db2 routes the command to each member of the data sharing group. The trace output goes to the destination that is specified on the DEST parameter.

Table 29. Keywords for use with the Db2 global trace

| Keyword | Valid with commands | Default | Other parameters | Comments and restrictions |
|---------|---------------------|---------------------------|-----------------------------------|---|
| TRACE | START | None | Type of trace | This keyword must be specified. A parameter does not need to be specified for the TRACE keyword if other keywords and their parameters provide sufficient detail to stop or display trace activity. |
| | DISPLAY | | | |
| | MODIFY | None | | |
| | STOP | None | | |
| CLASS | START | | Class number | None |
| | DISPLAY | | | |
| | MODIFY | | | |
| | STOP | | | |
| DEST | START | Established by trace type | GTF, RES, SMF SRV, or OPn | SRV is reserved for IBM service personnel. |
| | DISPLAY | | | |
| | STOP | | | |
| PLAN | START | * | in the range 1–8 valid PLAN names | If several PLANs are specified, only one AUTHID and TNO can be specified. If several values are specified, only one LOCATION value can be specified. |
| | DISPLAY | * | | |
| | STOP | * | | |
| AUTHID | START | * | in the range 1–8 valid AUTHIDs | If several AUTHIDs are specified, you only one PLAN and TNO can be specified. If several values are specified, only one LOCATION value can be specified. |
| | DISPLAY | * | | |
| | STOP | * | | |

Table 29. Keywords for use with the Db2 global trace (continued)

| Keyword | Valid with commands | Default | Other parameters | Comments and restrictions |
|----------|---------------------|---------------------------|---------------------------------------|---|
| RMID | START | * | in the range 1–8 RMIDs | None. |
| | DISPLAY | * | | |
| | STOP | * | | |
| TNO | DISPLAY | * | in the range 1–8 valid trace numbers | If several TNOs are specified, only one PLAN and AUTHID can be specified. TNO is required with MODIFY. If several values are specified, only one LOCATION value can be specified. |
| | MODIFY | * | | |
| | STOP | * | | |
| IFCID | START MODIFY | Set by trace type | IFCID number | When a trace type is specified, all IFCIDs specified by the trace type, classes and the IFCID are started. |
| COMMENT | START | Blanks | Any character string | The comment is displayed within the stop or start trace record that is sent to SMF, GTF, or SRV. |
| | DISPLAY | | | |
| | MODIFY | | | |
| | STOP | | | |
| TDATA | START | Established by trace type | COR, CPU, or TRA | The product section of a trace record can contain several headers. |
| LOCATION | START | * | in the range 1–8 valid location names | If several LOCATIONS are specified, only one PLAN, AUTHID, and TNO can be specified. |
| | DISPLAY | * | | |
| | STOP | * | | |

PSPI

Related reference

- START TRACE command (Db2) (Db2 Commands)
- MODIFY TRACE command (Db2) (Db2 Commands)
- DISPLAY TRACE command (Db2) (Db2 Commands)
- STOP TRACE command (Db2) (Db2 Commands)

Global trace facility

The Db2 global trace is used to obtain information about program flow, the entries to and exits from Db2 functions and modules, to help resolve problems. The global trace is intended to be used under the direction of IBM support personnel.

PSPI

This information explains the different destinations to which global trace records can be sent, how to start, change, display, and stop the Db2 global trace, how to locate the trace tables, and how the trace records in the trace tables are formatted.

To use this command, you must have one of the following authorities:

- SYSADM or SYSOPR authority
- Authority to issue start/stop trace commands (trace authority)
- Authority to issue the display trace command (display authority)

The following commands control the global trace:

- -START TRACE(GLOBAL)
- -DISPLAY TRACE(GLOBAL)
- -MODIFY TRACE(GLOBAL)
- -STOP TRACE(GLOBAL)

PSPI

Related concepts

[Accounting trace \(Db2 Performance\)](#)

[Performance trace \(Db2 Performance\)](#)

[Statistics trace \(Db2 Performance\)](#)

[Monitor trace \(Db2 Performance\)](#)

[Audit trace \(Db2 Performance\)](#)

Related reference

[Printing and analyzing global traces](#)

A trace must be active on the Db2 subsystem that you want to trace. For data sharing, the trace commands have member or group scope.

Starting the global trace

The Db2 global trace can be started automatically at startup (by parameters that are specified during the installation action or migration process) or with the -START TRACE(GLOBAL) command.

PSPI

FL 506 The global trace provides data related to the major functional and exceptional events that are taking place within Db2. When a trace is started, Db2 assigns it a trace number (TNO) from 1 to 64. This number mask is displayed in the standard trace record header externalized to SMF or GTF.

Specify "GLOBAL" when starting a trace to distinguish the Db2 global trace from other trace types, as displayed in the following example:

```
-START TRACE(GLOBAL)
```

When you install or migrate Db2, start a collection of global serviceability data. The events recorded by a Db2 global trace are identified by event identifiers (EIDs). EIDs appear in the global trace records that Db2 externalizes.

Specifying trace options

Some trace options have specific values for a global trace. Those options are described below. For information on specifying the keywords for other trace options, see [-START TRACE command \(Db2\) \(Db2 Commands\)](#).

Specifying a class

The CLASS keyword specifies a particular class or type of trace events for global trace.

- Class 1. Reserved for major functions and exceptions
- Class 2. Utility tracing and reserved for medium functions
- Class 3. All other module entries/exits
- Class 4. Reserved
- Class 5. Reserved
- Class 6. Reserved for installation use with IFI
- Class 7. For distributed events

- Class 8. For tracing SQL statements for distributed events at the requesting and responding locations.
- Class 9. For tracing DDF diagnostics information.
- Class 10. For tracing detailed database services address space (*ssnmDBM1*) storage usage information.

The default on the global trace command is to start trace classes 1, 2, and 3. "CLASS" normally should not be specified for global trace.

Important: Global trace classes 1, 2, and especially 3 can **significantly degrade system performance** and can produce an enormous amount of data if externalized to GTF or SMF.

If class 7 is specified, the following events are traced:

- VTAM exits to Db2
- All VTAM calls/returns
- Buffer sent/received
- Conversation allocation request queued in Db2.

If class 8 is specified the following events are traced:

- Requesting location SQL statements before modification
- Both requesting and responding location SQL statements after modification.

Specifying a destination

The DEST keyword specifies the location to which global trace data is sent.

Destinations include:

- RES—Resident trace table
- GTF—z/OS generalized trace facility (GTF)
- OP*n*—Destination for trace output
- SMF—z/OS System Management Facilities (SMF)
- SRV—Serviceability Routine.

FL 504 The default destination for a global trace is the resident trace table (RES), which is located above the line in the extended common service area (ECSA). Subsystem parameter TRACTBL determines if the RES exists and its size. Global trace data can also be sent to GTF, SMF, or OP*n* (where *n* is a value from 1 to 16 that corresponds to a specific destination). SMF is not recommended. SRV is reserved for IBM service personnel.

When specifying a location with the DEST keyword, use the appropriate abbreviation. For example, specify DEST(GTF) – not DEST(generalized trace facility):

```
-START TRACE(GLOBAL) DEST(GTF)
```

Trace data can be sent to more than one destination. This example sends data to both RES and GTF:

```
-START TRACE(GLOBAL) DEST(GTF,RES)
```

Specifying a resource manager

The RMID keyword limits the trace to the activity of a particular Db2 resource manager. Up to 8 resource managers can be specified. The default is an asterisk (*), which invokes the trace for any resource manager.

The following table identifies the RMIDs for a global trace.

Table 30. Resource manager identifiers (RMIDs)

| RMID Dec(Hex) | Resource Manager |
|----------------------|---|
| 1(01) | Initialization procedures |
| 2(02) | Agent services management |
| 3(03) | Recovery management |
| 4(04) | Recovery log management |
| 6(06) | Storage management |
| 7(07) | Subsystem support for allied memories |
| 8(08) | Subsystem support for SSI functions |
| 9(09) | Reserved |
| 10(0A) | Buffer management |
| 11(0B) | Reserved |
| 12(0C) | System parameter management |
| 13(0D) | Precompiler |
| 14(0E) | Data management |
| 15(0F) | Reserved |
| 16(10) | Instrumentation commands, trace, and dump services |
| 17(11) | LOB management |
| 18(12) | Data space management |
| 19(13) | Data Space management, AMS subcomponent |
| 20(14) | Service controller |
| 21(15) | Database utilities |
| 22(16) | Relational database support |
| 23(17) | General command processing |
| 24(18) | Message generator |
| 25(19) | Distributed Relational Data System Manager |
| 26(1A) | Instrumentation accounting and statistics |
| 27(1B) | Distributed Communications Resource Manager |
| 28(1C) | Distributed Transaction Manager (DDF address space (<i>ssnmDIST</i>)) |
| 29(1D) | Distributed Data Interchange Services |
| 30(1E) | Distributed Transaction Manager (DSCF address space) |
| 31(1F) | Group Manager |

Unlike the PLAN and AUTHID keywords, more than one trace does not start if more than one RMID is specified. This example invokes a single trace for 5 resource managers:

```
-START TRACE(GLOBAL) RMID(1,7,14,18,24)
```

PSPI

Related reference

[-START TRACE command \(Db2\) \(Db2 Commands\)](#)

Displaying global trace activity

The `-DISPLAY TRACE` command displays information about the traces that were started, including the options in effect.

PSPI

In a data sharing group, you can enter the `DISPLAY TRACE` command on each subsystem that you want to monitor, or you can enter the command with the `SCOPE(GROUP)` option on one subsystem to cause Db2 to issue the command on all members of the data sharing group. The command output from all members goes to the console from which the `-DISPLAY TRACE` command was issued.

Because other traces can be started and stopped while you view `-DISPLAY TRACE` output, the information presented by the `-DISPLAY TRACE` command might change immediately after the display is complete.

Each `-DISPLAY TRACE` option, except `TNO`, limits the effect of the command to active traces that were started using the same option, either explicitly or by default, with exactly the same parameter values.

For example, the following command lists only the active traces that were started using the options `GLOBAL` and `CLASS(1,2)`; it does not list, for example, any trace started using `CLASS(1)`.

```
-DISPLAY TRACE (GLOBAL) CLASS (1,2)
```

Information about a particular trace can be displayed by specifying its trace number. For example, the following command displays the second trace that you invoked.

```
-DISPLAY TRACE TNO(2)
```

When a trace number is specified, you do not need to specify any other keywords or parameters. The trace number provides a sufficient level of detail for Db2 to stop or display trace activity.

PSPI

Related reference

[-DISPLAY TRACE command \(Db2\) \(Db2 Commands\)](#)

Modifying global trace activity

The `-MODIFY TRACE` command changes the trace events (IFCIDs) being traced for any active trace. The `-MODIFY TRACE` command actually stops the existing trace and then starts the specified trace.

PSPI

For example, the following command modifies trace number 1 to collect only data from IFCIDs 106 and 131 (utility trace data and system parameters in effect at trace invocation). Any other data that was being collected is stopped.

```
-MODIFY TRACE(GLOBAL) CLASS(3) IFCID(106,131)  
TNO(1) COMMENT('SYS PARMS AND UTILITY TRACE')
```

If several traces are active, specify enough qualifying information to isolate the trace you want to change.

PSPI

Stopping global trace activity

The `-STOP TRACE` command terminates trace activity.

PSPI

In a data sharing group, you can stop the trace only on one subsystem, or, if the trace was started with SCOPE(GROUP), you can stop the trace for the entire group by issuing the -STOP TRACE command with the SCOPE(GROUP) option.

Stopping a trace might not require specifying all the keywords specified when starting the trace. Just provide sufficient detail to stop the trace.

The command -STOP TRACE is invalid. To stop all active traces, issue the following command:

```
-STOP TRACE(*)
```

You might, for example, have started a trace with the following command:

```
-START TRACE(GLOBAL) DEST(GTF) RMID(10,12,24)
```

This command contains several keywords that limit the trace. Assuming this is the only global trace active, you can issue this command to stop it:

```
-STOP TRACE(GLOBAL)
```

If several global traces were active and you wanted to stop only one of them, you need to provide more details in the -STOP TRACE command. For instance, assume three traces are started as follows:

```
-START TRACE(GLOBAL) DEST(GTF) RMID(10,12,24)
-START TRACE(GLOBAL) RMID(10,12,24)
-START TRACE(GLOBAL) AUTHID(SYSADM1)
```

To stop the first of these three traces without disturbing the other two, you can enter:

```
-STOP TRACE(GLOBAL) DEST(GTF)
```

Because only the first trace sends data to GTF, specifying the trace destination provides sufficient detail.

The trace number (TNO) can also be used to stop trace activity. For instance, you might have previously started two traces with the following commands:

```
-START TRACE(GLOBAL) DEST(GTF)
-START TRACE(GLOBAL) AUTHID(USER1)
```

If output from -START TRACE or -DISPLAY TRACE commands indicate that the traces that were started were assigned trace numbers 1 and 2, respectively, then the first trace can be stopped by entering:

```
-STOP TRACE(GLOBAL) TNO(1)
```

When a trace number is specified, you do not need to specify any other keywords or parameters. The trace number provides a sufficient level of detail for Db2 to stop or display trace activity.

◀ PSPI

Using global trace output

Trace output is based on the parameters specified for the -START TRACE(GLOBAL) command. Each record identifies one or more significant Db2 events.

Interpreting trace record formats

The destination of global trace records (specified with the DEST keyword of the -START TRACE command) affects their contents and, therefore, their interpretation. Data sent to the resident trace table (RES) contains only addresses and contents of registers. Data sent to z/OS generalized trace facility (GTF) and z/OS System Management Facilities (SMF) also may contain control blocks and storage areas.

GTF and SMF trace entries do not appear in Db2 formatted SVC dumps, but resident tables do appear if the appropriate keywords are specified. The following topics describe how to format, locate, and interpret

global trace data sent to the resident trace table. The process of locating and interpreting the resident trace table in the unformatted portion of the non-summary dump data set is described later in this book.

Related concepts

Interpreting the formatted global trace table

The formatted Db2 trace tables are printed in time sequence from the oldest to the most current. To begin an analysis, review the most current entries (these appear at the end). Continue analyzing the information by scanning backward.

When to use the Db2 trace table

Global trace data sent to the resident trace table usually appears at the end of the formatted section of a Db2-issued SVC dump.

You might need to use the Db2 global trace table when you suspect a WAIT or LOOP problem.

The Db2 trace table can be used to look for trace entries that relate to the CSECT that failed. If you notice, for example, that the trace table shows control entering but not leaving the CSECT that abended, then you know the location of the problem.

If you look at the trace entries just before entry into the failing CSECT and analyze the data items in these trace entries, you might be able to learn more about the problem. (For example, if one of the data items is a field that contains the values of input to the CSECT, then you might be able to detect bad input; this would show that the caller of the abending CSECT is the cause of the abend.)

Related concepts

The global trace table

The Db2 trace table contains entries for significant Db2 activities and is useful for diagnosing WAIT or LOOP problems.

Building a keyword string

You can systematically select *keywords* to describe a failure in Db2 for z/OS (Db2) or internal resource lock manager (IRLM). Keywords are predefined words or abbreviations that identify aspects of a program failure.

Type-of-failure keywords

A type-of-failure keyword describes an external symptom of a program failure.

Related tasks

Searching for known problems and solutions for Db2 for z/OS

Searching the IBM Support site is most effective if you include all the appropriate keywords in your search.

Interpreting the formatted global trace table

The formatted Db2 trace tables are printed in time sequence from the oldest to the most current. To begin an analysis, review the most current entries (these appear at the end). Continue analyzing the information by scanning backward.

The following fields are the most significant fields in each trace entry:

RMID

The resource manager ID that made the trace entry.

Event number

The event number unique to the resource manager.

Refer to [Figure 61 on page 219](#) and follow these guidelines for interpreting that trace table:

1. Determine the address of the failing EB (execution block). This helps pinpoint some of the significant trace entries to examine.

The address of the failing EB is listed under Key 14 of the VRA Diagnostic Information Report (at the beginning of the formatted dump) and at offset X'30' in the SDWA (system diagnostic work area). [“Printing and analyzing dumps” on page 181](#) provides more information.

For the trace table shown in the example, the failing EB is at X'01E30480'. (The SDWA and VRA Diagnostic Information Report are not shown.)

2. Turn to the end of the formatted trace table. The last entry is most current, but might not always correspond to the failing EB.
3. Scanning backwards through the table, locate the most recent entry for the failing EB.

In the example, the last entry is the most recent entry corresponding to the failing EB. The EID (event identifier) indicates this entry is for the agent services manager (RMID = X'02'), event number X'0001'. The EID trace code tables show this involves an entry in the SUSPEND execution unit, and that the address of the resource options block (ROB) is recorded (to the right of the DATA column). There is no corresponding exit from SUSPEND (because this is the most recent entry and also the end of the table).

4. You can continue to scan backwards through the trace table, interpreting the entries for the failing EB as needed.

Interpreting the unformatted global trace table

The formatted trace table contains information copied from the unformatted resident trace table, which is always included in the non-summary section of the dump data set.

A resident trace table contains fixed-length, 64-byte records. A given Db2 event is recorded in a single record if there are no more than five event-data items; otherwise, they are recorded in two adjacent records. The first record, which might be the only record, contains a "long" header and up to five event-data items. The second record, if there is one, contains a "short" header and any remaining data items (up to a maximum of six items in the second record). The short header of the second record contains only the time-of-day stamp, which matches the time-of-day stamp in the first record. The second doubleword is X'FFFFFFFF00000000', and the six doublewords contain the data items.

The contents of the first and second record are shown in the following tables.

Table 31. Unformatted Db2 trace record content for first record

| Decimal offset | Hex offset | Length (decimal) | Field Description |
|----------------|------------|------------------|--|
| 0 | 0 | 8 | Time of Day. The TOD clock. This field is zero if the TOD clock value is not available. In a formatted trace table, this field follows the "eye-catcher" TOD . |
| 8 | 8 | 0.5 | Number of event-data items in the record or record pair. This field does not appear in the formatted trace table. |
| 8.5 | 8.5 | 0.5 | Indicator of the type of event . (Possible values are shown below.) |
| 9 | 9 | 1 | RMID. One-byte ID of the reporting resource manager. In a formatted trace table, this field follows the "eye-catcher" EID . |
| 10 | A | 2 | Two-byte event number , unique for the specific resource manager. When combined with the RMID, this number makes a unique trace event identifier. |
| 12 | C | 1 | Internal function code . In a formatted trace table, this field follows the "eye-catcher" FUNC . |
| 13 | D | 1.5 | Twelve-bit primary ASID (the address space containing the data). In a formatted trace table, this field follows the "eye-catcher" ASID . Any data addresses in the item list point to objects in this address space. |
| 14.5 | E.5 | 1.5 | Twelve-bit ASID of the caller of the CSECT that invoked the trace facility. In a formatted trace table, this field follows the "eye-catcher" RET . |

Table 31. Unformatted Db2 trace record content for first record (continued)

| Decimal offset | Hex offset | Length (decimal) | Field Description |
|----------------|------------|------------------|---|
| 16 | 10 | 4 | Four-byte return address of the caller (from register 14). |
| 20 | 14 | 4 | Four-byte address of the EB control block that was current at the time of the event. In a formatted trace table, this field follows the "eye-catcher" EB . |
| 24 | 18 | varies | A series of up to five 8-byte data items (such as addresses or values) associated with the event. The total number of items in the record, or record pair, is given in the second field of the trace record (offset X'08'). In a formatted trace table, this field follows the "eye-catcher" DATA . |

Table 32. Unformatted Db2 trace record content for second record

| Decimal offset | Hex offset | Length (decimal) | Field description |
|----------------|------------|------------------|--|
| 0 | 0 | 8 | Time of Day . The 8-byte TOD clock. This field is zero if the TOD clock value is not available. In a formatted trace table, this field does not appear. |
| 8 | 8 | 4 | Doubleword of X'FFFFFFFF00000000' |
| 16 | 18 | varies | A series of up to six 8-byte data items (such as addresses or values) associated with the event. The total number of items in the record, or record pair, is given in the second field of the first record. |

Table 33. Db2 trace tag values and meanings

| TAG value (hex) | Tag Name In Formatted Trace Table | Definition |
|-----------------|---|--|
| 00 | blank | Used when none of the other predefined codes in this table is applicable |
| 01 | RMENTRY | Resource manager functional entry point, invoked via RMRQ |
| 02 | RMEXIT | Resource manager functional exit point |
| 03 | RAENTRY | Resource Application request functional entry point, invoked via RARQ |
| 04 | RAEXIT | Resource Application request functional exit point |
| 05 | BKNBCST | Begin notification sequence |
| 06 | ENDBCST | End notification sequence |
| 07 | Reserved | Reserved |
| 08 | Reserved | Reserved |
| 09 | CALENTRY | Resource manager functional entry point, invoked via CALL |
| 0A | CALLEXIT | Resource manager functional exit point |
| F0 | Not applicable in Db2-issued trace tables | Start trace control record, valid for GTF destination only |
| F1 | Not applicable in Db2-issued trace tables | Stop trace control record, valid for GTF destination only |

Distributed data facility global trace

The distributed data facility contains a serviceability trace.

All distributed allied agents and database access agents have an associated DDF trace table with the following characteristics:

- The trace table is fixed-length.
- Each trace table contains many 128-byte event entries.
- When a DDF agent block (DPSB) is allocated for an allied or database agent, the table is created and initialized.
- The trace table wraps when it becomes full.

Trace events can be added as needed to service the distributed data facility.

Correlating the Db2 trace with the z/OS trace

To associate entries in the Db2 trace table with entries in the z/OS trace table, compare time of day for both entries.

As [Table 31 on page 246](#) indicates, the time-of-day field in the Db2 trace entry is obtained from bytes 3 to 6 of the same clock from which z/OS obtains the time of day for its trace entries.

Interpreting GTF and SMF global trace records

Db2 global trace data written to GTF has a GTF ID of X'0FB9' and an FID of X'00'. Records longer than the 256-byte limit of GTF are spanned by Db2.

Db2 global trace data is written to SMF as a SMF type 102 record. The following list shows the general format of these trace records:

1. Each record begins with a standard SMF or GTF writer header.
2. Each record next contains a self-defining section.

The self-defining section always begins at the end of the writer header. It contains pointers used to find the sections of the record that contain the actual trace data. The self-defining section identifies:

- The offset to the resource manager data
- The length of one unique area
- The number of times the unique area repeats.

3. The first self-defining area contains a product section.

The product section has a standard Db2 header and identifies the record and RMID. The trace header section, present for all global trace records, contains the event ID (EID).

4. Each record contains a section for resource manager data.

Global trace classes

IFCIDs (instrumentation facility component identifiers), RMIDs (resource manager IDs), and event descriptions can help you interpret global trace data sent to SMF or GTF.

The following tables can help you interpret global trace data sent to SMF or GTF. Each table lists IFCIDs, RMIDs, and event descriptions. Nine classes are defined for global trace, one of which (class 4) is reserved by IBM for serviceability, and one (class 6) is user-defined.

Class 1 : reserved for major functions

| IFCID | RMID | Event |
|-------|------|---|
| 0106 | 26 | System parameters in effect at trace invocation |

Class 2 : reserved for medium functions

| IFCID | RMID | Event |
|-------|------|---|
| 0106 | 26 | System parameters in effect at trace invocation |

Class 3 : entry and exit trace

| IFCID | RMID | Event |
|-------|-------------|--|
| 0000 | Actual RMID | Module entry exit trace |
| 0038 | 04 | Log buffer write |
| 0046 | 02 | Begin exec unit switch |
| 0047 | 02 | Begin new SRB |
| 0048 | 02 | End new SRB |
| 0049 | 02 | Begin new TCB |
| 0050 | 02 | End new TCB |
| 0051 | 02 | Shared latch resume (serviceability-only information) |
| 0052 | 02 | Shared latch wait (serviceability only information) |
| 0056 | 02 | Exclusive latch wait (Serviceability only information) |
| 0057 | 02 | Exclusive latch resume (serviceability-only information) |
| 0068 | 07 | Begin ABEND |
| 0069 | 07 | End ABEND |
| 0070 | 07 | Begin COMMIT phase 2 |
| 0071 | 07 | End COMMIT phase 2 |
| 0072 | 07 | Begin CREATE THREAD |
| 0073 | 07 | End CREATE THREAD |
| 0074 | 07 | Begin TERMINATE THREAD |
| 0075 | 07 | End TERMINATE THREAD |
| 0076 | 07 | Begin end-of-memory request |
| 0077 | 07 | End end-of-memory request |
| 0080 | 07 | Begin establish exit |
| 0081 | 07 | End establish exit |
| 0082 | 07 | Begin identify |
| 0083 | 07 | End identify |
| 0084 | 07 | Begin prepare to COMMIT |
| 0085 | 07 | End prepare to COMMIT |
| 0086 | 07 | Begin sign-on |
| 0087 | 07 | End sign-on |
| 0088 | 07 | Begin sync |

| IFCID | RMID | Event |
|--------------|-------------|--|
| 0089 | 07 | End sync |
| 0093 | 02 | Entry to suspend |
| 0094 | 02 | Exit from suspend |
| 0106 | 26 | System parameters in effect at trace invocation |
| 0114 | 04 | Archive I/O begin |
| 0115 | 04 | Archive I/O end DASD |
| 0116 | 04 | Archive I/O end tape |
| 0117 | 04 | Begin read archive |
| 0174 | 03 | Begins archive log mode (QUIESCE) command processing |
| 0175 | 03 | Ends archive log mode (QUIESCE) command processing |
| 0252 | 10 | Begins GBP request |
| 0260 | 10 | End GBP request |
| 0265 | 31 | Begin SCA request |
| 0266 | 31 | End SCA request |
| 0267 | 31 | Begin CF rebuild request |
| 0268 | 31 | End CF rebuild request |
| 0364 | 01 | New address space being created or terminated |

Class 4 : reserved by IBM for serviceability

| IFCID | RMID | Event |
|--------------|-------------|---|
| 0106 | 26 | System parameters in effect at trace invocation |

Class 5 : work file and SQL parsing global trace

| IFCID | RMID | Event |
|--------------|-------------|---------------------------------------|
| 0190 | 22 | Overflow condition during hybrid join |
| 0249 | 14 | Monitor DBD invalidations |

Class 7 : distributed data facility global trace

| IFCID | RMID | Event |
|--------------|-------------|--------------------------|
| 0164 | 27 | Distributed transaction |
| 0165 | 27 | VTAM macro calls/returns |

Class 8 : distributed data facility global trace

| IFCID | RMID | Event |
|--------------|-------------|----------------------------|
| 0168 | 22 | Distributed SQL statements |

Class 9 : DRDA distributed data facility global trace

| IFCID | RMID | Event |
|-------|------|--|
| 0180 | 27 | VTAM buffer list entries that are sent or received |
| 0181 | 29 | DDM level 6B object generation |
| 0182 | 29 | FDOCA objects |

Class 10 : DBM1 storage usage global trace

| IFCID | RMID | Event |
|-------|------|---|
| 0217 | 20 | Detailed information about DBM1 storage usage |

Using the statistics trace for distributed diagnosis

The statistics class 4 can be used to diagnose Distributed Relational Database Architecture (DRDA) and distributed two-phase commit exception conditions.

For DRDA, the statistics class 4 trace writes the following records:

1. IFCID 0191

This record is written by DDIS when DDIS identifies a DRDA exception condition, or on behalf of either DTM or DRDS when those managers identify a DRDA exception condition.

2. IFCID 0192

This record is written by DCRM it identifies a DRDA exception condition.

3. IFCID 0193

This record is written by DTM when it identifies a DRDA exception condition.

4. IFCID 0194

This record is written by DCRM when it identifies a DRDA exception condition.

5. IFCID 0195

This record is written by DRDS when it identifies a DRDA exception condition.

The statistics class 4 trace writes these records for the distributed two-phase commit process: IFCIDs 0203 - 0210 and 0234 - 0238.

Related concepts

[Exception condition diagnostic procedures](#)

Several exception condition diagnostic procedures are available.

[Distributed two-phase commit error conditions](#)

Db2 support of distributed two-phase commit provides the detection of a number of events that can have a negative impact on data availability or that indicate or can lead to data inconsistency.

Using the performance trace for diagnosis

While the performance trace is primarily a tuning aid, it can sometimes be useful for diagnosis because degraded performance can also indicate a Db2 problem, one for which you would specify the PERFM keyword in building a keyword string to describe the problem.

You can use the performance trace to determine whether degraded performance represents such a problem or the need for subsystem tuning. Often degraded performance is a symptom of a WAIT or LOOP problem, and the performance trace can help you make this determination. In addition, performance trace classes 6 and 7 can help you detect locking problems.

Related concepts

[Performance trace \(Db2 Performance\)](#)

Trace field descriptions

You can find descriptions of trace records in the IFCID flat file (DSNWMSGs).

The DSNWMSGs file is available in the following locations:

- The most current version of DSNWMSGs is available only for clients who have Db2 13 for z/OS licenses. The information is in a PDF file. To locate this information, see [Db2 13 for z/OS IFCID flat file \(DSNWMSGs\)](#).
- An older version of DSNWMSGs is available in the *prefix*.SDSNIVPD(DSNWMSGs) data set. You can use the TSO or ISPF browse function to look at the field descriptions in *prefix*.SDSNIVPD(DSNWMSGs), even when Db2 is down. If you prefer to look at the descriptions in printed form, you can use ISPF to print a listing of the data set.

Some of the field descriptions in the DSNWMSGs file are marked with "S"; that indicates that those fields are to be used only for diagnosis, modification, and tuning.



Attention: Do not use diagnosis, modification, and tuning information as a programming interface.

You can load the DSNWMSGs file into a user-defined Db2 table. This approach has the advantage of providing you with access to the IFCID field descriptions through SQL SELECT statements in whatever order or format you choose. You can also use the power of SQL to tailor the information to meet the needs at your site before you print it. Because of the wide variation from customer to customer in how Db2 trace data is analyzed, you might find this method most convenient. DSNWMSGs contains sample SQL statements for creating a user-defined Db2 table and retrieving information from it, as well as LOAD utility control statements for populating the table.

Related concepts

[Types of Db2 traces \(Db2 Performance\)](#)

Related tasks

[Controlling traces \(Db2 Administration Guide\)](#)

Related reference

[-START TRACE command \(Db2\) \(Db2 Commands\)](#)

Call attachment facility traces

The call attachment facility, which enables users to connect to Db2 through TSO foreground, TSO background, or z/OS batch, provides diagnostic trace messages intended primarily for use by IBM service personnel.

The following topics describe and give an example of the call attachment facility trace stream.

Related reference

[Call attachment facility trace messages](#)

An application programmer that uses the call attachment facility (CAF) to write an application that attaches to Db2 can choose whether the messages are displayed or not. They display if the DSNTRACE *ddname* is allocated during CAF execution.

Producing trace messages

Call attachment facility trace messages are always included in the call attachment facility's trace table in SYSABEND or SYSUDUMP dumps. However, these trace messages can also be sent to a TSO terminal or to SYSOUT.

Procedure

Before starting an application, allocate *ddname* DSNTRACE.

DSNTRACE can be allocated to a TSO terminal or to SYSOUT. In TSO foreground, issue FREE DD(DSNTRACE) to send SYSOUT data to the printer.

Do not allocate DSNTRACE to a data set unless the application only connects to Db2 from a single task. The call attach DSNTRACE does not function in a multi-tasking environment because the call attachment facility does not serialize access to the trace data set. Writing trace records (SVC6F, DECIMAL SVC111) to the same JES2 SYSOUT data set results in ABEND378.

Related tasks

Activating the DSN trace stream (Troubleshooting problems in Db2)

Finding the trace table

SYSABEND or SYSUDUMP dumps of the call attachment facility address space publish a wrap-around trace table that includes trace messages, if the DSNTRACE data set was allocated at the time of the abend.

About this task

Each message is preceded by the eye-catcher >>>>. followed by accumulative number indicating the relative sequence in which the trace message was produced. Use this number to determine the most recent events if the trace table has wrapped.

Procedure

To find the trace table:

1. Locate register 6 at time of error. This contains the address of the call attachment facility control block (CAB).
Locate the "RTM2WA SUMMARY" section in the first few pages of the dump. Register 6 is published near the heading that reads "REGS AT TIME OF ERROR".
2. Find the beginning of the CAB. The eye-catcher CAB appears at right.
3. Scan down about 1,000 bytes into the CAB. The trace table appears nearby, preceded by the eye-catcher TRACE TABLE >>>>.

Interpreting trace messages

One class of messages is particularly significant because it helps identify where an error occurred (in Db2 or in an application address space, such as the call attachment facility).

This class of messages contains a line of equal signs and indicates events that occurred before and after calls to the Db2 program request handler. These messages occur in BEFORE/AFTER pairs:

```
BEFORE some event =====
AFTER  some event =====
```

After the BEFORE message is written, control passes from the call attachment facility to Db2. After the AFTER message is written, control has returned from Db2. If an abend occurs just after a DSNAB10I BEFORE message is written, the problem probably is in Db2. Otherwise, the problem is probably in the call attachment facility or an application.

The following figure illustrates a stream of trace messages.

```
DSNA800I DSNACA90 TCB=008F28E8 ENTERED DSNACA80 ACTION=CONNECT R1=000002A0 CABPTR=00014360 CABFLAG1=0B CABFLAG2=00
DSNA800I DSNACA90 TCB=008F28E8 CONNECT REQUEST          SSID=AAAA  TECBP=0000740C  SECBP=00007408  RIBW=00007420
DSNA826I DSNACA90 TCB=008F28E8 BEFORE VALIDITY CHECKING THE START-UP ECB AT ADDRESS 00007408
DSNA827I DSNACA90 TCB=008F28E8 AFTER  VALIDITY CHECKING THE START-UP ECB WHICH CONTAINS 00000000
DSNA826I DSNACA90 TCB=008F28E8 BEFORE VALIDITY CHECKING THE TERMINATE ECB AT ADDRESS 0000740C
DSNA827I DSNACA90 TCB=008F28E8 AFTER  VALIDITY CHECKING THE TERMINATE ECB WHICH CONTAINS 00000000
DSNA825I DSNACA70 TCB=008F28E8 CASSID=AAAA CABREL=318 CABSECP=00000000
DSNA813I DSNACA70 TCB=008F28E8 FRBP=000144F8 RAL=00000000 RALE=0001  FVLE=0002  PARM=00000000  PCNT=0000
DSNA814I DSNACA70 TCB=008F28E8 RC1=0000  RC2=00000000  FBACK=00000000  RHPC=00000000  QUAL=0001  RSV1=00
DSNA810I DSNACA70 TCB=008F28E8 BEFORE IDENTIFY =====
DSNA811I DSNACA70 TCB=008F28E8 AFTER  IDENTIFY =====
DSNA813I DSNACA70 TCB=008F28E8 FRBP=000144F8 RAL=00000000 RALE=0001  FVLE=0002  PARM=00000000  PCNT=0000
DSNA814I DSNACA70 TCB=008F28E8 RC1=0000  RC2=00000000  FBACK=00000000  RHPC=00000000  QUAL=0001  RSV1=00
DSNA828I DSNACA70 TCB=008F28E8 RIB ADDRESS=00BF220  RIBPTR ADDR=00007420
DSNA813I DSNACA70 TCB=008F28E8 FRBP=000144F8 RAL=00000000 RALE=0001  FVLE=0000  PARM=00000000  PCNT=0000
DSNA814I DSNACA70 TCB=008F28E8 RC1=0000  RC2=00000000  FBACK=00000000  RHPC=00000000  QUAL=0001  RSV1=00
DSNA810I DSNACA70 TCB=008F28E8 BEFORE ESTABLISH EXIT =====
DSNA811I DSNACA70 TCB=008F28E8 AFTER  ESTABLISH EXIT =====
DSNA813I DSNACA70 TCB=008F28E8 FRBP=000144F8 RAL=00000000 RALE=0001  FVLE=0000  PARM=00000000  PCNT=0001
DSNA814I DSNACA70 TCB=008F28E8 RC1=0000  RC2=00000000  FBACK=00000000  RHPC=00000000  QUAL=0001  RSV1=00
DSNA801I DSNACA90 TCB=008F28E8 LEAVE DSNACA90 CABFLAG1=CF CABFLAG2=00 CABPTR=00014360
DSNA800I DSNACA90 TCB=008F28E8 ENTERED DSNACA80 ACTION=OPEN R1=000002A0 CABPTR=00014360 CABFLAG1=CF CABFLAG2=00
DSNA808E DSNACA90 TCB=008F28E8 CANNOT HAVE 2 SSIDS (AAAA AND BBBB) FROM 1 TCB
DSNA807I DSNACA90 TCB=008F28E8 OPEN REQUEST          SSID=BBBB  PLAN=PLAINAME
DSNA801I DSNACA90 TCB=008F28E8 LEAVE DSNACA90 CABFLAG1=CF CABFLAG2=00 CABPTR=00014360
```

Figure 67. Sample call attachment facility trace stream

Each of the four wall messages shown is either preceded by or followed by a dump of the FRB (this is dummy data). Further, message DSN208E shows that the user committed an error: the user specified two different SSIDs on the CONNECT and OPEN calls. CAF user error messages begin with DSN2... .

If a Db2abend occurred during establish exit, then no trace messages appear after the second BEFORE message (DSNA810I . . . BEFORE ESTABLISH EXIT =====).

Related reference

Call attachment facility trace messages

An application programmer that uses the call attachment facility (CAF) to write an application that attaches to Db2 can choose whether the messages are displayed or not. They display if the DSNTRACE *ddname* is allocated during CAF execution.

IMS attachment facility traces and IMS log record

It is important to determine whether a problem is in the application program, IMS, the IMS attachment facility, or Db2.

The IMS attachment facility maintains two trace tables in storage. For many problems involving the IMS attachment facility, these two tables can be found in:

- An IMS attachment facility '5501' log record, or
- The WAL or WAU control blocks in a dump

IMS log entries

The IMS log contains entries (with an identifier of '5501FF') that are written by the IMS attachment facility.

The entries are written whenever one of the following unexpected error situations occurs:

- Request is denied because of prohibited circumstance.
- Request is denied because of an invalid sequence of requests.
- Request is denied because of a parameter error.

When these records are written to the IMS log, they contain connection information and a 20-entry wraparound trace table called WALTAREA. (This same trace area is also in the WAL and WAU.)

The connection information in each of these records includes:

- Time and date stamp
- IMSID
- Db2 subsystem ID
- PST number
- PSB name
- Application program name scheduled
- Plan name
- Authorization ID
- Recovery token
- Call type
- Region information.

The IMS log also contains entries (with identifiers of '5501FE') that the IMS attachment facility writes whenever a unit of recovery is processed during resolve indoubt processing. Each of these records includes:

- A time and date stamp
- Db2 subsystem ID
- FRB return code
- FRB reason code
- Recovery token status

Recovery token
 Resource name
 Correlation token.

IMS attachment facility X'5501FF00' log record

The IMS attachment facility writes the X'5501FF00' log record when unexpected error conditions occur between Db2 and the IMS attachment facility.

The format of the X'5501FF00' log record is shown in the following figure.

```

****  SNAP LOG RECORD X'5501FF00' START
LOC   NAME          SIZE      DESCRIPTION
000   WALSF  F      IMS LOG RECORD
      WALSF  F      - 5501FF FOR DB2
000   WALSF  FLN  FIXED(16) LENGTH OF THE RECORD
002   WALSF  FZZ  FIXED(16) RESERVED
004   WALSF  FID  FIXED(32) RECORD CODE X'5501FF00'
008   WALSF  FDT  CHAR(4)   DATE
00C   WALSF  FTM  CHAR(4)   TIME
010   WALIM  SCN  CONNECTION NAME TO DB2
010   WALIM  SID  CHAR(8)   IMSID OR VTAM ID
      (CONNECTION NAME)
018   WALSS  ID  CHAR(4)   DB2 SUBSYSTEM ID
01C   WALDISP  CORRELATION ID
01C   WALP  STD  CHAR(4)   PST DISPLAY TOKEN
020   WALP  SBN  CHAR(8)   PSB EBCDIC VALUE
028   WALAP  N  CHAR(8)   APPLICATION PROGRAM NAME
030   WALRES  N  CHAR(8)   RESOURCE NAME (PLAN)
038   WALAI  D  CHAR(8)   AUTHORIZATION ID
040   WALWF  FIXED(32) CREATE THREAD WEIGHT
044   WALR  C  RECOVERY TOKEN FOR ATTACH.
044   WALNI  D1  NETWORK ID-VTAM NODE NAME
      OR BATCH CONNECTION NAME
044   WAL1  SI  CHAR(4)   OR IMS SUBSYSTEM ID
048   WAL1  SN  CHAR(4)   IMS SUBSYSTEM ID CONTINUED
04C   WALNI  D2  NETWORK ID - UNIQUE NUMBER
      EITHER IMS OR BATCH SUPPLIED
04C   WALNI  D2A  FIXED(32) IMS SCHEDULE NUMBER (ONLINE)
050   WALNI  D2B  FIXED(32) IMS COUNTERS (ONLINE)
054   WALCT  YPE  CHAR(4)   CALL TYPE INDICATOR
058   WALSCH  F  SCHEDULING FLAGS
058   WALREC  F  CHAR(1)   U=UPDATES, N=NO UPDATES
059   WALMOD  F  CHAR(1)   MODE S=SINGLE, M=MULTIPLE.
05A   WALREG  T  BIT(8)    REGION TYPE
05B   WALREG  O  CHAR(1)   DEFAULT REGION SSM VALUE
05C   WALTE  YE  CHAR(4)   EYE CATCHER ID OF TRC7
060   WALTARE  A  CHAR(640)  TRACE AREA
2E0   *  FIXED(15)  RESERVED
2E2   WALER  L  FIXED(15)  ERROR LENGTH
2E4   WALER  N  *  ERROR VALUE - CREATE THREAD
****  SNAP LOG RECORD X'5501FF00' END

```

Figure 68. Format of X'5501FF00' log record

The format and content of each trace record in the WALTAREA trace is described in [“Contents of the IMS attachment facility trace table”](#) on page 257.

IMS attachment facility X'5501FE00' log record

The IMS attachment facility writes the X'5501FE00' log record any time resolve indoubt processing occurs between Db2 and the IMS attachment facility.

The format of the X'5501FE00' log record is shown in the following figure.

```

****  SNAP LOG RECORD X'5501FE00' BEGINNING
LOC   NAME          SIZE      DESCRIPTION
000   WALSFE          4          INDOUBT SNAP LOG RECORD
000   WALSFE LN      FIXED(16)  LENGTH OF THE RECORD
002   WALSFEZZ       FIXED(16)  RESERVED
004   WALSFEID       FIXED(32)  ID X'5501FE00'
008   WALSFE DT      FIXED(32)  INDICATES DATE.
00C   WALSFE TM      FIXED(32)  INDICATES TIME.
010   WALSFE SI     CHAR(4)     DB2 SUBSYSTEM NAME
014   WALSFE R1     CHAR(2)     RETURN CODE FROM DB2
016   WALSFE CT     FIXED(15)  COUNTS INDOUBT PROCESSED.
                                A COUNT OF THE NUMBER OF
                                TIMES RESOLVE INDOUBT
                                PROCESS HAS BEEN CALLED WITH
                                L (LAST) SPECIFIED. ALSO
                                USED AS A FLAG VALUE TO
                                DETERMINE THE FIRST TIME VS
                                THE OTHER
018   WALSFE R2     CHAR(4)     REASON CODE FROM DB2
01C   WALRIP        CHAR          RESOLVE INDOUBT PARM AREA
01C   RIURRQST     CHAR(4)     RECOVERY RESOLUTION REQUEST
                                'SHOW' - REQUEST UR
                                INFO
                                'IDBT' - STORED ON
                                GOOD SHOW
                                'COMM' - REQUEST UR
                                COMMIT
                                'ABRT' - REQUEST UR
                                ABEND
                                ' ' - ANYTHING
                                ELSE
                                - ERROR
020   RIURCRID     CHAR(12)    THREAD LEVEL CORRELATION_ID -
                                PROVIDED AS FEEDBACK
                                ON 'SHOW'
02C   RIURNID      CHAR(16)    UR LEVEL NETWORK_ID -
                                PROVIDED AS FEEDBACK
                                ON SHOW AND USED TO
                                IDENTIFY URS ON
                                COMM/ABRT
02C   RIURNIDC     CHAR(8)
034   RIURNIDB     BIT(64)
03C   RIURRESN     CHAR(8)     RESOURCE NAME PROVIDED AS
                                FEEDBACK ON 'SHOW'
044   RIURTOKN     BIN(16)    TOKEN FOR SHOW-NEXT-NID
046   RIURICNT     BIN(16)    NUMBER OF INDOUBTS FOR
                                CONNECTION
048   RIURCONN     CHAR(8)     CONNECTION NAME
                                PROVIDED AS FEEDBACK
                                ON SHOW AND USED TO
                                IDENTITY URS
                                ON COMM/ABRT
****  SNAP LOG RECORD X'5501FE00' END

```

Figure 69. Format of X'5501FE00' log record

IMS attachment facility trace diagnosis guidelines

Several steps can be taken to use the IMS attachment facility trace to help diagnose problems.

1. To determine if the problem occurred in the application program, see if the address in the PSW at the time of error falls within the address range of the application program.
2. Try to determine the cause of failure by reviewing abend completion code at the beginning of the dump.
 - If the abend completion code indicates a user abend, then try to find an explanation for that user abend completion code in the IMS messages and codes publication.
 - If the abend completion code is a system X'04E' or X'04F', locate the Db2 abend reason code value in register 15 and look up the reason code value in [Db2 reason codes \(Db2 Codes\)](#).
3. Locate the application program's last (most recent) save area.
4. Determine the last event the application program performed before the failure.

5. If the last event was a SQL call, locate the highest sequence number in the IMS attachment facility trace table. (The entry with the highest sequence number is the most recent.) Look at the WALTAREA trace area first to find the connection status.
6. Analyze that entry to see at what control status point the call is, by using the information in [Table 35 on page 259](#).
7. If the trace entry is complete (control status points A and E are nonblank), then analyze the WAU trace.
8. Looking at these trace tables should help in identifying where the problem occurred (that is, if it occurred in Db2, in the IMS attachment facility, or in IMS), and it should help identify the events that were occurring at the time of the failure (for example, a trace identifier of TRMT indicates that a thread has been terminated).
9. The Db2-DL/I batch support is packaged as part of the Db2 IMS attachment facility. Follow the procedures in ["Type-of-failure keywords" on page 12](#) when reporting problems with the Db2-DL/I batch support. Abends produced by the Db2-DL/I batch support are either SYSABEND or SYSUDUMP dumps. A SYS1.LOGREC entry is normally not produced.

Contents of the IMS attachment facility trace table

When an application program makes a call to Db2 (via an SQL request), resources can be allocated (plan ID) and the call can be processed.

The IMS attachment facility traces events, called *control status points*, such as resources being allocated, in the WALTAREA; the normal application call path is traced in the WAU trace table.

The following sections describe the format and contents of individual trace entries and indicate when individual fields within a trace entry are updated.

Sequence of IMS Trace Activities

The control status points are keys into the following table.

Control status point A

When IMS invokes the IMS attachment facility, the 32-byte trace entry is blanked out and replaced by an exit ID and unique ascending sequence number.

Control status point B

Just before the IMS attachment facility calls Db2, the call function fields are updated.

Control status point C

When Db2 returns control to the IMS attachment facility, the status fields are updated.

Control status point D

If the attachment facility calls one or more subroutines, the subroutine information is traced after the subroutine completes.

Control status point E

The return code is updated when the IMS attachment facility returns control to IMS.

Table 34. Format of IMS attachment facility trace entry

| Value and length | Trace ID | Sequence number | Call function | Call Status | Subroutines | RC |
|-------------------------|----------|-----------------|---------------|-------------|-------------|----|
| Filled in at CSP | A | A | B | C | D | E |
| Length in bytes | 4 | 2 | 6 | 6 | 12 | 2 |

Notes:

The control status points (CSP) of A, B, C, D, and E are defined as follows:

Trace identification, sequence number at CSP A: Each entry in the trace table begins with an "eye-catcher" to help identify the various trace identification field values in the trace table.

Trace identification:

- For the WAL trace, the trace identification field can be set to these events:

ABRT

Abort continue (rollback) a unit of recovery.

CMDO

Process a Db2 command.

COMC

Commit continue (phase 2) a unit of recovery.

CTHD

Create a thread to Db2.

ECHO

Determine if Db2 is operational.

ID00

Identify: Initialize a connection from the IMS application to Db2.

INIT

Initialize the IMS attachment facility work areas.

PREP

Prepare to commit (phase 1) a unit of recovery.

RI00

Resolve indoubt work units between IMS and Db2.

SIGN

Sign on a new user to Db2.

SOFF

Sign off a user to Db2.

SUBT

Handle subsystem termination of Db2 or IMS.

TRMI

Terminate Identify: Terminate the connection to Db2.

TRMT

Terminate the thread between Db2 and IMS.

- For the WAU trace, the trace identification field can be set to these events:

NORC

Normal call to process a SQL request.

SNOO

System not operational to handle exceptional conditions.

Sequence number:

The sequence number following the trace identifier reflects the order in which the trace entry was recorded in relation to the other trace entries. This sequence number can then be used to determine the correct sequence of events when the trace table wraps.

Call function at CSP B

Indicates that a request to Db2 has been made. This code is used internally; note only whether the value is blank or nonblank.

Call status at CSP C

Indicates that the request to Db2 has been completed and that control has returned to the IMS attachment facility. This Db2 return code and reason code is returned to the IMS Attach from Db2 for upon completion of the event.

Subroutine at CSP D

Called identification (traced after call to routine). This information is generally not applicable to diagnosing problems.

Return code at CSP E

The IMS attachment facility return code from the exit.

The character string "TRCx" precedes the first trace entry in each of the trace tables (WAL and WAU). Refer to The following table for information about the combinations of field values in a trace entry and about what these might indicate.

Table 35. IMS attachment facility trace control status points

| Control Status Points | | | | | |
|-----------------------|-----|-----|-----|---|---|
| A | B | C | D | E | Remarks |
| n | b | b | n/a | b | The call has entered the IMS attachment facility. |
| n | n | b | n/a | b | The call is in Db2. |
| n | n | n | n/a | b | The call has returned from Db2 and is in the IMS attachment facility or in a subroutine. |
| n | n/a | n/a | n/a | n | <ul style="list-style-type: none">• The call has returned to IMS,• The call is in IMS and never made it to the IMS attachment facility, or• The entry is a residual trace entry. |

Notes:

n

Represents nonblank fields.

b

Represents blank fields (that is, X'40').

n/a

indicates that this field contains information that is not applicable.

Locating the IMS attachment facility trace tables in a dump

The WAL and DSNMWAWU load modules belong to the region controller task. To be certain these trace tables are included in the dump, stop the region with a dump.

The formats for the WAL and the DSNMWAWU trace tables are the same except for two ID fields. The DSNMWAWU data is primarily about applications. The format of the information in the WAL load module includes connection information, the 20-entry trace table (WALTAREA), and a work area used when problems occur due to unavailable resources. For details, see [Table 34 on page 257](#). The trace table is always active.

The IMS attachment facility load modules containing these trace areas are:

WAL

The trace table in the IMS control region and in the IMS dependent region that indicates the control status of the connection between IMS and Db2.

DSNMWAWU

The trace table exists in the dependent region that indicates the application call status.

To find these trace tables in a dump, refer to [Table 35 on page 259](#) and [Figure 71 on page 261](#) and follow the steps below:

1. Locate the CDE for the IMS message region involved in the problem.
2. Find the names **WAL** and **DSNMWAWU** in the list of load modules and the WAL and DSNMWAWU addresses.

IMS attachment facility messages

When the attachment between Db2 and IMS is run in an IMS online environment, the attach messages are issued to the master terminal operator (MTO) and diagnostic information is issued to the IMS log.

In the batch environment, the Db2 IMS attach messages and IMS attach diagnostic information are issued to the DDOTV02 output data set.

IMS attachment facility message format

The IMS attachment facility writes messages when the attach package connects to or disconnects from Db2, and when any resolve indoubt problems occur.

The format of the attach messages is shown in the following figure.

| LOC | NAME | SIZE | DESCRIPTION |
|-----|------|-----------|----------------------|
| 000 | | FIXED(15) | LENGTH OF THE RECORD |
| 002 | | FIXED(15) | RESERVED |
| 004 | | CHAR(*) | ACTUAL MESSAGE TEXT |

Figure 70. IMS attachment facility message format

When abends occur

Abends can occur during Db2 command request processing or during Db2 non-command-related requests.

Abends during Db2 command request processing

An attachment facility CSECT in the IMS control region establishes an ESTAE routine when it processes a Db2 command request. If an abend occurs during the processing request, the ESTAE routine writes an entry to the SYS1.LOGREC data set. This entry includes information in all of the standard fields of a SYS1.LOGREC entry (described in the z/OS diagnostic techniques publication). In addition, information is recorded in the variable recording area (VRA), which includes:

- Db2 abend reason code from the IMS attachment facility
- Active trace entry of the IMS attachment facility
- The command module autodata area.

Abends during Db2 non-command-related requests

If an abend occurs when Db2 is not processing command requests, the IMS attachment facility does not establish a recovery environment; therefore, any abends that occur are handled by IMS.

```

CDE
676F78      NCDE 00000000  RBP 00676AD0  NM DFSRRCC00  EPA 000F5E78  XL/MJ 006AC918  USE 000100FB  ATTR 3B20000
676070      NCDE 006760A0  RBP 0069C980  NM DFSRRCC40  EPA 000F95B6  XL/MJ 00676060  USE 000200FB  ATTR 3B20000
676F58      NCDE 00676070  RBP 00676920  NM DFSPPCC20  EPA 000F50F0  XL/MJ 0069C6C0  USE 000200FB  ATTR 3B20000
676F58      NCDE 00676070  RBP 00676920  NM DFSPPCC20  EPA 000F50F0  XL/MJ 0069C6C0  USE 000200FB  ATTR 3B20000
6768A0      NCDE 00676030  RBP 00676818  NM FAFI5022  EPA 000FED16  XL/MJ 00676000  USE 000100E6  ATTR 0B20000
FD7880      NCDE 00FD7730  RBP 00000000  NM IGG019DK  EPA 00F88000  XL/MJ 00FD78A0  USE 00010000  ATTR 0B22000
66F860      NCDE 006763F8  RBP 00000000  NM DFSFSAW0  EPA 000F6390  XL/MJ 006768C0  USE 000100FB  ATTR 0320000
6763F8      NCDE 0066FE10  RBP 00000000  NM DSNMAPNX  EPA 00A4AE68  XL/MJ 006763E8  USE 000100F1  ATTR 03A2000
66FE10      NCDE 0066F720  RBP 00000000  NM DSNMWAU  EPA 000F6C10  XL/MJ 0066FE30  USE 000100FB  ATTR 0222000
66F720      NCDE 0066F760  RBP 00000000  NM DSNMBSRH  EPA 00670100  XL/MJ 0066F710  USE 000100E6  ATTR 31A2000
66F760      NCDE 0066F7A0  RBP 00000000  NM DSNMERR0  EPA 0066BD30  XL/MJ 0066F750  USE 000100E6  ATTR 31A2000
66F7A0      NCDE 0066F7D0  RBP 00000000  NM DSN31D00  EPA 0066C298  XL/MJ 0066F790  USE 000100E6  ATTR 39A2000
66F7D0      NCDE 00676228  RBP 00000000  NM DSNAPRH  EPA 00670318  XL/MJ 00676208  USE 000100E6  ATTR 31A2000
676228      NCDE 00676278  RBP 00000000  NM WAL  EPA 0066C5000  XL/MJ 00676248  USE 000100E6  ATTR 02A2000
676278      NCDE 006762B8  RBP 00000000  NM DSNMSB00  EPA 0066CB80  XL/MJ 00676268  USE 000100E6  ATTR 31A2000
6762B8      NCDE 006762F8  RBP 00000000  NM DSNMDR10  EPA 0066D280  XL/MJ 006762A8  USE 000100E6  ATTR 31A2000
6762F8      NCDE 00676338  RBP 00000000  NM DSNMDR00  EPA 0066D938  XL/MJ 006762E8  USE 000100E6  ATTR 31A2000
676338      NCDE 00676378  RBP 00000000  NM DSNMID00  EPA 0066E108  XL/MJ 00676328  USE 000100E6  ATTR 31A2000
676378      NCDE 006763A8  RBP 00000000  NM DSNMINIT  EPA 0066EAF8  XL/MJ 00676368  USE 000100E6  ATTR 31A2000
6763A8      NCDE 006768A0  RBP 00000000  NM DFSEEV7  EPA 00679048  XL/MJ 006763C8  USE 000100E6  ATTR 02A2000
FD7730      NCDE 00FD7760  RBP 00000000  NM IGG019DJ  EPA 00B28118  XL/MJ 00FD7750  USE 00030000  ATTR 0B22000
676030      NCDE 00676F58  RBP 00000000  NM DFSECP20  EPA 000F90C8  XL/MJ 00676020  USE 000100FB  ATTR 3320000
6760A0      NCDE 0069C3A0  RBP 00000000  NM DIRCA001  EPA 000F9748  XL/MJ 006760C0  USE 000100FB  ATTR 0222000
69C3A0      NCDE 006760E0  RBP 00000000  NM DIRCL001  EPA 00674000  XL/MJ 0069C070  USE 000100E6  ATTR 02A2000
6760E0      NCDE 00676120  RBP 00000000  NM WALESTR  EPA 00675310  XL/MJ 00676100  USE 000100E6  ATTR 02A2000
676120      NCDE 0069C000  RBP 00000000  NM EVTPSSTR  EPA 00679190  XL/MJ 00676140  USE 000100E6  ATTR 02A2000
69C000      NCDE 0069C040  RBP 00000000  NM WALEDSN  EPA 00675988  XL/MJ 0069C020  USE 000100E6  ATTR 02A2000
69C040      NCDE 0069C0C0  RBP 00000000  NM EVTPDSN  EPA 00679928  XL/MJ 0069C060  USE 000100E6  ATTR 02A2000
69C0C0      NCDE 0069C090  RBP 00000000  NM DFSSTR  EPA 000FD800  XL/MJ 00676FA0  USE 000100FC  ATTR 3122000
69C090      NCDE 0069C160  RBP 00000000  NM DFSLESE  EPA 00679839  XL/MJ 0069C0B0  USE 000100E6  ATTR 02A2000
69C360      NCDE 00676FC0  RBP 00000000  NM ESIESDCB  EPA 006798A8  XL/MJ 0069C380  USE 000100E6  ATTR 02A2000
676FC0      NCDE 0069C690  RBP 00000000  NM DFDDPD10  EPA 00679968  XL/MJ 00676FE0  USE 000100E6  ATTR 02A2000
69C690      NCDE 0069C940  RBP 00000000  NM DFSVC000  EPA 000F6869  XL/MJ 0069C680  USE 000100FB  ATTR 0320000
69C940      NCDE 00676F78  RBP 00000000  NM DFSRPRX0  EPA 000F6898  XL/MJ 0069C930  USE 000100FB  ATTR 1320000

LPA/JPA MODULE  WAL
66C500      000205C0  E6C1D340  0066C298  00670318  00000000  0066BD30  00000000  00670100  *...WAL..B.....*
66C520      000F6C10  03480000  5501FF00  00000000  00000000  E2E8E2F3  40404040  E2E2E3D9  *.....SYS3  SSTR*
66C540      F0F0F0F1  C4C6D2C6  F0F0F0F6  C6C1C6C9  F5F0F2F2  C6C1C6C9  F5F0F2F2  C4C6D2C6  *0001DFKF0006FAFI5022FAFI5022DFKF*
66C560      F0F0F0F6  00000004  E2E8E2F3  40404040  0000000C  00000001  E2D8D340  E4D404D9  *0006...SYS3  ....SQL UM.R*
66C580      E3D9C3F7  C9D5C9E3  00014040  40404040  40404040  40404040  40404040  40404040  *TRC7INIT..*
66C5A0      40400000  C9C4F0F0  00020001  00020001  00000000  00004040  40404040  40404040  *...ID00.....*
66C5C0      40400000  E2C9C7D5  00030001  00030001  00000000  00004040  40404040  40404040  *...SIGN.....*
66C5E0      40400000  C3E3C8C4  00040001  00040001  00000000  0000C2C8  00040404  40404040  *...CTHD.....BH..*
66C600      40400000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  *.....*
66C620      00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  *.....*

LINES 66C640-66C840 SAME AS ABOVE
66C860      00000000  00000000  00000000  00000000  C6D9C2F7  C6D9C240  00010004  0066C9FC  00060000  *.....FRB7FRB  ....I....*
66C880      00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  *.....*
66C8A0      00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  *.....*
66C8C0      00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  *.....*
66C8E0      0066C604  0066C7E4  00000004  00500000  5501FE00  00000000  00000000  00000000  *..F...GU.....SSTR*
66C900      00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  *.....*

LINE 66C920 SAME AS ABOVE
66C940      00000000  00000000  00000000  C3000000  00000000  00000000  00000000  00000000  *.....C.....*
66C960      00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  *.....*

LINE 66C980 SAME AS ABOVE
66C9A0      00000000  00000000  E2C1E2F7  00000000  00000000  006758BC  00000000  5066D42C  *.....SAS7.....M.*
66C9C0      00670318  00000001  0066C9F8  0066C564  0066C500  0066CB70  00679190  00A15A58  *.....18..E..E...H.....*
66C9E0      00A15A74  0067538C  00A15050  0066D857  0066C9F8  0066D292  0066C870  0066C554  *.....Q...18  K...H...E.*
66CA00      0066CA18  0066C540  0066D857  0066C808  8066C564  0000001C  E8C5E2F0  40F0F0F1  *.....E...Q...H...E.....YES0 001*
66CA20      00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  *.....*

```

Figure 71. Sample IMS attachment facility trace

Resource Recovery Services attachment facility traces

The Resource Recovery Services attachment facility (RRSAF), which enables users to connect to Db2 using Resource Recovery Services (RRS), provides diagnostic trace messages intended primarily for use by IBM service personnel.

Starting the RRSAF trace

If the RRSAF trace is started, messages are included in the RRSAF's trace table in SYSABEND or SYSUDUMP dumps.

Procedure

Allocate a data set with DD name DSNRRSAF before you start an RRSAF application.

For example, if you use a batch job to start an application, include a DD statement like the following:

```
//DSNRRSAF DD DUMMY
```

After you start the trace, you cannot turn it off. Because the trace can degrade performance, you should start it only when you need to diagnose a problem.

Finding the trace table

SYSABEND or SYSUDUMP dumps of the application address space include a wrap-around trace table with a maximum of 20 trace messages.

Procedure

To find the trace table:

1. Search the dump for the eye-catcher RRB.
2. Look past this eye-catcher for the characters RRSAF TRACE>>>>. Examine the eight bytes that immediately precede those characters. The first four bytes contain the number of entries in the trace table. The second four bytes contain the index of the last entry written.

Interpreting trace messages

Use the RRSAF trace messages to help diagnose a problem.

Each entry in the trace table has the format shown in the following table.

Table 36. Description of an RRSAF trace record

| Bytes | Contents of Field |
|-------|---|
| 1-4 | Abbreviation for the RRSAF function performed: <ul style="list-style-type: none">• 'SQL ' - SQL operation• 'IFCA' - IFC API call• 'IFCX' - IFC Translate call• 'IDEN' - IDENTIFY• 'CTHD' - CREATE THREAD• 'TTHD' - TERMINATE THREAD• 'TIDY' - TERMINATE IDENTIFY• 'SIGN' - SIGNON• 'ASIN' - AUTH SIGNON• 'XLAT' - TRANSLATE• 'S_ID' - SPAS_ID• 'S_IN' - SPAS_INIT_SP• 'S_TM' - SPAS_TERM_SP• 'S_SV' - SPAS_SERV_SP• 'S_DS' - SPAS_DISC• 'ASSO' - ASSOCIATE THREAD• 'DSSO' - DISSOCIATE THREAD |
| 5-8 | If the function is 'SQL ', the SQLCODE. Otherwise, the RRSAF return code. |
| 9-12 | RRSAF reason code. |
| 13 | Value of flag RRBFLAG1 after function completes |
| 14 | Value of flag RRBFLAG1 before function starts |
| 15 | Value of flag RRBFLAG2 after function completes |
| 16 | Value of flag RRBFLAG2 before function starts |

All functions that begin with SPAS are functions that Db2 performs for stored procedures running in a WLM-established address space.

RRSAF writes one entry before it runs a function and another entry after it runs the function.

After the before entry is written, control passes from RRSF to Db2. After the after entry is written, control has returned from Db2. If an abend occurs just after a before message is written, the problem is probably in Db2. Otherwise, the problem is probably in RRSF or the application.

The following figure shows an example of RRSF trace output in a dump.

```

43301D20 43301BB0 43300CAE 43301BB8 00000000 C3300C08 1 0000000D 2 0000000C 3 D9D9E2C1 *.....C.....RRSA*
43301D40 C640E3D9 C1C3C540 6E6E6E6E C9C4C5D5 00000000 00000000 CB000100 E2C9C7D5 *F TRACE >>>>IDEN.....SIGN*
43301D60 00000000 00000000 CFCB0101 C3E3C8C4 00000000 00000000 EFCF0101 E2D8D340 *.....CTHD.....SQL *
43301D80 00000000 00000000 FFFF0101 E2D8D340 00000000 00000000 FFFF0101 E2D8D340 *.....SQL.....SQL *
43301DA0 00000000 00000000 FFFF0101 E2D8D340 00000000 00000000 FFFF0101 E2D8D340 *.....SQL.....SQL *
43301DC0 00000000 00000000 FFFF0101 E2D8D340 00000000 00000000 FFFF0101 E2D8D340 *.....SQL.....SQL *
43301DE0 00000000 00000000 FFFF0101 E2D8D340 00000000 00000000 FFFF0101 E2D8D340 *.....SQL.....SQL *
43301E00 00000000 00000000 FFFF0101 E2D8D340 00000000 00000000 FFFF0101 00000000 *.....SQL.....*

```

Figure 72. Sample RRSF trace stream

| Figure Label | Description |
|--------------|--|
| 1 | Number of entries in the trace table |
| 2 | Index of the last entry in the trace table |
| 3 | Eye-catcher RRSF TRACE >>>> |
| 4 | Beginning of the first trace record, which records an RRSF IDENTIFY function |
| 5 | Return code for the IDENTIFY function |
| 6 | Reason code for the IDENTIFY function |
| 7 | RRSAF flags |
| 8 | Beginning of the last trace record, which records an SQL statement |

TSO attachment facility traces

The TSO attachment facility provides three tracing mechanisms, DSN trace streaming, CLIST tracing, and SPUFI trace streaming.

SPUFI diagnostic panels and the ABEND subcommand of the DSN command processor are other available diagnostic aids. These tools are helpful for gathering pertinent information about the processing that occurs in the TSO attachment facility; much of that information is helpful to IBM Support in resolving problems that occur.

The Db2 TSO attachment facility tracing maintenance system is intended for use either by IBM personnel or by a customer working with IBM Support.

Related tasks

[ABEND subcommand of the DSN command processor](#)

[The TSO attachment facility provides an ABEND subcommand that enables users to request and obtain a dump.](#)

[SPUFI diagnostic panels](#)

When activated, SPUFI diagnostic panels appear on the terminal as SPUFI is run. These panels display significant variable names and their assigned values (if any). These panels can be examined to verify that the information is correct.

Activating the DSN trace stream

The DSN trace stream consists of trace messages; each represents a specific function that has been or is about to be performed. View these messages on the screen or specify that they be collected in a data set, or both. These messages are especially valuable in diagnosing possible WAITs and LOOPs.

About this task

The DSN trace stream can be activated through explicit DSN commands or through DB2I.

Procedure

To activate the DSN trace stream, choose one of the following approaches:

- Activate the DSN trace stream using explicit DSN commands.
 - a) Start the DSN session with the TEST parameter, on the DSN command, set to a number that is greater than 100. Entering a number greater than 100 results in the tracing of all the CSECTs.

Use the following command:

```
DSN SYSTEM(subsystem_name) TEST(255)
```

where `subsystem_name` is the site-defined name of the subsystem.

A DSN session begins when DSN is entered from the TSO environment. To turn on the DSN trace stream during a DSN session, re-enter the DSN subcommand shown above within the current DSN session.

- b) When the DSN trace is activated, all trace messages are routed through the TSO message issuer routine (IKJEFF02) to the TSO SYSTSPRT DD statement. Optionally, the DSN trace information can be duplicated by allocating a separate data set with a DD name of DSNTRACE. The DSNTRACE data set can then be used to maintain a separate log for offline reference and diagnosis that contains only the trace messages produced by the DSN command. The DSNTRACE data set should be allocated prior to DSN invocation as a sequential data set, with RECFM=F, and LRECL=132.
- c) Review the DSN trace messages that appear on the terminal or in the data set that has collected the trace stream.

The text contained in these DSN trace messages often provides clues about the problem. For diagnosing problems, special messages clearly show control passing from DSN to Db2 and back. They can help to determine whether a problem is in Db2, DSN, or an application program.

Specifically, the trace messages that contain a line of equal signs indicate when control is being passed between the application and Db2. The series of equal signs can be thought of as a "wall" between DSN and Db2 that indicates in which area a given trace event occurs. Each of these trace messages begins with one of the following lines:

```
BEFORE some event =====  
AFTER  that event =====
```

Whenever a BEFORE message is displayed, this indicates that control has been passed to Db2. Whenever an AFTER message is displayed, this indicates that control has returned to the application program.

- d) Refer to the notes about the DSN trace stream for additional information.
- Activate the DSN trace stream using DB2I.
 - a) Invoke the DB2I primary menu (DSNEPRI).
 - b) Select OPTION 0, which is not shown on the menu.
 - c) Press ENTER to pass the panel that introduces the Db2 TSO MAINTENANCE SYSTEM.

3. Press ENTER to pass by the panel that introduces the Db2 TSO MAINTENANCE SYSTEM. The next panel provides choices between several TSO attachment facility tracing mechanisms.
4. Specify SYMLIST on the TRACE CLISTS option. This traces the most events.

The options on the TRACE CLISTS line are the same as those that can be specified on the CLIST CONTROL statement.

- Selecting LIST is equivalent to specifying CONTROL LIST; it traces TSO commands in the CLISTS.
- Selecting CONLIST is equivalent to specifying CONTROL LIST CONLIST; it traces TSO commands in the CLISTS and each CLIST statement before variable substitution occurs.
- Selecting SYMLIST is equivalent to specifying CONTROL LIST CONLIST SYMLIST; it traces TSO commands in the CLISTS, each CLIST statement before variable substitution occurs, and each CLIST statement after variable substitution occurs.

5. Run the application, and the CLIST trace messages display.

Results

If the CLIST tracing indicates a call to the DSN command processor that seems to have caused an error, then run the job again with DSN tracing set to a value greater than 100. Receiving messages from both the DSN and CLIST trace streams help IBM Support resolve the problem.

Trace messages must be viewed online; they cannot be sent to a data set.

Activating the SPUFI trace facility

The SPUFI trace stream consists of trace messages sent to the ISPF LOG. Also appearing in the ISPF LOG are messages from the ISPF trace facility.

Procedure

To activate the SPUFI trace facility:

1. Invoke the DB2I primary menu (DSNEPRI).
2. Select OPTION 0, which is not shown on the menu.
3. Press ENTER to pass by the panel that introduces the Db2 TSO MAINTENANCE SYSTEM. The next panel displays several TSO attachment facility tracing mechanisms.
4. Enter YES by the TRACE SPUFI option. The trace messages issued during the execution of SQL statements (in an input file) are written to the ISPF LOG.
5. Use the ISPF browse LOG option to review the SPUFI trace messages.

For more information about the ISPF browse LOG option and other ISPF commands, refer to the z/OS ISPF program reference publication.

6. To save the LOG, specify the KEEP option on the ISPF panel displayed when the session is terminated.

The SPUFI trace messages provide a high-level view of the processing events. This list describes significant factors to be aware of while reviewing the ISPF LOG and the SPUFI trace messages it contains.

- At the beginning of the ISPF LOG is the following message:

```
START OF ISPF LOG - - - - - SESSION # nnn -----
```

where nnn is the number of the session.

- Each subsequent message relates to some significant aspect of the processing involved in that session.
- The entries in the left column indicate the time (in minutes and seconds) that each corresponding trace message was issued.
- To the right of the time of the message is the message identifier itself; each SPUFI trace message begins with DSNE.

- To the right of the message identifier for most of the SPUFI messages is the name of the CSECT that issued the trace message; the names of each of these CSECTs are in the format:

```
DSNESMxx
```

where xx are numbers in the range 0–99 that uniquely identify the CSECT.

- The right column of the ISPF LOG describes what was happening when the message was issued.
- If processing completes normally, the end of the ISPF LOG shows this final ISPF message:

```
END OF ISPF LOG - - - - - SESSION # nnn -----
```

where nnn is the number of the session.

- If an abend occurs during the session, the SPUFI trace messages are interrupted, and one or more TSO or ISPF error messages are printed at the bottom of the ISPF LOG. These error messages indicate the abend completion code and other pertinent information.

Example

```
16:13 - DSNE454I DSNESM30 LEAVE YSTUFBUF WITH RBUFU NOW = 64
16:13 - DSNE420I DSNESM30 LOOPTOP
16:13 - DSNE445I DSNESM30 TYPE IS INTEGER OR SMALL INTEGER
16:13 - DSNE446I DSNESM30 BEFORE CONVERSION
16:13 - DSNE447I DSNESM30 AFTER CONVERSION
16:13 - DSNE449I DSNESM30 NO TRUNCATION OCCURRED
16:13 - DSNE454I DSNESM30 LEAVE YSTUFBUF WITH RBUFU NOW = 70
16:13 - DSNE454I DSNESM30 LEAVE YSTUFBUF WITH RBUFU NOW = 72
16:13 - DSNE420I DSNESM30 LOOPTOP
16:14 - DSNE445I DSNESM30 TYPE IS INTEGER OR SMALL INTEGER
16:14 - DSNE446I DSNESM30 BEFORE CONVERSION
16:14 - DSNE447I DSNESM30 AFTER CONVERSION
16:14 - DSNE449I DSNESM30 NO TRUNCATION OCCURRED
16:14 - DSNE454I DSNESM30 LEAVE YSTUFBUF WITH RBUFU NOW = 80
16:14 - DSNE599I DSNESM22 - BEGIN EXECUTION
16:14 - DSNE812I DSNESM22 ABOUT TO WRITE A LINE OF OUTPUT
16:14 - DSNE812I DSNESM22 ABOUT TO WRITE A LINE OF OUTPUT
16:14 - DSNE581I DSNESM22 - END EXECUTION 0
16:14 - DSNE404I DSNESM30 EXITED
16:14 - DSNE820I DSNESM68 - BEGIN EXECUTION
- SQL REQUEST = FETCH
16:14 - DSNE821I DSNESM68 - END EXECUTION -
- SQLCODE = 00000000
16:14 - DSNE403I DSNESM30 ENTERED
16:14 - DSNE409I DSNESM30 BEFORE MAIN LOOP
16:14 - DSNE420I DSNESM30 LOOPTOP
16:14 - DSNE453I DSNESM30 TYPE IS CHAR, VARCHAR, or LVARCHAR
16:14 - DSNE449I DSNESM30 NO TRUNCATION OCCURRED
16:14 - DSNE454I DSNESM30 LEAVE YSTUFBUF WITH RBUFU NOW = 8
16:14 - DSNE454I DSNESM30 LEAVE YSTUFBUF WITH RBUFU NOW = 20
16:14 - DSNE420I DSNESM30 LOOPTOP
16:14 - DSNE425I DSNESM30 TYPE IS CHARACTER
16:14 - DSNE449I DSNESM30 NO TRUNCATION OCCURRED
16:14 - DSNE454I DSNESM30 LEAVE YSTUFBUF WITH RBUFU NOW = 28
16:14 - DSNE454I DSNESM30 LEAVE YSTUFBUF WITH RBUFU NOW = 30
16:14 - DSNE420I DSNESM30 LOOPTOP
16:14 - DSNE420I DSNESM30 LOOPTOP
16:14 - DSNE425I DSNESM30 TYPE IS CHARACTER
16:14 - DSNE449I DSNESM30 NO TRUNCATION REQUIRED
16:14 - DSNE454I DSNESM30 LEAVE YSTUFBUF WITH RBUFU NOW = 31
16:14 - DSNE454I DSNESM30 LEAVE YSTUFBUF WITH RBUFU NOW = 36
```

Figure 74. Sample TSO attachment facility SPUFI trace messages

Related reference

[TSO attachment facility trace messages](#)

Some TSO trace messages have the same message number. They are listed in order of the CSECTs that issue them.

IRLM - Db2 activity trace

IRLM uses the z/OS component trace (CTRACE) to trace the flow of activity between IRLM and Db2.

IPCS CTRACE format, merge, and locate routines can be used to process buffer data, but the IPCS VERBX IRLM command provides more detailed output and is the preferred method. IRLM traces the following functions:

SLM

All calls to and from the system lock manager are traced.

XCF

All calls to and from the cross system coupling facility are traced.

XIT

Asynchronous interactions with the system lock manager are traced.

DBM

All calls to and from the DBMs are traced.

INT

IRLM traces member and group status events.

EXP

IRLM traces all exception events that occur during execution.

The XIT, EXP, and INT traces start automatically when you start the IRLM startup procedure.

The TRACE=YES parameter of the IRLM startup procedure lets you capture trace data to wraparound buffers at IRLM startup time. When you specify TRACE=YES, IRLM starts all traces.

You can use the z/OS TRACE CT command to start or stop specific traces and to specify whether the trace data wraps in the trace data sets. You cannot stop the EXP and INT traces using the TRACE CT command. This z/OS command requires that load module DXRRL183 be placed in the linklib.

IRLM traces are vital in resolving many of the problems in IRLM and Db2 that are related to lock requests. You should always activate these traces in a test or development environment. It is strongly recommended that you activate these traces in a production environment. To activate IRLM traces, specify TRACE=YES in the IRLM address space (IRLMPROC) startup procedure. The traces are written to internal wrap-around buffers.

Initially, IRLM allows 10 64 KB buffers for each trace type. The buffer storage is taken from ECSA, and the buffers are allocated as needed. You can change the number of buffers for each trace type using a command of this form:

```
MODIFY irlmproc,SET,TRACE=nnn
```

Related reference

[START irlmproc command \(z/OS IRLM\) \(Db2 Commands\)](#)

[MODIFY irlmproc,SET command \(z/OS IRLM\) \(Db2 Commands\)](#)

[TRACE CT command \(z/OS IRLM\) \(Db2 Commands\)](#)

Formatting the IRLM trace

Formatting the IRLM internal CTRACE entries is best accomplished with the IRLM dump formatter.

About this task

To format the IRLM trace, use one of the following syntaxes:

- `verbx IRLM 'SU=nnnn, TR=xxx'`

- IRLM 'SU=nnnn, TR=xxx, TY=DE'

where *nnnn* = irlmnm, and *xxx* = trace type (SLM, XCF, XIT, DBM, INT, or EXP)

IRLM trace buffers can be formatted using IPCS. The IRLM trace formatting load module DXRRL184 and the buffer find routine load module DXRRL186 must be available to use IPCS.

The following IPCS subcommands can be used to format IRLM trace buffers:

SETD DA(*dataset name*)

Set the trace data set to the default.

CTRACE QUERY

Query the trace components and their associated subtraces that are identified to CTRACE.

CTRACE COMP(*irlm*) SUB(*(sub)*)*zzz*

Format the trace buffers where *irlm* is the IRLM subsystem name for the CTRACE component and *sub* is one of the IRLM subtraces (SLM, XCF, DBM, INT, EXP, XIT).

zzz should be short or full depending on how much data is formatted. Short is the default.

CTRACE content

The IPCS VERBX IRLM command displays an IRLM DUMP format of the CTRACES.

Procedure

1. For single 132 character line output, use the following command:

```
VERBX IRLM 'SU=nnnn,TR=DBM'
```

```
032A 100-01: START A REQUEST      0204 00 0000 B46F40E681462606
      07D3D6C3D2F0F1              TKN: 0586D0C4
```

Figure 75. VERBX – 132 character line output

2. For DETAIL output, in addition to the 132 character summary line output, use the following command:

```
VERBX IRLM 'SU=nnnn,TR=DBM,TY=DE'
```

```

      Trace Header
+0000 010E0010 00000002 B46F40E6 81462606 |.....? Wa...
+0010 C4E7D9D9 D3F1F0F0 60F0F17A 40E2E3C1 |DXRRL100-01: STA
+0020 D9E340C1 40D9C5D8 E4C5E2E3 40404040 |RT A REQUEST
      Element type = RLPL      Length = 00C8      Address = 0583DA20
057B43AA +0000 00000000 00005A50 00000000 80000000 |.....!&.....
+0048 +0010 00000000 00000000 00000000 00000000 |.....
+0058 +0020 80BC9000 0581C048 001B0001 058D4910 |....a{.....
+0068 +0030 00000000 001B0001 00000000 00000000 |.....
+0078 +0040 00000000 00000000 D3D6C3D2 07D3D6C3 |.....LOCK.LOC
+0088 +0050 D2F0F100 00000000 00000000 00000000 |K01.....
+0098 +0060 00000000 00000000 00000000 80000000 |.....
+00A8 +0070 00000000 00000000 0000A248 00000000 |.....s.....
+00B8 +0080 02040000 12000000 00000000 00000000 |.....
+00C8 +0090 00000000 00000000 00000000 00000000 |.....
+00D8 +00A0 00000000 00000000 00000000 00000000 |.....
+00E8 +00B0 00000000 00000000 00000000 00000000 |.....
+00F8 +00C0 00000000 00000000 |.....|
      Element type = RLPL PT      Length = 0004      Address = 0583D8C8
057B447A +0000 058D4994              |...m|
```

Figure 76. VERBX – DETAIL output

Example

The DBM is the request from the DBMS to IRLM. The SLM traces are the entries made when IRLM forwards those lock requests to the SLM.

```

032A 100-01: START A REQUEST      0204 00 0000 B46F40E681462606
      07D3D6C3D2F0F1                TKN: 0586D0C4
      Trace Header
+0000 010E0010 00000002 B46F40E6 81462606 |.....? Wa...
+0010 C4E7D9D9 D3F1F0F0 60F0F17A 40E2E3C1 |DXRRL100-01: STA
+0020 D9E340C1 40D9C5D8 E4C5E2E3 40404040 |RT A REQUEST
      Element type = RLPL      Length = 00C8      Address = 0583DA20
057B43AA +0000 00000000 00005A50 00000000 80000000 |.....!&.....
+0048 +0010 00000000 00000000 00000000 00000000 |.....
+0058 +0020 80BC9000 0581C048 001B0001 058D4910 |.....a{.....
+0068 +0030 00000000 001B0001 00000000 00000000 |.....
+0078 +0040 00000000 00000000 D3D6C3D2 07D3D6C3 |.....LOCK.LOC
+0088 +0050 D2F0F100 00000000 00000000 00000000 |K01.....
+0098 +0060 00000000 00000000 00000000 80000000 |.....
+00A8 +0070 00000000 00000000 0000A248 00000000 |.....s.....
+00B8 +0080 02040000 12000000 00000000 00000000 |.....
+00C8 +0090 00000000 00000000 00000000 00000000 |.....
+00D8 +00A0 00000000 00000000 00000000 00000000 |.....
+00E8 +00B0 00000000 00000000 00000000 00000000 |.....
+00F8 +00C0 00000000 00000000 |.....|
      Element type = RLPL PT      Length = 0004      Address = 0583D8C8
057B447A +0000 058D4994 |...m|

```

Figure 77. IRLM DUMP format of a CTRACE record

The following CTRACE sample shows two types of traces for a lock request.

```

COMPONENT TRACE FULL FORMAT
COMP(IRLE)      SUBNAME((DBM))
**** 02/10/94
MNEMONIC  ENTRY ID      TIME STAMP      DESCRIPTION
-----
DBM        00000002  18:42:05.816178  RLPL format
+0000  ID.....  DXRRL100-01: START A REQUEST
+0020  TLA1..... 000100C8  07166220
+0028  RLPL..... 00000000  06545768  00000000  80000000  00000000
+003C  00000000  006B12C8  008FBBC0  0090B000  00906048
+0050  00316545  06545060  00000000  00316545  06545060
+0064  00000000  00000000  00000000  0423AD20  09000058
+0078  C8806D01  D7000000  00000000  00000000  00000000
+008C  00000000  00000000  80000000  00000000  00000000
+00A0  006B12C8  008FBBC0  02060000  8A000000  00000000
+00B4  00000000  006B5BE4  00000000  00000000  00000000
+00C8  00000000  00000000  00000000  00000000  00000000
+00DC  00000000  00000000  00000000  00000000  00000000
DBM        00000002  18:42:05.816406  RLPL format
+0000  ID.....  DXRRL100-02: REQUEST COMPLETED
+0020  TLA1..... 000100C8  07166220
+0028  RLPL..... 00000000  06545768  00000000  80000000  00000000
+003C  00000000  006B12C8  008FBBC0  0090B000  00906048
+0050  00316545  06545060  00000000  00316545  06545060
+0064  00000000  00000000  00000000  0423AD20  09000058
+0078  C8806D01  D7000000  00000000  00000000  00000000
+008C  00000000  00000000  80000000  00000003  00000000
+00A0  006B12C8  008FBBC0  02060000  8A000000  00000000
+00B4  00000000  006B5BE4  00000000  00000000  00000000
+00C8  00000000  00000000  00000000  0067027C  A743B4E5
+00DC  09010080  00000000  00080000  00000000  00000000
COMPONENT TRACE FULL FORMAT
COMP(IRLE)      SUBNAME((SLM))
**** 02/10/94
MNEMONIC  ENTRY ID      TIME STAMP      DESCRIPTION
-----
SLM        00000010  18:42:05.816193  RNA, RTE and UDB format
+0000  ID.....  DXRRL120-01: IXLLOCK OBTAIN
+0020  TLA1..... 00060020  00670238
+0028  RNA.....  09000058  C8806D01  D7000000  00000000  00000000
+003C  00000000  00000000  00000000
+0048  TLA2..... 000C0040  07166418
+0050  RTE.....  0423AD20  09000058  C8806D01  D7000000  00000000
+0064  00000000  00000000  00000000  00000000  00000008
+0078  C9D4E2C5  40404040  0423AD20  00000000  00000000
+008C  00000000
+0090  TLA3..... 000B0040  071663D8
+0098  UDB.....  C9D4E2C5  40404040  00000000  00000000  00080000
+00AC  00000000  00000000  00000000  00000000  40000000
+00C0  08000000  00000000  A8D1A743  B4D7B281  A8D1A743
+00D4  B4D7B281
SLM        00000020  18:42:05.816397  RNA and reason code
+0000  ID.....  DXRRL120-03: IXLLOCK RETURN
+0020  TLA1..... 00060020  00670238
+0028  RNA.....  09000058  C8806D01  D7000000  00000000  00000000
+003C  00000000  00000000
+0048  TLA2..... 00060004  0716637C
+0050  REAS..... 00000000

```

Figure 78. Sample of a CTRACE record

Finding the locks that belong to an agent

To diagnose lock contention problems between agents within Db2, it may be necessary to find and analyze all locks for a particular agent. To find these locks, only the RLPL records are needed.

To become oriented, find the lock request for the request's skeleton cursor table (CT) lock. The RLPL for this lock request contains the plan name in the resource name field. The resource name field begins at offset X'48' from the beginning of the RLPL and consists of a four-byte hash field, followed by a one-byte length field, followed by the 31-byte resource name field that contains the plan name. Therefore, you should find the plan name at offset X'4D' from the beginning of the RLPL. To find this lock request, scan the RLPL records looking for the plan name.

Within this lock request, the following items are used in finding the locks for a particular agent:

1. The two-byte work unit number (RLPWUN) at offset X'2A'.

2. The four-byte work unit ID address at offset X'2C'. This is the ACE or EB address.
3. The two-byte owning work unit number (RLPOWN) at offset X'36'.
4. The four-byte owning work unit address (RLPOWNID) at offset X'38'. This is an ACE address.

For a given work number (RLPWUN) span and the work unit ID address in the owning work unit (RLPOWU) remains the same.

For a given request, the work number, work unit ID address, and owning work unit address can change, depending on the overall processing involved for the request.

Within a work number span, the last associated RLPL is an UNLOCK request to release all locks held. This RLPL can be identified by:

- A four-byte token field (RLPLTKN) at offset X'44' of all zeros.
- A 36-byte resource name field at offset X'48' of all zeros.
- A one-byte function field (RLPLFUNC) at offset X'80' that contains X'03', UNLOCK.
- A one-byte duration field (RLPLDURA) duration field in byte X'82' with a value equal to or higher than the duration of any prior lock requests.
- A corresponding "owning work unit address" (RLPOWNID) at offset X'38'.

After finding the RLPL for the plan request, find the work unit number and the owning work unit address. It might now be necessary to go backwards in the CTRACE to find the first occurrence of the work number/RLPOWU combination. If the preceding trace entries are to be ignored, then scan forward using the work number and RLPOWU. The last entry in the CTRACE for this combination should be an UNLOCK request to release all locks.

Usually, entries follow this request, but they have a different work number/RLPOWU combination. Repeat the process for these combinations.

RLIPL (request identify parameter list)

RLPL records can be used to diagnose lock contention problems between agents within Db2.

Bytes 8-12

The identify return code

Bytes 64-68

The requesting work unit ID

Bytes 72-76

IRLM subsystem name

Bytes 76-84

Db2 subsystem name

Bytes 132-140

Returned User Token

Byte 143

The reason code

Byte 144

Subsystem return code

RLQD (query parameter list)

The RLQD begins with four bytes of global information, starting at offset X'00'. The remainder of the RLQD consists of alternate headers and data items. Each header contains a code that describes the type and length of the data item that follows. Each header is four bytes long and the data items are variable in length.

The following shows the content of the headers and the four types of data items. The last header in the RLQD always contains X'80' in the second byte to indicate that the following data item is the last one in the RLQD.

GLOBAL INFORMATION

Bytes 1-4

Global information

HEADER INFORMATION**Byte 1**

Type of data Item

X'01' Work unit summary data

X'02' Resource data

X'03' Lock data for held resource

X'04' Lock data for suspended request

Byte 2

Last data item

X'80' Last data item in the RLQD

Bytes 3 and 4

Length of data item **DATA ITEM X'01': WORK UNIT SUMMARY DATA:**

Bytes 1-8

Owning work unit ID

Bytes 9 and 10

Number of resources locked

Bytes 11 and 12

Number of suspended requests **DATA ITEM X'02': RESOURCE DATA**

Byte 1

Global scope flag X'80'

Bytes 5-40

Resource name **DATA ITEM X'03': LOCK DATA FOR HELD RESOURCE**

Bytes 1-4

RLB pointer

Bytes 5-13

Work unit ID

Bytes 13-29

State counters—8 halfword counters

Byte 29

Lock duration

Byte 30

Flags

X'80' Held lock is count by state

X'40' Held lock is resultant state

Byte 31

Current state **DATA ITEM X'04': LOCK DATA FOR SUSPENDED RESOURCE:**

Bytes 1-4

RLB pointer

Bytes 5-12

Work unit ID

Byte 13

The function that was requested. The functions are the same as in **Word 1 Byte 2-Request Function (FC)**

Byte 14

Lock duration

Byte 15

Flags

X'80' Suspended lock is count by state

X'40' Suspended lock is resultant state

IRLM SVC dump support

The IRLM uses SVC dump support to record errors or to dump its system status when an error occurs.

This SVC dump supports:

- Provides a machine-readable dump that can be submitted with an APAR.
- Ensures that dumps are issued for failures that occur within the IRLM address space, for failures that occur while executing IRLM code, or for failures that occur in Db2 IRLM exits that are coded within a Db2 address space.
- Allows a dump to be issued at any time; that is, a dump can be issued even when neither Db2 nor the IRLM abends.
- Offers the performance advantage of offline dump formatting.
- Includes support for the **MODIFY irllmproc,DIAG** command, which controls when certain types of IRLM diagnostic dumps are taken.

Related reference

[MODIFY irllmproc,DIAG command \(z/OS IRLM\) \(Db2 Commands\)](#)

Diagnosing EDM pool space problems using traces

You can use IFCID 0133 and IFCID 0134 to diagnose and collect diagnostic information about EDM pool space problems.

Procedure

PSPI

1. Use an IFCID 0133 trace if you are getting abnormal terminations like the following one:

```
ABNDND0E5 RC0C90101 DSNGEDLC:5009
```

A trace for IFCID 0133 helps IBM Software Support diagnose similar problems in the future.

To use the IFCID 0133 trace, issue the following command before an abend occurs:

```
-START TRACE (PERFM) CLASS(30) IFCID(133)
```

If you want to turn off the trace, you must stop it explicitly.

2. Use an IFCID 0134 trace if you are getting EDM pool full conditions. After you start this trace, Db2 generates a dump the first time an EDM Pool Full condition is detected.
Db2 turns off this trace after the first abend dump is taken.

To start the trace for IFCID 0134, issue the following command:

```
-START TRACE (PERFM) CLASS(30) IFCID(134)
```

PSPI

Results

After Db2 is refreshed, both traces will be inactive.

z/OS traces

z/OS traces, which are common to all products operating as formal subsystems of z/OS, are available for use with Db2. For information on using and interpreting this trace facility, refer to the z/OS diagnostic publications.

Writing Db2 log buffers to IFI

For diagnostic purposes, you might want to look at the contents of the Db2 log buffers as they are written to the active log.

PSPI

For example, you might want to capture log records from the log buffers during the time a job experiences repeatable failures. You might want to interactively capture a copy of the log buffers which are written to the active log, such that the copies of the log CIs can be transported or placed in a remote or local data set.

To begin writing Db2 log buffers to IFI, an application can issue the following Db2 command via the IFI COMMAND interface:

```
-START TRACE(P) CLASS(30) IFCID(126) DEST(OPX)
```

Where:

- P signifies to start a Db2 performance trace. Any of the Db2 trace types can be used (accounting, statistics, performance, audit, monitor, global).
- CLASS(30) is a user-defined trace class (31 and 32 are also user-defined classes).
- IFCID(126) activates Db2 log buffer recording.
- DEST(OPX) starts the trace to the next available internal Db2 online performance (OP) buffer. The size of this OP buffer can be explicitly controlled by the BUFSIZE keyword of START TRACE. Valid sizes range from 256 KB to 16M, and the size must be evenly divisible by 4.

When the START TRACE command takes effect, from that point forward, until Db2 terminates, Db2 recovery log manager (RLM) begins writing 4 KB log buffer VSAM control intervals (CIs) to the OP buffer as well as to the active log. As part of the IFI COMMAND invocation, the application specifies an ECB to be posted and a threshold to which the OP buffer is filled when the application is posted to obtain the contents of the buffer. The IFI READA request is issued to obtain OP buffer contents.

Reading specific log records: IFCID 129 or IFCID 306 can be used with an IFI READS request to return a specific range of log records from the active log into the return area the program has initialized. Enter the following command in the IFI program:

```
CALL DSNWLI(READS,ifca,return_area,ifcid_area,qual_area)
```

IFCID 129 or IFCID 306 must appear in the IFCID area.

PSPI

Related concepts

[Programming for the instrumentation facility interface \(IFI\) \(Db2 Performance\)](#)

Related reference

[-START TRACE command \(Db2\) \(Db2 Commands\)](#)

Db2 stand-alone log services: change log inventory and print log map

Db2 provides two offline (batch) utilities to assist with the management of the active logs, the archive logs, and the bootstrap data set (BSDS).

The *change log inventory* (DSNJU003) enables you to change the contents of the bootstrap data set (BSDS). The change log inventory utility is usually run with Db2 stopped because if it is run when Db2 is active, inconsistencies can result. Use `-STOP DB2 MODE(QUIESCE)` to stop the Db2 subsystem, then run the utility, and after that restart Db2 with the `-START DB2` command.

The *print log map* (DSNJU004) utility enables you to print the contents of the bootstrap data set (BSDS). For Db2 data sharing, DSNJU004 prints BSDS information about each Db2 subsystem in the data sharing group. The print log map utility can run when Db2 is active or inactive. If it is run when Db2 is active, the user's JCL and the Db2 started task must both specify `DISP=SHR` for the BSDS data sets.

Related reference

[DSNJU003 \(change log inventory\) \(Db2 Utilities\)](#)

[DSNJU004 \(print log map\) \(Db2 Utilities\)](#)

Diagnostic information in Db2 catalog tables

The Db2 catalog tables contain descriptive information about the data stored in Db2 databases.

These tables are updated by SQL statements. The catalog tables are useful diagnostic aids for the following problems:

Authorization problems

If authorization problems occur, a message is issued. Look at the following catalog tables:

- SYSIBM.SYSCOLAUTH
- SYSIBM.SYSTABAUTH
- SYSIBM.SYSDBAUTH
- SYSIBM.SYSRESAUTH
- SYSIBM.SYSPACKAUTH
- SYSIBM.SYSPLANAUTH
- SYSIBM.SYSUSERAUTH

Select the catalog table that corresponds to the format of the GRANT statement. This catalog table includes information about who has authority and who can grant it.

Index problems

If you receive a message from a utility (message identifier DSNUxxxx) saying the index is incorrect for a given table space, look first at the SYSIBM.SYSINDEXES table, which identifies the table and database. After the table name is known, look at the SYSIBM.SYSTABLES catalog table, which identifies the table space.

Catalog consistency

The sample library, SDSNSAMP, contains sample queries in member DSNTESQ that can be used to check for consistency between catalog tables.

RECOVER utility failure

If the RECOVER utility fails, either a message is issued or an abend occurs. You can use the REPORT utility to view recovery information that the RECOVER utility uses or you can review the SYSIBM.SYSCOPY catalog table. Try to determine which data sets and volumes of image copies were required by RECOVER.

Performance problems following RUNSTATS processing

If performance problems occur after running the RUNSTATS utility, check the data in the SYSIBM.SYSPLAN and SYSIBM.SYSPACKAGE catalog tables to confirm that any plans and packages associated with the table space on which the utility was run are rebound. The table spaces referred to by the plans are recorded in SYSIBM.SYSPLANDEP. The tables referred to by the packages are recorded in SYSIBM.SYSPACKDEP.

Performance problems following creation of a new index

If performance problems occur after a new index is created, check the data in the SYSIBM.SYSPLAN and SYSIBM.SYSPACKAGE tables to make sure that any plans and packages that use the table on which the index was created are rebound. The tables referred to by the plans are recorded in SYSIBM.SYSPLANDEP. The tables referred to by the packages are recorded in SYSIBM.SYSPACKDEP.

Shortage of EDM pool space

If you receive a message saying that there is insufficient EDM pool space, check the SYSIBM.SYSPLAN and SYSIBM.SYSPACKAGE catalog tables. Determine how many plans and associated packages are to be running concurrently and/or the total number of plans and packages accessed during the lifetime of Db2.

Check the KEEP_DYNAMIC field in SYSIBM.SYSPLAN and SYSIBM.SYSPACKAGE to determine whether plans and packages use the dynamic statement cache. Dynamic statement caching uses the EDM pool, so using the cache without increasing the size of the EDM pool can lead to storage shortages. Using a data space for cached dynamic statements can help alleviate storage shortages.

Conversion error during LOAD utility processing or during INSERT or UPDATE processing

If a conversion error during LOAD processing occurs, a message is issued. Check the SYSIBM.SYSCOLUMNS catalog table to ensure that fields and columns being loaded are compatible. You can also determine the cases in which nulls are permitted.

DASD problems

Check the SPACE column in the SYSIBM.SYSSTOGROUP, SYSIBM.SYSTABLESPACE, and SYSIBM.SYSINDEXES catalog tables. The SPACE column is correct and current only if STOSPACE utility was run. Furthermore, the SPACE column is only applicable to table spaces and indexes using STOGROUPS, and is not applicable to user-defined data sets.

Related tasks

[Calculating EDM pool sizes \(Db2 Installation and Migration\)](#)

SQLDA extension for binary XML data

Although there is no embedded SQL support for binary XML host variables, it is possible to set up an SQLDA to indicate that XML data is XML binary data.

PSPI

For OPEN, EXECUTE, FETCH, and CALL statements, the application program sets the fields in the SQLDA to provide Db2 with information about input or output host variables in the application program.

To describe a binary XML host variable, the SQLDA has the following format and content:

- A binary XML host variable that is not a file reference variable requires an SQLDA with two sets of SQLVARs.
- In a base SQLVAR, the SQLTYPE field is set to BLOB (404/405) and the SQLLEN field is set to 0
- The SQLNAME field of the base SQLVAR is set as follows: The length of SQLNAME is 8. The first four bytes of the data portion of SQLNAME are X'00000000'. The fifth byte is a flag field that indicates the type of host variable. The values of this field are as follows:
 - X'01'- XML variables (XML AS BLOB, XML AS CLOB, XML AS DBCLOB)
 - X'03'- binary XML variables

There is no change to the sixth to eighth of the SQLNAME field of the base SQLVAR

- In an extended SQLVAR, the SQLLONGLEN field is set to the length attribute of a binary XML variable.

To describe a binary XML file, the file reference variable only requires an SQLDA with one set of SQLVARs. The SQLTYPE field of a base SQLVAR is set to BLOB_FILE (916/917). The SQLNAME field of base SQLVAR is set to the same value as that of a binary XML variable.

There is no change to the output SQLDA for DESCRIBE and PREPARE INTO statements.

The binary XML bits on the host variables of all other data types are ignored.

PSPI

Db2 utilities for troubleshooting

The Db2 utilities (UT) subcomponent provides a number of utilities helpful for diagnosing various Db2 problems.

CHECK DATA

CHECK DATA performs the following functions:

- Reports information about each referential constraint violation it detects and offers options to copy invalid rows to other tables and to delete invalid rows
- For tables that contain LOB columns, checks for consistency between the base table space and the LOB table space
- For tables that contain XML columns, checks for consistency between the base table space and the XML table space
- For tables that contain XML columns, checks XML data integrity and validates against XML schemas for conformance.

For more information, see [CHECK DATA \(Db2 Utilities\)](#).

CHECK LOB

CHECK LOB checks large object (LOB) table spaces for structural defects and invalid LOBs. For more information see [CHECK LOB \(Db2 Utilities\)](#).

CHECK INDEX

CHECK INDEX allows one or more indexes to be checked simultaneously and issues warning messages when it finds an inconsistency. For more information, see [CHECK INDEX \(Db2 Utilities\)](#).

COPY

The COPY utility makes full or incremental copies to sequential data sets, including full imaged copies of index spaces. It produces as many as four copies of the image into four different data sets: two for the local site and two for the recovery site. Incremental copies contain only pages that have changed since the prior copy function; full copies contain the entire table space. For more information, see [COPY \(Db2 Utilities\)](#).

DIAGNOSE

The DIAGNOSE utility enables Db2 service representatives to diagnose problems with utilities. Up to 32 activities can be monitored through the TYPE parameter. These activities are defined as needed by the service staff. The DIAGNOSE can also be used for the following activities:

- Obtaining dumps for any utility abend, overriding any considerations that might suppress such a dump
- Send OBD and SYSUTIL information to SYSPRINT for review (DISPLAY option)
- Dumping both the Db2 MEPL and the application space (DSNUTILB) MEPL to SYSPRINT without forcing an SVC dump (DISPLAY MEPL option)
- Dumping an entry for a database in SYSIBM.SYSDATABASE to SYSPRINT (DISPLAY option)
- Forcing Db2 to abend after a specific Db2 message is specified or when a module trace ID is encountered (ABEND option with the MESSAGE or TRACEID option)

For more information, see [DIAGNOSE \(Db2 Utilities\)](#).

DSN1COMP

DSN1COMP is a stand-alone utility that estimates space savings to be achieved by Db2 data compression in data sets. The utility runs on data sets that contain uncompressed data and do not contain index spaces.

The DETAIL and SHOWTREE parameters are provided for use by IBM support centers to examine dictionary structures:

- the DETAIL option provides performance related information
- the SHOWTREE option prints the formatted dictionary tree to the SYSPRINT output data set

For more information, see [DSN1COMP \(Db2 Utilities\)](#).

DSN1COPY

The DSN1COPY stand-alone utility is used for copying VSAM and sequential data sets, performing validity checking on data, index and dictionary pages, and translating object identifiers (OBIDs) to help in moving data sets between different systems. It also prints hexadecimal dumps of data sets and databases.

DSN1COPY also reports if generic clustering is lost in certain catalog table spaces. Contact IBM Support if this occurs. For more information see [DSN1COPY \(Db2 Utilities\)](#).

DSN1LOGP

The DSN1LOGP stand-alone utility reads the recovery log and formats its contents for display. For data sharing, DSN1LOGP prints log records merged from the logs of more than one Db2 subsystem. Either a summary or a detailed report can be requested. For more information see [DSN1LOGP \(Db2 Utilities\)](#).

DSN1PRNT

DSN1PRNT stand-alone is used for printing hexadecimal dumps of table spaces, index spaces, image copy data sets, and sequential data sets, and formats the pages of these data sets. For more information see [DSN1PRNT \(Db2 Utilities\)](#).

DSN1SDMP

The purpose of the DSN1SDMP utility, intended to be used under the direction of IBM Support, is to take dumps and write trace records for diagnostic purposes in a simple and timely manner. The DSN1SDMP utility consists of two features:

- It can select on any data within a trace record and, if the selection passes, can optionally cause a dump. This function captures the first-failure data much closer to the time of error than was possible before.
- It can also write Db2 trace records to an z/OS data set, rather than to z/OS SMF or GTF data sets. This allows trace data to be used more quickly for diagnostic purposes.

For more information, see [DSN1SDMP \(Db2 Utilities\)](#).

MERGE COPY

The MERGECOPY utility can either merge several incremental copies to form a combined incremental copy or merge one or more incremental copies with a full copy to form a combined full copy. MERGECOPY can produce up to four image copies. For more information, see [MERGECOPY \(Db2 Utilities\)](#).

QUIESCE

The QUIESCE utility establishes a point of consistency, a quiesce point, for a table space.

The point of consistency is recorded as the current log RBA in the SYSIBM.SYSCOPY catalog table and can be used by RECOVER TORBA or TOLOGPOINT. TOLOGPOINT can accept an RBA or LRSN. In a data sharing group, the QUIESCE utility quiesces an object for the entire group and records a LRSN that can be used by RECOVER TOLOGPOINT.

For more information, see [QUIESCE \(Db2 Utilities\)](#).

RECOVER

The RECOVER utility recovers entire table spaces and/or index spaces, a data set within a table space or a partition of an index space, or a range of pages within a table space or index space.

To recover a table space, RECOVER looks at the ZPARAM to determine if the current system is at the local site or the recovery site and recovers from copies of data registered on the current system.

If you specify the LOCALSITE parameter, only local site image copies are used (even if the site is the recovery site). If you specify the RECOVERYSITE parameter, only recovery site image copies are used (even if the site is the local site).

For more information, see [RECOVER \(Db2 Utilities\)](#).

REBUILD INDEX

The REBUILD INDEX utility rebuilds one or more indexes within a table space, all indexes within a table space, or one or more logical or physical partitions of a partitioning index. To rebuild an

index, REBUILD INDEX deletes the erroneous index and builds a new index from the table. For more information, see [REBUILD INDEX \(Db2 Utilities\)](#).

REPAIR

The REPAIR utility is helpful for the diagnosis and repair of inconsistent data. Use the REPAIR utility with extreme caution; incorrect use of REPAIR can cause the loss of data integrity. The REPAIR utility functions useful for diagnosing and repairing problems are: LOCATE, VERIFY, REPLACE, DUMP, SET, and REPAIR DBD. For more information, see [REPAIR \(Db2 Utilities\)](#).

REPORT

The REPORT utility can help to determine the recovery history of a table space. It reports the following information:

- The SYSIBM.SYSCOPY catalog table recovery records (REPORT can print all the SYSCOPY records in the local system, the recovery system, or both.)
- Log ranges from the SYSIBM.SYSLGRNX directory table.
- Log data sets from the bootstrap data set (BSDS)
- Names of all table spaces in a table space set.

For more information, see [REPORT \(Db2 Utilities\)](#).

Related concepts

[Basic information about Db2 utilities \(Db2 Utilities\)](#)

[Db2 utilities \(Introduction to Db2 for z/OS\)](#)

Db2 commands for troubleshooting

You can use commands to perform the tasks that are required to control and maintain your Db2 subsystem.

DISPLAY DATABASE LRSN command for troubleshooting

The Db2 command DISPLAY DATABASE with the LRSN keyword displays the commit LRSN and the read LRSN for a particular page set or partition.

PSPI

The read LRSN is the LRSN at which read interest is acquired on a page set or partition. The commit LRSN is the LRSN of the oldest write claim on a page set or partition. This information can be used to diagnose performance issues that are related to over-locking or inefficient space reuse.

LRSN

Specifies that the Db2 database manager displays the commit LRSN and the read LRSN for a particular page set or partition. The LRSN keyword cannot be used with keywords other than DATABASE, SPACENAM, or PART.

The LRSN values are displayed as local timestamps, in this format:

```
mm/dd/yyyy hh:mm:ss
```

The parts of the displayed timestamp are the numeric values for the month, day, year, hour, minute, and second.

Depending on where the page set is accessed, the commit LRSN and read LRSN might not be available from the data sharing member that issues the command. When the commit LRSN or read LRSN is not available, N/A is displayed in the output.

The following example command displays the commit LRSN and the read LRSN for partition 1 of all table spaces whose names begin with TSNAME, in all databases whose names begin with DBNAME.

```
-DISPLAY DATABASE(DBNAME*) SPACENAM(TSNAME*) PART(1) LRSN
```

The output looks similar to this output:

```
DSNT360I -DB2A *****
DSNT361I -DB2A * DISPLAY DATABASE SUMMARY
* GLOBAL
DSNT360I -DB2A *****
DSNT362I -DB2A DATABASE = DBNAME1 STATUS = RW
DBD LENGTH = 383600
DSNT397I -DB2A
NAME      TYPE PART  LRSN                                TIMESTAMP
-----
TSNAME1  TS    1      COMMIT 00CD534D63D8BA000000 6/18/2016 15:23:49
          READ  00CD535930B89F000000 6/18/2016 16:16:37
***** DISPLAY OF DATABASE DBNAME1 ENDED *****
DSN9022I -DB2A DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
```

PSPI

DISPLAY THREAD SERVICE command for troubleshooting

PSPI

The DISPLAY THREAD(*) SERVICE command helps to identify and rectify some CPU stalls. The following options are available:

SERVICE(WAIT)

Displays threads that are in a long-term suspend. If a thread is suspended on a latch, the command displays latch details, and Db2 attempts to boost the holder to relieve the latch contention. If the thread is suspended on a lock, the command displays lock detail information.

SERVICE(STG)

Displays the information for SERVICE(WAIT), with an additional message that outlines the agent local storage consumption for each thread displayed.

For more information, see [-DISPLAY THREAD command \(Db2\) \(Db2 Commands\)](#). PSPI

Related concepts

[Commands for controlling Db2 and related facilities \(Introduction to Db2 for z/OS\)](#)

Related information

[Command types for Db2 for z/OS \(Db2 Commands\)](#)

Program call linkages

Db2 uses z/OS program calls as one form of program invocation.

Db2 defines program call linkage to z/OS with one of these *program call* (PC) instructions:

PC-ss

Permits an address "space switch"

PC-cp

Retains an address space as "current primary" and does not cause a space switch

The z/OS architecture defines three addressable address spaces—*primary*, *secondary* and *home*. If a space switch occurs (because of the PC-ss instruction mentioned above), one address space is considered primary and another is considered secondary. If a space switch does not occur (because a PC-cp instruction is used), the primary and secondary address spaces are the same (that is, one address space is considered both primary and secondary).

The *home address space* is always present, and contains the dispatchable unit of work before any program calls are issued. If a space switch does not occur, then the home address space is also both the primary and secondary address space. If a program call with a space switch is issued by the dispatchable unit (using PC-ss), the primary space becomes the secondary address space and the "switched to" address space becomes the primary address space. If the dispatchable unit issues another PC with a

space switch, the previous primary becomes the secondary and the "switched to" address space becomes the new primary address space

Db2 uses PC-cp to get into supervisor state and key 7. Both PC-cp and PC-ss are used to enter 31-bit addressing mode (AMODE). In addition, PC-ss defines a new primary address space.

Each resource manager defines its PC linkage relationships using either (1) the RMVT and RMFT control blocks for RMRQ calls from one resource manager to another or (2) the RAL and FVL control blocks for RARQ calls to resource managers from an application.

TSO attachment facility diagnostic aids

The TSO attachment facility provides two diagnostic aids, SPUFI diagnostic panels and ABEND subcommand of the DSN command processor.

“TSO attachment facility traces” on page 263 describes three other TSO attachment facility tools: DSN trace stream, CLIST trace, and SPUFI trace stream.

SPUFI diagnostic panels

When activated, SPUFI diagnostic panels appear on the terminal as SPUFI is run. These panels display significant variable names and their assigned values (if any). These panels can be examined to verify that the information is correct.

About this task

Using this option of the TSO maintenance system to change program variables can disrupt normal SPUFI operation.

Procedure

To use the SPUFI diagnostic panels:

1. Display the DB2I primary menu (DSNEPRI).
2. Select OPTION 0, which is not shown on the menu.
3. Press Enter to pass by the panel that introduces the Db2 TSO MAINTENANCE SYSTEM. The next panel offers choices between several TSO attachment facility tracing mechanisms.
4. Select the **DISPLAY SPUFI** option (option 4). Specify any options that you want, such as the SPUFI CSECTs to trace.
5. With the SPUFI diagnostic panels activated, return to the DB2I primary menu and select the **SPUFI** option.
6. On the SPUFI panel, enter the parameters required to execute an SQL statement. At that time, the first diagnostic panel is displayed. A sample of a SPUFI panel (with some of the variables assigned) is shown in the following figure.
7. Optional: Complete the following steps:
 - Change the display or trace options identified at top.
 - Print a screen image by pressing the PF key that was defined for that purpose (the ISPF print key).
 - Change the displayed panel by altering the last two characters in the field identifying the currently displayed panel. Then, press Enter.

For example, if the panel name is DSNEXP00, change 00 to some other valid number (20, 21, 30, 55, 56, 70, 80). To execute from the point at which you left off, press Enter.

Results

```
DSNEXP55          SPUFI INTERNAL VARIABLE DISPLAY
====>
DSNE599I DSNESM55 - BEGIN EXECUTION
CHANGE DISPLAY OR TRACE OPTIONS BELOW:
 1 TRACE SPUFI   ==> YES                YES STARTS/CONTINUES TRACE
 2 DISPLAY SPUFI ==> 123                ENTER 0 TO 99 OR OVER 100
 3 DISPLAY PANEL ==> DSNEXP55          ENTER SPUFI PANEL NAME
INTERNAL VARIABLES FROM DSN DSEB CONTROL BLOCK:
10 PREFIX ==> H443722    40 ILRCL ==> 80        70 ACTION ==> 4
11 LIB     ==> DSN       41 IVOL  ==>          71 ACODE  ==> 1
12 TYPE    ==> PLS      42 ORCFM ==> FB        72 RCODE  ==> 0
13 MEMB    ==> SPUFIIN  43 XXXXX ==>          73 FLAGS  ==> 000000F4
14 OTHER   ==>
15 OUTPU   ==> SEQ.DS
16 ISTAT   ==> C0       46 OSTAT ==> 00        76 OCLAS  ==> SYSDA
17 IMEMB   ==> SPUFIIN  47 OMEMB ==> .....    77 PGW    ==> 0
18 IDSN    ==> H443722.DSN.PLS        78 OLRCL  999
19 ODSN    ==> .....                70 OBSIZ  999
20 MODID   ==> DSNESM55                80 MSGID  ==>
21 DAIRF1  ==> .....
22 DAIRF2  ==> .....
.....
```

Figure 79. Sample SPUFI diagnostic panel

ABEND subcommand of the DSN command processor

The TSO attachment facility provides an ABEND subcommand that enables users to request and obtain a dump.

About this task

Use this subcommand when a problem is suspected (such as incorrect output) with another DSN subcommand, and the problem does not cause an abend and a dump to be generated.

Important: The ABEND subcommand is used for diagnostic purposes only, and is intended to be used only under the direction of IBM Support. Use it only when diagnosing a problem with DSN or Db2.

Procedure

To use this subcommand:

1. Allocate a SYSUDUMP data set.
2. Reissue the DSN subcommand you suspect is involved in the problem.
3. After receiving the DSN prompt, type ABEND.
4. After TSO issues a message saying that DSN has ended, press ENTER and wait.
5. After TSO issues a "READY" message, the dump can be browsed or printed.

Results

Sometimes, the information dumped is not the information that you want. Always use the ABEND subcommand as soon as possible after a problem is re-created to increase the chances of obtaining meaningful data. Do not press the ATTENTION key before issuing the ABEND subcommand; usually, the data is lost.

Related reference

[DSN command \(TSO\) \(Db2 Commands\)](#)

[ABEND subcommand \(DSN\) \(Db2 Commands\)](#)

[Printing and analyzing dumps](#)

You might need to print and analyze a memory dump to help to diagnose a problem.

SYS1 service aids

Internal resource lock manager (IRLM) diagnostic aids include SVC dumps of the IRLM address space in the SYS1.DUMPxx data sets, entries on the SYS1.LOGREC data set, and output from the MODIFY ...,STATUS command.

Other IRLM diagnostic aids include:

- z/OS component trace (CTRACE) entries.
- Db2 performance trace classes 6 and 7.
- IRLM messages and codes.

Related concepts

[IRLM - Db2 activity trace](#)

IRLM uses the z/OS component trace (CTRACE) to trace the flow of activity between IRLM and Db2.

[Using the performance trace for diagnosis](#)

While the performance trace is primarily a tuning aid, it can sometimes be useful for diagnosis because degraded performance can also indicate a Db2 problem, one for which you would specify the PERFM keyword in building a keyword string to describe the problem.

Related reference

[Facilities and tools for Db2 performance monitoring \(Db2 Performance\)](#)

SYS1.DUMPXX

The IRLM address space is dumped to a SYS1.DUMPxx data set with an SVC dump whenever the IRLM, or an SRB that is related to the IRLM, abnormally terminates.

You can use IPCS to format and display a dump of an IRLM address space. To view a list of parameters that you can use to extract information from the dump, issue this command in IPCS:

```
VERBX IRLM 'HELP'
```

The IRLM verb has a parameter, IRLMNM, which is the IRLM subsystem name. The default is IRLM. If you are using a subsystem name other than the default, you need to specify the parameter.

SYS1.LOGREC

When IRLM detects a program error, it generates an entry on the SYS1.LOGREC data set.

Use the IFCEREP1 service aid to obtain a listing of the SYS1.LOGREC data set containing the SYS1.LOGREC entries pertaining to the IRLM. For more information about this service aid, refer to the z/OS diagnostic techniques publication.

Output from the MODIFY command

When you diagnose IRLM problems, you need to know the levels of the IRLMs to which your Db2 subsystems are connected. This is especially important in the data sharing environment, where there might be several IRLMs, all at different levels.

The output from the command `MODIFY irlmproc,STATUS,ALLI` gives you information about the service level and function level of each IRLM in a data sharing group. The service level is the last APAR applied to the IRLM, or the release level of the IRLM, if no APARs have been applied. The function level is a number that IRLM increments when service to IRLM introduces a new function. The first two characters of the function level indicate the IRLM version. For example, 2. indicates IRLM 2.2. For a data sharing group, the output also includes the minimum function level and service level that the group can tolerate.

To determine the levels of the IRLMs in the group, you issue the command:

```
MODIFY IRLMPR21,STATUS,ALLI
```

It is also important to know the maintenance level of each IRLM module. To display this information, issue this modify command:

```
MODIFY irlmproc,STATUS,MAINT
```

Related reference

[MODIFY irlmproc,STATUS command \(z/OS IRLM\) \(Db2 Commands\)](#)

Data sharing problem diagnosis

Problem diagnosis in the data sharing environment is similar to diagnosing problems that occur in a single Db2 subsystem. The key difference is that the actions required to identify and resolve problems must be applied to more than one Db2 subsystem.

Data sharing environment

Data sharing requires a *z/OS sysplex*, a group of central processor complexes (CPCs) running z/OS that are connected through channel-to-channel (CTC) communications or through a coupling facility.

Data sharing uses IBM zSystems™ Parallel Sysplex® technology. The sysplex technology is supported by the z/OS Cross-System Coupling Facility (XCF) and Cross-System Extended Services (XES). Cross-System Coupling Facility services allow authorized applications on one system to communicate with applications on the same system or on other systems. Cross-System Extended Services allow authorized applications or subsystems that are running in a sysplex to share data by using a coupling facility.

In Db2 data sharing, Db2 subsystems belong to a Db2 data sharing group. Each Db2 subsystem, or group member, has the same direct access to the same data as any other member of the data sharing group.

Group members share databases, a single Db2 catalog and directory, and a single z/OS ICF catalog. Each member has its own IRLM, buffer pool, EDM pool, workfiles, log, and local BSDS. The sysplex provides support for other facilities that are required for data sharing that include a global lock manager, and shared storage.

Hangs in a data sharing environment

In a data sharing environment, there can be two types of hangs: single system hangs and group hangs (multi-system hangs).

A single system hang means that the agent that is hanging is contending for resources that are held by other agents on the same Db2 member. A single-system hang can be diagnosed with a console dump of only the one Db2, and its associated IRLM.

A group hang means that the agent or agents that are hanging are contending for resources that are held by other agents on other members. Diagnosing a group hang requires console dumps from multiple Db2 systems, and their associated IRLMs.

If a hang is in a sysplex and there is any doubt of the extent of the hang, take dumps of all instances of Db2 and their IRLMs.

If applications hang in the data sharing environment, there are several places where the problem can occur. The problem location might be:

- The local Db2/IRLM
- A peer member Db2/IRLM
- In the z/OS XES component within the sysplex

The probable cause for a hang in the data sharing environment is a problem with one of the following functions:

- P-lock negotiation
- Global locking
- IRLM Notify message sending
- Making the transition from group buffer pool simplex mode to duplex mode, or from duplex mode to simplex mode

Regardless of the type of hang or the symptoms of the hang, you can diagnose it by using the procedures in [“WAIT/LOOP keywords” on page 19](#).

Timeouts and deadlocks in a data sharing environment

A timeout or deadlock in the data sharing environment is similar to one in the non-data sharing environment, except that the problem might be caused by a lock that is held by a peer group member rather than a local or remote user on the same subsystem.

When a deadlock or timeout occurs, you receive one or more of the messages DSNT318I, DSNT375I, DSNT376I, or DSNT377I. Those messages display the holder of the lock and its Db2 member name and provide detailed diagnostic guidance.

Use the command `DISPLAY DATABASE` to gather information about the lock status of all members of the data sharing group. Specify the `LOCKS` keyword for information about transaction locks for table spaces, tables, index spaces, and partitions. Specify the `CLAIMERS` keyword for information about claims on table spaces, tables, index spaces, and partitions.

Issue the IRLM `MODIFY irlmproc,DIAG,HANG` command to collect SVC dumps for all the IRLM instances in the sysplex. This command should be used only under the direction of your IBM service representative. This command prevents the `TIMEOUT/LOCK SRB` from running until z/OS dump services reschedules it. After this command runs, you must ensure that IRLM returns to normal processing.

You can activate statistics trace class 3 or performance trace class 6 to get information about timeouts and deadlocks. IFCID 0196 gives timeout statistics, and IFCID 0172 gives deadlock statistics. Start the trace on the member where the application with timeout or deadlock problems runs. When a timeout or deadlock occurs, Db2 gathers information from all other members of the data sharing group.

Related reference

[-DISPLAY DATABASE command \(Db2\) \(Db2 Commands\)](#)

[Trace field descriptions \(Db2 Performance\)](#)

Related information

[DSNT318I \(Db2 Messages\)](#)

[DSNT375I \(Db2 Messages\)](#)

[DSNT376I \(Db2 Messages\)](#)

[DSNT377I \(Db2 Messages\)](#)

IRLM delays in a data sharing environment

IRLM can experience delays in child lock propagation

The following events can cause IRLM to experience delays in child lock propagation:

- Member recovery
 - XES delays work but does not inform IRLM of the failed connection.
- Loss of connectivity of a lock structure
 - XES delays work but does not request IRLM to rebuild or disconnect a lock structure.
- XES is slow to process a lock request for an undetermined reason.

IRLM can also experience a delay in P-lock negotiation. A common reason for a delay in P-lock negotiation is that an active log is full, and Db2 is waiting for the operator to mount an archive tape.

If your data sharing system experiences delays in child lock propagation or P-lock negotiation, you can request dumps of all IRLM instances in the data sharing group when a delay for child lock propagation lasts 45 seconds or more, or a delay for P-lock negotiation lasts two minutes or more. Issue this console command to request IRLM dumps for delays in child lock propagation:

```
MODIFY irlmproc,DIAG,DELAY
```

Issue this console command to request IRLM dumps for delays in P-lock negotiation:

```
MODIFY irlmproc,DIAG,PLOCK
```

Issue this console command to request IRLM dumps for either type of delay:

```
MODIFY irlmproc,DIAG,ALL
```

You need to issue `MODIFY irlmproc,DIAG` commands on only one member of a data sharing group. These commands are sent to all peer members to capture the same dumps from each member. A `MODIFY irlmproc,DIAG` command is active for only a single incident on a single member. After dump processing is triggered for a specific incident, dump collection for later incidents of that type are disabled. If diagnosis of another incident is needed, issue the `MODIFY irlmproc,DIAG` command again.

Related concepts

[Concurrency and locks in data sharing environments \(Db2 Data Sharing Planning and Administration\)](#)

Inconsistent data in a data sharing environment

As is true in the non-data-sharing environment, problems with inconsistent data in the data sharing environment might be caused by broken pages, inconsistent data pages and index pages, or down-level data sets.

One cause of inconsistent data occurs only in the data sharing environment: Db2 might be improperly maintaining communication or data coherency between members. If more than one group member returns incorrect output, it might be an indication of a buffer pool coherency problem. For diagnosis of this problem, you must provide merged logs from all members of the data sharing group. Use the `DSN1LOGP` utility to obtain them. To format the merged logs, include the following in your `DSN1LOGP` job:

- A GROUP DD statement in the JCL. This statement names a BSDS for one member of the data sharing group. Db2 can find the rest of the information from that one BSDS.
- The option `LRSNSTART` in the `DSN1LOGP` statement.

You might also be asked to provide the following for all members of the data sharing group:

- A dump of the group buffer pool (GBP) and shared communications area (SCA) list structures.

You can use an `z/OS DUMP` command similar to this one to obtain a dump of each group buffer pool:

```
DUMP COMM=(GBPx-dump-name)
nnn,STRLIST=(STRNAME=GBPx-structure-name,
CONNAME=conname,ACCESSTIME=NOLIMIT,
(STGCLASS=ALL,COCLASS=ALL,EDATA=SER)),END
```

To obtain a dump of the shared communications area, use a `DUMP` command similar to this one:

```
DUMP COMM=(SCA-dump-name)
nnn,STRLIST=(STRNAME=SCA-structure-name,
(LISTNUM=ALL,EDATA=SER)),END
```

- Dumps of all Db2 and IRLM address spaces
- Console logs
- `SYS1.LOGREC` data sets

Related tasks

[Identifying Db2 data inconsistency problems](#)

Data inconsistency problems occur for various reasons, including internal Db2 problems, I/O errors, or system problems.

Query parallelism problem diagnosis

Query parallelism is the ability to process read-only queries in parallel. Queries that are most likely to be processed in parallel have access paths that perform sequential prefetch against page sets that consist of multiple data sets.

Db2 uses query CP parallelism. A single Db2 subsystem on an z/OS system with multiple processors uses multiple parallel tasks to process a query.

Db2 uses multiple modes of query parallelism:

DB2 uses multiple modes of query parallelism:

Query I/O parallelism

Db2 prefetches data from several partitions of a table space at one time. A single processor processes the first request from each partition, then the second request from each partition, and so on. The processor does not wait for I/O, but there is only one processing task.

Query CP parallelism

A single Db2 subsystem on an z/OS system with multiple processors uses multiple parallel tasks to process a query.

Related concepts

[Parallel processing \(Db2 Performance\)](#)

Related tasks

[Enabling query parallelism \(Db2 Performance\)](#)

[Disabling query parallelism \(Db2 Performance\)](#)

Determine if a query problem is related to parallelism

The first question to ask when you encounter a problem in a parallel query is whether that problem is related to parallelism. To answer that question, disable all parallelism, then try the query again.

If the problem occurs whether or not you have parallelism enabled, you can assume that it is unrelated to parallelism. If the problem occurs only with parallelism enabled, follow these steps to determine which type of parallelism is causing the problem:

1. Disable all parallelism. Run the problem query again.
2. If the problem still occurs, enable CP parallelism. Run the query again.

When you run the query with CP parallelism, issue the following commands and collect the output:

DISPLAY BUFFERPOOL

The output shows the buffer pool thresholds for parallelism. Issue this command for all constrained buffer pools.

DISPLAY THREAD

Issue this command if you suspect a hang problem. The output shows PT in the status column for parallel threads. It also indicates which task is the originating task.

Disabling all parallelism

To disable parallelism, take one of the following actions:

- For queries that use static SQL statements, bind the plan or package that contains the queries with the option DEGREE(1).
- For queries that use dynamic SQL statements, run the following SQL statement before you run the queries:

```
SET CURRENT DEGREE='1'
```

Disabling CP Parallelism

To disable CP parallelism and leave I/O parallelism enabled

1. Determine which processors are online by issuing the z/OS command:

```
D M=CPU
```

2. Vary all but one processor offline by issuing the z/OS command:

```
CF CPU(n),OFFLINE
```

Disabling parallelism by using the resource limit facility

If you use the resource limit facility (RLF), you can disable CP parallelism for static or dynamic SQL statements by specifying the RLST values shown in the following table. Use RLF to disable parallelism for static statements only for diagnosis.

Table 37. RLST values for disabling parallelism

| RLFFUNC | RLFBIND | Effect |
|---------|---------|--|
| 4 | blank | Disable CP parallelism for dynamic statements |
| 4 | S | Disable CP parallelism for static plans or packages |
| 4 | A | Disable CP parallelism for static and dynamic statements |

These RLST values do not affect the way RLF governs the processor time for dynamic statements. To use RLF to change the parallelism mode for a plan or package, you must rebind the plan or package while RLF is active. For data sharing, RLF must be active on the Db2 member where the rebind occurs.

Related concepts

[Parallel processing \(Db2 Performance\)](#)

Related tasks

[Disabling query parallelism \(Db2 Performance\)](#)

[Enabling query parallelism \(Db2 Performance\)](#)

Related reference

[CURRENT DEGREE special register \(Db2 SQL\)](#)

[-DISPLAY BUFFERPOOL command \(Db2\) \(Db2 Commands\)](#)

[-DISPLAY THREAD command \(Db2\) \(Db2 Commands\)](#)

Types of parallelism problems

Parallel queries can experience several types of problems.

Hangs

The diagnostic procedures for hangs during parallel processing are similar to those for hangs during other types of Db2 processing.

One useful source of diagnostic information is the output that you receive when you run DSNWDMP with option DS=3 on dumps of the Db2 master address spaces. With DS=3, you can see what parallel tasks were running when the hang occurred.

Incorrect output

Diagnosing incorrect output problems for query parallelism is similar to diagnosing problems when there is no parallelism.

If you collected trace data for IFCID 0221, you can check that data for parallel processing errors. Examine the page range or key range values in the QW0221D sections. There is a QW0221D section for each parallel task. Ensure that the ranges in any two QW0221D sections do not overlap.

If the query produces output, you can obtain further diagnostic information by forcing a dump in CSECT DSNXROUA while the query runs. To do that, run utility DSN1SDMP with these control statements:

```
* START A GLOBAL TRACE TO TRACE MODULE ENTRIES AND EXITS
START TRACE=GLOBAL CLASS(3) DEST(OPX)
FOR(1)
* ABEND WHEN THE SELECT CRITERIA ARE MET
ACTION(ABENDTER(00E60100))
SELECT
* FIND THE LOCATION OF THE EVENT CODE
P2,06
* ABEND WHEN THE EVENT CODE IS DSNXROUA ENTRY
DR,00,X'023A'
```

Next, run the query and collect the resulting dump data.

No parallelism

The access path that Db2 chooses for a query is one factor that determines whether the query uses parallelism. For example, in a correlated subquery, the inner table access (single row retrieval) is not a good candidate for parallelism because the overhead of parallelism is too high. Db2 does not support parallel access for certain other access paths, such as an IN-list index scan.

Even when the access path is right for parallelism, Db2 might choose sequential access instead. Some reasons for this are:

- Host variable values limit the query to a single partition.
- ESA sort facility hardware is not available.
- Buffer pool resources are constrained.

If you think a query that is not being processed in parallel should be, collect the following information:

- A trace for IFCID 0221. This trace indicates whether buffer pools are constrained. IFCID 0221 records are written when performance trace class 8 is on.
- An SVC dump of the database services address space (*ssnmDBM1*).
- Execute the EXPLAIN statement for the problem queries and select all columns from the PLAN_TABLE. For column IBM_SERVICE_DATA, select HEX(IBM_SERVICE_DATA). The information in the PLAN_TABLE indicates whether the access path for the statement is right for parallelism.

ABENDs

When a parallel task abnormally terminates, the originating task returns SQLCODE -904 with reason code X'00E30101'. Db2 generates no dump for the originating task. Obtain the dump that is generated for the parallel task and analyze it as you would any other dump that is produced during a query.

Related concepts

[WAIT/LOOP keywords](#)

The symptoms for WAIT and LOOP keywords might not be distinguishable at first.

Related tasks

[INCORROUT modifier keyword](#)

Use this modifier keyword if the type-of-failure keyword is INCORROUT.

Related reference

[Format dumps by using the DSNWDMP statement](#)

You can use the DSNWDMP statement to specify the dump records to be used as input, and causes the Db2 dump formatter (DSNWDPRD) to be invoked, which formats the specified Db2 control blocks.

Diagnose parallelism problems by using traces

Db2 generates a number of trace records that are useful in diagnosing parallelism problems.



The IFCIDs for the trace records are 0221, 0222, and 0231. Those IFCIDs are in performance trace class 8. See the IFCID flat file (DSNWMSGs) for explanations of those trace records.

◀ PSPI

Related concepts

[Trace field descriptions \(Troubleshooting problems in Db2\)](#)

Chapter 5. Diagnostic aids for distributed data

Use traces, error messages, and -DISPLAY THREAD command output to diagnose distributed problems.

Diagnosing distributed data facility (DDF) failures

It is important to understand the flow of distributed requests and the procedures that can be used for doing problem determination for Db2 for z/OS in a distributed environment.

Distributed SQL application flow for VTAM connections

When the application program (the requester) issues the first SQL statement that references a remote database system, an LU6.2 conversation must be established with the remote database system (the server).

The APPC ALLOCATE verb is used to establish a conversation. When you look at a System Network Architecture (SNA) buffer trace, an ALLOCATE verb can easily be spotted, because the first path information unit (PIU) transmitted after the ALLOCATE is marked with the begin bracket (BB) flag. Path information unit is the structure for transmitting data through the network.

Once the LU6.2 conversation is allocated, the local Db2 system builds a message that represents the application's SQL statement. The APPC SEND_DATA verb is used to send the SQL statement to the server for processing. During transmission, the message that represents the SQL statement is broken up into path information units (PIUs), where the SQL data in each path information unit (PIU) is no larger than the request unit (RU) size that is selected by the communication administrator. If the data does not fill a complete PIU, the data remains in the APPC buffer until the PREPARE_TO_RECEIVE verb is issued.

System Network Architecture (SNA) pacing is what the receiver application can use to control the rate at which the sending application transmits data. A pacing window can be defined, which specifies the maximum number of path information units that can be transmitted by the sender before the sender must wait for the receiver to process the data. The sender must request pacing responses at the intervals that are dictated by the pacing window size.

The requester's Db2 system issues the APPC PREPARE_TO_RECEIVE verb to tell the server that it is waiting for an answer to the SQL statement. The PREPARE_TO_RECEIVE verb transmits the very last path information unit (PIU) for the SQL statement, and marks the path information unit (PIU) with the change direction (CD) indicator. When the path information unit (PIU) immediately follows an ALLOCATE verb, the path information unit (PIU) is also marked with the BB (Begin Bracket) indicator.

When the server detects the change direction (CD) indicator, it knows the SQL statement is complete and the requester is waiting to receive the answer to the SQL statement.

The server runs the SQL statement, and builds a reply message that represents the answer to the SQL statement. As before, the answer message is broken into path information units for transmission, with the last path information unit (PIU) in the answer marked change direction (CD). The change direction (CD) indicator tells the requester that the SQL answer is complete, and the server is waiting for the next SQL statement.

The requesting database returns the SQL answer to the application program, and waits for the next SQL request from the application.

The next SQL request is also converted into a message that represents the application's SQL statement.

The message is broken into path information units for transmission, where the last path information unit (PIU) is marked change direction (CD). In this case, the SQL request fits in a single PIU. As before, the change direction (CD) indicator tells the server that the SQL statement is complete, and the requester is awaiting the answer message.

The server sends the reply message, marking the last path information unit (PIU) with change direction (CD).

When the application program terminates, a DEALLOCATE verb is issued by the requesting database to tell the database server that the distributed database application is complete. The DEALLOCATE verb causes a path information unit (PIU) containing the conditional end bracket CEB flag to be transmitted to the server, which tells the server that the application is complete.

Distributed SQL application flow for TCP/IP connections

When the application program (the requester) issues the first SQL statement that references a remote database system, a connection must be established with the remote database system (the server).

The TCP/IP connect socket call is used to establish a conversation.

The requester's Db2 system issues the READ call to tell the server that it is waiting for an answer to the SQL statement.

Once the connection is established, the local Db2 system builds a message that represents the application's SQL statement. The WRITEV call is used to send the SQL statement to the server for processing.

When the application program terminates, a CLOSE call is issued by the requesting database to tell the database server that the distributed database application is complete. The CLOSE call tells the server that the application is complete.

DDF error messages

A distributed request can run unsuccessfully for a number of reasons.

- Local DDF is not started.
- Remote DDF is not started (if the remote system is a Db2).
- Remote system is not started.
- VTAM LU is not active.
- VTAM path errors occurred.
- VTAM failure occurred.
- Session or conversation failures occurred.
- Db2 communications database entries are incorrect.
- Security failure occurred.
- TCP/IP failure occurred.

Resource unavailable messages for DDF

For some errors in the distributed environment, the Db2 user or application receives:

```
SQLCODE -904; RESOURCE UNAVAILABLE
```

The following is returned to the user in the SQLCA: The Db2 reason code, a code that describes the type of the resource that is unavailable, and the resource name. Distributed data facility reason codes start with '00D3'.

RESOURCE TYPES for errors in a distributed environment are:

- 00001000 DDF
- 00001001 System conversation
- 00001002 Agent conversation
- 00001003 CNOS processing
- 00001004 Communications database
- 00001005 Database access agent
- 00001007 TCP/IP domain name

- 00001008 TCP/IP service name

VTAM resource information

For VTAM connections, the RESOURCE NAME contains:

- LU-name
- VTAM logon mode name
- VTAM return codes
- SNA sense data.

Four VTAM return codes are displayed in the Db2 error message:

- RTNCD: VTAM primary return code
- FDBK2: VTAM secondary return code
- RCPRI: APPC primary return code
- RCSEC: APPC secondary return code.

The SNA sense data is 4 bytes long and consists of:

- BYTE 0 + 1: Sense code: Category (Byte 0) and Modifier (Byte 1)
- BYTE 2 + 3: Sense code-specific information.

| Sense Data | | | |
|------------|----------|------------------------------------|--------|
| Sense Code | | | |
| Byte 0 | Byte 1 | Byte 2 | Byte 3 |
| Category | Modifier | Sense Code Specific Information | |

Figure 80. SNA sense data

The SNA sense data categories are:

- X'00': User sense data only (not for LU 6.2)
- X'08': Request reject
- X'10': Request error
- X'20': State error
- X'40': Request header (RH) usage error
- X'80': Path error.

If RTNCD contains '00' and FDBK2 contains '0B', the requested VTAM function was unsuccessful and RCPRI and RCSEC should be analyzed. Together with the SNA sense data, the problem can be identified.

CNOS processing for LU 'LUDBD1' (which is DB2 system 'SYDNEY') and VTAM logon mode 'SYSTOSYS' was unsuccessful. The VTAM return codes must be analyzed:

```
-----
SELECT * FROM SYDNEY.SYSIBM.SYSCOLUMNS;
-----
DSNT408I  SQLCODE = -904, ERROR:  UNSUCCESSFUL EXECUTION CAUSED BY AN
UNAVAILABLE RESOURCE. REASON 00D31029, TYPE OF RESOURCE 00001003, AND
RESOURCE NAME LUDBD1.SYSTOSYS.00.0B.0008.0001.08570003
DSNT415I  SQLERRP = DSNLXCNV SQL PROCEDURE DETECTING ERROR
DSNT416I  SQLERRD = 900 0 0 0 0 0 SQL DIAGNOSTIC INFORMATION
DSNT416I  SQLERRD = X'FFFFFFC7C' X'00000000' X'00000000' X'00000000'
X'00000000' X'00000000' SQL DIAGNOSTIC INFORMATION
-----
DSNE618I  ROLLBACK PERFORMED, SQLCODE IS 0
DSNE616I  STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
-----
```

The DB2 error message is shown below:

```
DSNT408I  SQLCODE = -904, ERROR:  UNSUCCESSFUL EXECUTION CAUSED BY AN
UNAVAILABLE RESOURCE. REASON 00D31029, TYPE OF RESOURCE 00001003,AND
RESOURCE NAME LUDBD1.SYSTOSYS.00.0B.0008.0001.08570003
```

SQLCODE -904, RESOURCE UNAVAILABLE, tells that the SQL statement could not be executed, because a specific resource was not available at the time the statement was executed:

```
+-----+
DSNT408I |SQLCODE = -904, ERROR:  UNSUCCESSFUL EXECUTION CAUSED BY AN |
|UNAVAILABLE RESOURCE.|
+-----+
```

DB2 reason code X'00D31029' tells that the requested VTAM function to allocate a conversation failed (VTAM returned a non-zero return code):

```
+-----+
|REASON 00D31029|
+-----+
```

Code '00001003' defines the resource type of 'CNOS processing':

```
+-----+
|TYPE OF RESOURCE 00001003|
+-----+
```

Because the VTAM function to perform CNOS processing failed, the specific resource name must be analyzed. The combination of RTNCD/FDBK2 '00/0B' tells that the VTAM function was unsuccessful and the APPC codes should be analyzed. The combination of RCPRI/RCSEC '0008/0001' tells that a conversation cannot be allocated. The SNA SENSE field '08570003' tells that the SSCP-SLU session is not active. In other words, DDF of 'SYDNEY' is not active.

```
+-----+
|RESOURCE NAME LUDBD1.SYSTOSYS.00.0B.0008.0001.08570003|
+-----+
```

```

LUNAME <-----+
Logon Mode Name <-----+
RTNCD (VTAM primary return code) <-----+
FDBK2 (VTAM secondary return code) <-----+
RCPRI (APPC primary return code) <-----+
RCSEC (APPC secondary return code) <-----+
SENSE (SNA sense code) <-----+

```

Figure 81. Db2 DDF error message in VTAM environment -- resource unavailable

TCP/IP resource information

For TCP/IP connections, the RESOURCE NAME contains:

- Value in column LINKNAME in table SYSIBM.IPNAMES
- Value in column IPADDR in table SYSIBM.IPNAMES
- TCP/IP error code from socket call gethostbyname or getservbyname

if the resource is a TCP/IP domain name, or

- Value in column LOCATION in table SYSIBM.LOCATIONS
- Value in column PORT in table SYSIBM.LOCATIONS
- TCP/IP error code from socket call gethostbyname or getservbyname

if the resource is a TCP/IP service name.

Example TCP/IP and Db2 error messages

The following figure shows an example of messages you receive on the z/OS console when a local Db2 subsystem attempts to connect to a remote Db2 subsystem, but DDF is down on the remote system.

```
DSNL511I ? DSNLIENO TCP/IP CONVERSATION FAILED
          TO LOCATION STLEC1B
          IPADDR=9.112.126.120 PORT=447
          SOCKET=CONNECT RETURN CODE=1128 REASON CODE=12F80291
```

Figure 82. Analyzing Db2 and TCP/IP error message on z/OS console

Message DSNL511I indicates that the TCP/IP CONNECT socket call failed. The CONNECT call establishes a connection between the socket for the local Db2 and the socket for the remote Db2. The IPADDR and PORT values are for the remote Db2.

TCP/IP startup problems

TCP/IP startup problems are common sources of unsuccessful DDF operation.

- The TCP/IP listener is not started because of a TCP/IP configuration problem, or because the DNS is not available during DDF startup.

In this situation, DDF startup completes, but the domain name in message DSNL004I is -UNKNOWN. Connections cannot be established until the TCP/IP listener is successfully established and TCP/IP services become available for DDF. After the initial connection failure, Db2 periodically attempts to reinitialize TCP/IP. When the connection is successful, Db2 issues message DSNL519I.

- A TCP/IP getaddrinfo socket call fails during DDF startup.

When this failure occurs, Db2 issues the following message:

```
DSNL512I ? DSNLILNR TCP/IP GETADDRINFO(myhostname) FAILED WITH
          RETURN CODE=1 AND REASON CODE=78AE1004
```

Possible causes:

- The DDF application cannot access the TCPDATA data set.
- The DNS is not available.
- There is no entry for the IP address of the local host in the DNS.
- There is no entry in the `/etc/hosts` file.
- There is no entry in the `hlq.HOSTS.ADDRINFO` data set.

The following search order is used to find the TCPDATA data set for the DDF application:

1. `/etc/resolv.conf` file
2. Data sets in the `//SYSTCPD DD` statement
3. `jobname.TCPIP.DATA`
4. `SYS1.TCPPARMS(TCPDATA)`
5. `TCPIP.TCPIP.DATA`

The first file in this list is used as the TCPDATA file. After this file is correctly defined to DDF, you can use the TRACE RESOLVER statement in this file to further diagnose a problem. The resolver trace displays the IP address that is being resolved, so you can confirm that the `gethostid` call was successful. The resolver trace also displays any errors that are encountered with the DNS or local files. The output of the resolver trace is in the JES log of the DDF started task, `DSNDIST`.

Db2 hangs during distributed processing

If a user or application "hangs" during distributed processing, the condition that causes the problem can be at the requesting (local) system, the serving (remote) system, or in the network (VTAM, TCP/IP, CTC, or NCP).

At a Db2 system, the command DISPLAY THREAD DETAIL can be used to get information about distributed threads. This command should be issued at both requesting and serving locations to get the status of the distributed thread.

The command -DISPLAY THREAD (*) TYPE(SYSTEM) can be used to identify the tokens of DB commands and system agents that can be canceled. After the tokens are identified, you can use the -CANCEL THREAD (token) command to cancel DB commands and system agents that are in progress.

Canceling a distributed thread that is hung in VTAM

When your network connection is through VTAM, a distributed thread can be:

- Active in VTAM
- Waiting in Db2 for VTAM notification that a particular event is completed
- Not in VTAM, not in Db2 (for example, waiting for user input).

If the distributed thread is not active in VTAM, it can be canceled by using the CANCEL DDF THREAD command. A distributed thread should be canceled at the requesting location. In this case, both the allied thread (at the requesting location) and the database access threads (at the serving location) are terminated with an SVC dump. If the distributed thread is canceled at the serving site, only the database access thread is terminated with a dump.

If a distributed thread is hung in VTAM, the following VTAM command can be used to terminate the session:

```
V NET,TERM,SID=SESSION-ID
```

To terminate a session, the session identifier must be known. The following procedure can be used to find the session identifier and terminate the thread:

1. Use the Db2 command DISPLAY THREAD LOCATION DETAIL to identify the hung thread. Get the session identifier (field SESSID).

A Db2 command DISPLAY THREAD LOCATION DETAIL is used to find the hung thread. In this example, it is authorization ID SYSOPR by using Db2 plan CAN1. The session identifier is 'F0EF951D7B824660':

```
?DIS THD(*) LOC(*) DET
DSNV401I ? DISPLAY THREAD REPORT FOLLOWS -
DSNV402I ? ACTIVE THREADS - 056
NAME      ST A   REQ ID      AUTHID   PLAN      ASID      TOKEN
TSO       TR   178 SYSOPR      SYSOPR   CAN1      0012      3
-IMSNET.LUDBD2.A0FFF131B239=3 ACCESSING DATA AT
--SYDNEY
--LOCATION          SESSID          A ST TIME
--SYDNEY          F0EF951D7B824660 S 8927509332750
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I ? DSNVDT '?DIS THD' NORMAL COMPLETION
```

Figure 83. Session identifier

2. Use the VTAM command D NET, ID=1uname, SCOPE=ALL: Take the last 7 bytes of the session identifier that is provided by the DISPLAY THREAD DETAIL command from step 1 to correlate the VTAM session identifier (field SID). In the following figure, the VTAM session identifier is 'E2EF951D7B824660'.

```

D NET, ID=LUDBD2, SCOPE=ALL
IST097I DISPLAY ACCEPTED
IST075I NAME = LUDBD2, TYPE = APPL
IST486I STATUS= ACTIV, DESIRED STATE= ACTIV
IST861I MODETAB=DB2MODES USSTAB=***NA*** LOGTAB=***NA***
IST934I DLOGMOD=***NA***
IST597I CAPABILITY-PLU ENABLED ,SLU ENABLED ,SESSION LIMIT NONE
IST654I I/O TRACE = OFF, BUFFER TRACE = OFF
IST271I JOBNAME = DBD2DIST, STEPNAME = DBD2DIST
IST171I ACTIVE SESSIONS = 0000000004, SESSION REQUESTS = 0000000000
IST206I SESSIONS:
IST634I NAME          STATUS          SID          SEND RECV VR TP NETID
IST635I LUDBD1       ACTIV-S       E2EF951D7B824660 000A 0010 0 0 IMSNET
IST635I LUDBD1       ACTIV-S       E2EF951D7B82465F 001D 0000 0 0 IMSNET
IST635I LUDBD1       ACTIV-P       E3EF951D7C824D69 0000 000F 0 1 IMSNET
IST635I LUDBD1       ACTIV-P       E3EF951D7C824D68 0002 0002 0 1 IMSNET
IST314I END

```

Figure 84. Session identifier

3. Use the session identifier that is provided by the VTAM DISPLAY command from step 2 to terminate the session by way of the VTAM TERMINATE command `V NET, TERM, SID=session-id`.

```

V NET, TERM, SID=E2EF951D7B824660
IST097I VARY ACCEPTED
MSG0: PLEASE STAND BY .....
IST455I SID=E2EF951D7B824660 SESSIONS ENDED

```

Figure 85. Session identifier

After the session is terminated, APPC primary/secondary return codes of RCPRI=0048, RCSEC=0000 are returned to Db2. This combination of RCPRI and RCSEC indicates that the conversation was terminated because the session, which was used by the conversation, was terminated. This combination of RCPRI/RCSEC is called "Resource failure, no retry". At the remote site (subsystem recognition character "!"), APPC primary/secondary return codes of RCPRI=004C, RCSEC=0000 are returned to Db2. This combination is called "Resource failure, retry".

Canceling a distributed thread with a TCP/IP connection

When your network connection is through TCP/IP, a distributed thread can be:

- Active in TCP/IP
- Waiting in Db2 for TCP/IP notification that a particular event is completed
- Not in TCP/IP, not in Db2 (for example, waiting for user input)

A distributed thread that uses a TCP/IP connection can be canceled by using the Db2 command `CANCEL DDF THREAD`. Use the Db2 command `DISPLAY THREAD DETAIL` to identify the hung thread. A distributed thread should be canceled at the requesting location. In this case, both the allied thread (at the requesting location) and the database access threads (at the serving location) are terminated with an SVC dump. If the distributed thread is canceled at the serving site, only the database access thread is terminated with a dump.

Problem determination procedures

Distributed processing problems fall into several general categories.

Abends

All programs can abnormally end (ABEND) at one time or another. Most applications have some sort of recovery processing to try to avoid actually terminating the application. VTAM and TCP/IP are fairly typical in this respect, and usually attempt to recover before they terminate.

If there is no other indication of a problem, check whether a dump was taken. For an ABENDOC4 or ABENDOA9, usually an SVC dump of the address space is taken. In general, for complete diagnosis of a VTAM or TCP/IP problem, CSA must be dumped.

From the dump, the location of the failure can be determined. This information is often included in a dump summary, formatted at the top of the dump. This information can be used to check in INFOSYS (if INFOSYS is available at your site) for known problems in this area. If INFOSYS is not available or a fix cannot be found, then contacting the IBM Support Center or IBM Service with the following information often resolves the problem.

- Abend code
- Failing module name

This information can be gathered from the dump, or sometimes from the console log messages that are issued at abend time. If the module name is not included in the dump summary information, then locate the PSW address (second word of the PSW) in the dumped storage. Work backwards through the dump (towards zero) until an eyecatcher is encountered. The eyecatcher includes the module name. VTAM module names all start with 'IST'. TCP/IP module names all start with 'TC'. (The eyecatcher is alphanumeric data that is written at the front of the module. It can be read from the EBCDIC conversion area on the right side of the dump. For example, in the dump, the module name could be x'C9E2E3D6D9C6C2C1'. In the EBCDIC conversion area, you would see, 'ISTORFBA'.)

- Offset of PSW into this module

Usually, before the eyecatcher is an x'47....' instruction, which branches around the eyecatcher and into the module code. This instruction is the start of the module. Subtract the address of this instruction from the PSW address to determine the offset. Remember that this is in hex, not decimal. If the module is in LPA, an LPA map can be generated. The LPA map indicates the start of the module. Work out the offset by using this start address.

- Latest maintenance to hit this module

This information can be gathered either from the dump, or by using SMP/E. Again, find the eyecatcher in the dump. This information normally includes an assembly date and the latest PTF to be applied to the module. Write down both the assembly date and the PTF number.

- Registers at the time of the abend

This information can be taken either from the console log or the dump. Register 15 sometimes contains a return code that helps problem diagnosis; for example, on an ABEND0A9.

- Recent maintenance that is applied

PTFs or APARs that were recently applied might contribute to the circumstances that caused the problem.

- Recent changes to the system

In general, problems occur only after changes are made to a system. Keep an open mind when you look at recent changes. Even changes that appear unconnected can be the cause of the problem.

- Frequency of the abend

How often the abend occurred can give some idea of the magnitude of the problem. If the abend is occurring frequently, it indicates that something fairly fundamental is wrong. If it is a one-time occurrence, it might be an obscure set of circumstances that caused the problem.

IBM Support might request more information. The console log around the time of the abend and the dump should be kept in case further diagnosis of the problem requires more information from the dump.

Abends in VTAM

VTAM abends fall into two categories: those where there is an abend in the VTAM address space, and those where the abend occurs in the VTAM code in a user (like DDF) address space.

The link pack area (LPA) has VTAM code, which is accessible by all address spaces. VTAM has its control blocks in the common service area (CSA). These too, are accessible by all address spaces. In this way, each address space can do VTAM processing in its own address space. In general, this is I/O processing: sending and receiving data across the application interface. In this way the DDF

address space (*ssnmDIST*) might abend in a VTAM module and cause a dump of the DDF address space (*ssnmDIST*) to be taken.

VTAM has several z/OS abend codes that are reserved for VTAM's use only. These are ABEND0A7, 0A8, 0A9, 0AA, 0AB, 0AC, and 0AD. The most commonly occurring of these abends is ABEND0A9. The value in general-purpose register 15 (GPR15) gives some indication of the problem.

Sometimes the ABEND0A9 is preceded by an ABENDOC4. Always look for an indication of a previous error. This could be another abend, or an error message to the user or the console. It is not necessarily a VTAM message, and might not occur immediately before the abend. LOGREC is a good source for finding abends that might have been missed on the console by the operators. It is important to start diagnosing a problem when the first sign of trouble occurs, as later problems are often part of the aftermath of the original problem. Recovery routines normally try the failing process again before eventually terminating.

Performance problems and hang situations

When the problem is a hang, it can be difficult to determine the cause of the problem.

Always look for any indication of a problem on the console log. If the log contains nothing of immediate interest, try to determine the extent of the hang. Try to put the users that are hung into groups with common characteristics:

Are only users of one particular application hung?

For example, local Db2 (on DBD1) requests are working, but requests to another remote Db2 (DBD2) are hanging.

Are local requests on the remote host (DBD2) working?

If not, this situation would appear to be a problem on the other host. Look for an abend, loop, or wait type problem. Check the remote host console for any relevant messages. Check whether Db2 commands on the remote host (DBD2) work.

For example, if no commands work, this situation could mean a problem in Db2. If most commands work, but a cancel thread command does not, this situation might indicate a problem in the DDF address space (*ssnmDIST*) or in VTAM or TCP/IP.

If local requests are working, then the problem would appear to be in either the network or the DDF address space (*ssnmDIST*).

Is any network traffic flowing between the two hosts?

Check whether there are any other network users hung. If there are, then this situation is most likely a network problem.

Are all the hung users in one particular part of the network?

For example, all hung users might have terminals on one controller. In this case, the problem could be the controller. Display the status of the controller, for it might need to be used again. Another example could be that all the hung users are accessing the network through one particular link and this link is having a large numbers of errors. This situation could result in slow response times because of error recovery and retries. Investigate the link problem.

When a device or line that is connected to an NCP goes inoperative (INOP), NCP generates a miscellaneous data record (MDR). This information is sent to the owning SSCP. These records are found in LOGREC and can be viewed by requesting an EREP report. The records can also be seen online by using NetView Hardware Monitor (or an equivalent). These records contain information about why the resource went INOP. They are very useful for problem determination.

Is any particular address space using very high CPU utilization?

If so, monitor this situation, as the address space might be in a loop.

Are all users hung?

If so, this could be a more fundamental problem with the operating system. Check whether any z/OS commands are working.

In any hang situation, the results (or lack of results) from various displays can give a clearer picture of the scope of the problem. The more information that is available, the easier the problem diagnosis is, and, usually, the faster the resolution is. It is a discouraging task to find a problem in a VTAM dump when only told that it was hanging. It can also be time-consuming.

Questions to ask that are specific to VTAM

What is the status of the remote application LU (LUDBD2)?

The command `D NET, ID=<remote DDF>, SCOPE=ALL` issued on the host that owns the application indicates if the application is active and has any sessions with the local DDF. If the status is not 'ACTIV', check the meaning of the status in the manual and take appropriate action.

The same command that is issued on the local host displays the CDRSC status. Make sure this is ACTIV (or ACT/S) also.

Are sessions already set up?

The command above indicates whether there are active sessions. There are three sessions that are required for system use; one with a LOGMODE of SNASVCMG, and two with SYSTOSYS LOGMODE. If there are no sessions, then check the virtual route between the subareas.

Are the sessions working?

By repeating the displays, check the send and receive counts on the user sessions. Also, a `DISPLAY THREAD(*) DETAIL` command shows whether there is a conversation on each session. The users might be hung waiting for a conversation to end. When a session becomes available for a new conversation, this user's remote database access is processed.

Is the virtual route between the two subareas open and active?

Use a `D NET, ROUTE` command to see if the virtual route is operational. If the virtual route is blocked, this indicates there could be a storage shortage problem somewhere in the network. This might be a host or an NCP storage shortage.

Is the CDRM to CDRM session between the two hosts active?

The command `D NET, ID=<CDRM name>` can be used to display the status of the CDRMs, these should both be 'ACTIV'.

Are all users that are in session working and only those that are causing a session to be established hung?

In this circumstance, the problem could be a VTAM problem. Most of the send and receive data processing is done in the user's address space. All session establishment is done in the VTAM address space.

Do VTAM commands work?

If so, then VTAM seems to be working. If not, then VTAM might be in a wait (unlikely) or loop (more likely) or perhaps VTAM is unable to get a share of the CPU.

VTAM paging: Is VTAM doing a large amount of paging?

If so, what looks like a loop in VTAM could be VTAM running a long chain of control blocks.

If the hang occurs only on a session when one particular request is made, often a VTAM buffer trace can be used to see the last PIUs flowing on the session. These PIUs can often hold the key to the problem.

Loops

When a loop is suspected, there are various actions that can be taken.

CPU usage

Usually a loop can be readily found by displaying CPU usage. It is necessary to have some idea of "normal" CPU usage for the suspected address space, for comparison purposes. In general, any address space that uses an unusually high amount of CPU should be suspected.

Paging rates

The paging rates can be monitored to check for any abnormally high rates. Normal usage is again needed for comparison purposes.

Loop recording

The 3090 and 308X CPUs have a hardware facility to record up to 490 PSWs. This facility is activated from the hardware console and can be used to trace a loop. The output is dumped when an SVC or stand-alone dump is taken. This information can be helpful when you debug a loop problem. For more information about this facility, consult the documentation for your particular CPU.

Dumps

Usually when a loop is encountered, there is not much that can be done to recover the situation. Usually, a dump must be taken and the system restarted. Dumps can be taken in any of the following ways:

z/OS DUMP command

If the CPU has several processors, z/OS commands might still be working. If so, then, requesting a dump of the looping address space might be sufficient to obtain a dump.

Restart dump

If the CPU has only a single processor, then a restart dump can be taken.

Stand-alone dump

If neither of the previous dumping methods is appropriate, then a stand-alone dump can be taken. Normally, this option is the last choice as the entire system is taken down, and an IPL is required to recover.

With a multi-processor CPU a loop can appear as degraded performance. For example, if the CPU has four processors and one is processing the loop, this leaves three CPUs for normal processing. This reduction in processing power normally causes some performance degradation. Any users that are associated with the looping address space are normally hung.

Before you contact the IBM Support Center, try to have the following information available:

- Looping modules. Try to determine the modules that are involved in the loop by using [Failing module \(z/OS Communications Server: SNA Diagnosis\)](#).
- Maintenance level of the modules. Gather this information from SMP/E.
- Messages. Look for any messages on the console that might have triggered the loop. If one message is being repeatedly issued, include it in the description of the loop.
- Trace output. Have the output from any traces taken available, together with any information gained from the trace.
- Dump
- Console log

Have the following items available in case further diagnosis is required.

VTAM internal trace: If the VIT is active, this can give some insight into the problem. If the loop causes entries to be written to the VIT, then the trace table is likely to wrap before the operators have determined that VTAM is in a loop. However, if the loop does not cause trace entries to be written, the VIT can be invaluable in determining the event that initiated the loop.

Storage shortages

Diagnosing storage shortage problems can be difficult. However, with a dump and an idea of where the shortage occurs, the problem source can be identified.

When you contact the IBM Support Center, have available as much relevant information as possible. Remember to check for any changes that were made at the site. Many applications use VTAM's or TCP/IP's services. Changes that are made for one application can adversely affect another application.

Storage shortages in VTAM

Storage shortages can manifest themselves in any of the following ways:

VTAM messages

A VTAM message might be received at the console, indicating a storage shortage. This message normally indicates the z/OS subpool where the shortage occurred. Using this subpool number, the area of storage that experiences the shortage can be determined. For example, subpool 231 (SP231) is CSA, while subpool 17 (SP17) is in the address space private area.

The most common storage shortage problems involve CSA. VTAM uses large amounts of CSA for its control blocks and buffers that contain data that is being sent around the network. This problem can be because of various causes. For example, if one application is flooding another with data, and the second application is unable to receive the data and process it at an adequate rate, then the buffers build up in VTAM storage. This situation can be avoided by using session pacing.

If VTAM is unable to get enough storage to issue a message, then the normal storage shortage message is accompanied by an IST999E message.

RCPRI and RCSEC

The Primary return code (RCPRI) and secondary return code (RCSEC) are both given to DDF when an APPCCMD macro completes. On occasion, these codes might indicate that there is a storage shortage in VTAM. For example, if RCPRI is x'0084' and RCSEC is x'0000', then this indicates a storage shortage while VTAM was receiving data or sending a pacing response. An RCPRI of x'0098' with RCSEC of x'0000' indicates there is a temporary storage shortage while sending data. Usually this return code means that the send request temporarily depleted the buffer pool to such an extent that the pool must be expanded. The expansion did not occur before the completion of the APPCCMD macro.

SNA sense code

Some SNA sense codes indicate there might be a storage shortage also. For example, a user might be accessing data from the remote database, and receive '084C0000'. Checking this sense code in the manual indicates that there is a permanent insufficient resource condition. This resource could be storage. Other sense codes, such as '800A000' indicate a storage type problem, but not necessarily a storage shortage.

Abends

There are a series of z/OS abends, which indicate storage shortage problems. For example, ABEND878 and ABEND80A. These are uncommon in VTAM.

Hangs

Depending on the storage shortage and the processing that is occurring, the storage shortage could manifest itself in a hang situation. For example, if a virtual route becomes blocked because of storage shortages, then all the sessions that were using that session hangs until the storage shortage is relieved and the virtual route becomes open again. When any of these indications of a storage problem is received, the following steps can be used to find out more information about the shortage:

Determine the area of storage shortage:

This task is important, as the VTAM display command has information about the VTAM CSA usage. This information does not help diagnosis if the storage shortage is in VTAM private. However, perhaps CSA usage should be checked anyway.

The area can be determined by checking the z/OS subpool number (for example, as given in a message). The following subpools are in the CSA area:

SP227, SP228, SP231, SP239, and SP241.

Display buffer usage:

If the storage shortage occurred in one of the CSA subpools, then a `D NET ,BFRUSE` command can be used to determine the amount of CSA storage that is being used by VTAM. Usually the buffer use display gives a good idea of which VTAM pool is causing the storage shortage.

Monitor buffer usage:

When you look at a buffer shortage, it is often helpful to know whether the onset of the problem was gradual or immediate. If regular buffer usage displays are done, then gradual increases in buffer use can be seen. These gradual increases might take days to manifest themselves into storage shortage problems. In fact, if VTAM is taken down regularly, the storage shortage symptom might not be seen.

Another benefit of regular monitoring of the buffers is that when a problem does occur "normal" buffer usage for that host is known, so some comparison of the buffer values can be done.

If the onset of the buffer shortage is very fast, then check the system console (or log), looking for some event that has triggered this problem. This could be virtually anything. For example, an NCP has a problem and large amounts of data that was heading out onto the network is now caught in VTAM while recovery of the NCP is attempted.

Create a dump file

To finally determine the cause of the problem, a dump is usually necessary. When you take the dump, be sure to include CSA in the dumping options. Without CSA the dump is almost useless. The VTAM diagnosis manual describes how to find each of the VTAM pools, and what control blocks are allocated in each pool.

Traces

The SMS option on the VTAM Internal Trace can be helpful when you look at storage-related problems. When used with other options (PSS, SSCP and PIU for example), it can give a much clearer insight into the processing that is causing the storage shortage.

The SMS storage trace can be used to monitor the buffer pool usage if regular displays are inconvenient.

Storage shortage problems fall into the following main categories:

No session pacing

In this case, the VTAM IO buffers (IOBUFs) are being flooded by one application. Finding the IOBUFs in the dump can often lead to the application at fault. Usually the problem starts after the introduction of (or changes to) an application. Look for any changes in the system that could have caused the problem.

Control blocks not freed

There have been problems in the past where control blocks were not freed when they were no longer needed. These control blocks gradually fill large amounts of CSA. This type of problem can normally be found with a search in INFOSYS, if this is available at the site, or RETAIN, if you call IBM Support or IBM Service. If a known problem is not found, then the IBM Support Center should be contacted for further problem diagnosis.

Buffer pool unable to expand

If the buffer pool is scheduled for expansion by VTAM, but for some reason is unable to do so, a storage shortage error can be received. This could be a transient condition while the buffer pool is being expanded.

If this is not a transient condition, check that the pool is eligible for expansion. That is, there is an expansion limit and expansion number that is assigned to the buffer pool. If dynamic expansion has not been allowed, then the pool might have used all of its base allocation. In this case, the base allocation should be increased.

Incorrect output

Incorrect output problems can take various forms. In general, the problems can be grouped into two categories. The first category is where the remote access request fails. An SQL return code and SNA sense code or TCP/IP return code and reason code are received. In the second category the request works, but the information that received is not correct.

REQUEST FAILS WITH RETURN CODE: When a remote database access fails, normally the user receives a Db2 DDF message. These codes should be investigated and depending on the type of error, the appropriate action taken.

VTAM example

A user receives an SNA sense x'800A0000'. Checking this sense code in the SNA formats manual tells us that either the PIU was too long, or there was not sufficient buffering available for the PIU.

To diagnose the problem, it is easiest to see the PIUs to check whether the length is over the RUSIZE maximum that is defined in the MODETABLE entry for the session. To do this task, take a VTAM buffer trace of the attempted remote access by starting the trace with ID=<DB2 LU>. Optionally, either (or both) of the Db2 systems can be traced: that is, the local or remote Db2 LU. If the remote Db2 system is traced, we see the PIUs (and the lengths of these PIUs) that the remote Db2 is sending. Large amounts of data are coming from the remote host, so these large PIUs are more likely to be too long than the smaller requests from the local Db2. If the local Db2 is traced, the large PIUs should be seen arriving from the remote Db2. In this trace, look for a PIU that has sense included as part of the RU.

Often a search in INFOSYS, or RETAIN reveals information about the effect of particular VTAM definitions. This often leads to a resolution. If not directly, then indirectly by indicating new areas that can be checked for errors.

Of course, each problem must be treated individually, depending on the information received from the sense codes. Often a VTAM buffer trace helps in understanding a problem that results in a SNA sense code. The trace shows exactly what is happening on the session and the sequence of events that lead up to the issuing of the sense.

Inconsistent data

Under normal circumstances, inconsistent data is not caused by the network. If there is a network problem, all the network users are likely to be affected, not just one application. So, inconsistent data is more likely to be a problem within the database.

For example, if a user does a remote update and the commit fails to complete, the data on the database could be updated or not, depending on how far the commit progressed. A Db2 code, for example, x'00D300FE' might be received when a DDF thread is canceled. In these circumstances, a local request for the updated data should be made to check the actual state of the data to avoid inconsistent data being given to the users.

Unexpected messages

Sometimes the first indication of a problem is a message received at the console.

VTAM example

Message DSNL013I VTAM OPEN ACB FAILED ERROR=90 is received. This message indicates a problem was encountered while DDF was trying to open its ACB.

In this case, an error code of 90 means that VTAM could not find a resource in the VTAM definitions that matches the name in the ACB's APPLID field. This is probably because the application major node is not active. Often an open ACB error is because of the ACB not closing properly when the application was last deactivated. The console log should be checked around the time of the last deactivation for any sign of an error.

TCP/IP example

A Db2 data sharing group consists of 2 members, DB2A and DB2B. These messages appear on the z/OS console of member DB2B when DDF starts on DB2B:

```
DSNL512I -DSNB DSNLILNR TCP/IP LISTEN FAILED WITH  
          RETURN CODE=1115 AND REASON CODE=12E00291  
DSNL004I -DSNB DDF START COMPLETE  
          LOCATION STLEC2  
          LU        USIBMSY.SYEC2B  
          GENERICLU -NONE  
          DOMAIN   V8EC113.STL.IBM.COM  
          TCPPORT   446
```

All members of the data sharing group share port number for incoming requests. In this instance, DB2A is already using the port number (446), so DB2B gets error DSNL512I, which indicates that the LISTEN socket call failed. This means that DB2B cannot service incoming connection requests. Db2 retries the LISTEN call every 3 minutes. When DB2A stops, DB2B can use port 446, and Db2 issues message DSNL004I, which indicates that DDF started successfully on DB2B.

Related information

[z/OS Communications Server: SNA Messages](#)

[Return Codes \(z/OS Communications Server: SNA Programmer's LU 6.2 Guide\)](#)

Diagnostic tools for DDF and VTAM

Various tools are available to help diagnose problems in DDF.

VTAM traces

The VTAM buffer trace, the VTAM internal trace, and the z/OS I/O trace can be used to determine problems during distributed processing, including investigating the flows between logical units (LUs).

z/OS generalized trace facility (GTF)

The z/OS generalized trace facility (GTF) allows users to write records to a trace file. These traces can then be formatted into a more easily read form.

GTF can be started by issuing the following command:

```
S GTFDDF.GTF  
(S      is the z/OS Start command  
  GTFDDF is the name of the GTF procedure  
  GTF    is the name of this job. Use this name  
         when you want to stop GTF.)
```

GTF must be initialized with the USR option when you run the Db2 traces, the VTAM buffer, or the VTAM internal trace. The USR option allows VTAM or Db2 to write user (USR) to the GTF file. When you use the VTAM IO trace, RNIO should be included among the GTF options. The RNIO parameter includes all VTAM network activity in the trace. The output data set for the GTF trace (the default is called SYS1.TRACE) must be large enough to avoid wrapping the traces. If the GTF trace data is on DASD and fills up, it wraps around to the beginning and the initial trace data is lost. In particular, the VTAM internal trace can create many records.

After the traces finish running, the GTF job should be stopped. After the traces are completed, the GTF job should be stopped, by using the z/OS STOP command, which closes the data set holding the user records. The records can then be formatted.

Path information units (PIUs)

The path information unit (PIU) is the basic unit of information transfer throughout the network. The PIU contains the information necessary to reach its destination logical unit (LU) to maintain the protocols that the session is using, and to carry the data that the user transmits.

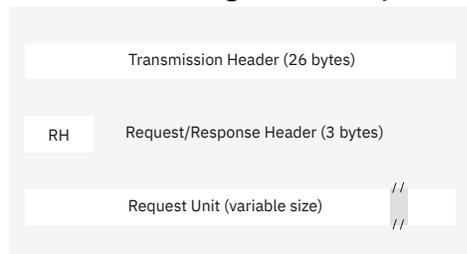


Figure 86. The format of a PIU

The following major parts of the PIU are shown in the previous figure:

- Transmission Header (TH)
- Request or Response Header (RH)
- Request Unit (RU).

The following figure displays an example of a BIND PIU.

```

LUDBD3 /LUDBD1      LRC(000,000)      INBOUND
TH=40000001 20008000 0000000F 0000000E 1D00006D 006A0018 0085 RH=6B8000
31001307 B0B050B3 00888585 88000602 00000000 *.....&;.heeh.....*
00000010 23000006 D3E4C4C2 C4F12400 0902E2D5 *.....LUDBD1....SN*
C1E2E5C3 D4C70903 00EF951D 79D77865 0E04C9D4 *ASVCMG....n..P....IM*
E2D5C5E3 4BD3E4C4 C2C4F100 06D3E4C4 C2C4F360 *SNET.LUDBD1..LUDBD3-*
15E3EF95 1D79D778 650CC9D4 E2D5C5E3 4BE5E3C1 *.T.n..P...IMSNET.VTA*
D4C50E0E F3C9D4E2 D5C5E34B D3E4C4C2 C4F12C0A *ME..3IMSNET.LUDBD1..*
01084040 40404040 4040                                     *...*
```

Figure 87. A BIND path information unit

Transmission header (TH)

The transmission header (TH) is used for routing information. It includes both the destination and origin logical unit (LU) addresses, and a length field.

In [Figure 87 on page 308](#), the origin subarea address is 4 bytes long at offset X'C', and the element address is 2 bytes long at offset X'14'. So, the origin address in our BIND PIU is subarea X'0000000E', element X'006A'.

The destination subarea address is 4 bytes long at offset X'8', and the element address is 2 bytes long at offset X'12'. The destination address in [Figure 87 on page 308](#) is subarea X'0000000F', element X'006D'.

At offset X'18' in the transmission header, there is a length field. This length does not include the length of the transmission header, but only the length of the request/response header (RH) and the request unit (RU). In [Figure 87 on page 308](#) the length is X'85'. In the example, there is a request/response header of 3 bytes; the request unit is X'82' bytes long. VTAM uses the length byte to ensure that the correct length of data is received.

Request/response header (RH)

The request/response header (RH) is used for maintaining the session's protocols. Each bit in the 3 bytes is an indicator.

The following bits are relevant while you use the distributed data facility (DDF):

BYTE 0 - X'6B'

- Request response indicator (RRI - byte 0 bit 0). If this bit is on (1), this PIU is a response PIU. If it is off (0), then the PIU is a request.

- Sense data included (SDI - byte 0 bit 5).

If this bit is on, then the first 4 bytes of the RU is SNA sense data. This sense data can often help to determine the cause of a problem.

BYTE 1 - X'80'

Pacing indicator (PI - byte 1 bit 7)

- If this bit is on and the PIU is a request, it is called a *pacing request*. A number of PIUs have been sent across the session and the origin LU is asking for confirmation that the destination LU is capable of taking more data. The LU that receives this PIU would normally (if there were no resource shortages) send back a pacing response.
- The pacing response (with both the RRI and PI bits on) is normally sent with no user data. That is, it is sent on its own. The pacing request, however, is usually sent on a PIU that contains data.

BYTE 2 - X'00'

- Begin bracket indicator (BBI - byte 2 bit 0)
- End bracket indicator (EBI - byte 2 bit 1)
- Conditional End Bracket Indicator (EBI - byte 2 bit 7). These 3 bits control the use of bracketing on the session. Brackets are used around each distributed data facility (DDF) conversation.
- Change Direction Indicator (CDI - byte 2 bit 2). This bit is used in the control of the half duplex session. On a half duplex session, only one LU can be sending at a time. The other LU, at this stage, must be receiving. The CDI allows the receiving LU to send data after the receive is complete.

Related reference

[Request/Response Headers \(RHs\) \(Systems Network Architecture Formats\)](#)

Request unit (RU)

The Request Unit is the information that is being sent between the two network addressable units (NAU) on the session.

Depending on the session the RU can take different forms.

SSCP sessions:

- The first bytes of the RU indicate a request or response code. Depending on the code, the rest of the RU takes a format specific for that code. For example, a CDINIT (Cross Domain INITiate) has a code of X'818641' while a SESSST (session started) RU has X'810686' as the first 3 bytes of the RU.

LU to LU sessions:

- The first byte might contain a code like the code in the SSCP sessions. The BIND has a code of X'31' and has a standard format.

The first bytes of the RU might be a Function Management Header (FMH). The distributed data facility (DDF) uses FMH5 to start conversations and for CNOS. It also uses FMH7 for errors. Usually, the FMH is followed by other data that is being sent.

The request unit (RU) can be data only, in a format (or it might be unformatted) that is recognized only by the applications on the session.

```

USRFD FEF ASCB 00F9D980          JOBN DBD2DIST
      BUFF LUDBD1 /LUDBD2      LRC(000,000)      OUTBOUND
      VTAM  TH=40000000 00000000 0000000E 0000000F 1C00006B 006E0007 0350 RH=0B90A0
                                270502FF 0003D000 400430F0 F3F20000 00000000 *.....032.....*
                                00000000 00000000 00000000 00000000 00000003 *.....*
                                26EFFE41 550010D4 C2C84000 00032641 080060D4 *.....MBH.....-M*
                                C9C24000 00008B00 00000000 00000000 010002A0 *IB.....*
                                EFCF1437 699C01A0 EFCF1437 699C0100 00000000 *.....*
                                00000000 00000000 00000000 00000000 00000000 *.....*
                                00000000 00000000 00000000 00000000 00000000 *.....*
                                00000000 00000000 000000C9 D4E2D5C5 E34040D3 *.....IMSNET L*
                                E4C4C2C4 F24040A0 EFCF1437 91008000 0000E2E3 *UDBD2 .....j.....ST*
                                E4E3E3C7 C1D9E36D D7D9D6C4 404041      *UTTGART_PROD . *
TIME      48910.013156

```

Figure 88. Outbound VTAM buffer entry

Related concepts

Path information units (PIUs)

The path information unit (PIU) is the basic unit of information transfer throughout the network. The PIU contains the information necessary to reach its destination logical unit (LU) to maintain the protocols that the session is using, and to carry the data that the user transmits.

Related reference

[Function Management \(FM\) Headers \(Systems Network Architecture Formats\)](#)

VTAM buffer traces

The VTAM buffer trace shows the contents of the message buffers as messages flow inbound and outbound from VTAM.

To start a VTAM buffer trace, the user must first start the z/OS generalized trace facility (GTF) with option=USR. Then, an z/OS modify command is used to start and stop the trace.

To start the trace:

```
F VTAMDDF,TRACE,TYPE=BUF,ID=LUDBD2
```

To stop the trace:

```
F VTAMDDF,NOTRACE,TYPE=BUF,ID=LUDBD2
```

```
(F          is the z/OS modify command
VTAMDDF    is the name of the VTAM startup procedure
TRACE      indicates that the trace is being turned
           on
NOTRACE    indicates that the trace is being turned
```

```
           off
TYPE=BUF   indicates that this command affects the
           buffer trace
ID=LUDBD2 indicates which node is to be traced,
           or is to no longer be trace.)
```

The 'ID=' parameter on the command causes the messages to and from this resource to be traced. When you trace session setup attempts, selection of the resource to be traced is important.

After the trace is started, re-create the problem situation that needs diagnosing. After you trace the problem, stop the trace to avoid writing unnecessary records to the GTF data set.

Now, the programs available to format the VTAM buffer trace are ACFTAP and IPCS.

The VTAM buffer trace is primarily used to determine the cause of session and network-related problems. The trace contains the transmission header (TH), the request/response header (RH), and up to 256 bytes of request unit (RU) data. Because the distributed data facility uses a RUSIZE of 4 KB, the buffer trace is of limited use when trying to capture the data that is sent from the session. The data can be traced by using the Db2 trace that specifies IFCIDs 160 and 161.


```

USRFD FF0 ASCB 00F9DB00          JOBN DBD2DIST
      VTAM BUFFERS              MAXU    MAXQ    AVNO    TEXP    MBUF    TOTL
      IO      00000008  00000000  000000C8  00000000  000000C8  000000C8
      PP      00000000  00000000  00000000  00000000  00000000  00000000
      LP      00000008  00000000  0000003B  00000000  00000040  00000040
      WP      0000000A  00000000  00000044  00000000  0000004E  0000004E
      NP      00000000  00000000  00000000  00000000  00000000  00000000
      LF      00000002  00000000  00000027  00000000  00000029  00000029
TIME    41772.136371
USRFD FF0 ASCB 00F9DB00          JOBN DBD2DIST
      VTAM BUFFERS              MAXU    MAXQ    AVNO    TEXP    MBUF    TOTL
      CR      00000008  00000000  000000C1  00000000  000000C8  000000C8
      UE      00000000  00000000  00000000  00000000  00000000  00000000
      SF      00000004  00000000  00000060  00000000  00000064  00000064
      SP      00000000  00000000  00000002  00000000  00000002  00000002
      AP      00000000  00000000  00000010  00000000  00000010  00000010
TIME    41772.136411

```

Figure 93. SMS trace printed by IPCS

VTAM internal trace

Under normal circumstances, the VTAM internal trace is run at the request of IBM service personnel. A great deal of information is available from these VTAM internal trace entries.

The VTAM internal trace (VIT) can be run internally to VTAM or externally to GTF. When VTAM is started the default is to have the trace running internal to two 4 KB pages with no options. When the trace is running internally, the only method of getting the trace records is to dump VTAM. The trace can then be viewed in the unformatted dump, or it can be formatted by using the VTAMMAP options in IPCS. Beside not having to dump VTAM, an additional advantage to running the VTAM internal trace to GTF is that GTF timestamps each entry. A VTAM internal trace run internally has no timestamps.

GTF must be started before the VTAM internal trace is started. Use the following commands to stop the VTAM internal trace running internally and to start it running to GTF.

```

F VTAMDDF,NOTRACE,TYPE=VTAM,OPTION=END
(F is the z/OS modify command
VTAMDDF is the name of the VTAM startup procedure
NOTRACE indicates that the trace is being turned off
TYPE=VTAM indicates that this command affects the VTAM
internal trace
OPTION=END stops the VIT and frees the pages in storage.
F VTAMDDF,TRACE,TYPE=VTAM,MODE=EXT
TRACE indicates that the trace is being turned on
MODE=EXT causes the VIT records to be sent to GTF.)

```

After the trace is running externally, the trace can be started with the options that you require:

```

F VTAMDDF,TRACE,TYPE=VTAM,OPTION=(APPC,PIU,API,MSG)
OPTION=xxx indicates which record types are to be traced.

```

After the event is traced, you can stop the VTAM internal trace with the following command:

```

F VTAMDDF,NOTRACE,TYPE=VTAM,OPTION=ALL
(OPTION=ALL requests the VIT remain active, but trace
nothing
OPTION=END stops the VIT.)

```

Then, stop GTF so that the trace data can be formatted.

Related reference

[Failing module \(z/OS Communications Server: SNA Diagnosis\)](#)

VTAM internal trace options

Trace options should be tailored to the requirements of the problem so that unnecessary record types are not included in the trace.

The following options are of most interest to distributed data facility (DDF) diagnosis:

APPC

The Advanced Program to Program Communication option writes entries each time an application issues an APPC command by using the APPCCMD macro, and each time one of these commands is posted back as complete.

API

The API option traces the non-APPC application interface macros that are being issued and posted back. There are also trace entries that are written when the non-LU 6.2 exits (for example, the LOGON and SCIP exits) are driven.

PIU

The PIU option gives details of all the PIUs that flow through this VTAM.

MSG

Each time a message is issued by VTAM or an operator issues a VTAM command, an entry is made in the VIT. This entry can be useful when you become oriented in a large VIT, and should be included when you run traces. Db2 messages do not appear in the VIT.

The programs available to format the VTAM internal trace are the same as for the VTAM buffer trace. They are IPCS (depending on the level of z/OS) and ACFTAP.

Storage Management Services (buffer use) trace

The Storage Management Services trace provides information about the use and availability of VTAM buffer pools.

The trace entries are written after a specified number of requests for VTAM buffers. The IBM default causes an entry after every 1000 (x'3E8') buffer requests. This threshold is specified by the halfword at label RACBSNAP in VTAM module ISTRACON. This can be changed by using z/OS's SPZAP (SUPERZAP) facility.

This trace facility is very similar to the display buffer use command in VTAM. It should not be confused with the SMS option in the VTAM internal trace.

GTF should be started with option=USR before you start the SMS trace with the following command:

```
F VTAMDDF,TRACE,TYPE=SMS,ID=VTAMBUF
(F is the z/OS modify command
VTAMDDF is the name of the VTAM startup procedure
TRACE indicates that the trace is being turned on
TYPE=SMS indicates the storage management services
trace
ID=VTAMBUF required for the SMS trace.)
```

The formatters can again be used to print the SMS trace. ACFTAP, however, gives only a hex dump of the trace record. IPCS formats the SMS trace records into a more readable form. [Figure 93 on page 314](#) shows a record that is formatted by using IPCS.

There is an entry for each of the VTAM buffer types:

- AP APBUF, Application Program Pageable pool
- CR CRPLBUF, Copied RPL pool
- IO IOBUF, IO buffer pool
- LF LFBUF, Large Fixed pool
- LP LPBUF, Large Pageable pool
- SF SFBUF, Small Fixed pool
- SP SPBUF, Small Pageable pool
- WP WPBUF, Working Set Pageable pool.

PPBUFs, NPBUFs, and UEBUFs are no longer used by VTAM, though they still appear in the output listings.

The following columns are in the output:

- MAXU. Records the maximum number of buffers in use in the pool at any one time.
- MAXQ. Records the maximum number of requests for buffers that were queued waiting for storage at any one time.
- AVNO. Number of buffers available when the trace entry was written.
- TEXP. Number of times the buffer pool expanded.
- MBUF. Records the maximum number of buffers, used and unused, that were in the pool. This includes the static and expansion parts of the pool.
- TOTL. Records the total number of buffers in the pool at the time the trace entry was written.

Each time a trace entry is written these counters are reset. So, MAXU, MAXQ, TEXP, and MBUF are counters relevant since the last trace entry was written.

Similarly, for a display of buffer use by the D NET, BFRUSE command, the values in the resulting display are valid since the last trace entry was written; or, since the last VTAM startup, if the SMS trace is not running.

Exception condition diagnostic procedures

Several exception condition diagnostic procedures are available.

DRDA exception condition diagnostic procedures

Db2 defines a DRDA exception condition as an event that represents one of several conditions.

1. A reply message that is received from the AS, defined by Distributed Data Management (DDM) Level 6 as valid for the DDM command, but that is other than the normal reply message or reply data object defined for that command.
2. A reply message or reply data object returned from the AS in response to a DDM command that is not valid for the DDM command, or is structurally incorrect. In the latter case, a parsing error was detected by Db2.
3. A reply message or reply data object that is both valid and structurally correct but whose contents are inconsistent with the semantic of the reply message or reply data object.
4. A command or command data object that is not valid, as defined by DDM Level 6, or is structurally incorrect. In the latter case, a parsing error was detected by Db2.

In terms of the DDM process model, a DRDA exception condition might be detected in either the SQLAM, AGENT, or CMNMGR layers. The Db2 implementations of these managers are contained within the data communications resource manager (DCRM), the distributed transaction manager (DTM), distributed relational database systems (DRDS), and distributed data interchange services (DDIS).

DRDA exception event notification

When DDF resource managers identify DRDA exception conditions, event notification is sent to several destinations.

1. The application if detected by the requester
2. The requester through a DDM reply message if detected by the server
3. One or more trace records written to the statistics class 4 trace.

While event notification of all DRDA exception conditions is targeted to various destinations, the amount of diagnostic information sent to some destinations is limited. This limitation is due primarily to the inability of Db2 to influence either the structure or content of information. For example, the SQLCA structure and contents are determined mainly by the SQL language. The one destination which Db2 alone can define is the structure and content of the statistics class 4 trace records. Db2 designed these trace records to provide, for each DRDA exception condition, a rich amount of relevant information. This information is contributed by one or more DDF resource managers, which are sufficient to determine the exact nature of the DRDA exception condition.

DRDA exception condition trace records

Several statistic class 4 trace records are available for diagnosis of DRDA exception conditions.

IFCID 0191

This record is written by DDIS when it identifies a DRDA exception condition, or on behalf of either DTM or DRDS when they identify a DRDA exception condition.

IFCID 0192

This record is written by DCRM when it identifies a DRDA exception condition.

IFCID 0193

This record is written by DTM when it identifies a DRDA exception condition.

IFCID 0194

This record is written by DCRM when it identifies a DRDA exception condition.

IFCID 0195

This record is written by DRDS when it identifies a DRDA exception condition.

The first 8 characters of each DDF diagnostic trace record contain an eyecatcher with the following format.

| IFCID | Hexadecimal | Printable |
|-------|---------------------|-----------|
| 0191 | X'C4D9C4C1F0F1F9F1' | DRDA0191 |
| 0192 | X'C4D9C4C1F0F1F9F2' | DRDA0192 |
| 0193 | X'C4D9C4C1F0F1F9F3' | DRDA0193 |
| 0194 | X'C4D9C4C1F0F1F9F4' | DRDA0194 |
| 0195 | X'C4D9C4C1F0F1F9F5' | DRDA0195 |

Db2 DRDA exception condition reason codes

In most cases, Db2 provides a 32-bit reason code as the common token sent to all destinations to be notified of the DRDA exception condition.

PSPI

Db2 requester role

If the DRDA exception condition is identified at the Db2 AR, then the reason code is placed within the SQLERRD1 field within the SQLCA. Additionally, the reason code is externalized within the detailed data, subfield X'82', subvector X'96' (WHY) of the Tivoli® NetView for z/OS alert and the IFCID 0191, 0192, 0193, 0194 and 0195 trace records. This reason code is directly traceable to an SQL statement that is issued by an application. It is the common token that identifies the alert and all relevant trace records produced. The reason code is the starting point for the DRDA exception condition diagnosis.

Db2 database server role

If the DRDA exception condition is identified at the Db2, then the reason code is placed within the server diagnostic information scalar, included within the DDM reply message that is returned to the requester. The reason code is externalized within the detailed data, subfield X'82', subvector X'96' (WHY) of the Tivoli NetView for z/OS alert and the IFCID 0191 trace records. This reason code is directly traceable to a DDM command that is received from the AR. It is the common token that identifies the alert and all relevant trace records produced. The reason code is the starting point for the DRDA exception condition diagnosis.

PSPI

Additional diagnostic options for DRDA exceptions

Under direction of IBM Service, a dump can be forced when some DRDA exception conditions are encountered instead of generating a trace record. This dump provides more diagnostic information not present in the trace alone.

To force the dump, the IFCID 299 trace must be activated. In general, the dump reason code is the DRDA exception reason code, but might be different in some cases. The reason code is identified by IBM Service when you are instructed to take this course of action.

DRDA summary

Before you use the Statistics Class 4 trace records, be familiar with the distributed relational database architecture.

DDM/FDOCA models

The DDM object model and the basic concepts of formatted data object content architecture (FDOCA) geometry and their application to the description of DRDA data are reviewed here.

DDM commands and reply messages

The DDM commands that are defined for Level 6 of the architecture include all commands unique to DRDA.

For each DDM command, which can be sent from an AR to an AT, DDM defines the collection of reply messages, which the AT can send in response to the command. Db2 supports this semantic for all valid reply messages that are received in response to all DDM commands sent by Db2 to an AT.

DDM command data and reply data

For each DDM command that can be sent from an AR to an AS, DDM defines the collection of command data objects that might accompany the command. Additionally, DDM defines the valid DDM objects that the AS might send in response to that command.

Except for EXCSATRD returned in response to EXCSAT, all command and reply data are either FDOCA descriptors or FDOCA data described by some FDOCA descriptor.

DRDA object descriptors

FDOCA models data or collections of data, based on relating finite and discrete geometrical spaces of arbitrary dimension.

Regarding describing DRDA objects, the application of the FDOCA geometry is restricted to simple data array (SDA), group data array (GDA), and row layout (RLO) triplets. Each SDA, GDA, and RLO are assigned, through the metadata definition triplet (MDD), a unique DRDA type. In the case of simple data arrays, the DRDA type is always a data type supported by DRDA. The DRDA data types might be mapped directly to SQL data types. Each group is assigned a DRDA type and describes an ordered collection of other groups or simple data arrays (possibly including length overrides). A row is assigned a DRDA type and describes an ordered set of elements, each of which is selected from one or more groups. An array is assigned a DRDA type and describes a finite number of rows.

Local identifiers (LIDs)

All FDOCA Descriptors are identified by a local identifier (LID). It is local because its definition is restricted to the current data stream that contains the descriptor.

The LID is merely a name by which the descriptor might be referenced. Thus, array descriptors reference row descriptors (RLOs) by LID, row descriptors reference groups (GDAs), and simple data arrays (SDAs) by LID, and groups reference other groups (GDAs) and simple data arrays (SDAs) by LID. The LID is distinct from the DRDA type.

Early DRDA descriptors

During application requester (AR) or server (AS) connection processing, a subset of the DRDA object descriptions is fixed and cannot be changed during that connection. These descriptors are called the

DRDA early descriptors and consist of simple data arrays (SDAs), group data arrays (GDAs), rows, and arrays (RLOs).

The simple data arrays describe each data type supported by DRDA and are called the early environmental descriptors. Additionally, the early descriptors include few groups, rows, and arrays (GDAs and RLOs). After committed, the AR/ must send only the DRDA object to the /AR; the descriptor is not sent.

An AR or AS commits support to the early descriptors at two distinct points during connection processing.

1. The early DRDA group, row, and array descriptors are established during EXCSAT/EXCSATRD processing.
2. The early environmental descriptors are established during ACCRDB/ACCRDBRM processing. These represent the 84 supported DRDA data types and are fixed and identical across these environments: QTDSQL370, QTDSQLX86, QTDSQL400, QTDSQLC, and QTDSQLVAX. Accepting one of these environments commits the AR or AS to support all 84 DRDA data types in a specific machine representation. While the DRDA data types cannot change, data type representations might be changed at various points in the processing of commands and replies.

Early descriptor LIDs

The LID, which is assigned to each early descriptor (SDA, Group, Row, Array), is identical to its DRDA type. This assignment is fixed and cannot be changed.

There are 84 DRDA data types and naming a particular environment, QTDSQL370, QTDSQLX86, QTDSQL400, QTDSQLC, or QTDSQLVAX, therefore identifies 84 MDDs and corresponding SDAs. The MDD contains the DRDA data type, and the SDA contains the LID by which each early environmental descriptor might be referenced. For each Environmental Descriptor, the DRDA type is the DRDA data type and therefore the LID is identical to the DRDA data type. In order to describe data of a specific DRDA data type, the AR/ must identify only the 1-byte LID (DRDA data type) and this resolves to a complete early environmental descriptor that consists of an MDD and SDA.

DRDA describes the following DDM reply data objects through early descriptors:

- The SQLCARD is described as an early FDOCA row.
- The SQLDARD is described as an early FDOCA array.

DRDA describes the following DDM command data objects through early descriptors:

- The SQLSTT is described as an early FDOCA row.
- The SQLNUM is described as an early FDOCA row.
- The SQLOBKNAM is described as an early FDOCA row.
- The SQLSTTVRB is described as an early FDOCA array.

Late DRDA descriptors

There is a class of DRDA objects whose description is not known at connection time, but can be only known at SQL statement run time.

These objects include descriptions of input host variables that are passed by the application in support of OPNQRY and EXCSQLSTT, and descriptions of the answer set returned in response to an OPNQRY or SQL static SELECT (EXCSQLSTT). In these cases, the number of host variables or columns, their SQL data types and lengths are only known when the application runs the statement. The description of the data is assembled dynamically and sent to the AR/ along with (but preceding) the data. They are called late descriptors.

The DRDA data types are fixed at ACCRDB/ACCRDBRM processing. The SQL types of all input host variables and constituent columns of an answer set must be mappable to one of these DRDA data types. If this mapping cannot be done, then the AR/ supports some data type, which is not defined within DRDA. It is in violation of DRDA. Otherwise, a DRDA data type can be identified.

Db2 representation of early and late descriptors

It is important to understand how Db2 represents early and late descriptors.

The DDIS FDLIDLST structure

Every FDOCA array consists of one or more FDOCA rows and all FDOCA rows represent elements of an FDOCA group.

Command and reply data are either described as FDOCA rows or arrays. Db2 creates data structures to support data retrieval of DDM command/reply data objects that are described by an Early or Late Row Descriptor. This data structure is a list of FDOCA Local Identifiers (LIDs) and is called an FDLIDLST. A collection of FDLIDLST structures is created for every FDOCA row to be retrieved from command or reply data. Thus, the Db2 unit of data transfer is an FDOCA row.

An FDOCA row is an ordered collection of elements of an FDOCA group. FDOCA groups are recursive. The elements are other FDOCA groups or Simple Data Arrays (SDAs). The DRDA objects that are described by FDOCA are usually defined in terms of nullable groups, that is all elements of the group are present in the object if the group null indicator value is 0, or none of the elements of the group are present in the object if the group null indicator value is positive (empty set).

An FDOCA group might be modeled as a tree such that the root represents the group occurrence and nonterminal nodes represent other FDOCA groups. Terminal nodes consist of Simple Data Arrays (SDAs), and always represent a datum of a unique DRDA data type. Given an FDOCA row, which orders the elements of an FDOCA group, G, there is a (parent) FDLIDLST representing G (root node) and an FDLIDLST for each non-terminal group that is contained within G. Terminal nodes up consist of SDAs and always represent a datum of a unique DRDA data type. The hierarchical (tree) structure of the FDOCA group is represented by one or more FDLIDLSTs. Each FDLIDLST retrieves the subset of the data that is described by the parent FDLIDLST.

The FDLIDLST header defines the total number of elements (GDAs or SDAs) comprising the group. This is followed by one LIDENT entry for each of these elements. The LID is stored in LIDENTLI and the DRDA type (SDA or GDA) is stored within LIDENTTY. The entry defines an SDA (terminal node) if the DRDA type (LIDENTTY) is less than X'50'. In this case, the value of LIDENTO1 is the offset relative to the Level 6b data stream (QW01916B) of the source data. If the value of LIDENTTY is greater than X'50', the entry represents a non-terminal node (another group) and the value of LIDENTO1 is the offset relative to the 0191 record of the child FDLIDLST, which describes that group.

FDLIDLST structures for early descriptors

Several FDLIDLST structures retrieve objects that are described by Late Descriptors.

SQLCARD

The SQLCARD is described as a row, consisting of all elements of the nullable group SQLCAGRP. The SQLCAGRP contains three nonterminal SDAs, SQLSTATE, SQLCODE, and SQLERRPROC and the nullable group SQLCAXGRP, which itself contains 20 SDAs. The FDLIDLST to retrieve an SQLCARD:

1. Parent FDLIDLST containing one entry with DRDA type (LIDENTTY) of X'54'. This is the parent and the value of LIDENTO1 is the offset to the child FDLIDLST.
2. The child FDLIDLST contains four entries, one each for SQLCODE, SQLSTATE, SQLERRPROC, and SQLCAXGRP. The value of LIDENTTY for SQLCODE, SQLSTATE, and SQLERRPROC are DRDA data types (less than X'50'). The data must be retrieved exactly as defined by the LIDENT lengths and representation. The value of LIDENTTY for the fourth entry is X'52' and this is the DRDA type of SQLCAXGRP, an FDOCA group. Thus, the LIDENTO1 points to a third FDLIDLST structure.
3. The SQLCAXGRP FDLIDLST structure contains 20 entries all of which are terminal nodes (SDAs). These entries describe the representations and lengths of the data, which is retrieved from the data stream.

SQLDARD

The content and structure of the SQLDARD depends on the DDM Levels of the AR and AT. If both AR and AT are both at DDM Level 6 and higher:

The SQLDARD is described as an open array, such that row 1 is an SQLCARD, row 2 is an SQLNUMROW, and rows 3 through (N+2) are SQLDAROW instances. The SQLDARD thus couples an SQLCA with an SQLDA. The number of SQLDAROW instances, N, is given by row 2, SQLNUMROW. This row contains the only element from the nonnullable SQLNUMGRP, an SDA of type Integer 2. The SQLDAROW consists of all elements of the nonnullable SQLDAGRP group. Because data retrieval is at the row level, the FDLIDLSTs required to retrieve the SQLDARD consist of one FDLIDLST to process the SQLCARD, one FDLIDLST to retrieve the SQLNUMROW and one FDLIDLST to retrieve each of the N SQLDAROW instances.

1. The FDLIDLST to retrieve an SQLCARD has been described previously as a parent FDLIDLST, child FDLIDLST containing three terminal SDAs, and one subordinate FDLIDLST describing the 20 terminal SDAs comprising the SQLCAXGRP
2. The FDLIDLST to retrieve an SQLNUMROW contains a single entry, defining the terminal SDA of type integer 2.
3. The FDLIDLST to retrieve an SQLDAROW contains 11 entries, each defining terminal SDAs, and one subordinate FDLIDLST describing the two terminal SDAs comprising the SQLUDTGRP. NOTE: the definitions of the terminal SDAs are dependent on the DDM levels of the AR and the AT. Check the DRDA reference for the contents at DDM Level 6.

The following information is true if either or both of the AR or AT has a DDM Level less than 6:

The SQLDARD is described as an open array, such that row 1 is an SQLCARD, row 2 is an SQLNUMROW, and rows 3 through (N+2) are SQLDAROW instances. The SQLDARD thus couples an SQLCA with an SQLDA. The number of SQLDAROW instances, N, is given by row 2, SQLNUMROW. This row contains the only element from the nonnullable SQLNUMGRP, an SDA of type Integer 2. The SQLDAROW consists of all elements of the nonnullable SQLDAGRP group. Because data retrieval is at the row level, the FDLIDLSTs required to retrieve the SQLDARD consist of one FDLIDLST to process the SQLCARD, one FDLIDLST to retrieve the SQLNUMROW and one FDLIDLST to retrieve each of the N SQLDAROW instances.

1. The FDLIDLST to retrieve an SQLCARD is described as a parent FDLIDLST, child FDLIDLST containing three terminal SDAs, and one subordinate FDLIDLST describing the 20 terminal SDAs comprising the SQLCAXGRP.
2. The FDLIDLST to retrieve an SQLNUMROW contains a single entry, defining the terminal SDA of type integer 2.
3. The FDLIDLST to retrieve an SQLDAROW contains 11 entries, each defining terminal SDAs.

NOTE: the definitions of the terminal SDAs are dependent on the DDM levels of the AR and the AT. Check the DRDA reference for the contents at DDM Level less than 6.

SQLSTT

The SQLSTT row consists of all occurrences of the SQLSTTGRP. The SQLSTTGRP is a nonnullable group that contains two SDAs: a variable SBCS character string and a variable Mixed character string. The FDLIDLST contains two entries, one each for the two SDAs and retrieval of the SQLSTT row must always retrieve both an SBCS variable character string and a Mixed variable character string. Only one of these strings can have a positive length.

SQLOBKNAM

The SQLOBKNAM row consists of all occurrences of the SQLOBJGRP. The SQLOBJGRP is a nonnullable group that contains two SDAs: a variable SBCS character string and a variable Mixed character string. The FDLIDLST contains two entries, one each for the two SDAs and retrieval of the SQLSTT row must always retrieve both an SBCS variable character string and a Mixed variable character string. Only one of these two strings can have a positive length.

SQLSTTVRB

The SQLSTTVRB is described as an open array, such that row 1 is an SQLNUMROW row and rows 2 through (N+1) are SQLVRBROW instances. The SQLNUMROW consists of the single element of

the nonnullable SQLNUMGRP. The SQLVRBROW consists of the nine occurrences of the nonnullable SQLVRBGRP group.

1. Because SQLNUMGRP is not nullable, the FDLIDLST structure to retrieve the SQLNUMROW contains a single LIDENT, defining a 2-byte integer, the contents of which is the number of SQLVRBROW instances.
2. Since SQLVRBGRP is not nullable, the FDLIDLST structure to retrieve an SQLVRBROW instance contains nine terminal SDAs.

NOTE: the definitions of the terminal SDAs are dependent on the DDM levels of the AR and the AT. Check the DRDA reference for the contents at DDM Level less than 6.

FDLIDLST structures for late descriptors

Several FDLIDLST structures retrieve objects that are described by Early Descriptors.

SQLDTARD

The SQLDTARD array is an open array of SQLCADTA rows. Each SQLCADTA row consists of all occurrences of the SQLCAGRP (DRDA type is X'54'), followed by all occurrences of LIDs defined in the nullable SQLDTAGRP. If there are N elements (triplets) contained within the SQLDTAGRP, then the FDLIDLST consists of the following items:

1. Parent SQLCAGRP FDLIDLST containing a single entry.
2. Child FDLIDLST containing three terminal SDAs and a non-terminal GDA, pointing to a subordinate SQLCAXGRP FDLIDLST.
3. SQLCAXGRP subordinate FDLIDLST containing 20 terminal SDAs to retrieve the SQLCAXGRP.
4. Parent FDLIDLST containing a single entry (DRDA type is X'D0'), pointing to a child SQLDTAGRP FDLIDLST.
5. Child FDLIDLST containing N entries each representing a terminal SDA.

The FDLIDLST structure follows.

```

FDLIDLST DSECT
FDLIDCNT DS    F          Total # LIDs this Group
*          Each LID is described by one
*          LIDENT entry. The LIDENT entries
*          are contiguous to FDLIDLST.
FDLIDLST DS    F          Last LIDLSTE processed
FDLIDF   DS    X          Flag byte
*          '80'X Optimization Flag
          DS    XL3       Reserved
FDLIDMX  DS    F          Max length if FDLIDF='80'X
          DS    XL8       Reserved
FDLIDHLL EQU   *-FDLIDLST Sizeof(FDLIDLST header)

*****
*   Define Mapping for LID element within FDLIDLST.          *
*   There is one LIDENT for each of the total number of LIDs *
*   specified by FDLIDCNT. The LIDENTs follow FDLIDLST.    *
*****
LIDENT   DSECT          FDOCA LID entry Mapping
LIDENTLI DS    X          FDOCA LID to be processed
*          this is identical to DRDA type
*          unless described by a Late
*          Environmental Descriptor
LIDENTTY DS    X          DRDA Type (GDA or SDA)
LIDENTFD DS    X          FDOCA Data Type Representation
LIDENTNU DS    X          Null Byte if nullable
LIDENTLT DS    F          Length of data in bytes
*          upper bound for variable
LIDENTO1 DS    F          If LIDENTTY >= X'50' then offset
*          relative to 0191 record to child
*          FDLIDLST. If LIDENTTY < X'50'
*          then offset relative to objdss
LIDENTO2 DS    F          Offset if partial object
LIDENTVA DS    X          High Order Byte if Variable
*          data and LIDENT state is SV2
LIDENTST DS    X          FD_LIDENT State

```

```

*          values are defined below
LIDENTVR DS   X          0 - data is not variable
*          1 - data is variable with
*          halfword prefix
*          2 - data is variable null
*          terminated
*          3 - data is variable with
*          one-byte prefix
LIDENT37 DS   X          0 - data is numeric,
*          representation is S/370
*          1 - data is numeric,
*          representation is 80X86
*          2 - data is numeric,
*          representation is /400
LIDENTAL DS   F          Actual size of object (may differ
*          from LIDENTLT is data is variable
LIDENTSD DS   F          0
LIDENTRT DS   A          Data routine or 0
LIDENTDL DS   F          Max length of data or 0
LIDENTF  DS   X          Flag
*          '80'X Data is placeholder
*          DS   XL3      Reserved
LIDENTLL EQU  *-LIDENT   Sizeof(LIDENT)

```

```

*****
* Define LIDENT states:
* State 0: Initial/Final State
* NOTE:
* States SV1, SV2, SV3, SN1 and SN2 are only valid
* if the data described by the LIDLST may span multiple
* DDM objects. This is only true if the LIDLST describes
* row data which is being processed via Limited Block
* protocols. In this case, row data may span multiple
* query blocks, each of size qryblksz, each containing
* one qrydta object.
*
* For fixed data, the size of the datum is determined from
* the size specified by the DRDA SDA specification, or
* if a Late Descriptor, from a length override. Once spec-
* ified, the length cannot change. For variable data, the
* size specified from the DRDA SDA specification or length
* override is an upper bound (a maximum). The actual length
* of the datum is transmitted as a halfword preceding the
* datum. Thus, it is possible that all or part of the half-
* word prefix is transferred in block n and the remainder
* is transferred in other blocks.
* The following LIDENT states support partitioning of
* the null indicator (if nullable), halfword length pre-
* fix and data.
*
* State SV1: Variable Data - 0 bytes of halfword ll
* State SV2: Variable Data - 1 byte of halfword ll
* State SV3: Variable Data - 2 bytes of halfword ll - no
* data retrieved
* State SN1: Nullable Data - Null Indicator not available
* State SN2: Nullable Data - Null Indicator available - no
* data retrieved
*****
*
LIDST0 EQU X'00'
LIDSTV1 EQU X'01'
LIDSTV2 EQU X'02'
LIDSTV3 EQU X'03'
LIDSTV4 EQU X'04'
LIDSTN1 EQU X'10'
LIDSTN2 EQU X'20'

```

Figure 94. FDLIDLST structure

The DDIS RDTA structure

There exists one parent FDLIDLST structure and 1 or more child FDLIDLSTs for every FDOCA row to be retrieved from the data stream. This is true for early and late FDOCA rows. However, a second data structure, the DDIS RDTA structure, is created to support Late Descriptors.

There are two DRDA late descriptors defined: the SQLDTA describes the host variables that are sent in support of EXCSQLSTT or OPNQRY and the SQLDTARD describes the answer set returned from OPNQRY

or EXCSQLSTT. Both descriptors have a simple geometry that is built around the late group descriptor, SQLDTAGRP.

The SQLDTA Late Descriptor is built from an SQLDTAGRP and describes input host variables. It is mapped to an input SQLDA. Similarly, the SQLDTARD, also built from the SQLDTAGRP, describes an output answer that is set and is mapped to an output SQLDA. The RDTA structure correlates the SQLDTAGRP elements (SDAs) in terms of SQL data types, lengths, and character representation (CCSID).

The RDTA structure is given the following figure. RDTALID contains the address of the parent SQLDTAGRP FDLIDLST structure that is required to retrieve data that is described by the late SQLDTA/SQLDTARD. For every triplet contained within the SQLDTAGRP, an RDTAENT element exists which defines the SQL attributes for that DRDA data type.

| | | | |
|----------|-------|-----------|---|
| RDTA | DSECT | | Relational Data Block |
| RDTAID | DS | H | RDTA ID |
| RDTALEN | DS | H | RDTA Length |
| RDTAEYE | DS | CL4 | RDTA Eye Catcher |
| RDTDA | DS | A | A(DB2 Sqlda) |
| RDTALID | DS | F | offset within 0191 record of QW0191LT section |
| * | | | of QW0191LT section |
| RDTAROW | DS | A | A(Input Host Variables) |
| RDTALED | DS | 0XL4 | |
| RDTAGLID | DS | X | SQLDTAGRP RLO LID and |
| * | DS | XL2 | Reserved |
| RDTANUM | DS | F | Total Number RDTAENT slots |
| RDTANPH | DS | F | Number of placeholders |
| RDTAFLG | DS | X | Flag |
| | | | '80'X RDTARAWP is used |
| | DS | XL3 | Reserved |
| RDTALL | EQU | *-RDTA | Sizeof(RDTA Header) |
| * | | | |
| | | | |
| RDTAENT | DSECT | | RDTA Entry Mapping - one entry for |
| * | | | each SDA processed in sqldtagrp |
| RDTTYP | DS | H | SQLTYPE |
| RDTUBT | DS | H | SQLSUBTYPE |
| RDTACCSI | DS | XL2 | Derived CCSID |
| RDTALGTH | DS | 0H | SQL Length |
| RDTALEN1 | DS | X | Decimal Precision |
| RDTALEN2 | DS | X | Decimal Scale |
| RDTAFLG1 | DS | X | Flag |
| | | | '80'X Placeholder only |
| | | | '40'X Mapped type |
| | DS | X | Reserved |
| RDTAOTYP | DS | H | Original SQLTYPE, if mapped |
| RDTARAWP | DS | A | Raw data pointer, if mapped |
| | DS | XL4 | Reserved |
| RDTALL | EQU | *-RDTAENT | Sizeof(RDTAENT) |

Figure 95. RDTA structure

The DDIS ZEDA structure

The SQLDTA describes the input host variables that are sent in support of EXCSQLSTT or OPNQRQ command. It has a simple geometry that is built around the late group descriptor, SQLDTAGRP, and it is mapped to an input SQLDA.

The ZEDA structure is used internally to support the late descriptors that are retrieved from an SQLDTA object, which represents the input host variables that are received at the application server. This structure is traced for Db2 serviceability purpose only.

DDIS IFCID 0191 trace record structure

The IFCID 0191 record always consists of a header, QW0191HD, and one or more other sections. The exact structure of the 0191 record is dependent upon the Db2 parse state, DDM command and command/reply data object, and nature of the DRDA exception condition.



In addition to the QW0191HD section, the IFCID 0191 record might contain one or more of the following sections:

QW0191CR

There is one of these sections for each DDM command, command data object, reply message, and reply data object processed (successfully or unsuccessfully) by DDIS. This section summarizes salient information that is extracted during Db2 processing.

QW0191FD

There is one of these sections for every DDM QRYDSC object or FDODSC scalar contained within an SQLDTARD or SQLDTA object processed (successfully or unsuccessfully) by DDIS. These objects (scalars) contain FDOCA Late Descriptors (SQLDTA or SQLDTARD) and this section summarizes the state of the FDOCA geometry as processed by Db2. This section is also present if an OUTOVR descriptor is sent with a command to override the format of output data.

QW0191RT

There is one of these sections for every DDM QRYDTA object or FDODTA scalar contained within an SQLDTARD or SQLDTA object processed (successfully or unsuccessfully) by DDIS. These objects (scalars) contain data that is described by an FDOCA late descriptor that is previously processed and this section contains the Db2 RDTA data structure that is required to process the data. The format of the RDTA data structure is described below.

QW0191LT

There is one of these sections for every DDM command data object or reply data object that is described by an FDOCA early or late descriptor and has been processed (successfully or unsuccessfully) by DDIS. This section contains the Db2 FDLIDLST data structure that is required to retrieve the data from the command or reply data object. The format of the FDLIDLST data structure is described below.

QW0191EA

There is one of these sections for every DDM FDODTA scalar contained within an SQLDTA object that is processed successfully or unsuccessfully by DDIS. This section is provided for Db2 serviceability purpose only.

QW01916B

There is one of these sections for every DDM command or command data object received from a requester, or reply message or reply data object received from a database server. This section contains the actual level 6b data stream received.

PSPI

DDIS IFCID 0191 common diagnostic procedures

The diagnosis of DRDA exception conditions that are detected by DDIS always begins with the 32-bit Db2 reason code.

PSPI

If the DRDA exception condition is detected at the requester, then the reason code is presented to the application through the SQLERRD1 field within the SQLCA. Additionally, it is included within the Detailed Data section of the Tivoli NetView for z/OS Alert and the QW0191HD section of the IFCID 0191 trace record.

If the DRDA exception condition is detected at the database server, then the reason code is included within the Server Diagnostic information scalar (SRVDGN) and returned as part of the DDM reply message sent to the requester. Additionally, the reason code is included within the Detailed Data section of the Tivoli NetView for z/OS Alert and the QW0191HD section of the IFCID 0191 trace record.

PSPI

Interpreting IFCID 0191 records

Each IFCID 0191 trace record contains a QW0191HD section, one QW0191CR section, and one QW01916B section for each DDM level 6b object that was successfully or unsuccessfully parsed by DDIS.

PSPI

Depending upon the DDM object that is processed, there might be associated with the QW0191CR section a QW0191FD section, a QW0191RT section, and a QW0191LT section:

- The QW0191FD section is present if the DDM object is either a QRYDSC or the FDODSC scalar contained within the SQLDTA or SQLDTARD objects.
- The QW0191RT section is present if the DDM object is either a QRYDTA or the FDOTA scalar contained within the SQLDTA or SQLDTARD objects.
- The QW0191LT section is present if the DDM object is either a command data object or a reply data object, described by an FDOCA early or late descriptor.

How to read the QW0191HD section

The QW0191HD section contains the following fields:

QW0191RS

Contains the DDIS reason code, which uniquely describes the DRDA exception condition. This reason code should be used as the basis for further diagnosis of the DRDA exception condition.

QW0191MN

Contains the name of the DDF module (DRDS, DDIS, DTM) which requested the 0191 record.

QW0191NO

Contains the number of this IFCID 0191 record instance relative to the total number as specified in QW019190.

QW0191TO

Contains the total number of IFCID 0191 records that are required to capture all DDM objects and FDOCA data.

QW0191FL

Contains the length of the DDM object or FDOCA data that is retrieved from the data stream and that resulted in the DRDA Exception Condition.

QW0191MI

A unique 2-byte identifier, which identifies the processing point within the module that is defined in QW0191MN at which the 0191 request was made.

QW0191TK

A unique 12 character error token.

QW0191C1

The code point of the DDM command whose processing resulted in the request for the 0191 record.

QW0191PA

The DDIS parse state of the command.

- If this is 'P1', the remaining sections in the 0191 record represent reply messages and reply data objects that were received by the AR.
- If this is 'P2', the remaining sections in the 0191 record represent command data objects that are received from the AR.

QW0191LN

The location name of the AR (if QW0191PA is 'P2'), or the location name of the (if QW0191PA is 'P1').

QW0191RN

The total number of reply messages that are returned from the AS in response to the command. This is 0 if QW0191PA is 'P2'.

QW01910N

The total number of OBJDSSs received from the AR.

QW0191DN

The total number of level 6b data streams received all of which have the same request correlator.

QW0191ET

Defines the Db2 response to a detected error.

- If QW0191ET is 0, then Db2 detected no syntactic or semantic errors in the data stream.
- If QW0191ET is 1, then the Db2 AR detected an error or exception condition, which resulted in formatting the SQLCA with an SQLCODE and SQLSTATE. In this case, QW0191SS contains the SQLSTATE.
- If QW0191ET is 2, then the Db2 detected an error or exception condition, which resulted in a DDM reply message that is being sent to the AR. The code point of the DDM reply message is contained in QW0191C2.

QW0191PT

If QW0191PA is 'P1', then QW0191PT contains the last five states and events of the top-level DDIS Reply Parser (DSNLZRPA). Similarly, if QW0191PA is 'P2', the QW0191PT contains the last five states and events of the top-level DDIS Request Parser (DSNLZSPA). These parsers are implemented as finite state machines, consequently, the (state,event) trace provides the last path through the parser.

How to read the QW0191CR section

If the value of QW0191PA is 'P1', then one QW0191CR section is included for every reply message and reply data object processed in response to the command. If the value of QW0191PA is 'P2', then one QW0191CR section is included for the command and one section for every command data object that was processed.

The QW0191CR section contains the following fields:

QW0191PS

Contains the parse state of the DDM object. If the value is 'DRDUCC', then this object was successfully parsed. If the value is 'DRDAFAIL', then the object parse failed.

QW0191C3

Contains the code point of the command, command data object, reply message, or reply data object.

QW0191NM

Contains the DSS number, carrying the level 6b object that is defined by QW0191C3.

QW0191OF

Contains the offset relative to the start of the 0191 record of QW01916B section. It contains a level 6b RQSDSS, OBJDSS, or RPYDSS containing the object that is defined by QW0191C3.

If QW0191PS is 'DRDUCC', then the value of QW0191FO is 0. If QW0191PS is 'DRDAFAIL', then the contents of QW0191FO are the offset relative to the 0191 record of the point at which the parse failed. This offset points within the data stream beginning at QW01916B. The cause of the parse failure depends on the object as defined by QW0191C3.

If the value of QW0191C3 is X'241A' (QRYDSC) or X'2413' (SQLDTARD), then the object contains a Late Descriptor. In this case, QW0191D1 contains the offset relative to the start of the 0191 record of a QW0191FD section, describing the progress of the parse of the Late Descriptor.

If the value of QW0191C3 is X'241B' (QRYDTA), or X'2413' (SQLDTARD), then the object contains data that are described by a late descriptor. In this case, QW0191D2 contains the offset relative to the start of the 0191 record of a QW0191RT section. This section contains a Db2 RDTA data structure, which correlates the DRDA types that are extracted from the Late Descriptor with the SQL type.

If the value of QW0191C3 is X'243E' (SQLOBKNAM), X'2414' (SQLSTT), X'2419' (SQLSTTVRB), X'2408' (SQLCARD), X'2411' (SQLDARD), X'241B' (QRYDTA) or X'2413' (SQLDTARD), then the object contains data that is described by an early or late FD:OCA descriptor. In this case, QW0191D3 contains the offset relative to the start of the 0191 record of a QW0191LT section. This section contains a Db2 data structure

(FDLIDLST) which is a representation of the early or late descriptor and used to retrieve data from the data stream.

If the value of QW0191C3 is X'2412' (SQLDTA), then the object contains data that is described by an early or late FD:OCA descriptor. In this case, QW0191D4 contains the offset relative to the start of the 0191 record of a QW0191EA section. This section contains an internal Db2 structure that is provided for Db2 serviceability purpose only.

How to read the QW0191FD section

This section is present within the 0191 record only if the object is a Late Descriptor (SQLDTA or SQLDTARD). The section is found from offset QW0191D1 within the QW0191CR section.

QW0191LD

Contains the number of late environmental descriptors that are sent as part of the Late Descriptor.

QW0191L1

Contains the SQLDTAGRP LID extracted from the descriptor. If this is 0, the SQLDTAGRP was not processed.

QW0191L2

Contains the SQLCADTA LID extracted from the SQLDTARD descriptor. This is 0 if QW0191PA is 'P2'. It is also 0 if QW0191PA is 'P1' and the SQLCADTA array descriptor has not been processed.

QW0191L3

Contains the SQLDTA LID extracted from the SQLDTA descriptor. This is 0 if QW0191PA is 'P1'. It is also 0 if QW0191PA is 'P2' and the SQLDTA array descriptor has not been processed.

QW0191L4

Contains the SQLDTARD LID extracted from the SQLDTARD descriptor. This is 0 if QW0191PA is 'P2'. It is also 0 if QW0191PA is 'P1' and the SQLDTARD array descriptor has not been processed.

QW0191GN

Contains the total number of triplets (excluding Continue Previous Triplet) which reference DRDA types by LID, and are contained within the SQLDTAGRP. This is the number of input host variables that are being passed (QW0191PA is 'P2') or the number of columns in the answer set (QW0191PA is 'P1').

QW0191F0

Is on if the geometry of the SQLDTAGRP descriptor is correct. All LIDs specified by the constituent SQLDTAGRP triplets must define a valid DRDA type. If the SQLTAGRP specifies LIDs outside the range of X'02' through X'49' as late environmental descriptors, then the SQLTAGRP Meta Data Definition (MDD) (DRDA type X'D0') must precede the SQLTAGRP GDA and each Late Environmental Descriptor must be specified by its 7-byte MDD and 12-byte SDA.

QW0191F1

Is on if the geometry of the SQLCADTA row descriptor is correct. The SQLCADTA row must define all occurrences of the SQLCAGRP followed by all elements of the SQLDTAGRP. If QW0191PA is 'P1', then the DDM object is either a QRYDSC or the FDODSC scalar of the SQLDTARD. If QW0191PA is 'P2', then the DDM object is the FDODSC scalar of the SQLDTA object.

If the SQLTAGRP specifies late environmental descriptors, then the SQLCADTA RLO must be preceded by the Meta Data Definition (MDD) for type X'E0'.

In addition to satisfying the SQLDTAGRP requirements, the SQLCADTA row descriptor must define the row as consisting of all elements of the (nullable) SQLCAGRP followed by all occurrences of elements of the SQLDTAGRP. The reference to SQLCAGRP and SQLDTAGRP is by LID. The reference to SQLCAGRP must be X'54', the reference to SQLDTAGRP must be identical to the LID, which appeared within the SQLDTAGRP descriptor.

QW0191F2

Is on if the geometry of the SQLDTA descriptor is correct. The SQLDTA descriptor must be such that all late environmental descriptors precede the SQLTAGRP descriptor. The SQLDTA/SQLDTARD geometry must be such that all late environmental descriptors precede the SQLDTAGRP descriptor. Also, all LIDs contained within the constituent SQLDTAGRP triplets must have been defined previously by some Late Environmental Descriptor, or default to the DRDA type. In the latter case, the LID must be

within the range X'02' through X'4F' and X 'C8' through X'CF'. If any of these requirements are not satisfied, then the SQLDTA/SQLDTARD geometry is incorrect.

In addition to satisfying the SQLDTAGRP requirements, the SQLDTA descriptor must define the row as all occurrences of the SQLDTAGRP. The reference to the SQLDTAGRP is via LID and this reference must be identical to the LID specified in the SQLDTAGRP descriptor. If the SQLTAGRP specifies late environmental descriptors, then the SQLDTA RLO must be preceded by the Meta Data Definition (MDD) for type X'E4'. If these requirements are not satisfied, the SQLDTA geometry is incorrect.

QW0191F3

Is on if the geometry of the SQLDTARD row is correct. Following the SQLCADTA row, the descriptor must define the SQLDTARD array as an (open) array such that each row is one occurrence of the SQLCADTA row. Reference to the SQLCADTA row is by LID, thus, the LID referenced within the SQLDTARD array must be identical to the LID, which appeared in the SQLCADTA row descriptor. If the SQLTAGRP specifies late environmental descriptors, then the SQLDTA RLO must be preceded by the Meta Data Definition (MDD) for type X'E4'. If any of these requirements are not satisfied, the SQLDTARD geometry is incorrect.

How to read the QW0191RT section

The QW0191RT section is included only if the object contains a late descriptor. Therefore, this section is included only if the object is a QRYDSC, SQLDTA, SQLDTARD, or OUTOVR. Both QRYDSC and SQLDTARD contain the late SQLDTARD descriptor, both of which map to a Db2 output SQLDA. The FDODSC scalar in the SQLDTA object contains the late SQLDTA descriptor, and this maps to a Db2 input SQLDA.

The OUTOVR descriptor contains override type information for the output SQLDA and only applies if LOB data formats are being overridden (either from LOB data value to LOB locator or vice versa).

This section includes the Db2 RDTA data structure that is built from the SQLDTAGRP. There is one RDTAENT entry for every LID processed as one of the elements of the SQLDTAGRP. The value of RDTANUM is identical to the number of SDA references contained in the SQLDTAGRP.

For each RDTAENT entry within this structure, the Db2 derived SQL type, subtype, CCSID, and length are recorded. The RDTA is the source of the input or output Db2 SQLDA. The DRDA type that is specified in the SQLDTAGRP triplets is used to find the 12-byte SDA for that DRDA type. If the SDA describes the data as character, then subtype is 1 for SBCS, 2 for Mixed and 3 for Graphic. Given the subtype of character data, the CCSID is derived from the CCSID overrides in effect at the time of descriptor processing.

The selection precedence is as follows:

1. If Late Environmental Descriptor describes this LID, select CCSID from 12-byte SDA.
2. If no late environmental descriptor, then choose command local CCSID as specified via TYPDEFOVR command/reply data object.
3. If neither late environmental descriptor nor TYPDEFOVR, then choose CCSID as specified at ACCRDB/ACCRDBRM time.

RDTDA contains the address of the Db2 input/output SQLDA that was created for DRDS. The SQLDA is not traced. RDTALID contains an offset to the QW0191LT section. This points to the same section pointed to by QW0191D3. The contents of RDTAROW are 0. RDTAGLID is the SQLDTAGRP LID extracted from the SQLDTA or SQLDTARD descriptor and RDTARLID is the SQLCADTA row LID extracted from the SQLDTARD descriptor. These values are identical to the values of QW0191DG and QW0191CA, appearing within the QW0191FD section.

If the DRDA Exception Condition occurred during the data retrieval, the RDTA specifies the SQLDA that was presented to the RDS Resource Manager (input SQLDA) or the DRDS Resource Manager (output SQLDA).

How to read the QW0191LT section

This section includes the Db2 FDLIDLST data structure that is used to retrieve data that is described by an early or late descriptor. All early descriptors FDLIDLSTs are pre-built and are part of Db2. Local copies

from the read only pre-built structures are made on a demand basis. The FDLIDLSTs that support late descriptors are dynamically created during descriptor processing.

There is one FDLIDLST for every FDOCA group (nullable or nonnullable) to be retrieved from the data stream. The Db2 unit of data transfer is an FDOCA row. Each row contains one or more groups, and the QW0191LT section captures the FDLIDLSTs required to retrieve all groups that constitute a row. When contained in a QW0191LT section, the contents of the FD_LIDENT_PTR fields are the offset relative to the data stream (QW01916B section) of the start of the data that is processed for that entry.

How to read the QW01916B section

The QW01916B section contains a level 6b header, defining an RQSDSS, RPYDSS, or OBJDSS data stream, followed by the data stream.



IFCID 0191 trace record common diagnostic procedures

The diagnosis of a DRDA exception condition that is identified by DDIS begins with one or more IFCID 0191 trace records and the Db2 reason code that is recorded within the QW0191RS field of the QW0191HD section.



For each reason code, a diagnostic procedure exists which documents the specific nature of the IFCID 0191 records.

How to interpret IFCID 0192 records

The correct format for DDM data streams is described in DDM under the term "DSS". Essentially, DDM data streams consist of one or more DSSs, each of which contains a DDM level 6a header. Flags in the DDM level 6a header tell the receiver (Db2) about the relationship between the current DSS and other DSSs in the data stream.

The DCRM component of Db2 is responsible for validating the content of DDM level 6a data streams, as described by the DDM term DSS. DDM defines several syntax error and protocol error conditions, which are produced when invalid DDM data streams are detected. These errors are produced for one of the following reasons:

- A DDM level 6a header contains an invalid flag setting or data value.
- The flag settings or data values in a DDM level 6a header are not consistent with the flag settings or data values in the previous DDM level 6a header.

In order to describe these errors, the DCRM component records trace data for two events:

1. A DDM data stream is detected which contains a syntax error. When an error of this type is encountered, a DDM SYNTAXRM message is sent to describe the error. Within the SYNTAXRM message, the DDM SYNERRCD (syntax error code) and the DDM SVRCOD (severity code) describe the cause and severity of the error. For these errors, the reader should refer to the DDM SYNERRCD documentation, to determine the nature of the error.
2. A DDM data stream is detected which contains a protocol error. When an error of this type is encountered, a DDM PRCCNVRM message is sent to describe the error. Within the PRCCNVRM message, the DDM PRCCNVCD (conversational protocol error code) and the DDM SVRCOD (severity code) describe the cause and severity of the error. For these errors, the reader should refer to the DDM PRCCNVCD documentation to determine the nature of the error.

In either case, DCRM records the following information in the trace record:

- The nature of the error (protocol or syntax)
- The DDM severity code (SVRCOD)
- The DDM error code (PRCCNVCD for protocol errors, SYNERRCD for syntax errors)

- The offset of the current DDM level 6a header in the data stream. This offset is measured from the start of the first DDM level 6a header in the data stream.
- The current DDM level 6a header, as defined in the DDM DSS term. This header might or might not be the cause of the error, since many of the errors indicate an inconsistent condition between the current DDM level 6a header and the previous DDM level 6a header. Thus, it is possible that the error was caused by the previous DDM level 6a header.
- The first 250 bytes of data in the data stream that follows the current level 6a header
- The offset of the previous DDM level 6a header, if the error was detected in a header other than the first. This offset is measured from the start of the first DDM level 6a header in the data stream.
- The previous DDM level 6a header, as defined in the DDM DSS term.

How to interpret IFCID 0193 records

These records document the inconsistency between the SQLCODE returned by the AS for a commit or rollback command, and the disposition of the Unit of Work reported by the AS. For example, if a commit command was sent, and an ENDUOWRM reply message was received with UOWDSP = '01'X (commit disposition), and the SQLCODE in the SQLCA was -204, then the IFCID record would contain the values -204,'C','C'. The complete data stream for the reply, including the SQLCARD and the ENDUOWRM, is in the IFCID 0191 record, which is always written when an IFCID 0193 record is written.

PSPI

Distributed two-phase commit error conditions

Db2 support of distributed two-phase commit provides the detection of a number of events that can have a negative impact on data availability or that indicate or can lead to data inconsistency.

Each of these events is the result of an error condition or recovery actions that follow an error condition. These error situations place Db2 in the role of recovery coordinator, recovery participant, or both.

All of these events are traced to statistics class 4 and some of these events result in a Tivoli NetView for z/OS alert. The Tivoli NetView for z/OS alerts include alert models A001 - A006.

Related concepts

Diagnostic tools for DDF and VTAM

Various tools are available to help diagnose problems in DDF.

Statistics class 4 trace records

The statistics class 4 trace includes error situations that involve indoubt threads during the distributed two-phase commit process.

PSPI

The trace records available for diagnosis of indoubt thread error conditions are IFCIDs 0203 - 0210 and 0234 - 0236. For a complete description of these and the other statistics class 4 IFCIDs (0191 - 0195 and 0238), see SDSNIVPD(DSNWMSGs). The IFCIDs described here are helpful in the diagnosis and resolution of indoubt thread error conditions.

IFCID 0203

IFCID 0203 reports a heuristic decision that forces a commit or rollback for a distributed indoubt thread. A heuristic decision is made when errors prevent or significantly delay automatic indoubt thread resolution.

This record is written when:

- A Db2 RECOVER INDOUBT command is issued
- During the resynchronization process

This IFCID is often produced under a system service task rather than the Db2 agent's task. For this reason, the fields in this trace record should take precedence over similar values that are found in the product information portion of the IFC trace data. For example, the LUWID in the product information describes the Db2 system service task. The LUWID in this record describes the Db2 agent's thread.

IFCID 0204

IFCID 0204 is written when Db2 attempts to reconnect to a remote system that requests a cold start.



A cold start means that the remote system has no memory of the work that was in progress when the previous connection failed. Db2 produces this record only when Db2 has memory of threads whose outcome must be resolved.

If Db2 was the coordinator of one or more of those threads, the partner system might have been indoubt when the cold start occurred. The Db2 coordinator assumes that the participant recovery log record is lost or damaged and indoubt threads cannot be resolved.

If the partner system was the coordinator of one or more of those threads, the local Db2 subsystem might be indoubt, requiring manual intervention with the RECOVER INDOUBT command.

This IFCID is often produced under a system service task rather than the Db2 agent's task. For this reason, the fields in this trace record should take precedence over similar values that are found in the product information portion of the IFC trace data. For example, the LUWID in the product information describes the Db2 system service task. The LUWID in this record describes the Db2 agent's thread.

IFCID 0207

IFCID 0207 reports when heuristic damage is detected during two-phase commit resynchronization.

Heuristic damage occurs when an operator forces an indoubt unit of work to commit or rollback, and the operator's choice conflicts with the outcome chosen by the coordinator of the unit of work. If the alerting Db2 is the participant, the damage is at this Db2. If the alerting Db2 is the coordinator, the damage is at the participant.

This trace record is recorded by the Db2 subsystem that makes the heuristic decision and any Db2 coordinator immediately upstream from that subsystem. Other Db2 subsystems that are involved in the unit of work do not produce this trace record.

This IFCID is often produced under a system service task rather than the Db2 agent's task. For this reason, the fields in this trace record should take precedence over similar values that are found in the product information portion of the IFC trace data. For example, the LUWID in the product information describes the Db2 system service task. The LUWID in this record describes the Db2 agent's thread.

IFCID 0209

IFCID 0209 is written when a communication failure occurs after 1 and before or during phase 2 of the resynchronization process.

The thread that experiences the communication failure might still be indoubt at the participant location. If Db2 is the participant, the status is indoubt. If Db2 is the coordinator, the status is commit or rollback.

IFCID 0234

IFCID 0234 records when CICS or IMS attempts to perform indoubt thread resolution for a network identifier (NID) which Db2 does not recognize.

This error can occur due to the following reasons:

- a Db2 conditional restart
- a Db2 restart after your recover all of Db2 to a previous point in time

- an IMS or a CICS start with downlevel recovery logs

When a restart loses indoubt logical units of work, other downstream participants might be left with indoubt threads.

IFCID 0235

IFCID 0235 is written when a Db2 conditional restart results in partial information about a LUWID that might require resynchronization with a coordinator or participant. Incomplete information about the LUWID prevents resynchronization.

This IFCID is often produced under a system service task rather than the Db2 agent's task. For this reason, the fields in this trace record should take precedence over similar values that are found in the product information portion of the IFC trace data. For example, the LUWID in the product information describes the Db2 system service task. The LUWID in this record describes the Db2 agent's thread.



Chapter 6. Data management

You have several options when identifying and diagnosing inconsistent data.

Resolving inconsistencies manually

If you cannot resolve an issue by using the RECOVER utility, you might need to manually determine the cause of a data or index inconsistency and resolve the problem. Be sure to keep a complete record of the actions you take.

Analyzing 00C9010X or 00C902XX abends

The 00C9010x and 00C902xx abend reason codes are issued along with informational error messages (usually DSNI013I or DSNI014I) when logic errors, such as record length inconsistencies, are found.

Referential integrity constraint violations do not cause 00C9010x or 00C902xx abends because Db2 does not depend on referential constraints in order to operate. Referential constraint violations do, however, cause incorrect results in processing.

- For 00C90101 or 00C90105 abends, the error messages that issued include all pages in use at the time of abend. Generally, the pages have no inconsistency problem. However, if you receive several error messages or abends that identify problems with the same set of pages, it can be an inconsistency problem.
- For 00C902xx abends, the error messages issued usually include all pages that are involved in the inconsistency problem.

The error messages provide the following information for each page:

REASON

The abend reason code that is issued with the message.

TYPE

A type code that identifies the type of resource involved.

NAME

The name of the resource (database name, space name, and page number).

CONN-ID

The connection ID, which, when combined with the correlation ID (CORR-ID), and the logical-unit-of-work ID (LUW-ID), identifies the communication path involved.

CORR-ID

The correlation ID, which, when combined with the connection ID (CONN-ID), and the logical-unit-of-work ID (LUW-ID), identifies the communication path involved.

LUW-ID

The logical-unit-of-work ID, which, when combined with the connection ID (CONN-ID), and correlation ID (CORR-ID), identifies the communication path involved.

Much of this information also appears in the abend's diagnostic area of the associated SVC dump. Register 13 is at offset X'4C' in the SDWA. Offset X'F4' from the value in register 13 is the address of the CT. At offset X'78' in the CT is the address of the abend's diagnostic area. The following figure illustrates the format and contents of this diagnostic area. It begins with a 96-byte prefix area, which is followed by a number of 96 byte "subareas," each containing information about 1 page that is involved in the problem. The diagnostic area contains as many contiguous subareas as there are pages that are involved in the problem.

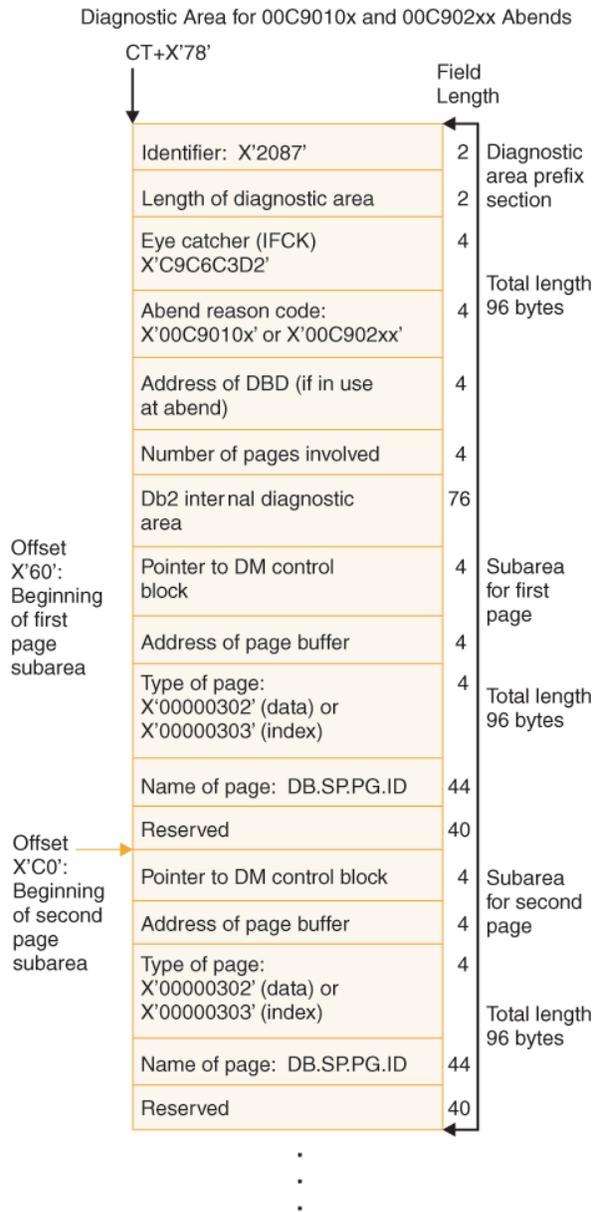


Figure 96. 00C9010X and 00C902XX SVC dump diagnostic area format

The name of page field is abbreviated as DB.SP.PG.ID:

DB

Database name: eight characters, followed by a one-character delimiter

SP

Table space (file page set) or index space (index page set) name: eight characters, followed by a three-character delimiter

PG

Page number: eight characters, followed by a four-character delimiter

ID

The ID of a record in a data page or the index subpage number in a segmented leaf page: two characters, followed by a one-character delimiter

For example:

```
DSN8DAPP.DSN8SDEP.X'000002'.X'01' (as it appears in messages)
DSN8DAPP.DSN8SDEP.X.000002..X.01. (as it appears in dumps)
```

Names are stored in character format. If the ID is unavailable, the field contains blanks. If the high-order bit of the ID field is on, the ID identifies an anchor point in a hash page, meaning that the Db2 directory or catalog is involved in the problem. Although the database name, the space name, and the page number are included in DSNI013I and DSNI014I messages that are issued, the ID (when available) is only found in the diagnostic area of the dump.

Related concepts

Type-of-failure keywords

A type-of-failure keyword describes an external symptom of a program failure.

Analyzing the SVC dump

You can analyze the SVC dump that is issued for the abend that you received.

Procedure

To analyze the SVC dump:

1. For each subarea in the diagnostic area of the dump, examine the ID in the name of page field.
2. If the ID is not included in the subarea for a given page, other diagnostic information that is produced before or along with the abend might indicate the record that is involved. However, you might have to analyze all the records in all the named pages. In cases of incomplete information, contact IBM Support.
3. If you found one or more IDs, locate each record that is identified by the corresponding ID. For 00C9010x abends, each ID identifies the record in use at the time of abend. For 00C902xx abends, each ID identifies a specific record that is involved in the inconsistency.
4. Examine the records that you found and try to determine where the inconsistency exists.
5. Analyze the structure that contains each record and verify that the connecting RID pointers are correct.
6. Be aware that the inconsistency might involve a missing record. In this case, the PG field identifies the page that was supposed to contain the indicated record. The DM might have tried unsuccessfully to find the record in the page.
7. In some operations, such as a DELETE, one of the pages that are involved in the inconsistency might not be identified at all because processing on that page had already completed at the time of the abend. In cases of incomplete information, contact IBM Support.
8. In the case of a 00C9010x abend, further information about the source of the problem might be obtained from knowing what DM operation was in process at the time the problem was detected. Use the following steps to determine what was going on:
 - a) Find the CSECT name identifier (last 5 bytes of the CSECT name) in the field named VRARRK5 of the variable recording area (VRA) in the SDWA. In an SVC dump, the VRA begins at offset X'190' in the SDWA. The first 3 bytes are always "DSN".
 - b) Look up this CSECT name in the online CSECT Directory and read the description.

In most cases, the description of the CSECT provides some indication of what was going on at the time of the abend.

For example, suppose DSNIBHUN was the name of the CSECT that issued the 00C9010x abend. The description indicates that this CSECT is involved with processing recovery log records. The problem in this case might be an inconsistency within the log record that is being processed or an inconsistency between the log record and the page to which the log record is being applied. If a page is involved in the problem, it is identified in a DSNI014I message and corresponding diagnostic area in the dump.
9. For 00C902xx abend reason codes, it is possible for more pages to be identified as being involved in the problem than the number of pages that are actually involved. These additional pages are included because they were in use at the time of the abend.
10. If no problem is found after you analyze the records that are involved, it is still possible that an inconsistency exists between a database descriptor (DBD) and the data or index involved. If a DBD

was involved at the time of the abend, the address of the DBD is in the prefix of the diagnostic area. Use this address to locate the DBD in the dump.

If you find one or more inconsistencies in the pages you analyzed in the SVC dump, determine whether the problem exists on DASD or only in virtual storage

- a) Obtain a REPAIR, DSN1COPY, or DSN1PRNT dump of each page that is involved in the problem.
- b) Analyze the dump to determine whether the problem you found in the SVC dump also exists on DASD.
 - If the problem does not appear in the dump of the pages, indicating that the transaction that was in process at the time the SVC dump was issued caused the problem, but only exists in virtual storage. This transaction is likely to cause the same problem when it is processed again.
 - If the problem does appear in the dump of the pages as well as in the SVC dump, the problem exists on DASD. Continue with the next step.
- c) If the problem exists on DASD:
 - Use the RECOVER utility to recover the page set containing the pages that are involved.

If RECOVER completes successfully, you still must verify that the problem is resolved on DASD:

 - i) Obtain a REPAIR, DSN1COPY, or DSN1PRNT dump of the pages involved.
 - ii) Look at the dump to determine whether the inconsistency is resolved.
 - iii) If the inconsistency is resolved, you should be able to continue normal operations. If it is not resolved, use the REPAIR utility to manually resolve the problem.

Related concepts

Analyzing 00C9010X or 00C902XX abends

The 00C9010x and 00C902xx abend reason codes are issued along with informational error messages (usually DSNI013I or DSNI014I) when logic errors, such as record length inconsistencies, are found.

Diagnosing DBD inconsistencies

You can locate a database descriptor (DBD) by using the REPAIR DBD utility or in a dump, and analyze the structure of the DBD.

Related tasks

Running REPAIR

The REPAIR utility is used to resolve data or index inconsistencies manually.

Related reference

Printing and analyzing dumps

You might need to print and analyze a memory dump to help to diagnose a problem.

DSN1COPY (Db2 Utilities)

DSN1PRNT (Db2 Utilities)

REPAIR (Db2 Utilities)

Analyzing 00C90102 abends

The 00C90102 abend reason code is issued when an inconsistency is found within a data or index page and automatic recovery is not successful.

Before a 00C90102 abend reason code is issued, Db2 issues a message (DSNI011I or DSNI012I), indicating that a page is damaged. These messages identify the page number and page type, the name of the module that detected the inconsistency, and the trace code that describes the event in process.

If you were unable to successfully resolve the inconsistency by using the RECOVER utility, and there is an SVC dump associated with the problem, start an analysis by finding the trace code. SVC dumps can provide some useful information, but might not always reflect the data actually residing on DASD, so do not use it as a basis for repairing data with the REPAIR utility. Instead, use a dump that is produced by REPAIR, DSN1COPY or DSN1PRNT.

Finding the trace code

The trace code describes the event that was in process at the time of abend and is shown in the messages that are issued for the abend (the value after the ERQUAL variable in the message), as well as in the SVC dump that is issued for the abend.

If for some reason the trace code cannot be determined from the messages, you can find it in the SVC dump in the following locations:

- In the last 2-byte field of the dump title on the first page. The trace code appears after the colon that follows the failing CSECT name.
- In the VRA diagnostic information report and the VRA area of the SDWA or SYS1.LOGREC. The trace code is associated with the key X'CE' (field name VRARRK6). See [“The variable recording area \(VRA\)” on page 205](#) for information about where these data areas appear in an SVC dump.

Do not refer to the Db2 trace table in the dump; subsequent events might have overwritten necessary information.

Use the trace code to determine what is wrong with the page. This error is the first error Db2 detected in the page, but there can be others.

Trace code X'FFFx' is one indication the Db2 catalog or directory is involved. Trace code X'0000' indicates an abend occurred during RECOVER processing, causing Db2 to mark the page as inconsistent or "broken".

Refer to the information in [“Analyzing a data page” on page 339](#) and [“Analyzing an index page” on page 342](#) when you have a REPAIR, DSN1COPY or DSN1PRNT dump of the inconsistent page. An SVC dump can also be used for analysis, but do not use it as the basis for changing data with the REPAIR utility. SVC dumps might not reflect the data as it appears on DASD. In SVC dumps, the address of the damaged page is in register 9 at offset X'3C' in the SDWA.

Analyzing a data page

Analyze the data page when you have a REPAIR, DSN1COPY or DSN1PRNT dump of the inconsistent page.

About this task

To determine whether a data page is damaged, you probably must check the page header, the ID map, the records and holes, and the total free space by working through the procedure that follows. If you find an error, continue checking all areas because other errors might exist as well. All errors must be resolved to restore the page to a consistent state.

The following figure illustrates the structure of a data page.

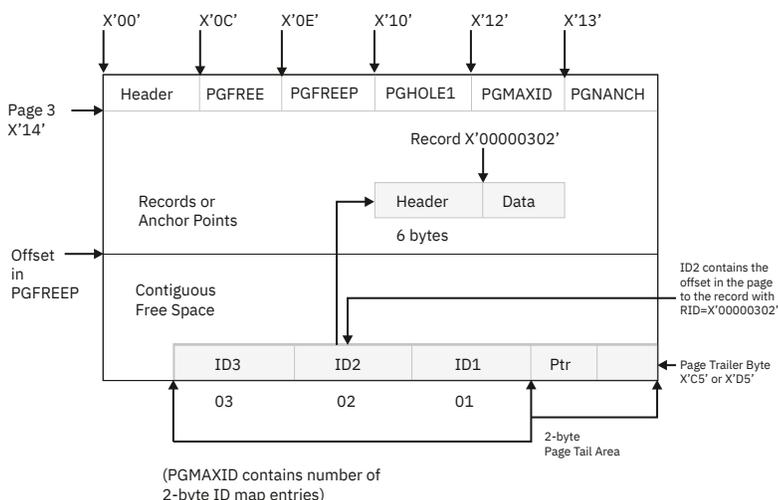


Figure 97. Format of file page set data page

Follow all steps in this procedure, even after you detect an error. All errors must be resolved, not just the first one you find. If data sharing, the relative byte address (RBA) field should have an LRSN value.

Procedure

To analyze a data page:

1. Check the page header (PGHEAD)

- Check PGCOMB, a 1-byte field at offset X'00'.
 - If Bit 3 = B'1', the last byte in the page must be X'D5'.
 - If Bit 3 = B'0', the last byte in the page must be X'C5'.
- Check PGFLAGS, a 1-byte field at offset X'0B'.
 - If Bit 0 = B'1', the data manager (DM) set the "broken" bit on to indicate a damaged page.
 - If Bit 0 = B'0', the DM did not set the "broken" bit. However, a problem can still exist.
 - Bits 1 and 5 should be B'0', indicating a data page.
 - If Bit 3 = B'1', the page is a space map page of either a segmented (Bit 2 is also B'1') or nonsegmented file page set. Do not follow the remaining procedures. Call an IBM support center.

2. Check whether the Db2 directory (DSNDB01) is involved.

- Check PGNANCH, a one-character field at offset X'13'.

If PGNANCH > X'00', bit 4 of PGFLAGS (at offset X'0B') must be B'1', indicating that the page is a hash page in the Db2 directory. PGNANCH contains the number of 8-byte anchors points in the page. Each anchor point contains a forward and backward pointer. For all anchors points, verify that either both pointers are X'00000000' or both pointers contain valid record identifiers (RIDs). A valid RID has a nonzero value in the first 3-byte portion and a nonzero value in the last byte.

3. Define three working variables:

- #IDS to accumulate the number of ID map entries.
- TOTAL HOLE to accumulate the total amount of storage that is contained in holes.
- CURRENT OFFSET to accumulate the offset to the part of the page that is being analyzed.

4. Analyze the space that contains records and holes, which begins at offset X'14' + (8×PGNANCH).

This space ends at the offset stored in PGFREEP (a two-character field at offset X'0E'). Starting with the first record or hole, perform the following checks, depending on what is stored in the page at the current point of interest.

a. Record or Overflow Record (first byte is X'00' or X'20'):

- Verify that the ID map entry whose number is contained in PGSBID (a one-character field at offset X'05') contains the offset of this record. Add 1 to the value currently in the working variable that named #IDS.
- Add the value in PGSLTH (a two-character field at offset X'01' in the record) to CURRENT OFFSET to get the offset to the next record or hole.

b. Pointer Record (first byte is X'40'):

- Add 6 bytes to CURRENT OFFSET to get the offset to the next record or hole.

c. Large Hole (first byte is X'80'):

- Add the value in PGHLTH (a two-character field at offset X'01' in the hole) to TOTAL HOLE.
- Add the value in PGHLTH to CURRENT OFFSET to get the offset to the next record or hole.

d. Small Hole (first byte is X'81', X'82', X'83' or X'84'):

- Add the length of the hole to TOTAL HOLE. The length is the value of the second nibble in the first byte (for example, the first byte of a 2-byte short hole is X'82').
- Add the length to CURRENT OFFSET to get the offset to the next record or hole.

Repeat these steps until CURRENT OFFSET is equal to the value in PGFREEP.

5. Analyze the chain of large holes by checking PGHOLE1, a two-character field at offset X'10' in the data page. If PGHOLE1 is not X'0000', use the value in PGHOLE1 as the offset to the first hole, and perform the following tests:
 - a) Verify that a large hole is at the offset (the value of the first byte should be X'80').
 - b) Use the value in PGHCHAIN (a two-character field at offset X'03' of the large hole) as the offset to the next hole and repeat this step if PGHCHAIN > X'0000'. If PGHCHAIN = X'0000', continue with the next step.
6. Analyze the ID map and page tail area. If not all existing ID map entries are in use (#IDS (used entries) does not equal PGMAXID (used + free entries)), the next-to-the-last byte in the page should contain the number of the first free ID map entry in the chain.
 - a) The ID map entry pointed to should not be currently in use (high-order bit should be on).
 - b) Use the low-order portion of the free ID map entry as an ID map entry number, and repeat these steps until you reach the end of the chain of free ID map entries.
7. Verify PGMAXID, a one-character field at offset X'12' of the page, contains the number of ID map entries that have been allocated, including both used and free entries.
8. Verify PGFREE, a two-character field at offset X'0C' of the page, contains the amount of total free space in the page.

- For 4-KB pages (X'1000', or 4096, bytes):

```
free space = X'1000' - X'02' for tail area
             - (2×PGMAXID) - PGFREEP
             + X'TOTAL HOLE' = PGFREE
```

- For 8-KB pages (X'2000', or 8192, bytes):

```
free space = X'2000' - X'02' for tail area
             - (2×PGMAXID) - PGFREEP
             + X'TOTAL HOLE' = PGFREE
```

- For 16-KB pages (X'4000', or 16384, bytes):

```
free space = X'4000' - X'02' for tail area
             - (2×PGMAXID) - PGFREEP
             + X'TOTAL HOLE' = PGFREE
```

- For 32-KB pages (X'8000', or 32768, bytes):

```
free space = X'8000' - X'02' for tail area
             - (2×PGMAXID) - PGFREEP
             + X'TOTAL HOLE' = PGFREE
```

PGMAXID, at offset X'12' of the page, contains the number of allocated ID map entries. PGFREEP, at offset X'0E' of the page, contains the offset to the contiguous free space in the page. TOTAL HOLE is the figure that you computed in [Step 4](#) for the total space that is occupied by holes in the page.

Results

If you found errors and determined what must be modified by way of the REPAIR utility, continue with “Running REPAIR” on [page 342](#). If no errors or inconsistencies were detected, the page is probably not damaged.

Analyzing an index page

Analyze the index page when you have a REPAIR, DSN1COPY or DSN1PRNT dump of the inconsistent page.

Procedure

Run the CHECK INDEX utility.

This checks the index page header, the physical index page header, the subpage directory, the logical index page header, and the page entries. All errors must be resolved to restore the page to a consistent state.

- If you found errors and determined what must be modified by way of the REPAIR utility, continue with [“Running REPAIR” on page 342](#).
- If you detect no errors or inconsistencies, the index page is probably not damaged.

Running REPAIR

The REPAIR utility is used to resolve data or index inconsistencies manually.

About this task

(Use the CHECK utility to manually resolve referential constraint inconsistencies.) Use the following task after you locate and analyzed the damaged page in a dump and you failed to resolve the problem with the RECOVER utility.

Do not use an SVC dump as the basis for changing data by way of REPAIR. SVC dumps do not always reflect the data as it exists on DASD. Use a dump that is produced by REPAIR, DSN1COPY, or DSN1PRNT instead. Always verify that the page has not changed on DASD by using the VERIFY keyword of REPAIR to verify the log RBA (PGLOGRBA) at offset X'01' of the page header.

Be extremely careful if you are changing the Db2 catalog or directory; refer to Physical Formats and Diagrams for detailed information about the contents and structure of the catalog and directory.

Refer to [Syntax and options of the REPAIR control statement \(Db2 Utilities\)](#) for the complete syntax of the REPAIR utility

Procedure

To run REPAIR:

1. Invoke the REPAIR utility with the LOG YES option and the DUMP control statement, specifying the pages that you suspect are damaged.
Then, verify that the dump you received contains the pages that you want.
2. If you know which page is damaged and you can see how to resolve the error, repair the page, and reset the "broken" bit. Invoke the REPAIR utility with the REPLACE RESET DATA control statement. Keep track of your actions in case anything must be "undone" later.
3. If you determined that the page is not damaged but merely has the "broken" bit on, reset the "broken" bit. Invoke the REPAIR utility with the REPLACE RESET control statement.
4. If you cannot determine how to resolve the problem by way of REPAIR or are unable to do so, contact IBM Support.

Results

If the REPAIR utility was used with the LOG NO option, do not use the RECOVER utility on the data that is modified by REPAIR. Because REPAIR activities are not logged, the RECOVER utility does not recognize the changes that are made with REPAIR. After you successfully use REPAIR (all inconsistencies in the page set are resolved), take an image copy of the data. When the image copy is successfully completed, use the RECOVER utility on the data. This task is because RECOVER uses the new image copy of the

data that you modified with REPAIR. If the image copy of the repaired data should become damaged and fallback to a prior image copy occurs, the result of using REPAIR is lost, and the inconsistency reappears.

Inconsistency resolution with RECOVER TABLESPACE and RECOVER INDEX

The RECOVER and REBUILD INDEX utilities provide the easiest way to recover inconsistent data that is indicated by abend reason codes 00C9010x or 00C902xx.

Related reference

[RECOVER \(Db2 Utilities\)](#)

[REBUILD INDEX \(Db2 Utilities\)](#)

RECOVER preparation

Before you run RECOVER on a table space or index space, determine the name of the page set and the number of the page that is involved in the data or index inconsistency.

To prepare to run RECOVER, use the most convenient of the following procedures:

- Review the DSNIXxxx messages you received. The format of the NAME category is DB . SP . PG

where

DB = database name

SP = table space (file page set) or index space (index page set)

PG = page number

If the database name of the inconsistent resource is DSNDB01 (directory) or DSNDB06 (catalog), there can be inconsistency problems in the hash chains in the directory or catalog, and subsequent DDL activity can cause more damage. Before proceeding, stop further DDL activity by issuing the following command:

```
-START DATABASE (DSNDB06) ACCESS (RO)
```

- Another way to find the page number of the damaged page is to locate register 8 at offset X'38' in the system diagnostic work area (SDWA) of the SVC dump that is issued for the abend. The SDWA is at the beginning of the dump. Register 8 contains the address of the buffer manager (BM) block. At offset X'10' in that block is the 3-byte hexadecimal number of the inconsistent page.
- A third method for determining the number of the damaged page and the name of a damaged page set is to locate either the variable recording area (VRA) or the VRA diagnostic information report. In an SVC dump, the VRA begins at offset X'190' in the SDWA. The VRA diagnostic information report is at the beginning of the formatted section of the dump. The page number in the dump where the formatted section begins can be found as the last entry of Print Dump Index or Table of Contents in the dump (indexed by "Formatted From DSNWDMP").

In the VRA, offset X'02' contains the type code. If the type code is 302, the damaged page is a data page; a type code of 301 or 303 indicates an index subpage or index page. Offset X'16' of the VRA contains the number of the damaged page and offset X'0B' contains the name of the page set involved in the inconsistency.

In the VRA diagnostic information report, this information is associated with the following keys:

X'D6'

The field name is VRARRK14; it contains the type code.

X'D7'

The field name is VRARRK15; it contains the name of the resource that is inconsistent. The page number is at offset X'14' and the page set name is at offset X'09' of this field.

To determine which pages in a table space are logically in error, issue the command

```
-DISPLAY DATABASE(database-name) SPACENAM(ts-name) LPL
```

Related concepts

The variable recording area (VRA)

More diagnostic information for Db2 abend reason codes is placed in the variable recording area (VRA) of the system diagnostic work area (SDWA) and is extracted and displayed in the VRA Diagnostic Information Report. This data can be produced by common recording routines and certain Db2 subcomponents.

Running RECOVER TABLESPACE and REBUILD INDEX

RECOVER can be used to recover an entire file page set, a single data page in a page set, or pages in error range. RECOVER can be used for table spaces and index spaces. REBUILD INDEX can be used to recover a single index or multiple indexes that are associated with a table space.

Procedure

To run RECOVER TABLESPACE and REBUILD INDEX:

1. Run RECOVER TABLESPACE for inconsistencies in a table space, supplying the page set name and page number. Run REBUILD INDEX for an inconsistent index, supplying the name of the index. All indexes in a table space can be rebuilt, that is, every index on every table in the table space. Or specify a number of individual indexes (index spaces) to recover.
2. Verify that the recovery was successful. Message DSNU500I indicates successful data recovery and message DSNU259I indicates a successful index rebuild.
3. If the inconsistency cannot be resolved with RECOVER TABLESPACE or REBUILD INDEX, use the SVC dump that is issued for the abend. Continue with [“Resolving inconsistencies manually” on page 335](#).

If an SVC dump is not available, but you know the page number that is involved, obtain a REPAIR, DSN1COPY or DSN1PRNT dump of the inconsistent page. The REPAIR utility can be used with the DUMP option to obtain a dump, or, for 00C90102 abends, DSN1COPY can be used with the CHECK option to obtain a dump. After a dump of the page that is involved in the inconsistency is obtained, continue with [“Resolving inconsistencies manually” on page 335](#).

4. When in doubt about what page to recover and a dump is not available, contact the IBM Support Center or IBM Service.

Related reference

[REBUILD INDEX \(Db2 Utilities\)](#)

[RECOVER \(Db2 Utilities\)](#)

[REPAIR \(Db2 Utilities\)](#)

[DSN1COPY \(Db2 Utilities\)](#)

[DSN1PRNT \(Db2 Utilities\)](#)

Diagnosing DBD inconsistencies

You can locate a database descriptor (DBD) by using the REPAIR DBD utility or in a dump, and analyze the structure of the DBD.

Using REPAIR DBD

REPAIR DBD is an extension to the REPAIR utility. It is designed to help maintain consistent database definitions between the Db2 catalog and directory.

Running REPAIR DBD, requires SYSADM authority, SYSCTRL, or installation SYSOPR authority.

REPAIR DBD provides the following abilities:

- Compare the definition of a database in the catalog with its definition in the directory.
- Rebuild the directory from the information in the catalog.
- Remove an inconsistent database descriptor (DBD) from the catalog and the directory.

Db2 must be operational when REPAIR DBD is run. Also, REPAIR assumes referential integrity in the catalog. You can run the CHECK DATA utility or sample queries from member DSNTESTQ of the *prefix.SDSNSAMP* library to verify data consistency within the catalog.

To aid in the diagnosis of an inconsistent DBD, run REPAIR DBD with the DIAGNOSE option. The output from REPAIR DBD contains two DBDs - the actual DBD on DASD and a DBD reconstructed from information in the Db2 catalog and directory. If you can obtain a copy of the inconsistent DBD with REPAIR DBD, continue with [“Analyzing a DBD”](#) on page 349. If you use REPAIR DBD to obtain a copy of the inconsistent DBD is not successful, continue with [“Finding a DBD in a dump”](#) on page 345.

Related reference

[REPAIR \(Db2 Utilities\)](#)

Finding a DBD in a dump

When you analyze problems that involve a DBD, it is necessary to obtain an SVC dump that contains the DBD or a dump that is produced by REPAIR, DSN1COPY or DSN1PRNT.

If you use an SVC dump that contains the DBD to analyze the inconsistency, be aware that the DBD that appears in the dump is not contained in DBD01 records, but rather in memory as one contiguous block or a chain of DBD blocks. Also, the area of memory that appears in the SVC dump might not include the entire DBD. To obtain a REPAIR, DSN1COPY or DSN1PRNT dump of the DBD, you need to know which DBD is involved and follow the steps under [“Analyzing a repair, DSN1COPY or DSN1PRNT dump”](#) on page 347.

If you do not know which DBD is involved and are unable to obtain a dump, contact an IBM support center.

Analyzing an SVC dump issued by a DSNGDxxx module

When an SVC dump is issued by a DSNGDxxx module, a programming error was encountered while a DBD was being built or modified. In this case, there is nothing that can be done to analyze the structure of the DBD because the structure is not completed.

About this task

After you complete the following steps, contact IBM Support for assistance.

Procedure

To analyze an SVC dump that is issued by a DSNGDxxx module:

1. Print the dump that was issued.
2. Make a note of the SQL operations that were being performed before the dump was issued. If you cannot determine what SQL operations were going on, review the description of the abend code. This information might help IBM Support understand and resolve the problem.
3. To prevent more DDL activity from causing further damage, issue a **-START DATABASE** command for read-only (RO) access for the catalog database, DSNDB06:

GUIP

```
-START DATABASE (DSNDB06) ACCESS (RO)
```

GUIP

Analyzing an SVC dump issued by a DSNIxxxx or DSNKxxxx module

Some SVC dumps issued by DSNIxxxx or DSNKxxxx modules contain DBDs.

About this task

If an SVC dump is obtained from a DSNIxxxx or DSNKxxxx module and you suspect that a DBD is involved in the problem, follow the following procedure to determine whether a DBD is included in the dump.

Procedure

To analyze an SVC dump that is issued by a DSNIxxxx or DSNKxxxx module:

1. Review the abend reason code:

| Option | Description |
|---|--|
| If the abend reason code is 00C90102: | Continue with step “2” on page 346 until the DBD is found, or until it is determined that there is no DBD in the dump. |
| If the abend reason code is 00C90101, 00C9011x, or 00C902xx: | Obtain the address of the DBD in the diagnostic area of the dump by analyzing the 00C9010X or 00C902XX abends. After you locate the DBD in the dump, continue with “Analyzing a DBD” on page 349. |

2. Locate the system diagnostic work area (SDWA) at the beginning of the dump.
3. In the field named SDWAGR13 (at offset X'4C' in the SDWA) is register 13. The value at offset X'F4' from the value in register 13 is the address of the cursor table (CT). At the beginning of the CT should be the following value:

```
X'200F'
```

If this value is not in the dump at this location, there is no DBD in the dump.

4. Use the DMPR or the CTDB to find the DBD:

| Option | Description |
|--|---|
| Using the DMPR control block to find the DBD: | <ol style="list-style-type: none"> a. CTDMPR in the CT contains either a 0 or the address of the DMPR (dump request) control block. If the value is 0, there is no DBD in the dump. b. At the beginning of the DMPR should be the following value: <pre>X'20DFxxxC4D4D7D9'</pre> <p>If this value is not in the dump at this location, there is no DBD in the dump.</p> c. At offset X'40' from the start of the DMPR control block is an array of 16 8-byte elements. These elements represent the last 16 DBD requests. Each element contains a 2-byte control ID, followed by a 2-byte length field, followed by a 4-byte address field, DMPRRADR. Each DMPRRADR field contains the starting address of a requested DBD. d. At the beginning of each DBD should be the following value: <pre>X'2039xxxxC4C2C440'</pre> e. If the DBD you are looking for is not at one of the 16 DMPRRAD addresses, use the CTDB to find the DBD. |
| Using the CTDB to find the DBD: | <ol style="list-style-type: none"> a. CTCUBPT in the CT contains either a 0 or the address of the CUB (cursor block). If the value is 0, there is no DBD in the dump. b. At the beginning of the CUB should be the following value: |

| Option | Description |
|--------|--|
| | <pre data-bbox="477 205 669 233">X'2015xxxxC3E4'</pre> <p data-bbox="464 264 1360 294">If this value is not in the dump at this location, there is no DBD in the dump.</p> <p data-bbox="436 308 1446 369">c. CUBCTDBP in the CUB contains the address of a CTDB (cursor table DBD block). At the beginning of the CTDB should be the following value:</p> <pre data-bbox="477 396 719 424">X'200DxxxxC3E3C4C2'</pre> <p data-bbox="464 455 1360 485">If this value is not in the dump at this location, there is no DBD in the dump.</p> <p data-bbox="436 499 1422 560">d. CTDBPTR in the CTDB contains either 0 or the address of a DBD. If the value is 0, there is no DBD in the dump.</p> <p data-bbox="436 575 1154 604">e. At the beginning of the DBD should be the following value:</p> <pre data-bbox="477 632 719 659">X'2039xxxxC4C2C440'</pre> |

5. Continue with “Analyzing a DBD” on page 349.

Related concepts

Analyzing 00C9010X or 00C902XX abends

The 00C9010x and 00C902xx abend reason codes are issued along with informational error messages (usually DSN1013I or DSN1014I) when logic errors, such as record length inconsistencies, are found.

The system diagnostic work area (SDWA)

Each SVC dump that is requested by a Db2 functional recovery routine usually contains an SDWA with information about the status of the subsystem at the time of the error. Typically, the SDWA is a starting point for diagnosis.

Analyzing a repair, DSN1COPY or DSN1PRNT dump

If you know which DBD is involved in the problem and you do not have an SVC dump that includes the DBD, obtain a REPAIR dump of the various sections of the DBD or a DSN1COPY or DSN1PRNT dump of the data set that contains the DBD.

About this task

If the data set is large, you might want to dump the individual pages that the following procedures point you to, rather than expend the resources necessary to obtain a dump of the entire data set. However, dumping individual pages can involve a great deal of work.

This task explains how to obtain a dump of specific data areas by using the REPAIR utility. You can also obtain a dump of these areas by using the DSN1COPY or DSN1PRNT utility.

If a DSN1PRNT dump is used to analyze the inconsistency, the FORMAT option should not be specified when the dump is printed. This option separates the header and anchor point area from the rest of the page, labels the individual fields of the header, and does not show the offsets from the beginning of the page, making it difficult to analyze the page.

Procedure

To obtain a REPAIR dump:

1. DBDs are stored in the Db2 directory in the DBD01 page set (*vsamcatalogname*.DSNDBD.DSNDB01.DBD01.I0001.A001) in DBDR and DBDS records. The DBDR (parent) record is connected to the DBDS (child) record via a link. Use the following steps to locate the DBDR:
 - a. The anchor point of the DBD contains the RID of the first DBDR on the hash chain. Use the hashing algorithms to find the page number and anchor point.

- b. Use the REPAIR utility with the DUMP control statement to dump the page of anchor points.
The REPAIR utility expects page numbers to be given in hexadecimal, and offsets in either decimal or hexadecimal.
- c. After the anchor point is found, use the REPAIR utility with the DUMP control statement to dump the page that is specified in the first RID (if the page has not already been dumped).
2. Use the ID map entry value in the fourth byte of the first RID to locate the first DBDR record in the hash chain.
3. After you located the first DBDR in the chain, determine whether the DBID at offset X'16' in the DBDR matches the DBID of the DBD being checked.

| Option | Description |
|------------------------------------|---|
| If the DBID matches: | Continue with step “4” on page 348. |
| If the DBID does not match: | Follow the hash chain (the RID at offset X'06' of the DBDR) forward until you locate the DBDR with a DBID that matches the DBD in question. If a REPAIR, DSN1COPY or DSN1PRNT dump of the entire page set is not available, dump the page that the RID points to. |

The following figure illustrates the structure of DBDRs in the hash chain of an anchor point.

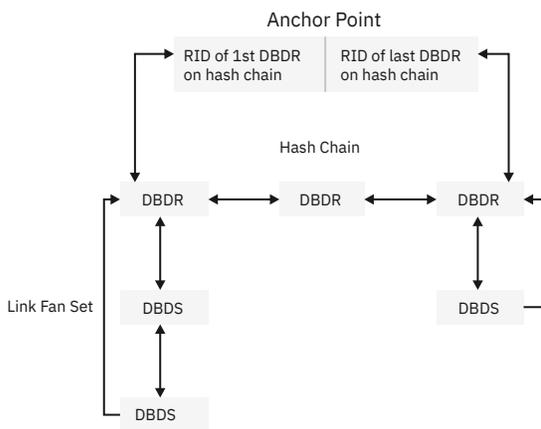


Figure 98. DBDRs in hash chain

4. Next, find the DBDS (child) records.
5. To locate the first DBDS record in the link fan set, use the RID at offset X'0E' of the DBDR. A RID of 0 indicates that no child exists for this parent record.
6. Use the forward link pointer at offset X'06' of the first child record to locate the second child record.
7. Continue to use the forward link pointers to locate successive child records in the link fan set. The forward link pointer of the last child record contains the RID of the parent record; the high-order bit of the last byte in the RID should be turned on.
8. After you locate the parent (DBDR) and all the child (DBDS) records for the DBD, logically concatenate the data areas of these records so they can be analyzed as if they were in one contiguous section.
9. Continue with “Analyzing a DBD” on page 349.

Related reference

[DSN1COPY \(Db2 Utilities\)](#)

[DSN1PRNT \(Db2 Utilities\)](#)

[REPAIR \(Db2 Utilities\)](#)

Analyzing a DBD

After you locate the DBD in the dump, the cause of the inconsistency can now be analyzed.

A DBD in virtual storage (memory) can have many DBD sections, resulting from DDL CREATE statements. Conceptually, however, a DBD is a contiguous block of information that contains object descriptors (OBIDs) for various DM objects. Each OBD has an identifier that is called an OBID. The OBID is an index into the OBDDMAP. The offset to a particular OBD from the beginning of the DBD header is equal to the value in OBDDMAP (OBID). For instance, if the OBID is 4, the offset to that OBD is stored at OBDDMAP(4), the fourth word of the OBDDMAP. The OBDDMAP begins at offset X'48' from the start of the DBD header.

The following figure illustrates these relationships and can aid in analysis.

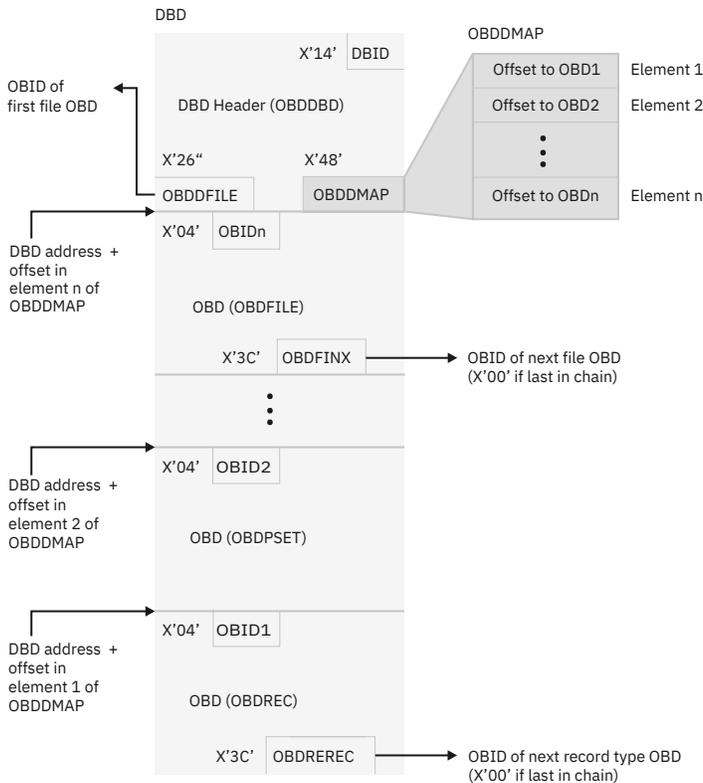


Figure 99. DBD and file OBD structure

The procedure that is provided explains how to go through all OBDs of all DBDs and how to analyze their structure. This process is extremely time consuming. When an error or inconsistency is found, you can choose to stop the analysis and assume that the problem is the only problem, or you can continue in an attempt to find other potential errors.

If you suspect that a certain OBID is involved in the problem, you can locate the corresponding OBD directly, and use parts of the procedure for analyzing a DBD (depending on what type of OBD is involved).

Begin with the first procedure, [“Checking the DBD header and chain of files”](#) on page 350, which explains how to start analyzing the structure of the DBD. This general procedure refers you to other more specific procedures that follow it.

The procedure for checking the structure of the DBD applies to DBDs for user-defined databases only. Assume that the structure of system-defined DBDs for Db2 areas such as the catalog are correct.

Locating an OBD (given the OBID of the OBD) involves using the OBID value as an index into OBDDMAP. When doing this task, mark the OBID to indicate that it is being used. When you search a chain, if an OBID is encountered that is already marked, the OBD is pointed to by more than one other OBD; that is, it is in two chains. If the OBD is not a relationship OBD between tables in the same database, this is an error (only relationship OBDs in the same database can appear in two chains).

Checking the DBD header and chain of files

After you locate the DBD in the dump, the DBD header and chain of files can be analyzed for inconsistencies.

Procedure

To check the DBD header and chain of files:

1. Check the DBD header.

The control block identifier (first 2 bytes of the DBD header) is X'2039'. At offset X'04' is the DBD eye-catcher.

2. Cross-check the DBD with the Db2 catalog.

Issue the following SQL statement, specifying the DBID for the DBD being checked.

PSPI

```
SELECT NAME FROM SYSIBM.SYSDATABASE
WHERE DBID=dbid;
```

The result from this SELECT statement should be one row, and the name should equal the name found in the field named OBDDBNAM. If the number of rows that are returned is not 1 or the name does not equal the name found in OBDDBNAM, make a note of the error.

3. Check the OBD file chain.

The OBDDFILE field in the OBDDBD contains the OBID of the first file OBD (OBDFILE). The OBDFINX field in the OBDFILE contains either 0 or the OBID of the next file OBD. (A zero indicates that the last file OBD in the chain.)

For each file OBD (indicated by an OBDDTYPE field that equals binary B'10') on the file chain, mark it and perform the check that is described under [“Checking the files” on page 350](#).

If the file chain is damaged (that is, the OBD located is not a file OBD), make a note of the error.

4. After you complete this procedure, no errors are found and if all OBIDs are marked, the structure of the DBD is probably consistent. If there are one or more unmarked OBIDs, then there is one or more errors.
5. If any errors are found when you analyze the structure of the DBD, contact the IBM support center and describe the analysis that was done.

Checking the files

After you locate the DBD in the dump, the files can be analyzed for inconsistencies.

Procedure

To check each file that is located by OBD:

1. Cross-check the file OBD with the catalog. To do this task, issue the following SQL statements, replacing dbid with the DBID of the DBD being checked and obid with the OBID of the file that is being checked.

PSPI

```
SELECT NAME, PSID FROM SYSIBM.SYSTABLESPACE
WHERE DBID=dbid
AND OBID=obid;
```

PSPI

The result is one row, and the name and PSID matches the OBDFNAME and OBDFIPS, in OBDFILE.

If the number of rows that are returned is not 1, or if the name and PSID do not match, make a note of the error.

2. Check the data page set of the file OBD. Use the OBID contained in the OBDFIPS field to locate, check, and mark the data page set OBD (OBDPSET). The value for OBDDTYPE is binary B'11'. To check the data page set OBD:
 - Determine whether the value of the OBDPJOB field matches the file OBID.
 - Cross-check the page set OBD with the catalog. Issue the following SQL statement, replacing obdfname with the value found in the OBDFNAME field of the file OBD, and replacing dbddbname with the database name of the DBD being checked (that is, the value found in the DBDDBNAM field of the DBD header).

GUPI

```
SELECT COUNT(*) FROM SYSIBM.SYSTABLEPART
WHERE TSNAME=obdfname
AND DBNAME=dbddbname;
```

GUPI

The result is 1 if the table space is a simple table space (that is, if OBDPJOB equals 0). If the table space is partitioned, the result is equal to the value of OBDPJOB (in OBDPS).

3. Check the OBD record chain within the file. The OBDFIREC field in the OBDFILE contains the OBID of the first record OBD (OBDREC). The OBDREREC field in OBDREC contains either a 0 or the OBID of the next record OBD. A 0 in OBDREREC indicates that it is the last record OBD.

Check the record chain for the file. Mark each record OBD (indicated by an OBDDTYPE field that equals binary B'01') in the record chain. Perform the check that is described under [“Checking the records”](#) on page 351.

If the record chain is damaged (that is, the OBD located is not a record OBD), make a note of the error.

4. Return to the procedure that referred you here.

Checking the records

After you locate the DBD in the dump, the records can be analyzed for inconsistencies.

Procedure

To check each record that is located by OBD:

1. If the record OBD is marked, make a note that it is a duplicate OBD (indicating an error) and return to the procedure that referred you here. If it is not marked, continue with the next step.
2. Determine whether the OBDREFIL field of the OBDREC matches the file OBID. If not, make a note of the error and continue.
3. Cross-check the OBD with the catalog. Issue the following SQL statement, replacing dbid with the DBID of the DBD being checked, and obid with the OBID of the record OBD being checked.

PSPI

```
SELECT NAME, CREATOR FROM SYSIBM.SYSTABLES
WHERE DBID=dbid
AND OBID=obid;
```

PSPI

The result is one row of output; if you receive more or less than one row, make a note of the error.

4. Issue the following SQL statement, replacing name with the NAME you received in the row of output from the previous SELECT, and creator with the CREATOR you received.

GUIP

```
SELECT COUNT(*) FROM SYSIBM.SYSCOLUMNS
WHERE TBNAME=name
AND TBCREATOR=creator;
```

GUIP

The result is equal to the value of the OBDRENFDF field of OBDREC; this field contains the number of columns in the table.

If the result is not as previously described, make a note of the error.

5. Check the index chain of the record. The OBDRECHI field of the OBDREC contains the OBID of the first index OBD (OBDFS). The OBDFSFSC field of the OBDFS contains either 0 or the OBID of the next index OBD. 0 indicates the last index OBD.

Check the index fan set chain. Mark each index fan set OBD (indicated by an OBDDTYPE field that equals B'00') defined on the record. Perform the check that is described under [“Checking the indexes”](#) on page 359.

If the index chain is damaged (that is, the OBD located is not an index type OBD), make a note of the error.

6. Check the referential integrity (RI) relationship chains of the record.
 - a) Parent record: The OBDRELPA field of the OBDREC contains the OBID of the first RI relationship OBD (OBDFS) in which the record is a parent. The OBDFSFSP field of the OBDFS contains either a 0 or the OBID of the next RI relationship OBD in which the record is a parent. A 0 indicates the last RI relationship OBD.

Be sure that the record that is a parent in an RI relationship has a primary key and a primary index. The OBDRDEF field in the OBDREC equals binary B'0' if a primary key and an index exist.

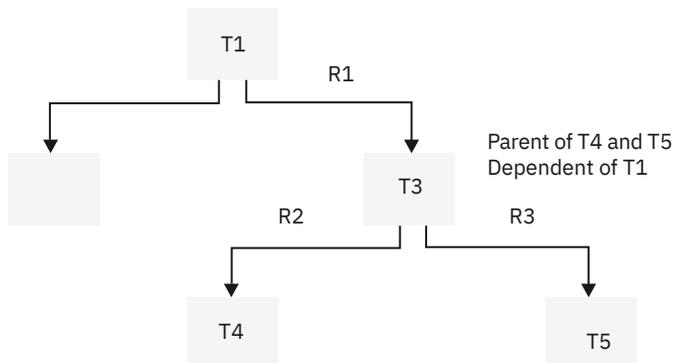
- b) Dependent record: The OBDRELCH field of the OBDREC contains the OBID of the first RI relationship OBD (OBDFS) in which the record is a child. The OBDFSFSC field of the OBDFS contains either a 0 or the OBID of the next relationship OBD in which the record is a child. A 0 indicates that it is the last RI relationship OBD.
7. Check the RI relationship fan set chains. Mark each RI relationship fan set OBD (indicated by an OBDDTYPE field that equals binary B'00' and an OBDFSDC field that equals binary B'1') defined on the record. Perform the checks that are described under [“Checking the referential integrity relationships”](#) on page 353

If the RI relationship chain is damaged (that is, the OBD located is not an RI relationship-type OBD), make a note of the error.

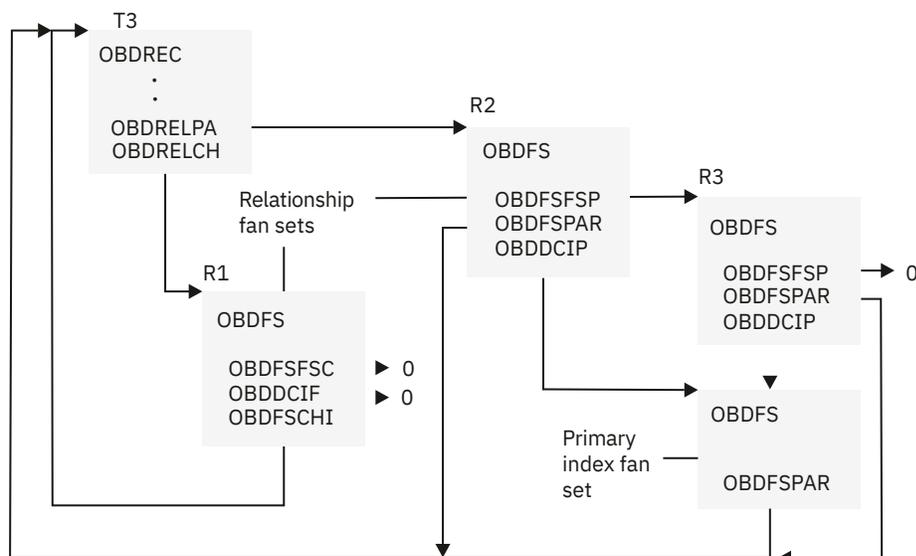
8. Check the auxiliary relationship chains. To do this task, examine the first and second extensions to the OBDREC. Find the first extension (OBDRX) by adding the offset in OBDREXT0 to the address of the OBDREC. Find the second extension (OBDRX2) by adding the offset in OBDREXT2 (which is in OBDRX) to the address of OBDRX. If OBDRXLOB (in OBDRX) equals B'1' and OBDRXILO (in OBDRX) equals B'0', then OBDRX2RA (in OBDRX2) contains the OBID of the first auxiliary relationship OBD (OBDRX2RA) for the table. The OBDANEXT field of the OBDRX2RA contains either 0 or the OBID of the next OBDRX2RA. Perform the checks that are described in [“Checking the auxiliary relationships”](#) on page 355. If the relationship chain is damaged (that is, the OBD located is not an auxiliary relationship-type OBD), make a note of the error.
9. Return to the procedure that referred you here.

Example

External representation:



Internal representation:



A parent record type OBD is incomplete if there is no primary index on the parent key.
A dependent record type OBD might or might not have an index on a foreign key.

Figure 100. Example of an RI relationship fan set chain

Checking the referential integrity relationships

After you locate the DBD in the dump, the referential integrity (RI) relationships can be analyzed for inconsistencies.

Procedure

To check the RI relationships:

1. “Checking the records” on page 351 illustrates the structure of relationship fan set chains. Refer to that figure to perform the following checks. These checks apply to informational referential constraints as well as enforced referential constraints.
 - a) For each relationship fan set OBD (OBDFS):
 - If OBDDCCHR is non-zero, then the dependent in the relationship resides in a different database than the parent. The DBID of the other database is in the OBDDCCHD field of this relationship fan set.

Make a note that when you check the relationship chains for database OBDDCCHD, you should ensure that the OBDDCPAR field contains the OBID of this relationship fan set, and that OBDDCPAD contains the DBID of this database.

- Check that OBDFSPAR is equal to the OBID of the parent record type OBD (OBDREC). The OBID of OBDREC is stored in OBDRECID.
- If OBDRDEF is equal to binary B'0' OBDREC, then OBDDCIP (in OBDFS) contains the OBID of the index fan set for the primary index. Verify the existence of that index fan set.

If OBDRDEF is equal to binary B'1' in OBDREC, then the definition of OBDREC is incomplete (there is no primary index on the primary key) and OBDDCIP is meaningless.

- Cross-check the OBD with the catalog. Issue the following SQL statement, replacing DBID with the DBID of the DBD being checked, and OBID with the OBID of the relationship.

PSPI

```
SELECT R.RELNAME, R.REFTBCREATOR, R.REFTBNAME
FROM SYSIBM.SYSRELS R,
     SYSIBM.SYSTABLES T
WHERE R.REFTBCREATOR = T.CREATOR
      AND R.REFTBNAME = T.NAME
      AND T.DBID = dbid
      AND R.RELOBID1 = obid
```

PSPI

This returns the name of the relationship with respect to the parent table.

b) Relationship chain in which record is a dependent

For each relationship fan set OBD (OBDFS):

- If OBDDCPAR is non-zero, then the parent in the relationship resides in a different database than the dependent. The dbid of the other database is in the OBDDCPAD field of this relationship fan set.

Make a note that when you check the relationship chains for database OBDDCPAD, you should ensure that the OBDDCCHR field contains the obid of this relationship fan set, and that OBDDCCHD contains the dbid of this database.

- Check that OBDFSCHI is equal to the OBID of the dependent record type OBD (OBDREC). The OBID of OBDREC is stored in OBDRECID.
- If OBDDCIF is not equal to zero, then it contains the OBID of the index fan set on the foreign key. Verify the existence of that index fan set.

If OBDDCIF is equal to zero, then there is no index on the foreign key.

- Cross-check the OBD with the catalog. Issue the following SQL statement, replacing dbid with the DBID of the DBD being checked, and OBID with the OBID of the relationship.

PSPI

```
SELECT R.RELNAME R.CREATOR, R.TBNAME
FROM SYSIBM.SYSRELS R,
     SYSIBM.SYSTABLES T
WHERE R.CREATOR = T.CREATOR
      AND R.TBNAME = T.NAME
      AND T.DBID = dbid
      AND R.RELOBID1 = obid
```

PSPI

This returns the name of the relationship with respect to the dependent table.

2. To check the number of foreign keys, issue the following SQL statement:

GUIP

```
SELECT COUNT(*) FROM SYSIBM.SYSFOREIGNKEYS
WHERE CREATOR = creator
AND RELNAME = relname
```

GUIP

To check the number of primary keys, issue:

GUIP

```
SELECT COUNT(*)
FROM SYSIBM.SYSINDEXES I,
     SYSIBM.SYSKEYS K
WHERE I.UNIQUERULE = 'P'
AND I.TBCREATOR = reftbcreator
AND I.TBNAME = reftbname
AND I.CREATOR = K.IXCREATOR
```

GUIP

3. Return to the procedure that referred you here.

Checking the auxiliary relationships

After you locate the DBD in the dump, the auxiliary relationships can be analyzed for inconsistencies.

Procedure

For each auxiliary relationship OBD (OBDRA):

1. Check that the OBD located is an OBDRA. For an OBDRA, the first 3 bits of OBDFLAGS are B'010' and OBDATYP3 is B'1'. If these fields do not contain the correct values, return to the procedure that referred you here.
2. Check that the OBDABT contains the OBID of the record OBD for the base table. If the base table space is partitioned, run the following SQL statement, replacing dbname with the name of the database and tsname with the name of the base table space.

GUIP

```
SELECT COUNT(*) FROM SYSIBM.SYSTABLEPART WHERE
DBNAME = 'dbname' AND TSNAME = 'tsname';
```

GUIP

The result is equal to the value in field OBDAPARTN of OBDRA. This field contains the number of partitions in the base table space.

3. If the base table space is partitioned, there is an OBDAPART section for each partition. Element *i* in the OBDAPART array corresponds to partition *i* of the base table space. If the base table space is not partitioned, there is a single OBDAPART section.

Each OBDAPART section contains:

- a. OBDAAT (OBID of the OBDREC for the auxiliary table)
 - b. OBDAPS (OBID of the OBDPS for the LOB table space)
 - c. OBDAFS (OBID of the OBDFS for the auxiliary index)
 - d. OBDAFI (OBID of the OBDFILE for the LOB table space)
 - e. OBDAIPS (OBID of the OBDPS for the auxiliary index)
4. Use the following queries to cross-check the values in the OBDAPART fields with the catalog.
 - Replace column with the value in field OBDAACOLNO (the column number) in OBDRA.
 - Replace obid with the value in OBDAAT.

- Replace obid1, obid2, ... obidn with the values in OBDAAT.
- To cross-check the value in OBDAAT with the catalog, run one of the following SQL statements. In these SQL statements, make the following changes:
 - Replace column with the value in field OBDACOLNO (the column number) in OBDRA.
 - Replace obid with the value in OBDAAT.
 - Replace obid1, obid2, ... obidn with the values in OBDAAT.
 - For a nonpartitioned base table space, run this statement:

PSPI

```
SELECT A.AUXTBNAME, A.COLNAME, A.TBNAME FROM
  SYSIBM.SYSCOLUMNS C,
  SYSIBM.SYSAUXRELS A,
  SYSIBM.SYSTABLES T
WHERE C.NAME = A.COLNAME
      AND C.COLNO = 'column'
      AND T.NAME = A.AUXTBNAME
      AND T.OBID = 'obid';
```

PSPI

The value of A.AUXTBNAME should be the name of the auxiliary table that is created for column A.COLNAME in base table A.TBNAME.

- For a partitioned base table space, run the following query:

PSPI

```
SELECT A.AUXTBNAME, A.COLNAME, A.TBNAME FROM
  SYSIBM.SYSCOLUMNS C,
  SYSIBM.SYSAUXRELS A,
  SYSIBM.SYSTABLES T
WHERE C.NAME = A.COLNAME
      AND C.COLNO = 'column'
      AND T.NAME = A.AUXTBNAME
      AND T.OBID IN (obid1, obid2, ..., obidn);
```

PSPI

The values of A.AUXTBNAME should be the names of the auxiliary tables that are created for column A.COLNAME in base table A.TBNAME.

- To cross-check the value in OBDAPS with the catalog, run one of the following SQL statements. In these SQL statements, make the following changes:
 - Replace column with the value in field OBDACOLNO (the column number) in OBDRA.
 - Replace obid with the value in OBDAPS.
 - Replace obid1, obid2, ... obidn with the values in OBDAPS.
 - For a nonpartitioned base table space, run this SQL statement:

PSPI

```
SELECT S.NAME, A.COLNAME, A.TBNAME FROM
  SYSIBM.SYSCOLUMNS C,
  SYSIBM.SYSAUXRELS A,
  SYSIBM.SYSTABLES T,
  SYSIBM.SYSTABLESPACE S
WHERE C.NAME = A.COLNAME
      AND C.COLNO = column
      AND T.NAME = A.AUXTBNAME
      AND T.TSNAME = S.NAME
      AND S.PSID = obid;
```

PSPI

The value of S.NAME should be the name of the LOB table space that is associated with A.COLNAME in base table A.TBNAME.

- For a partitioned base table space, run this SQL statement:

PSPI

```
SELECT S.NAME, A.COLNAME, A.TBNAME FROM
  SYSIBM.SYSCOLUMNS C,
  SYSIBM.SYSAUXRELS A,
  SYSIBM.SYSTABLES T,
  SYSIBM.SYSTABLESPACE S
WHERE C.NAME = A.COLNAME
      AND C.COLNO = column
      AND T.NAME = A.AUXTBNAME
      AND T.TSNAME = S.NAME
      AND S.PSID IN
      (obid1, obid2, ... , obidn);
```

PSPI

The values of S.NAME should be the names of the LOB table spaces that are associated with A.COLNAME in base table A.TBNAME.

- To cross-check the value in OBDAFS with the catalog, run the following SQL statements. In these SQL statements, make the following changes:
 - Replace column with the value in field OBDAFNO (the column number) in OBDAF.
 - Replace obid with the value in OBDAFS.
 - Replace obid1, obid2, ... obidn with the values in OBDAFS.
 - For a nonpartitioned base table space, run this SQL statement:

PSPI

```
SELECT I.NAME, A.AUXTBNAME,
  A.COLNAME, A.TBNAME FROM
  SYSIBM.SYSCOLUMNS C,
  SYSIBM.SYSAUXRELS A,
  SYSIBM.SYSTABLES T,
  SYSIBM.SYSINDEXES I
WHERE C.NAME = A.COLNAME
      AND C.COLNO = column
      AND T.NAME = A.AUXTBNAME
      AND T.NAME = I.TBNAME
      AND I.OBID = obid;
```

PSPI

The value of I.NAME should be the name of the auxiliary index on table A.AUXTBNAME associated with A.COLNAME in base table A.TBNAME.

- For a partitioned base table space, run this SQL statement:

PSPI

```
SELECT I.NAME, A.AUXTBNAME,
  A.COLNAME, A.TBNAME FROM
  SYSIBM.SYSCOLUMNS C,
  SYSIBM.SYSAUXRELS A,
  SYSIBM.SYSTABLES T,
  SYSIBM.SYSINDEXES I
WHERE C.NAME = A.COLNAME
      AND C.COLNO = column
      AND T.NAME = A.AUXTBNAME
      AND T.NAME = I.TBNAME
      AND I.OBID IN
      (obid1, obid2, ... , obidn);
```

PSPI

I.NAME should be the names of the auxiliary indexes on tables A.AUXTBNAME and associated with A.COLNAME in base table A.TBNAME.

- To cross-check the value in OBDAFI with the catalog, run one of the following SQL statements. In these SQL statements, make the following changes:
 - Replace column with the value in field OBDA COLNO (the column number) in OBDRA.
 - Replace obid with the value in OBDAFI.
 - Replace obid1, obid2, ... obidn with the values in OBDAFI.
 - For a nonpartitioned base table space, run this SQL statement:

PSPI

```
SELECT S.NAME, A.COLNAME, A.TBNAME FROM
  SYSIBM.SYSCOLUMNS C,
  SYSIBM.SYSAUXRELS A,
  SYSIBM.SYSTABLES T,
  SYSIBM.SYSTABLESPACE S
WHERE C.NAME = A.COLNAME
      AND C.COLNO = column
      AND T.NAME = A.AUXTBNAME
      AND T.TSNAME = S.NAME
      AND S.OBID = obid;
```

PSPI

The value of S.NAME should be the name of the LOB table space that is associated with A.COLNAME in base table A.TBNAME.

- For a partitioned table space, run this SQL statement:

PSPI

```
SELECT S.NAME, A.COLNAME, A.TBNAME FROM
  SYSIBM.SYSCOLUMNS C,
  SYSIBM.SYSAUXRELS A,
  SYSIBM.SYSTABLES T,
  SYSIBM.SYSTABLESPACE S
WHERE C.NAME = A.COLNAME
      AND C.COLNO = column
      AND T.NAME = A.AUXTBNAME
      AND T.TSNAME = S.NAME
      AND S.OBID IN
      (obid1, obid2, ... , obidn);
```

PSPI

The values of S.NAME should be the names of the LOB table spaces that are associated with A.COLNAME in base table A.TBNAME.

- To cross-check the value in OBDAIPS with the catalog, run the following SQL statements. In these SQL statements, make the following changes:
 - Replace column with the value in field OBDA COLNO (the column number) in OBDRA.
 - Replace obid with the value in OBDAIPS.
 - Replace obid1, obid2, ... obidn with the values in OBDAIPS.
 - For a nonpartitioned base table space, run this SQL statement:

PSPI

```
SELECT A.COLNAME, I.NAME, A.AUXTBNAME, T.NAME FROM
  SYSIBM.SYSCOLUMNS C,
  SYSIBM.SYSAUXRELS A,
  SYSIBM.SYSTABLES T,
  SYSIBM.SYSINDEXES I
WHERE C.NAME = A.COLNAME
      AND C.COLNO = column
```

```
AND T.NAME = A.AUXTBNAME
AND T.NAME = I.TBNAME
AND I.ISOBID = obid;
```

PSPI

The value of I.NAME should be the name of the auxiliary index on table A.AUXTBNAME associated with A.COLNAME in base table T.NAME.

- For a partitioned base table space, run this SQL statement:

PSPI

```
SELECT I.NAME, A.AUXTBNAME, A.COLNAME, A.TBNAME FROM
  SYSIBM.SYSCOLUMNS C,
  SYSIBM.SYSAUXRELS A,
  SYSIBM.SYSTABLES T,
  SYSIBM.SYSINDEXES I
WHERE C.NAME = A.COLNAME
      AND C.COLNO = column
      AND T.NAME = A.AUXTBNAME
      AND T.NAME = I.TBNAME
      AND I.ISOBID IN (obid1, obid2, ... , obidn);
```

PSPI

The values of I.NAME should be the names of the auxiliary indexes on tables A.AUXTBNAME associated with A.COLNAME in base table A.TBNAME.

Checking the indexes

After you locate the DBD in the dump, the indexes can be analyzed for inconsistencies.

Procedure

To check each index fan set located by ODB:

1. If the index OBD is marked, make a note that it is a duplicate OBD (indicates an error) and return to the procedure that referred you here. If it is not marked, continue with the next step.
2. Determine whether the OBDFSCHI field of the OBDFS matches the record OBID. If not, make a note of the error and continue.
3. Cross-check the OBDFS in the DBD with the Db2 catalog. Issue the following SQL statement, replacing dbid with the DBID of the DBD being checked, and obid with the OBID of the index fan set.

PSPI

```
SELECT NAME, CREATOR FROM SYSIBM.SYSINDEXES
WHERE DBID=dbid
      AND OBID=obid;
```

PSPI

The result is one row is returned. If more or less than one row is returned, make a note of the error.

4. Issue the following SQL statement, using the NAME you received from the previous SELECT and the CREATOR you received.

GUIP

```
SELECT COUNT(*) FROM SYSIBM.SYSKEYS
WHERE IXNAME=name
      AND IXCREATOR=creator;
```

GUIP

The result is equal to the value of OBDKNFDO in OBDIFS; this field contains the number of columns in the key. If the result is not as described, make a note of the error.

5. Use the OBID contained in the OBDIPSET field to locate, check, and mark the index page set OBD. The value for OBDDTYPE is binary B'11'. To check the index page set OBD:

- Determine whether the OBDPSOB field matches the index OBID.
- Compare the value of the OBDPSET field in the DBD to values in the SYSIBM.SYSINDEXPART catalog table. Issue the following SQL statement, replacing the NAME returned in step “3” on page 359 and the CREATOR returned in step “3” on page 359.

GUIP

```
SELECT COUNT(*) FROM SYSIBM.SYSINDEXPART
WHERE IXNAME=name
AND IXCREATOR=creator;
```

GUIP

The result is equal the value of OBDPSANM in OBDPS if the index is a clustering index for a partitioned table space. If the index is not a clustering index for a partitioned table space, the result is 1.

If the results are not as described, make a note of the error.

- Return to the procedure that referred you here.

Resolve the inconsistent DBD

You can contact IBM Support after you narrow down the probable cause.

After you follow the procedures to resolve the inconsistent DBD and compare the SVC dump of the DBD (the DBD as it appears in memory) to the dump produced by REPAIR, DSN1COPY or DSN1PRNT (the DBD as it appears in permanent storage), probable cause of the inconsistent DBD is narrowed down.

Next, contact IBM Support and describe the analysis that was done. Do not attempt to resolve the problem with the REPAIR utility. IBM Support requires the dumps that you worked with to resolve the inconsistent DBD.

Chapter 7. Trace messages and codes

The DSNWEIDS file is shipped with Db2. It provides brief descriptions of trace codes that the global trace facility issues. The TSO attachment facility, call attach facility, and SPUFI issue their own trace messages.

Db2 trace codes

A readable z/OS data set is shipped with Db2, which describes the Db2 trace codes that are issued by the global trace facility. These trace codes are known as *event identifiers* (EIDs). The data set is called *prefix.SDSNSAMP(DSNWEIDS)*.

Use the TSO or ISPF browse function to look at these descriptions online, even when Db2 is down. To look at the descriptions in printed form, use ISPF to print a listing of the data set.

Alternatively, the data set can be loaded into a user-defined Db2 table. This approach has the advantage of providing access to the EID descriptions through SQL SELECT statements in whatever order or format required. SQL can also be used to tailor the information before you print it.

Maintaining EID descriptions

Periodically, updated data is included on Db2 corrective service tapes. These tapes are shipped with the ++HOLD statement, with instructions to replace the original data set with the updated data set. In this way, the EID descriptions are kept current with Db2 code.

Loading EID descriptions into a table

You can load EID descriptions through SQL SELECT statements in whatever order or format that is required.

About this task

Procedure

To load EID descriptions into a table:

1. Run the following CREATE statements, under SYSADM authority. The SQL can be changed to place the tables in a different table space.

```
CREATE TABLESPACE EIDTS
  IN DSN8D13A
  USING STOGROUP DSN8G130;
CREATE TABLE DSN8D10.EID_DESCRIPTIONS
  (RMID          CHAR(6) NOT NULL,
   EID           CHAR(9),
   CSECT        CHAR(8),
   DESCRIPTION   CHAR(29),
   TYPE         CHAR,
   TRACEITEMS   CHAR(14),
   SEQ          INTEGER NOT NULL)
  IN DSN8D13A.EIDTS;
```

2. Use the following LOAD command to load the table with the data in *prefix.SDSNSAMP(DSNWEIDS)*. Change the JCL on the LOAD command to suit your environment. Make sure that the INDDN parameter identifies a DD statement that points to the data as it is shipped with Db2.

```
LOAD DATA INDDN(SYSREC) LOG(NO)
  INTO TABLE DSN8D10.EID_DESCRIPTIONS
  (RMID          POSITION(1:6)  CHAR(6),
   EID           POSITION(8:16) CHAR(9),
   CSECT        POSITION(18:25) CHAR(8),
   DESCRIPTION   POSITION(27:55) CHAR(29),
   TYPE         POSITION(57)   CHAR,
```

```
TRACEITEMS    POSITION(59:72) CHAR(14),
SEQ           POSITION(74:80) INTEGER EXTERNAL(6)
```

3. After the table is loaded, run the following command to start the database:

```
-START DATABASE (DSN8D13A) SPACENAM (EIDTS) ACCESS (RW)
```

Retrieving EID descriptions

SQL can be used to retrieve the EID descriptions in whatever format or order that is required. The SQL statements can be run as would any other SQL (for example, through SPUFI, QMF, or DSNTEP2).

About this task

PSPI

Some sample SQL statements follow that might be useful in analyzing the Db2 global trace. Consider printing a formatted report of the results from the first query as a reference. Be aware, however, that this information is licensed. The terms and conditions of the licensing agreement, as stated at the beginning of the file, apply to any hardcopy that is generated.

Procedure

- To retrieve all EID descriptions, ordered numerically by RMID:

```
SELECT DISTINCT *
  FROM DSN8D10.EID_DESCRIPTIONS
 ORDER BY SEQ;
```

- To retrieve only the EID descriptions for RMID 04-04:

```
SELECT DISTINCT *
  FROM DSN8D10.EID_DESCRIPTIONS
 WHERE RMID = 'AGREE' OR RMID = '04-04'
 ORDER BY SEQ;
```

- To retrieve only the EID description for EID 10-0A:

```
SELECT DISTINCT *
  FROM DSN8D10.EID_DESCRIPTIONS
 WHERE RMID = 'AGREE' OR EID = '10-0A'
 ORDER BY SEQ;
```

- To retrieve only the EID descriptions for CSECT DSNICFLD:

```
SELECT DISTINCT *
  FROM DSN8D10.EID_DESCRIPTIONS
 WHERE RMID = 'AGREE' OR CSECT = 'DSNICFLD'
 ORDER BY SEQ;
```

- To retrieve only the EID description for EID 3376-0D30 in RMID 21-15:

```
SELECT DISTINCT *
  FROM DSN8D10.EID_DESCRIPTIONS
 WHERE EID = 'AGREE' OR
 (RMID = '21-15' AND EID = '3376-0D30')
 ORDER BY SEQ;
```

PSPI

TSO attachment facility trace messages

Some TSO trace messages have the same message number. They are listed in order of the CSECTs that issue them.

PSP1

When a message is printed, it is preceded by the name of the issuing CSECT. To find a particular message, look for the issuing CSECT shown in an " eye-catcher" box before each block of messages that is issued by one CSECT. Then, look for the message within that group that has the same text as the message you received.

This trace facility is primarily intended for use either by IBM personnel or by a system programmer that works with IBM Support.

The following table shows the beginning page number for each CSECT.

CSECT: DSNECP00

| Message text | Explanation |
|--|--|
| ENTER DSNECP00 R1 <i>token1 token2</i> | CSECT DSNECP00 has been entered from TSO. The first hexadecimal token is register 1. The second hexadecimal token is not used. |
| AFTER STAX R6,R15 <i>token1 token2</i> | The STAX macro has been issued. The first hexadecimal token is register 6, which contains the pointer to the connection information block (CIB). The second hexadecimal token is register 15, the return code from the STAX macro. |
| MAIN LOOP TOP CIBP00C,CIBSTIP1 <i>token1 token2</i> | This is the start of the main loop in CSECT DSNECP00. The first hexadecimal token is CIBP00C, the loop control variable. The second hexadecimal token is CIBSTIP1, the user attention indication variable. |
| BEFORE ATTACH R2,R3 <i>token1 token2</i> | Load module DSNECP00 is about to attach load module DSNECP10. The first hexadecimal token is register 2, a pointer to DSNE0CIB. The second hexadecimal token is register 3, the ATTACH ECB. |
| BEFORE WAIT CIBEXECB,CIBSBECB <i>token1 token2</i> | The WAIT macro is about to be issued. The first hexadecimal token is CIBEXECB, the EXIT ECB. The second hexadecimal token is CIBSBECB, the ATTACH ECB. |

| Message text | Explanation |
|---|---|
| WAIT WAS JUST POSTED CIBEXECB,CIBSBECB token1 token2 | <p>The WAIT was just posted by either the Db2 (the EXIT ECB) or by DSNECP10, the DSN subtask.</p> <p>The first hexadecimal token is CIBEXECB, the EXIT ECB.</p> <p>The second hexadecimal token is CIBSBECB, the ATTACH ECB.</p> |
| DSNECP10 WAS POSTED CIBSBECB,CIBSTIP1 token1 token2 | <p>DSNECP10 has completed.</p> <p>The first hexadecimal token is CIBSBECB, the DSNECP10 ECB.</p> <p>The second hexadecimal token is CIBSTIP1, the user attention indicator variable.</p> |
| BIG IF BOTTOM CIBEXECB,CIBSBECB token1 token2 | <p>Execution is at the bottom of the main IF statement in the CSECT.</p> <p>The first hexadecimal token is CIBEXECB, the EXIT ECB.</p> <p>The second hexadecimal token is CIBSBECB, the DSNECP10 subtask ECB.</p> |
| MAIN LOOP BOTTOM CIBP00C,CIBSTIP1 token1 token2 | <p>The program is at the bottom of the main loop.</p> <p>The first hexadecimal token is CIBP00C, the loop control variable. The second hexadecimal token is CIBSTIP1, the attention indicator variable.</p> |
| EXIT DSNECP00 CIBEXECB,CIBSBECB token1 token2 | <p>The DSN command processor is terminating. Control will return to TSO.</p> <p>The first hexadecimal token is CIBEXECB, the EXIT ECB.</p> <p>The second hexadecimal token is CIBSBECB, the DSNECP10 subtask ECB.</p> |
| ENTER ZSTOMP CIBP00C,CIBMAXRC token1 token2 | <p>Control is entering the subroutine that will DETACH DSNECP10.</p> <p>The first hexadecimal token is CIBP00C, the main-loop control variable.</p> <p>The second hexadecimal token is CIBMAXRC, the maximum return code that has occurred so far in DSN processing.</p> |
| AFTER CS R2,CIBSBTCB token1 token2 | <p>The compare and swap (CS), that checks the TCB before detaching it, has just occurred. The two tokens should be identical for the CS to occur.</p> <p>The first hexadecimal token is the contents of register 2.</p> <p>The second hexadecimal token is CIBSBTCB, the TCB address.</p> |

| Message text | Explanation |
|--|---|
| BEFORE DETACH ZTCBPTR,CIBSTIP1 token1 token2 | <p>The DSNECP00 task is about to DETACH the DSNECP10 subtask.</p> <p>The first hexadecimal token is ZTCBPTR, the address of the TCB to be detached.</p> <p>The second hexadecimal token is CIBSTIP1, the attention indicator variable.</p> |
| AFTER DETACH ZTCBPTR,CIBSTIP1 token1 token2 | <p>DETACH has completed.</p> <p>The first hexadecimal token is ZTCBPTR, the address of the TCB detached.</p> <p>The second hexadecimal token is CIBSTIP1, the attention indicator variable.</p> |
| CIBPUTGT WAS ON AND USER PRESSED ATTENTION token1 token2 | <p>CIBPUTGT being on means that the DSN initialization processing is complete and that normal attention processing is enabled.</p> <p>The first hexadecimal token is ZTCBPTR, the address of the TCB that was processing at the time of attention.</p> <p>The second hexadecimal token is CIBSTIP1, which indicates whether the user pressed attention.</p> |
| ENTER DSNECP00 ESTAI ROUTINE, R0=12 token1 token2 | <p>The ESTAI was entered with no SDWA.</p> <p>The first hexadecimal token is unpredictable.</p> <p>The second hexadecimal token is the attention indicator variable, CIBSTIP1.</p> |
| CIBSTIP1=ATTN token1 token2 | <p>The user pressed the attention key.</p> <p>The hexadecimal tokens are not used.</p> |
| ENTER DSNECP00 ESTAI ROUTINE, ABEND CODE, REASON CD token1 token2 | <p>The ESTAI routine in DSNECP00 has been entered with an SDWA.</p> <p>The first hexadecimal token is the ABEND code.</p> <p>The second hexadecimal token is the reason code.</p> |
| ABEND ADDRESS FROM PSW token1 token2 | <p>An SDWA was available and the abend address is in token one.</p> <p>The first hexadecimal token is SDWAPMKA.</p> <p>The second hexadecimal token is not used.</p> |
| CIBSTIP1=ATTN token1 token2 | <p>The user pressed the attention key.</p> <p>The hexadecimal tokens are not used.</p> |

CSECT: DSNECP01

| Message text | Explanation |
|---|--|
| ENTER DSNECP01, R6= <i>token1 token2</i> | Attention routine has been entered. The first hexadecimal token is unpredictable. The second hexadecimal token is register 6, the address of the connection information block (CIB). |
| "No Text" | There is no text after the leading blanks in the input buffer. The first hexadecimal token is ZI, the offset into the buffer. The second hexadecimal token is the message length. |
| <i>firstchar token1 token2</i> | The message text (<i>firstchar</i>) contains the first character in the input buffer. The first hexadecimal token contains the buffer index to ' <i>firstchar</i> '. The second hexadecimal token contains the size of the buffer. |
| ZINPUT= <i>token1 token2</i> | Result of scanning the input buffer has been determined. The first hexadecimal token is not used. The second hexadecimal token contains the character to be used for cancel/resume decision. |
| NO C INPUT, MXRPLEN= <i>token1 token2</i> | No C for cancel was found. The first hexadecimal token contains the scan limit. The second hexadecimal token is not used. |
| LEAVE DSNECP01 <i>token1 token2</i> | DSNECP01 is about to end. The first hexadecimal token is the contents of register 15. Register 15 contains the results of the FREEMAIN that just occurred. The second hexadecimal token is unpredictable. |
| NO PUTGET MESSAGE, R15= <i>token1 token2</i> | The PUTGET failed. The first hexadecimal token contains register 15 after PUTGET. The second hexadecimal token is not used. |
| AFTER COMPARE/SWAP, R2, ZTCBPTR <i>token1 token2</i> | DSNECP01 is serializing DETACH of subtask. The first hexadecimal token contains register 2. The second hexadecimal token contains the TCB pointer of the task to be detached. |

| Message text | Explanation |
|---|--|
| BEFORE DETACH, ZTCBPTR = token1 token2 | <p>DSNECP01 is about to DETACH subtask.</p> <p>The first hexadecimal token contains the address of the TCB to be detached.</p> <p>The second hexadecimal token is unpredictable.</p> |
| DSNECP01 AFTER DETACH, R15= | <p>The DETACH has been performed.</p> <p>The first hexadecimal token contains the return code from DETACH.</p> <p>The second hexadecimal token is unpredictable.</p> |
| CSECT: DSNECP10 | |
| Message text | Explanation |
| ENTER DSNECP10, R6 token1 token2 | <p>CSECT DSNECP10 has been entered.</p> <p>The first hexadecimal token contains register 6, the address of the connection information block (CIB).</p> <p>The second hexadecimal token is unpredictable.</p> |
| AFTER DSNECP20 token1 token2 | <p>CSECT DSNECP20 has finished initializing control blocks.</p> <p>The hexadecimal tokens are not used.</p> |
| CIBSCANC: subcmd token1 token2 | <p>subcmd is the current subcommand being processed.</p> <p>The first hexadecimal token is CIBMAXRC, the maximum return code so far.</p> <p>The second hexadecimal token is CIB10RTC, an internal return code.</p> |
| BEGIN DSN COMMAND token1 token2 | <p>User has entered DSN as a subcommand.</p> <p>The first hexadecimal token is the contents of CIBSESS, a session control variable.</p> <p>The second hexadecimal token is the contents of CIB10RTC, which is an internal return code.</p> |
| CIBMAXRC,CIB10RCT token1 token2 | <p>Execution is just before the check for the new maximum return code.</p> <p>The first hexadecimal token contains CIBMAXRC, the internal maximum return code.</p> <p>The second hexadecimal token contains CIB10RCT, which is the current subcommand's return code.</p> |

| Message text | Explanation |
|---|---|
| EXIT DSNECP10, MAXRC, CIB10RTC <i>token1 token2</i> | <p>DSNECP10 is about to return to DSNECP00.</p> <p>The first hexadecimal token contains CIBMAXRC, the internal maximum return code.</p> <p>The second hexadecimal token contains CIB10RTC, the return code from the current subcommand.</p> |
| ENTER DSNECP10 ESTAERTN, R0, R6 <i>token1 token2</i> | <p>The ESTAE routine for DSNECP10 has been entered.</p> <p>The first hexadecimal token contains register 0. If it is 12, then no SDWA was obtained.</p> <p>The second hexadecimal token contains register 6, the address of the connection information block (CIB).</p> |
| ADDR(DSNECP10),PSW ADDR <i>token1 token2</i> | <p>An SDWA was obtained.</p> <p>The first hexadecimal token is the current address of DSNECP10.</p> <p>The second hexadecimal token contains SDWANXT1.</p> |
| USER ATTENTION RECOGNIZED, R0, R6 <i>token1 token2</i> | <p>The ABEND was caused by user attention; dumping will be suppressed.</p> <p>The first hexadecimal token is the contents of register 0.</p> <p>The second hexadecimal token is register 6, the address of the connection information block (CIB).</p> |
| CIBMAXRC <i>token1 token2</i> | <p>The ESTAE routine is about to return to ABTERM (RTM) processing.</p> <p>The first hexadecimal token contains ZMAXRC, the maximum return code.</p> <p>The second hexadecimal token is not used.</p> |
| END DSNECP10 ESTAERTN, MAXRC, R15 <i>token1 token2</i> | <p>Control is now leaving the ESTAE routine.</p> <p>The first hexadecimal token contains MAXRC.</p> <p>The second hexadecimal token is the contents of register 15.</p> |
| CSECT: DSNECP12 | |
| Message text | Explanation |
| ENTER DSNECP12, CIBRTRY=, CIBIDFRB= <i>token1 token2</i> | <p>CSECT DSNECP12 has been entered.</p> <p>The first hexadecimal token is CIBRRTY, the retry parm from the DSN command.</p> <p>The second hexadecimal token is CIBIDFRB, the Identify FRB address.</p> |

| Message text | Explanation |
|--|---|
| CIBCORID=cibcorid CIBMODE=token1 token2 | <p>This message shows the correlation identifier and the mode (foreground or background).</p> <p>The first hexadecimal token contains CIBMODE.</p> <p>The second hexadecimal token is zero.</p> |
| BEFORE IDENTIFY DB2 CALL =====, token1 token2 | <p>CSECT DSNECP12 is about to establish the Identify level connection with Db2. Control is now passing to Db2.</p> <p>The first hexadecimal token contains FRBRC1, which is not used at this time.</p> <p>The second hexadecimal token contains FRBRC2, which is not used at this time.</p> |
| AFTER IDENTIFY DB2 CALL =====, token1 token2 | <p>Control has just returned from the identify call.</p> <p>The first hexadecimal token contains FRBRC1, the return code from the identify.</p> <p>The second hexadecimal token contains FRBRC2, the reason code from the identify.</p> |
| BEFORE ESTABLISH-EXIT DB2 CALL ===== token1 token2 | <p>DSN is about to hand control to Db2 for the Establish EXIT call.</p> <p>The first hexadecimal token is FRBRC1.</p> <p>The second hexadecimal token is FRBRC2.</p> |
| AFTER ESTABLISH-EXIT DB2 CALL ===== token1 token2 | <p>Control has now returned from Db2.</p> <p>The first hexadecimal token is FRBRC1.</p> <p>The second hexadecimal token is FRBRC2.</p> |
| EXIT DSNECP12, FRBRC1=, FRBRC2= token1 token2 | <p>The identify attempt is complete.</p> <p>The first hexadecimal token contains FRBRC1, preceded by two bytes of zeros.</p> <p>The second hexadecimal token is FRBRC2.</p> |
| FRB FIELD CONTENTS FOLLOW (CIBIDFRB) token1 token2 | <p>The FRB contents are about to be written.</p> <p>The first hexadecimal token contains the address of the identify FRB.</p> <p>The second hexadecimal token is not used.</p> |
| FRBRAL(PTR), FRBRALE(BIN15), FRBFVLE(BIN15) token1 token2 | <p>More of FRB.</p> <p>The first hexadecimal token contains FRBRAL.</p> <p>The second hexadecimal token contains FRBFVLE and FRBFVLE.</p> |

| Message text | Explanation |
|---|---|
| FRBPARM(PTR), FRBPCNT(BIN15) token1 token2 | <p>More of FRB.</p> <p>The first hexadecimal token is the FRBPARM pointer.</p> <p>The second hexadecimal token is the FRBPCNT.</p> |
| FRBRC1(BIN15), FRBRC2(CHAR4) token1 token2 | <p>More of FRB.</p> <p>The first hexadecimal token is FRBRC1.</p> <p>The second hexadecimal token is FRBRC2.</p> |
| FRBFBACK(PTR), FRBRHPC(BIN32) token1 token2 | <p>More of FRB.</p> <p>The first hexadecimal token is the FRBFBACK pointer.</p> <p>The second hexadecimal token is the FRBRHPC.</p> |
| FRBQUAL(BIN15), FRBRSV1(BIN15) token1 token2 | <p>More of FRB.</p> <p>The first hexadecimal token is FRB QUAL.</p> <p>The second hexadecimal token is FRBRSV1.</p> |
| SSID- cibssid ,CONID- cibconid token1 token2 | <p>The IDENTIFY was done for Db2 'ssid' and connection ID 'conid'.</p> <p>The first hexadecimal token contains ZSSID.</p> <p>The second hexadecimal token contains ZCONID.</p> |
| CONTY- cibconty ,ZFRB token1 token2 | <p>The connection type used in this IDENTIFY was 'conty'.</p> <p>The first hexadecimal token contains ZCONTY.</p> <p>The second hexadecimal token contains ZFRB.</p> |
| CSECT: DSNECP13 | |
| Message text | Explanation |
| ENTER DSNECP13, R6=, CIBRFRB= token1 token2 | <p>CSECT DSNECP13 (create thread) has been entered.</p> <p>The first hexadecimal token is the connection information block (CIB) pointer.</p> <p>The second hexadecimal token is CIBRFRB, the FRB pointer.</p> |
| BEFORE CREATE THREAD DB2 CALL =====R6,CIBCTFRB token1 token2 | <p>DSNECP13 is about to complete the connection to Db2 with a create thread call to the PRH.</p> <p>The first hexadecimal token is the address of the connection information block (CIB).</p> <p>The second hexadecimal token is the FRB address.</p> |

| Message text | Explanation |
|--|--|
| AFTER CREATE THREAD DB2 CALL=====R6,CIBCTFRB <i>token1 token2</i> | Control has just returned from Db2. The first hexadecimal token contains register 6, the address of the connection information block (CIB). The second hexadecimal token is CIRRFRB, the FRB address. |
| R6=, CIBCTR TN= <i>token1 token2</i> | Control is about to leave DSNECP13. The first hexadecimal token is register 6, the address of the connection information block (CIB). The second hexadecimal token is CIBCTR TN, a create thread return code. |
| EXIT DSNECP13 <i>token1 token2</i> | Now leaving CSECT DSNECP13 The first hexadecimal token is not used. The second hexadecimal token is the address in CIBRFRB. |
| FOLLOWING ARE FRB FIELD CONTENTS, CIBCTFRB= <i>token1 token2</i> | This and following messages are a dump of the FRB pointed to by CIBCTFRB. These messages precede and follow the CREATE THREAD call. The first hexadecimal token contains CIBCTFRB. The second hexadecimal token is not used. |
| FRBRAL(PTR), FRBRALE(BIN15), FRBFVLE(BIN15) <i>token1 token2</i> | More of FRB. The first hexadecimal token is FRBRAL. The second hexadecimal token contains FRBRALE, the resource access list entry and FRBFVLE, the function vector list entry. |
| FRBPARM(PTR), FRBPCNT(BIN15) <i>token1 token2</i> | More of FRB. The first hexadecimal token is FRBPARM. The second hexadecimal token is FRBPCNT. |
| FRBRC1(BIN15), FRBRC2(CHAR4) <i>token1 token2</i> | More of FRB. The first hexadecimal token is the FRBRC1. The second hexadecimal token is the FRBRC2. |
| FRBKNAME: fback <i>token1 token2</i> | This message lists the contents of the area pointed to by FRBFBACK. The hexadecimal tokens are not used. |
| FRBFBACK PTR(31), FRBRHPC <i>token1 token2</i> | More of FRB. The first hexadecimal token is FRBFBACK. The second hexadecimal token is FRBRHPC. |

| Message text | Explanation |
|--|---|
| FRBQUAL(BIN15), FRBRSV1(BIN15) token1 token2 | More of FRB. The first hexadecimal token is FRBQUAL. The second hexadecimal token is FRBRSV1. |
| CIBPLNID : planid | The CREATE THREAD was done for plan 'planid.' The hexadecimal tokens are not used. |
| CIBCORID : corid | The CREATE THREAD was done by using 'corid' as the correlation ID. The hexadecimal tokens are not used. |
| ZINDOUBT : indoubt | The CREATE THREAD was done by using 'indoubt' for indoubt status. The hexadecimal tokens are not used. |
| CSECT: DSNECP14 | |
| Message text | Explanation |
| ENTER DSNECP14, SIBDFLAG=, ADDR(SIBDCL)= token1 token2 | This message announces entry into CSECT DSNECP14. The first hexadecimal token is SIBDFLAG, a byte of DCLGEN flag bits. The second hexadecimal token is ADDR(SIBDCL), the DCLGEN section of the SIB. |
| BEFORE LI SETUP,PARSE WAS GOOD CIBRFRB,CIBFRMLI token1 token2 | This message indicates that preparation for the create thread is in progress. The first hexadecimal token is CIBRFRB, the address of the FRB. The second hexadecimal token is CIBFRMLI, which contains the address to which the language interface will branch. |
| AFTER DSNECP66, SIBDSQLD=, SQLD= token1 token2 | The PREPARE, DESCRIBE, and SELECT are now successfully completed. The first hexadecimal token is SIBDSQLD, the SQLDA address. The second hexadecimal token is SQLD. |
| BEGIN COMMENT token1 token2 | The CSECT is about to begin writing the first comment block. The hexadecimal tokens are not used. |

| Message text | Explanation |
|---|---|
| BEFORE CALL DSNECP67, SIBDOFF=, ZBIN31= <i>token1 token2</i> | <p>The CSECT just completed the first comment block.</p> <p>The first hexadecimal token contains the current value of SIBDOFF, the offset into the main output buffer.</p> <p>The second hexadecimal token is contains ZBIN31.</p> |
| AFTER DSNECP67, SIBDOFF= <i>token1 token2</i> | <p>Control that is just returned from DSNECP67.</p> <p>The first hexadecimal token contains the current value of SIBDOFF, the offset into the main output buffer.</p> <p>The second hexadecimal token is not used.</p> |
| FINISHED SECOND COMMENT BLOCK, SIBDOFF= <i>token1 token2</i> | <p>The second comment block is now complete.</p> <p>The first hexadecimal token contains the current value of SIBDOFF, the offset into the main output buffer.</p> <p>The second hexadecimal token is not used.</p> |
| BEFORE TERMINATE THREAD, SIBDOFF= <i>token1 token2</i> | <p>DCLGEN is starting to shutdown and is about to terminate its thread.</p> <p>The first hexadecimal token contains the current value of SIBDOFF, the offset into the main output buffer.</p> <p>The second hexadecimal token is not used.</p> |
| BEFORE CALL DSNECP69 TO CLOSE & DEALLOCATE SIBDOFF= <i>token1 token2</i> | <p>DCLGEN continues its cleanup process.</p> <p>The first hexadecimal token is SIBDOFF.</p> <p>The second hexadecimal token is not used.</p> |
| BEFORE FREEMAIN, SIBDSQLS=, SIBDSQLD= <i>token1 token2</i> | <p>The FREEMAIN of the SQLDA is about to happen.</p> <p>The first hexadecimal token contains the size of the SQLDA.</p> <p>The second hexadecimal token contains the address of the SQLDA.</p> |
| LEAVE DSNECP14, SIBDFLAG=, SIBDOFF= <i>token1 token2</i> | <p>DCLGEN has just completed. Control will return to the calling CSECT, DSNECP19.</p> <p>The first hexadecimal token contains the value of SIBDFLAG, the DCLGEN flag area.</p> <p>The second hexadecimal token contains the current value of SIBDOFF, the offset into the main output buffer.</p> |

| Message text | Explanation |
|--|---|
| ENTER DECORATE, SIBDOFF= <i>token1 token2</i> | <p>Control has passed to subroutine DECORATE, which writes decorative lines around comment blocks.</p> <p>The first hexadecimal token contains the current value of SIBDOFF, the offset into the main output buffer.</p> <p>The second hexadecimal token is not used.</p> |
| LEAVE DECORATE, SIBDOFF= <i>token1 token2</i> | <p>DECORATE is finished and will return control to the mainline code.</p> <p>The first hexadecimal token contains the current value of SIBDOFF, the offset into the main output buffer.</p> <p>The second hexadecimal token is not used.</p> |
| ENTER RESETBUF, SIBDOFF= <i>token1 token2</i> | <p>Control has passed to subroutine RESETBUF, which clears the output buffer and resets SIBDOFF.</p> <p>The first hexadecimal token contains the current value of SIBDOFF, the offset into the main output buffer.</p> <p>The second hexadecimal token is not used.</p> |
| LEAVE RESETBUF, ZBIGBUF(1:25)= 25bytes <i>token1 token2</i> | <p>The first 25 bytes of ZBIGBUF are shown.</p> <p>The first hexadecimal token contains the current value of SIBDOFF, the offset into the main output buffer.</p> <p>The second hexadecimal token is not used.</p> |
| ENTER WRITECOM, SIBDOFF= <i>token1 token2</i> | <p>Control has now passed to subroutine WRITECOM, which writes a line of comment block output.</p> <p>The first hexadecimal token contains the current value of SIBDOFF, the offset into the main output buffer.</p> <p>The second hexadecimal token is not used.</p> |
| LEAVE WRITECOM, SIBDOFF= <i>token1 token2</i> | <p>Control will now return to the mainline code.</p> <p>The first hexadecimal token contains the current value of SIBDOFF, the offset into the main output buffer.</p> <p>The second hexadecimal token is not used.</p> |

CSECT: DSNECP16

| Message text | Explanation |
|---|--|
| ENTER DSNECP16, ADDR(SIB)=, ADDR(SIBRUN)= token1 token2 | CSECT DSNECP16 has been entered. The first hexadecimal token contains the subcommand information block (SIB) address. The second hexadecimal token contains the address of SIBRUN, the RUN section of the SIB. |
| BEGIN TSO COMMAND CIBCPPL=,CPPLECT token1 token2 | Start processing TSO command sequence. The first hexadecimal token is CIBCPPL. The second hexadecimal token is CPPLECT. |
| userinput token1 token2 | This message echoes the users input from the PUTGET. The first hexadecimal token is the command length (ZCMDLEN). The second hexadecimal token is not used. |
| AFTER PUTGET ZRETCODE=, CPPLCBUF= token1 token2 | This is more information regarding the PUTGET that just completed. The first hexadecimal token is the return code from the PUTGET. The second hexadecimal token is CPPLCBUF. |
| END MODE CPPLCBUF= token1 token2 | TSO command has been invoked or PUTGET failed. The first hexadecimal token is CPPLCBUF. The second hexadecimal token is not used. |
| CIBMAC= BEFORE FREEMAIN, CPPLCBUF=, ZCMDLEN=token1 token2 | About to freemain the TSO command buffer. The first hexadecimal token contains CPPLCBUF, the address of the command buffer about to be freed with FREEMAIN. The second hexadecimal token contains ZCMDLEN, the length of the command buffer. |
| END TSO COMMAND token1 token2 | The TSO command processing is completed. The hexadecimal tokens are not used. |
| BEFORE LINK TO PROGRAM SIBRDCB=, ADDR(SIBRUN)= token1 token2 | About to LINK to application program. The first hexadecimal token contains SIBRDCB. The second hexadecimal token contains the address of SIBRUN. |

| Message text | Explanation |
|--|---|
| FRBRC1(BIN15), FRBRC2(CHAR4) token1 token2 | More of the FRB. The first hexadecimal token is FRBRC1. The second hexadecimal token is FRBRC2. |
| FRBFBACK(PTR), FRBRHPC(BIN32) token1 token2 | More of the FRB. The first hexadecimal token is FRBFBACK. The second hexadecimal token is FRBRHPC. |
| FRBQUAL(BIN15), FRBRSV1(BIN15) token1 token2 | More of the FRB. The first hexadecimal token is FRBQUAL. The second hexadecimal token is FRBRSV1. |
| ZPARAM1(PTR31) , ZPARAM2(PTR31) token1 token2 | This displays more of the FRB parameter list. The first hexadecimal token is ZPARAM1. The second hexadecimal token is ZPARAM2. |
| ZPARAM3(PTR31) , ZFWD0 token1 token2 | This displays more of the FRB parameter list. The first hexadecimal token is ZPARAM3. The second hexadecimal token is ZFWD0. |
| CSECT: DSNECP18 | |
| Message text | Explanation |
| ENTER DSNECP18 CIBTRMOP=, R6= token1 token2 | CSECT DSNECP18 has been entered. The first hexadecimal token contains CIBTRMOP. The second hexadecimal token is R6, the base register for the main DSN control block (the CIB). |
| CIBTRMOP NOT BLANK, CIBTRMOP= token1 token2 | There is a termination option. The first hexadecimal token is CIBTRMOP. The second hexadecimal token is not used. |
| BEFORE TERMINATE DB2 CALL ===== token1 token2 | Control is about to be passed to Db2 to do the TERMINATE. The first hexadecimal token is CIBTRMOP, the termination option. The second hexadecimal token is not used. |
| AFTER TERMINATE DB2 CALL ===== token1 token2 | Control has returned from Db2 to DSN. The hexadecimal tokens are not used. |

| Message text | Explanation |
|---|--|
| EXIT DSNECP18 <i>token1 token2</i> | DSNECP18 is about to return to caller. The first hexadecimal token is SIBRCODE, the main DSN return code. The second hexadecimal token is not used. |
| FRB FIELDS FOLLOW (CIBFRB): <i>token1 token2</i> | The following messages show the FRB contents. The first hexadecimal token is the FRB address. The second hexadecimal token is not used. |
| FRBRAL(PTR), FRBRALE(BIN15), FRBFVLE(BIN15) <i>token1 token2</i> | More of the FRB. The first hexadecimal token is FRBRAL. The second hexadecimal token is FRBRALE followed by FRBFVLE. |
| FRBPARAM(PTR), FRBPCNT(BIN15) <i>token1 token2</i> | More of the FRB. The first hexadecimal token is FRBPARAM. The second hexadecimal token is FRBPCNT. |
| FRBRC1(BIN15), FRBRC2(CHAR4) <i>token1 token2</i> | More of the FRB. The first hexadecimal token is FRBRC1. The second hexadecimal token is FRBRC2. |
| FRBFBACK(PTR), FRBRHPC(BIN32) <i>token1 token2</i> | More of the FRB. The first hexadecimal token is FRBFBACK. The second hexadecimal token is FRBRHPC. |
| FRBQUAL(BIN15), FRBRSV1(BIN15) <i>token1 token2</i> | More of the FRB. The first hexadecimal token is FRBQUAL. The second hexadecimal token is FRBRSV1. |
| CSECT: DSNECP19 | |
| Message text | Explanation |
| ENTER DSNECP19 <i>token1 token2</i> | This message announces entry into CSECT DSNECP19, which does individual subcommand processing and issues the main DSN PUTGET. The first hexadecimal token is CIBPTR, the base for the CIB, (the main DSN control block). The second hexadecimal token is not used. |

| Message text | Explanation |
|---|--|
| BEFORE TEST CPPLCBUF,CIBEXECB <i>token1 token2</i> | <p>If CPPLCBUF, the address of the command buffer, is zero, this CSECT issues a PUTGET, writes 'DSN', and reads in a new command buffer.</p> <p>The first hexadecimal token is CPPLCBUF.</p> <p>The second hexadecimal token is CIBEXECB.</p> |
| AFTER PUTGET CPPLCBUF=, ZRETCODE= <i>token1 token2</i> | <p>This message occurs after the PUTGET has completed. The PUTGET got DSN a subcommand to process.</p> <p>The first hexadecimal token is CPPLCBUF, the address of the buffer holding the current subcommand.</p> <p>The second hexadecimal token is the return code from the PUTGET.</p> |
| <i>userinput</i> | <p>This message echoes the user input from the PUTGET.</p> <p>The first hexadecimal token is the command length.</p> <p>The second hexadecimal token is the return code from the PUTGET.</p> |
| BEGIN ZCLEANER, ZPLACE=, ZSIZE= <i>token1 token2</i> | <p>DSNECP19 is about to clear an area in the subcommand information block (SIB).</p> <p>The first hexadecimal token is ZPLACE.</p> <p>The second hexadecimal token is ZSIZE.</p> |
| CLEAR LOOPTOP, ZPLACE=, ZSIZE= <i>token1 token2</i> | <p>Each time this message appears, another 255-byte portion of the SIB is about to be zeroed.</p> <p>The first hexadecimal token is ZPLACE.</p> <p>The second hexadecimal token is ZSIZE.</p> |
| END ZCLEANER, ZPLACE= <i>token1 token2</i> | <p>The varying area of the subcommand information block (SIB) has now been zeroed.</p> <p>The first hexadecimal token is ZPLACE.</p> <p>The second hexadecimal token is not used.</p> |
| BEFORE SUBCOMMAND SELECT, CIBSCANC=cibscanc <i>token1 token2</i> | <p>Subcommand processing is about to begin. The main SELECT statement follows.</p> <p>The hexadecimal tokens are not used.</p> |
| BEFORE CALL DSNECP21 ADDR(DSNECP21)= <i>token1 token2</i> | <p>DSNECP19 is about to call DSNECP21 to parse the DSN command.</p> <p>The first hexadecimal token is the address of DSNECP21.</p> <p>The second hexadecimal token is not used.</p> |

| Message text | Explanation |
|--|--|
| BEFORE CALL DSNECP40 (BIND) ADDR(DSNECP40)= token1 token2 | <p>DSNECP19 is about to call DSNECP40 to parse the BIND subcommand.</p> <p>The first hexadecimal token is the address of DSNECP40.</p> <p>The second hexadecimal token is not used.</p> |
| BEFORE CALL DSNECP29 ADDR(DSNECP29)= token1 token2 | <p>The BIND parse was successful and DSNECP29 is about to be called to perform dynamic allocation for BIND.</p> <p>The first hexadecimal token is the address of DSNECP29.</p> <p>The second hexadecimal token is not used.</p> |
| BEFORE CALL DSNECP41 (REBIND) ADDR(DSNECP41)= token1 token2 | <p>DSNECP19 is about to call DSNECP41 to parse the REBIND subcommand.</p> <p>The first hexadecimal token is the address of DSNECP41.</p> <p>The second hexadecimal token is not used.</p> |
| BEFORE CALL DSNECP42 (FREE) token1 token2 | <p>DSNECP19 is about to call DSNECP42 to parse the FREE subcommand.</p> <p>The hexadecimal tokens are not used.</p> |
| BEFORE CALL DSNECP26 (RUN) token1 token2 | <p>DSNECP19 is about to call DSNECP26 to parse the RUN subcommand.</p> <p>The hexadecimal tokens are not used.</p> |
| BEFORE CALL DSNECP24 (DCLGEN) token1 token2 | <p>DSNECP19 is about to call DSNECP24 to parse the DCLGEN subcommand.</p> <p>The hexadecimal tokens are not used.</p> |
| BEFORE CALL DSNECP17 (SUBSYSTEM CMD) token1 token2 | <p>DSNECP19 is about to call DSNECP17 to process a Db2 command.</p> <p>The hexadecimal tokens are not used.</p> |
| BEFORE LINK DSNESM00 (SPUFI) R6=, R7= token1 token2 | <p>DSNECP19 is about to call DSNECP17 to process a Db2 command.</p> <p>The first hex token contains the contents of R6, the CIB address.</p> <p>The second hex token contains the contents of R7, the address of the SPUFI load module (DSNESM00).</p> |

| Message text | Explanation |
|--|---|
| AFTER LINK TO SPUFI R6=, ZRETCode= token1 token2 | <p>SPUFI just completed and DSNECP19 regained control.</p> <p>The first hex token contains the contents of R6, the CIB address.</p> <p>The second hex token contains the return code from the LINK.</p> |
| BEFORE ABEND 00C50101 R6=token1 token2 | <p>The user entered the ABEND subcommand with the TEST parameter on the DSN command set to a nonzero value. The next thing to happen will be a system ABEND that takes a dump.</p> <p>The first hexadecimal token is R6.</p> <p>The second hexadecimal token is not used.</p> |
| UNRECOGNIZED SUBCOMMAND, ASSUME TSO COMMAND cibscanc ADDR(DSNECP31)=token1 token2 | <p>An unrecognized subcommand has been detected. DSN will try to ATTACH it. The following message shows the subcommand.</p> <p>The first hexadecimal token is the address of csect DSNECP31, which will attempt to ATTACH a load module called whose name is stored in cibscanc.</p> <p>The second hexadecimal token is not used.</p> |
| cibscanc token1 token2 | <p>cibscanc is the unrecognized command string that will be attached.</p> <p>The hexadecimal tokens are not used.</p> |
| END SUBCOMMAND ROUTINE token1 token2 | <p>The unrecognized subcommand processing is complete.</p> <p>The hexadecimal tokens are not used.</p> |
| BEFORE FREEMAIN, CPPLCBUF,ZCMDLEN token1 token2 | <p>The command buffer is about to be freed.</p> <p>The first hexadecimal token is CPPLCBUF.</p> <p>The second hexadecimal token is ZCMDLEN.</p> |
| EXIT DSNECP19 CIBSCANC=cibscanc R6=, CPPLPTR=token1 token2 | <p>CSECT DSNECP19 has completed processing one subcommand. Control now returns to CSECT DSNECP10. The current subcommand is cibscanc.</p> <p>The first hex token is R6, the CIB base register.</p> <p>The second hex token is CPPLPTR, the pointer to the CPPL.</p> |

CSECT: DSNECP20

| Message text | Explanation |
|---|---|
| ENTER DSNECP20 R6,CIBCPPL <i>token1 token2</i> | <p>This message announces entry into CSECT DSNECP20, which initializes the parse and scan parameter lists.</p> <p>The first hexadecimal token is the CIB address.</p> <p>The second hexadecimal token is CIBCPPL.</p> |
| <i>cibpromp token1 token2</i> | <p>This message displays the prompting string that will be written to the terminal or batch-output stream when CSECT DSNECP19 issues a PUTGET.</p> <p>The hexadecimal tokens are not used.</p> |
| PPLDONE PPLPTR,PPLANS <i>token1 token2</i> | <p>The parse parameter list has been built. It will be used by all of the DSN CSECTs that do parsing.</p> <p>The first hexadecimal token is PPLPTR.</p> <p>The second hexadecimal token is PPLANS.</p> |
| CSPLDONE CSPLPTR,CSOAPTR <i>token1 token2</i> | <p>The command scan parameter list has been built. It will be used by CSECT DSNECP22.</p> <p>The first hexadecimal token is CSPLPTR.</p> <p>The second hexadecimal token is CSOAPTR.</p> |

CSECT: DSNECP21

| Message text | Explanation |
|---|---|
| ENTER DSNECP21 CIBPTR=, CIBHDECP= <i>token1 token2</i> | <p>This message announces entry into CSECT DSNECP21, which parses the DSN command.</p> <p>The first hexadecimal token is CIBPTR, the address of the main DSN control block.</p> <p>The second hexadecimal token is the address of csect DSNHDECP.</p> |
| BEFORE PARSE CALL PPLPTR=, CPPLPTR= <i>token1 token2</i> | <p>The CSECT is about to call IKJPARS.</p> <p>The first hexadecimal token is the Parse Parameter List address.</p> <p>The second hexadecimal token is the Command Processor Parameter List address.</p> |
| AFTER PARSE, R15= <i>token1 token2</i> | <p>This message is displayed after the call to IKJPARS has taken place.</p> <p>The first hexadecimal token is register 15.</p> <p>The second hexadecimal token is not used.</p> |

| Message text | Explanation |
|---|---|
| TEST LEVEL: CIBTESTL=token1 token2 | <p>The TEST parameter of the DSN command has been parsed.</p> <p>The first hexadecimal token is CIBTESTL level.</p> <p>The second hexadecimal token is not used.</p> |
| SUBSYS ID: subsys CIBHDECP=token1 token2 | <p>The subsystem parameter of the DSN command has been parsed.</p> <p>The first hexadecimal token is the contents of CIBHDECP.</p> <p>The second hexadecimal token is not used.</p> |
| RETRY COUNT: CIBRTRY=token1 token2 | <p>The RETRY parameter of the DSN command has been parsed.</p> <p>The first hexadecimal token is the RETRY value.</p> <p>The second hexadecimal token is not used.</p> |
| EXIT DSNECP21, R15= token1 token2 | <p>CSECT DSNECP21 has now completed processing. Control will now return to the calling CSECT, DSNECP19.</p> <p>The first hexadecimal token is register 15.</p> <p>The second hexadecimal token is not used.</p> |
| CSECT: DSNECP22 | |
| Message text | Explanation |
| ENTER DSNECP22 token1 token2 | <p>This message announces entry into CSECT DSNECP22, which scans the various DSN subcommands.</p> <p>The hexadecimal tokens are not used.</p> |
| BEFORE CALL IKJSCAN token1 token2 | <p>This message is displayed before the call to IKJSCAN that will determine what subcommand the user entered.</p> <p>The hexadecimal tokens are not used.</p> |
| ZCMDLEN, CSOAFGL token1 token2 | <p>This message reports results from the call to IKJSCAN.</p> <p>The first hexadecimal token is the length of the command.</p> <p>The second hexadecimal token is CSOAFGL.</p> |
| CIBSCANC=cibscanc token1 token2 | <p>cibscanc is the subcommand the user entered (via the PUTGET in DSNECP19).</p> <p>The first hexadecimal token is register 15.</p> <p>The second hexadecimal token is CSOAFGL.</p> |

| Message text | Explanation |
|---|---|
| ZI, CSOAFLG <i>token1 token2</i> | Bad input to IKJSCAN was detected. The first hexadecimal token is ZI. The second hexadecimal token is CSOAFLG. |
| BLANK OR NULL, ZI=, CSOAFLG= <i>token1 token2</i> | This message appears only if the subcommand was a null or blank string. The first hexadecimal token is ZI. The second hexadecimal token is CSOAFLG. |
| DB2 COMMAND FOUND | A Db2 command has been found by recognizing the system start character in the command buffer. The hexadecimal tokens are not used. |
| CIBSCANC= <i>token1 token2</i> | This message contains the current value of CIBSCANC, the subcommand that the user entered. The hexadecimal tokens are not used. |
| EXIT DSNECP22 <i>token1 token2</i> | CSECT DSNECP22 has just completed. Control will return to the calling CSECT, DSNECP19. The hexadecimal tokens are not used. |
| CSECT: DSNECP23 | |
| Message text | Explanation |
| ENTER DSNECP23, SIBRSW1= <i>token1 token2</i> | This message announces entry into CSECT DSNECP23. The first hexadecimal token is SIBRSW1. The second hexadecimal token is 0. |
| END INITIALIZATION, ZOHDAIR=, ZS99RB= <i>token1 token2</i> | Initialization is now complete. The first hexadecimal token is the address of ZOHDAIR. The second hexadecimal token is the address of ZS99RB. |
| BEGIN DYNALLOC, SIBRSW1= <i>token1 token2</i> | The CSECT is starting to build DYNALLOC text units. The first hexadecimal token is SIBRSW1. The second hexadecimal token is not used. |
| DSN TU, ZTUNDX=, ZPTR=, DSNAME: <i>token1 token2</i> | Building the dsname text unit (TU). The first hexadecimal token is ZTUNDX, the index to this TU. The second hexadecimal token is ZPTR, the pointer to the TU. |

| Message text | Explanation |
|---|---|
| <i>dsname token1 token2</i> | <p>Part of the S99 parameter list is displayed, the dsname.</p> <p>The hexadecimal tokens are not used.</p> |
| DDN TU, ZTUNDX=, ZPTR= <i>token1 token2</i> | <p>The DDNAME text unit (TU) has been built.</p> <p>The first hexadecimal token is ZTUNDX, the index to this TU.</p> <p>The second hexadecimal token is ZPTR, the pointer to the TU.</p> |
| DSORG TU, ZTUNDX=, ZPTR= <i>token1 token2</i> | <p>The return DSORG text unit (TU) has been built.</p> <p>The first hexadecimal token is ZTUNDX, the index to this TU.</p> <p>The second hexadecimal token is ZPTR, the pointer to the TU.</p> |
| DS STATUS TU, ZTUNDX=, ZPTR= <i>token1 token2</i> | <p>The data set status text unit (TU) has been built.</p> <p>The first hexadecimal token is ZTUNDX, the index to this TU.</p> <p>The second hexadecimal token is ZPTR, the pointer to the TU.</p> |
| NDISP TU, ZTUNDX=, ZPTR= <i>token1 token2</i> | <p>The normal disposition text unit (TU) has been built.</p> <p>The first hexadecimal token is ZTUNDX, the index to this TU.</p> <p>The second hexadecimal token is ZPTR, the pointer to the TU.</p> |
| CDISP TU, ZTUNDX=, ZPTR= <i>token1 token2</i> | <p>The conditional disposition text unit (TU) has been built.</p> <p>The first hexadecimal token is ZTUNDX, the index to this TU.</p> <p>The second hexadecimal token is ZPTR, the pointer to the TU.</p> |
| PSWD TU, ZTUNDX=, ZPTR= <i>token1 token2</i> | <p>The password text unit (TU) has been built.</p> <p>The first hexadecimal token is ZTUNDX, the index to this TU.</p> <p>The second hexadecimal token is ZPTR, the pointer to the TU.</p> |
| BEFORE DYNALLOC, S99RBPTR=, ZPTR= <i>token1 token2</i> | <p>About to issue DYNALLOC SVC.</p> <p>The first hexadecimal token is S99RBPTR.</p> <p>The second hexadecimal token is ZPTR, the pointer to the text unit.</p> |

| Message text | Explanation |
|---|--|
| AFTER DYNALLOC, R15=, S99RSC= token1 token2 | <p>After the DYNALLOC SVC.</p> <p>The first hexadecimal token is register 15.</p> <p>The second hexadecimal token is S99RSC.</p> |
| END DYNALLOC, SIBRCODE=, SIBRSW1= token1 token2 | <p>The DYNALLOC function is completed.</p> <p>The first hexadecimal token is SIBRCODE, the return code from this routine.</p> <p>The second hexadecimal token is SIBRSW1, a byte of status switches.</p> |
| OPEN COMPLETE, SIBRCODE=, SIBRSW1= token1 token2 | <p>The OPEN function has been completed.</p> <p>The first hexadecimal token is SIBRCODE, the return code from this routine.</p> <p>The second hexadecimal token is SIBRSW1, a byte of status switches.</p> |
| BEGIN DEALLOC, SIBRSW1= token1 token2 | <p>Starting data set deallocation.</p> <p>The first hexadecimal token is SIBRSW1.</p> <p>The second hexadecimal token is not used.</p> |
| DALLOC TU, ZTUNDX=, ZPTR=, DSNAME: token1 token2 | <p>The deallocation text unit has been built.</p> <p>The first hexadecimal token is ZTUNDX.</p> <p>The second hexadecimal token is ZPTR.</p> |
| dsname token1 token2 | <p>Data set name used in the deallocation text unit.</p> <p>The hexadecimal tokens are not used.</p> |
| AFTER DYNALLOC, R15=, S99RSC= token1 token2 | <p>After the DYNALLOC SVC.</p> <p>The first hexadecimal token is register 15.</p> <p>The second hexadecimal token is S99RSC.</p> |
| EXIT DSNECP23, SIBRCODE=, SIBRSW1= token1 token2 | <p>DSNECP23 is about to return to caller.</p> <p>The first hexadecimal token is SIBRCODE.</p> <p>The second hexadecimal token is SIBRSW1.</p> |
| BEFORE CALL TO DAIRFAIL, R15=, S99RSC= token1 token2 | <p>DAIRFAIL is about to be invoked.</p> <p>The first hexadecimal token is register 15.</p> <p>The second hexadecimal token is S99RSC.</p> |
| AFTER DAIRFAIL, R15=, R1= token1 token2 | <p>After DAIRFAIL call.</p> <p>The first hexadecimal token is register 15.</p> <p>The second hexadecimal token is register 1.</p> |

CSECT: DSNECP24

| Message text | Explanation |
|--|--|
| ENTER DSNECP24, CIBPTR=, ADDR(SIBDCL)= <i>token1 token2</i> | This message announces entry into CSECT DSNECP24, which parses the DCLGEN command. The first hexadecimal token is CIBPTR. The second hexadecimal token is the address of SIBDCL. |
| BEFORE CALL IKJPARS, R1= <i>token1 token2</i> | This message is displayed just before the call to IKJPARS. The first hexadecimal token is register 1. The second hexadecimal token is zero. |
| AFTER PARSE, RETCODE R15=<i>token1 token2</i> | This message is displayed after the call to IKJPARS has taken place. The first hexadecimal token is register 15. The second hexadecimal token is not used. |
| TABLENAME: <i>tabname</i> ZPARMLEN=<i>token1 token2</i> | This message indicates the table parameter, <i>tabname</i> , from the DCLGEN subcommand. The first hexadecimal token is ZPARMLEN. The second hexadecimal token is zero. |
| LIBRARY : <i>libname</i> ZPARMLEN=<i>token1 token2</i> | This message shows the library parameter, <i>libname</i> , from the DCLGEN subcommand. The first hexadecimal token is ZPARMLEN. The second hexadecimal token is zero. |
| MEMBER : <i>memname</i> ZPDEBIN=<i>token1 token2</i> | This message shows the member parameter, <i>memname</i> , from the DCLGEN subcommand. The first hexadecimal token is ZPDEBIN. The second hexadecimal token is not used. |
| PASSWORD : <i>password</i> ZPDEBIN=<i>token1 token2</i> | This message shows the password parameter from the DCLGEN subcommand. The first hexadecimal token is ZPDEBIN. The second hexadecimal token is not used. |
| ACTION : <i>action</i> ZPDEBIN=<i>token1 token2</i> | This message shows the action parameter from the DCLGEN subcommand. The first hexadecimal token is ZPDEBIN. The second hexadecimal token is not used. |

| Message text | Explanation |
|---|---|
| WILL GET LANGUAGE FROM DSNHDECP ZDECPTR=, ZPTR= <i>token1 token2</i> | <p>The user did not enter a language parameter on the DCLGEN subcommand, so the value specified at installation time and written in the DECP CSECT will be used.</p> <p>The first hexadecimal token is ZDECPTR, the base pointer for DECP.</p> <p>The second hexadecimal token is ZPTR, the address of the language parameter's PDE.</p> |
| LANGUAGE : <i>language</i> ZPDEBIN=<i>token1 token2</i> | <p>This message shows the language parameter from the DCLGEN subcommand.</p> <p>The first hexadecimal token is ZPDEBIN.</p> <p>The second hexadecimal token is not used.</p> |
| NAMES : <i>names</i> ZPDEBIN=, SIBDNAML= <i>token1 token2</i> | <p>This message shows the names parameter from the DCLGEN subcommand.</p> <p>The first hexadecimal token is ZPDEBIN.</p> <p>The second hexadecimal token is SIBDNAML, the length of the NAMES parameter.</p> |
| STRUCTURE:<i>structure</i> ZPDEBIN=, SIBDNAML= <i>token1 token2</i> | <p>This message shows the structure parameter from the DCLGEN subcommand.</p> <p>The first hexadecimal token is ZPDEBIN.</p> <p>The second hexadecimal token is SIBDSTRL, the length of the STRUCTURE parameter.</p> |
| WILL GET DELIMITER FROM DSNHDECP ZDECPTR=, DECPSDL= <i>token1 token2</i> | <p>This message indicates that the user did not enter either QUOTE APOST on the DCLGEN subcommand. The APOST/QUOTE setting will be taken from DSNHDECP, a CSECT that is initialized for the whole system at installation time.</p> <p>The first hexadecimal token is ZDECPTR, a pointer to DECP.</p> <p>The second hexadecimal token is the DECP byte containing the delimiter setting.</p> |
| APOST <i>token1 token2</i> | <p>This message indicates that the APOST option was specified on the DCLGEN subcommand. This message and the next (QUOTE) are mutually exclusive.</p> <p>The first hexadecimal token is ZSQLDLIM, the PDE returned from parse.</p> <p>The second hexadecimal token is not used.</p> |

| Message text | Explanation |
|---|---|
| QUOTE <i>token1 token2</i> | <p>This message indicates that the QUOTE option was specified on the DCLGEN subcommand. This message and the previous (APOST) are mutually exclusive.</p> <p>The first hexadecimal token is ZSQLDLIM, the PDE returned from parse.</p> <p>The second hexadecimal token is not used.</p> |
| LABEL : label ZPDEBIN=, SIBDFLG2= <i>token1 token2</i> | <p>This message indicates the LABEL option used in the DCLGEN command. It will be either YES or NO.</p> <p>The first hexadecimal token is ZPDEBIN.</p> <p>The second hexadecimal token is the second DCLGEN flag byte.</p> |
| LEAVE DSNECP24 , SIBDFLAG=, SIBRCODE= <i>token1 token2</i> | <p>This CSECT has just completed processing. Control will return to the calling CSECT, DSNECP19.</p> <p>The first hexadecimal token is SIBDFLAG.</p> <p>The second hexadecimal token is SIBRCODE.</p> |
| CSECT: DSNECP25 | |
| Message text | Explanation |
| ENTER DSNECP25 <i>token1 token2</i> | <p>This message announces entry into CSECT DSNECP25, which prepares the DCLGEN command string before the string is input to the TSO parser.</p> <p>The first hexadecimal token is CIBPTR.</p> <p>The second hexadecimal token is the address of SIBDCL.</p> |
| STRING (LANGUAGE) FOUND AT: <i>token1</i> | <p>This message indicates that the LANGUAGE parameter was found in a DCLGEN command string.</p> <p>The token gives the offset into the command string of the parameter.</p> |
| BLANK FOLLOWS STRING AT: <i>token1</i> | <p>This message is issued if a blank is found between a DCLGEN parameter and its argument.</p> <p>The token gives the offset into the command string of the blank.</p> |
| LAST SCAN CHARACTER: <i>token1</i> | <p>This message indicates that the first non-blank character after the DCLGEN LANGUAGE, TABLE, STRUCTURE, or NAMES parameter has been found.</p> <p>The token is the position of this character.</p> |

| Message text | Explanation |
|---|---|
| BLANK FOLLOWS PAREN AT <i>token1</i> | <p>This message indicates that a blank follows an opening parenthesis within the TABLE parameter argument in a DCLGEN command string.</p> <p>The token gives the offset into the command string of the blank.</p> |
| LANGUAGE IS C | <p>This message indicates that DCLGEN output will be in the C language.</p> |
| STRING (TABLE) FOUND AT: <i>token1</i> | <p>This message indicates that the TABLE parameter was found in a DCLGEN command string.</p> <p>The token gives the offset into the command string of the parameter.</p> |
| STRING (STRUCTURE) FOUND AT: <i>token1</i> | <p>This message indicates that the STRUCTURE parameter was found in a DCLGEN command string.</p> <p>The token gives the offset into the command string of the parameter.</p> |
| STRING (NAMES) FOUND AT: <i>token1</i> | <p>This message indicates that the NAMES parameter was found in a DCLGEN command string.</p> <p>The token gives the offset into the command string of the parameter.</p> |
| FOUND: <i>token1 token2</i> | <p>This message indicates that the table name in a DCLGEN statement contains a period.</p> <p>The first token gives the offset into the command string of the period.</p> <p>The second token gives one of six states in which the string can be:</p> <ul style="list-style-type: none"> • 0 - First part of the table name string • 1 - Blank after first part of the string • 2 - Period separating two parts of the table name • 3 - Blank after a separating period • 4 - Second string after a separating period • 5 - End of the table name |

| Message text | Explanation |
|--|--|
| QUOTE FOUND: <i>token1 token2</i> | <p>This message indicates that a delimiter was found in the TABLE parameter value in a DCLGEN command string.</p> <p>The first token gives the offset into the command string of the character.</p> <p>The second token gives one of six states in which the string can be:</p> <ul style="list-style-type: none"> • 0 - First part of the table name string • 1 - Blank after first part of the string • 2 - Period separating two parts of the table name • 3 - Blank after a separating period • 4 - Second string after a separating period • 5 - End of the table name |
| S0 FOUND: <i>token1 token2</i> | <p>This message indicates that a DBCS shift-out character was found in the TABLE parameter value in a DCLGEN command string.</p> <p>The first token gives the offset into the command string of the character.</p> <p>The second token gives one of six states in which the string can be:</p> <ul style="list-style-type: none"> • 0 - First part of the table name string • 1 - Blank after first part of the string • 2 - Period separating two parts of the table name • 3 - Blank after a separating period • 4 - Second string after a separating period • 5 - End of the table name |
| SI FOUND: <i>token1 token2</i> | <p>This message indicates that a DBCS shift-in character was found in the TABLE parameter value in a DCLGEN command string.</p> <p>The first token gives the offset into the command string of the character.</p> <p>The second token gives one of six states in which the string can be:</p> <ul style="list-style-type: none"> • 0 - First part of the table name string • 1 - Blank after first part of the string • 2 - Period separating two parts of the table name • 3 - Blank after a separating period • 4 - Second string after a separating period • 5 - End of the table name |

| Message text | Explanation |
|---|--|
| BLANK FOUND: <i>token1 token2</i> | <p>This message indicates that the table name in a DCLGEN statement contains a blank.</p> <p>The first token gives the offset into the command string of the blank.</p> <p>The second token gives one of six states in which the string can be:</p> <ul style="list-style-type: none"> • 0 - First part of the table name string • 1 - Blank after first part of the string • 2 - Period separating two parts of the table name • 3 - Blank after a separating period • 4 - Second string after a separating period • 5 - End of the table name |
|) FOUND: <i>token1 token2</i> | <p>This message indicates that a closing parenthesis was found within the argument of the TABLE parameter in a DCLGEN command string.</p> <p>The first token gives the offset into the command string of the closing parenthesis.</p> <p>The second token gives one of six states in which the string can be:</p> <ul style="list-style-type: none"> • 0 - First part of the table name string • 1 - Blank after first part of the string • 2 - Period separating two parts of the table name • 3 - Blank after a separating period • 4 - Second string after a separating period • 5 - End of the table name |
| OTHER CHAR FOUND: <i>token1 token2</i> | <p>This message indicates that a character other than a delimiter, a DBCS shift-out or shift-in character, a blank or a right parenthesis was found in the TABLE parameter value in a DCLGEN command string.</p> <p>The first token gives the offset into the command string of the character.</p> <p>The second token gives one of six states in which the string can be:</p> <ul style="list-style-type: none"> • 0 - First part of the table name string • 1 - Blank after first part of the string • 2 - Period separating two parts of the table name • 3 - Blank after a separating period • 4 - Second string after a separating period • 5 - End of the table name |
| ENTER COPYNME | <p>This message is issued on entry to the COPYNME routine, which copies a STRUCTURE or NAMES argument to a temporary buffer.</p> |

| Message text | Explanation |
|--|---|
| APOSTROPHE FOUND AT: <i>token1</i> | <p>This message indicates an apostrophe was found within a STRUCTURE or NAMES argument in a DCLGEN command string.</p> <p>The token gives the offset into the command string of the apostrophe.</p> |
| APOSTROPHE FOLLOWS APOSTROPHE AT: <i>token1</i> | <p>This message indicates two consecutive apostrophes were found within a STRUCTURE or NAMES argument in a DCLGEN command string.</p> <p>The token gives the offset into the command string of the first apostrophe.</p> |
| SO FOUND AT: <i>token1</i> | <p>This message indicates that a DBCS shift-out character was found within the argument of a NAMES or STRUCTURE parameter in a DCLGEN command string.</p> <p>The token gives the offset into the command string of the shift-out character.</p> |
| SI FOUND AT: <i>token1</i> | <p>This message indicates that a DBCS shift-in character was found within the argument of a NAMES or STRUCTURE parameter in a DCLGEN command string.</p> <p>The token gives the offset into the command string of the shift-in character.</p> |
| BLANK IN DB OR DLMED STRING AT: <i>token1</i> | <p>This message is issued if a blank is found within a STRUCTURE or NAMES argument in a DCLGEN command string, and the blank is within a DBCS string or a delimited string.</p> <p>The token gives the offset into the command string of the blank.</p> |
| BLANK INDICATES END OF STRING AT: <i>token1</i> | <p>This message is issued when the first blank is found at the end of a STRUCTURE or NAMES argument in a DCLGEN command string.</p> <p>The token gives the offset into the command string of the blank.</p> |
| BLANK FOUND AT: <i>token1</i> | <p>This message is issued if a blank is found within a STRUCTURE or NAMES argument in a DCLGEN command string.</p> <p>The token gives the offset into the command string of the blank.</p> |

| Message text | Explanation |
|---|---|
|) FOUND AT: <i>token1</i> | <p>This message indicates that a closing parenthesis was found within the argument of a NAMES or STRUCTURE parameter in a DCLGEN command string.</p> <p>The token gives the offset into the command string of the closing parenthesis.</p> |
| TRANSLATING CHAR: <i>token1 token2</i> | <p>This message indicates that a character in a DCLGEN command string is being translated to uppercase.</p> <p>The first token gives the value of the character.</p> <p>The second token gives the offset into the command string of the character.</p> |
| OUTPUT STRING LENGTH: <i>token1</i> | <p><i>token1</i> is the length of the modified copy of the DCLGEN command string.</p> |
| INPUT STRING LENGTH: <i>token1</i> | <p><i>token1</i> is the length of the original DCLGEN command string.</p> |
| IN AND OUT INDEXES: <i>token1 token2</i> | <p>This message gives the offsets into the input and output data strings when the COPYNME routine has finished copying a NAMES or STRUCTURE parameter value into the output string.</p> <p>The first token is the offset into the input string.</p> <p>The second token is the offset into the output string.</p> |
| EXIT COPYNME | <p>This message is issued on exit from the COPYNME routine.</p> |
| CSECT: DSNECP26 | |
| Message text | Explanation |
| ENTER DSNECP26, R6=, CIBCPPL= <i>token1 token2</i> | <p>This message announces entry into CSECT DSNECP26, which parses the RUN subcommand.</p> <p>The first hexadecimal token is register 6.</p> <p>The second hexadecimal token is CIBCPPL.</p> |
| AFTER IKJPARS, R15= <i>token1 token2</i> | <p>This message is displayed after the call to IKJPARS has taken place.</p> <p>The first hexadecimal token is register 15.</p> <p>The second hexadecimal token is not used.</p> |

| Message text | Explanation |
|---|--|
| PROGRAM : <i>program token1 token2</i> | This message shows the program parameter, program, from the RUN subcommand. The first hexadecimal token is register 15. The second hexadecimal token is ZPARMLEN. |
| PLANID : <i>planid token1 token2</i> | This message shows the plan parameter from the RUN subcommand. The first hexadecimal token is register 15. The second hexadecimal token is ZPARMLEN. |
| LIBRARY : <i>libname token1 token2</i> | This message shows the library parameter from the RUN subcommand. The first hexadecimal token is register 15. The second hexadecimal token is ZPARMLEN. |
| PASSWORD : <i>password token1 token2</i> | This message shows the password parameter from the library parameter of the RUN subcommand. The first hexadecimal token is SIBRLPLN. The second hexadecimal token is zero. |
| CP: <i>xxx, R15= token1 token2</i> | xxx will be YES or NO, to indicate whether or not the CP parameter was entered. The first hexadecimal token is register 15. The second hexadecimal token is zero. |
| SIBRSW1= <i>sibrsw1 token1 token2</i> | This message displays the byte of RUN subcommand switches called SIBRSW1. The first hexadecimal token is SIBRSW1. The second hexadecimal token is not used. |
| END DSNECP26, SIBRCODE=, CIBCPPL= <i>token1 token2</i> | This CSECT has just completed. Control will now return to the calling CSECT, DSNECP19. The first hexadecimal token is SIBRCODE. The second hexadecimal token is CIBCPPL. |
| CSECT: DSNECP27 | |
| Message text | Explanation |
| ENTER DSNECP27, SIBP27IN= <i>token1 token2</i> | This message announces the entry into CSECT DSNECP27, which formats messages from BIND and Db2 command processing. The first hexadecimal token is SIBP27IN, the address of the message buffer. The second hexadecimal token is zero. |

| Message text | Explanation |
|--|--|
| ZLEN= token1 token2 | The message buffer length is displayed. The first hexadecimal token is ZLEN. The second hexadecimal token is zero. |
| MSG BUF(73CHARS): msgbuf token1 token2 | A portion of the actual message buffer (including LLOO) is displayed. The hexadecimal tokens are not used. |
| BEFORE DO-WHILE, ZBUFPtr=, ZBUFEND= token1 token2 | About to start displaying lines of the message. The first hexadecimal token is ZBUFPtr. The second hexadecimal token is ZBUFEND. |
| TOP OF LOOP, ZLEN=, ZBUFPtr= token1 token2 | This message is displayed once for each line of the message. The first hexadecimal token is ZLEN. The second hexadecimal token is ZBUFPtr. |
| END DSNECP27, R15=, SIBRCODE= token1 token2 | CSECT DSNECP27 is about to return to its caller. The first hexadecimal token is register 15. The second hexadecimal token is SIBRCODE. |

CSECT: DSNECP28

| Message text | Explanation |
|--|--|
| ENTER DSNECP28, R1=, CIBFRB= token1 token2 | This message announces entry into CSECT DSNECP28, which does the initialization processing for the first SQL call from an application program. The first hexadecimal token is register 1. The second hexadecimal token is the FRB address. |
| CALL DSNECP12, CIB=, CIBFRB= token1 token2 | An IDENTIFY request is being made. The first hexadecimal token is the address of the CIB. The second hexadecimal token is the address of the FRB. |
| AFTER DSNECP12, CIB=, CIBIDRTN= token1 token2 | The IDENTIFY request has been completed. The first hexadecimal token is the address of the CIB. The second hexadecimal token is return code from the Identify routine. |

| Message text | Explanation |
|---|---|
| CALL DSNECP13, CIB=, CIBCTFRB= <i>token1 token2</i> | <p>A CREATE-THREAD request is about to be made.</p> <p>The first hexadecimal token is the address of the CIB.</p> <p>The second hexadecimal token is the address of the FRB.</p> |
| AFTER DSNECP13, CIBCTRTN=, CIBCTFRB= <i>token1 token2</i> | <p>The CREATE-THREAD request has been completed.</p> <p>The first hexadecimal token is the return code from Create Thread.</p> <p>The second hexadecimal token is the address of the FRB.</p> |
| BEFORE SQL CALL NUMBER ONE CIBFRMLI=, DSNETRAP= <i>token1 token2</i> | <p>Both the Create Thread and any Identify done in DSNECP28 were successful. DSNECP28 is about to send the first SQL request to the program request handler.</p> <p>The first hexadecimal token is the address to which DSNECP28 will branch. This field will contain either the address of the program request handler or DSNETRAP.</p> <p>The second hexadecimal token is the address of DSNETRAP, a CSECT that writes a trace message just before branching to the program request handler.</p> |
| CREATE THREAD OR IDENTIFY FAILED CIBFRMLI=, TRAP=<i>token1 token2</i> | <p>Either an Identify or Create Thread request originating in DSNECP28 has failed. The normal first SQL call will not be made. Rather, control will return to the language interface, which will make an XLAT call.</p> <p>The first hexadecimal token is the address to which DSNECP28 will branch. This field will contain either the address of the program request handler or DSNETRAP.</p> <p>The second hexadecimal token is the address of DSNETRAP, a CSECT that writes a trace message just before branching to the program request handler.</p> |
| EXIT DSNECP28, RETURNING TO DSNELI FRBRC1, FRBRC2 <i>token1 token2</i> | <p>This message is produced just before CSECT DSNECP28 returns to the language interface.</p> <p>The first hexadecimal token is FRBRC1.</p> <p>The second hexadecimal token is FRBRC2.</p> |

| Message text | Explanation |
|--|--|
| EXIT DSNECP28,CIBFRFB,CIBFRMLI <i>token1 token2</i> | <p>CSECT DSNECP28 has completed processing and is about to return to caller.</p> <p>The first hexadecimal token is the FRB address.</p> <p>The second hexadecimal token is where the language interface will go for the actual SQL processing. Normally, it points to the program request handler.</p> |

CSECT: DSNECP29

| Message text | Explanation |
|---|---|
| ENTER DSNECP29, ADDR(SIBBIND)= <i>token1 token2</i> | <p>This message announces the entry into CSECT DSNECP29, which performs dynamic allocation services for the BIND subcommand.</p> <p>The first hexadecimal token is the address of SIBBIND.</p> <p>The second hexadecimal token is zero.</p> |
| END INITIALIZATION, ADDR(ZOHDAIR)=, ADDR(ZS99RB)= <i>token1 token2</i> | <p>Initialization has completed.</p> <p>The first hexadecimal token is the address of ZOHDAIR.</p> <p>The second hexadecimal token is the address of ZS99RB.</p> |
| BEFORE DDNAME ALLOC, S99RBPTR=, S99TUPAR(1:4)= <i>token1 token2</i> | <p>CSECT DSNECP29 is checking for the presence of a DBRMLIB (DDNAME).</p> <p>The first hexadecimal token is S99RBPTR.</p> <p>The second hexadecimal token is the first 4 bytes of the ddname.</p> |
| AFTER DYNALLOC1, R15=, S99RSC= <i>token1 token2</i> | <p>The first dynamic allocation call just took place.</p> <p>The first hexadecimal token is the return code from the call.</p> <p>The second hexadecimal token is S99RSC, the reason code.</p> |
| BEFORE RETURN DSORG, S99RBPTR=, S99TUPAR(1:4)= <i>token1 token2</i> | <p>CSECT DSNECP29 is about to check on the data set organization of the DBRMLIB.</p> <p>The first hexadecimal token is S99RBPTR.</p> <p>The second hexadecimal token is the first four bytes of S99TUPAR.</p> |

| Message text | Explanation |
|--|--|
| AFTER RETURN DSORG, S99TUPAR(1:4)=, S99RSC= token1 token2 | <p>The DYNALLOC call has completed.</p> <p>The first hexadecimal token is the data set organization code.</p> <p>The second hexadecimal token is S99RSC, the reason code.</p> |
| 1ST TU, ZTUNDX=, ZPTR=, DSNAME:, ADDR(DSN): token1 token2 | <p>The dsname text unit has been built. The next message contains the DSNAME.</p> <p>The first hexadecimal token is the TU index value (ZTUNDX).</p> <p>The second hexadecimal token is the TU pointer (ZPTR).</p> |
| S99TUPAR(1:S99TULNG) token1 token2 | <p>This message is the data set name.</p> <p>The hexadecimal tokens are not used.</p> |
| 2ND TU, ZTUNDX=, ZPTR= token1 token2 | <p>The return DDNAME text unit has been built.</p> <p>The first hexadecimal token is the TU index value (ZTUNDX).</p> <p>The second hexadecimal token is the TU pointer (ZPTR).</p> |
| 3RD TU, ZTUNDX=, ZPTR= token1 token2 | <p>The return DSORG text unit has been built.</p> <p>The first hexadecimal token is the TU index value (ZTUNDX).</p> <p>The second hexadecimal token is the TU pointer (ZPTR).</p> |
| 4TH TU, ZTUNDX=, ZPTR= token1 token2 | <p>The data set status text unit has been built.</p> <p>The first hexadecimal token is the TU index value (ZTUNDX).</p> <p>The second hexadecimal token is the TU pointer (ZPTR).</p> |
| 5TH TU, ZTUNDX=, ZPTR= token1 token2 | <p>The normal disposition text unit has been built.</p> <p>The first hexadecimal token is the TU index value (ZTUNDX).</p> <p>The second hexadecimal token is the TU pointer (ZPTR).</p> |
| 6TH TU, ZTUNDX=, ZPTR= token1 token2 | <p>The conditional disposition text unit has been built.</p> <p>The first hexadecimal token is the TU index value (ZTUNDX).</p> <p>The second hexadecimal token is the TU pointer (ZPTR).</p> |

| Message text | Explanation |
|--|---|
| 7TH TU, ZTUNDX=, ZPTR= token1 token2 | <p>The password text unit has been built.</p> <p>The first hexadecimal token is the TU index value (ZTUNDX).</p> <p>The second hexadecimal token is the TU pointer (ZPTR).</p> |
| BEFORE DS ALLOC, S99RBPTR=, ZPTR= token1 token2 | <p>This CSECT is about to issue the DYNALLOC SVC.</p> <p>The first hexadecimal token is S99RBPTR.</p> <p>The second hexadecimal token is ZPTR.</p> |
| AFTER DYNALLOC3, R15=, S99RSC= token1 token2 | <p>After the third DYNALLOC SVC.</p> <p>The first hexadecimal token is register 15.</p> <p>The second hexadecimal token is S99RSC.</p> |
| CONCATENATION COUNT= token1 token2 | <p>This message shows the number of data sets to be concatenated for BIND.</p> <p>The first hexadecimal token is SIBBCCB.</p> <p>The second hexadecimal token is zero.</p> |
| BEFORE CONCAT, S99RBPTR=, ZPTR= token1 token2 | <p>CSECT DSNECP29 is about to request dynamic concatenation.</p> <p>The first hexadecimal token is S99RBPTR.</p> <p>The second hexadecimal token is ZPTR.</p> |
| AFTER DYNALLOC4, R15=, S99RSC= token1 token2 | <p>The fourth dynamic allocation call has just occurred.</p> <p>The first hexadecimal token is the return code from that call.</p> <p>The second hexadecimal token is S99RSC.</p> |
| CALL DSNECP30 (BIND), ADDR(SIBBIND)= token1 token2 | <p>CSECT DSNECP30 is about to be called.</p> <p>The first hexadecimal token is the address of SIBBIND.</p> <p>The second hexadecimal token is not used.</p> |
| AFTER DSNECP30, SIBRCODE=, ADDR(SIBBIND)= token1 token2 | <p>CSECT DSNECP30 has returned control to DSNECP29.</p> <p>The first hexadecimal token is SIBRCODE.</p> <p>The second hexadecimal token is the address of SIBBIND.</p> |
| BEFORE DECONCAT, S99RBPTR=, SIBCCBP= token1 token2 | <p>CSECT DSNECP29 is about to request dynamic deconcatenation.</p> <p>The first hexadecimal token is S99RBPTR.</p> <p>The second hexadecimal token is SIBCCBP.</p> |

| Message text | Explanation |
|---|--|
| AFTER DYNALLOC5, R15=, S99RSC= token1 token2 | <p>The fifth dynamic deallocation has just occurred.</p> <p>The first hexadecimal token is register 15, the return code from the deallocation.</p> <p>The second hexadecimal token is S99RSC, the reason code from the deallocation.</p> |
| BEFORE UNALLOC, S99RBPTR=, ZTUPTR(1)= token1 token2 | <p>CSECT DSNECP29 is about to request dynamic deallocation.</p> <p>The first hexadecimal token is S99RBPTR.</p> <p>The second hexadecimal token is ZTUPTR(1).</p> |
| AFTER DYNALLOC6, R15=, S99RSC= token1 token2 | <p>The sixth dynamic allocation call has just occurred.</p> <p>The first hexadecimal token is the return code from that call.</p> <p>The second hexadecimal token is the reason code from that call.</p> |
| EXIT DSNECP29, SIBRCODE= token1 token2 | <p>CSECT DSNECP29 is about to return to caller.</p> <p>The first hexadecimal token is SIBRCODE, the return code from this CSECT's execution.</p> <p>The second hexadecimal token is not used.</p> |
| BEFORE CALL TO DAIRFAIL, R15=, S99RSC= token1 token2 | <p>DAIRFAIL is about to be invoked.</p> <p>The first hexadecimal token is register 15.</p> <p>The second hexadecimal token is S99RSC.</p> |
| AFTER DAIRFAIL, R15=, R1= token1 token2 | <p>After DAIRFAIL call to format an error message.</p> <p>The first hexadecimal token is register 15, the return code from DAIRFAIL.</p> <p>The second hexadecimal token is register 1, the address of DFPARMS.</p> |
| BEFORE DBRMLIB UNALLOC, S99RBPTR=, ZPTR= token1 token2 | <p>CSECT DSNECP29 is about to request freeing of DBRMLIB DDNAME.</p> <p>The first hexadecimal token is S99RBPTR.</p> <p>The second hexadecimal token is ZPTR.</p> |
| AFTER DYNALLOC7, R15=, S99RSC= token1 token2 | <p>The seventh dynamic allocation call has completed.</p> <p>The first hexadecimal token is register 15.</p> <p>The second hexadecimal token is S99RSC.</p> |

CSECT: DSNECP30

| Message text | Explanation |
|---|--|
| ENTER DSNECP30 <i>token1 token2</i> | <p>This message announces entry into CSECT DSNECP30, which performs the Db2 calls needed for BIND, REBIND, or FREE.</p> <p>The hexadecimal tokens are not used.</p> |
| END DSNECP30, R15= <i>token1 token2</i> | <p>CSECT DSNECP30 is about to return to its caller.</p> <p>The first hexadecimal token is register 15.</p> <p>The second hexadecimal token is zero.</p> |
| HERE COMES THE FRB CIBFRB <i>token1 token2</i> | <p>This and following messages are a dump of the function request block (FRB) pointed to by CIBFRB. These messages will precede and follow the TERMINATE call.</p> <p>The first hexadecimal token is the FRB address.</p> <p>The second hexadecimal token is zero.</p> |
| FRBRAL(PTR), FRBRALE(BIN15), FRBFVLE(BIN15) <i>token1 token2</i> | <p>More of FRB.</p> <p>The first hexadecimal token is FRBRAL.</p> <p>The second hexadecimal token is FRBRALE followed by FRBFVLE.</p> |
| FRBPARM(PTR), FRBPCNT(BIN15) <i>token1 token2</i> | <p>More of FRB.</p> <p>The first hexadecimal token is FRBPARM.</p> <p>The second hexadecimal token is FRBPCNT.</p> |
| FRBRC1(BIN15), FRBRC2(CHAR4) <i>token1 token2</i> | <p>More of FRB.</p> <p>The first hexadecimal token is FRBRC1.</p> <p>The second hexadecimal token is FRBRC2.</p> |
| FRBFBACK(PTR), FRBRHPC(BIN32) <i>token1 token2</i> | <p>More of FRB.</p> <p>The first hexadecimal token is FRBFBACK.</p> <p>The second hexadecimal token is FRBRHPC.</p> |
| FRBQUAL(BIN15), FRBRSV1(BIN15) <i>token1 token2</i> | <p>More of FRB.</p> <p>The first hexadecimal token is FRBQUAL.</p> <p>The second hexadecimal token is FRBRSV1.</p> |

CSECT: DSNECP31

| Message text | Explanation |
|--|---|
| ENTER DSNECP31, CIBSCANC=command, CIBCPPL=, R6= token1 token2 | <p>This message announces entry into CSECT DSNECP31, which attaches potential TSO commands. SIBSCANC is the subcommand to be attached.</p> <p>The first hexadecimal token is CIBCPPL.</p> <p>The second hexadecimal token is register 6, the base register for the CIB.</p> |
| CPPLCBUF: cbuf ZCMDLEN=, ZCMDOFF= token1 token2 | <p>This message echoes the command buffer to be processed by DSNECP31.</p> <p>The first hexadecimal token is ZCMDLEN, the command buffer length.</p> <p>The second hexadecimal token is ZCMDOFF, the command buffer offset.</p> |
| FOUND "HELP" ONLY ADDR(ZHLPCBUF)=, ZHLPOFF= token1 token2 | <p>HELP command processing is beginning. This command buffer contains only HELP. It is not followed by any other operand.</p> <p>The first hexadecimal token is the address of the CPPLCBUF used for HELP processing.</p> <p>The second hexadecimal token is the command offset into this buffer.</p> |
| USER ENTERED "HELP SOMETHING" token1 token2 | <p>HELP command processing is beginning. This command buffer contains HELP followed by some other operand.</p> <p>The hexadecimal tokens are not used.</p> |
| CBUF BEFORE DSNECP22 cbuf ZCMDLEN=, ZCMDOFF= token1 token2 | <p>DSNECP31 is about to call DSNECP22 to scan the command buffer. The current buffer is displayed.</p> <p>The first hexadecimal token is the LL part of the buffer.</p> <p>The second hexadecimal token is the OO part of the buffer.</p> |
| NEW CIBSCANC IS cibscanc ZCMDLEN=, CPPLCBUF= token1 token2 | <p>The first word after HELP (in the command buffer) is being displayed.</p> <p>The first hexadecimal token is ZCMDLEN, the first 2 bytes in the command buffer.</p> <p>The second hexadecimal token is CPPLCBUF, the command buffer address.</p> |

| Message text | Explanation |
|--|---|
| NOT A DSN SUBCMD ZCMDLEN=, CPPLCBUF= <i>token1 token2</i> | The user requested HELP for a string that has not been recognized as a DSN subcommand. The first hexadecimal token is ZCMDLEN, the first 2 bytes in the buffer. The second hexadecimal token is CPPLCBUF, the command buffer address. |
| NOT A HELP COMMAND ZSIBPTR=, ECTSCMD= <i>token1 token2</i> | The command being processed has been scanned and found not to be a HELP subcommand. The first hexadecimal token is the SIB address. The second hexadecimal token is the CPPL address. |
| ECTPCMD, ECTSCMD <i>pcmd scmd</i> SIBTDCB= <i>token1 token2</i> | ECTPCMD and ECTSCMD are listed. The first hexadecimal token is SIBTDCB. The second hexadecimal token is zero. |
| BEFORE ATTACH===== ZECB=, CIBCPPL= <i>token1 token2</i> | CSECT DSNECP30 is about to ATTACH the potential command. The first hexadecimal token is ZECB. The second hexadecimal token is the address of the CPPL. |
| AFTER ATTACH===== ZECB=, CIBRC= <i>token1 token2</i> | CSECT DSNECP30 is about to ATTACH the potential command. The first hexadecimal token is ZECB. The second hexadecimal token is the return code from the ATTACH. |
| BEFORE WAIT ZECB=, ADDR(ZECB)= <i>token1</i> <i>token2</i> | CSECT DSNECP30 is about to WAIT on the completion of the subtask. The first hexadecimal token is the ECB. The second hexadecimal token is the address of ZECB. |
| AFTER WAIT ZECB=, ZTCB= <i>token1 token2</i> | The wait has been posted. The first hexadecimal token is ZECB, the posted ECB. The second hexadecimal token is ZTCB, the TCB of the attached process. |
| BEFORE DETACH, ZTCB= <i>token1 token2</i> | CSECT DSNECP30 is about to detach the subtask. The first hexadecimal token is ZTCB, the command's TCB. The second hexadecimal token is zero. |

| Message text | Explanation |
|---|---|
| AFTER DETACH R15=token1 token2 | <p>The subtask has been detached.</p> <p>The first hexadecimal token is register 15, the return code from the DETACH.</p> <p>The second hexadecimal token is zero.</p> |
| BEFORE DS ALLOC, S99RBPTR=, TUPTR= token1 token2 | <p>CSECT DSNECP31 is requesting removal of 'in-use' bits left over from subtask.</p> <p>The first hexadecimal token is S99RBPTR.</p> <p>The second hexadecimal token is TUPTR.</p> |
| AFTER DYNALLOC, S99RSC=, R15= token1 token2 | <p>The DYNALLOC request has completed.</p> <p>The first hexadecimal token is S99RSC.</p> <p>The second hexadecimal token is register 15.</p> |
| ZECB & ZMASK=, ZECB= token1 token2 | <p>CSECT DSNECP30 is checking for an 806 system ABEND, which probably indicates typographic errors on the user's subcommand.</p> <p>The first hexadecimal token is ZECB ANDed with ZMASK.</p> <p>The second hexadecimal token is the ECB.</p> |
| TCB ADDRESS ZTCB=, ZECB= token1 token2 | <p>The subtask completed with other than an 806 system ABEND.</p> <p>The first hexadecimal token is ZTCB, the subtask TCB.</p> <p>The second hexadecimal token is the ECB.</p> |
| BEFORE CALL TO DAIRFAIL ZR15=, S99RSC= token1 token2 | <p>CSECT DSNECP30 is requesting translation of DYNALLOC failure.</p> <p>The first hexadecimal token is ZR15.</p> <p>The second hexadecimal token is S99RSC.</p> |
| AFTER DAIRFAIL, R15=, ADDR(DFPARMS)= token1 token2 | <p>The DAIRFAIL request has completed.</p> <p>The first hexadecimal token is register 15, the return code from the DAIRFAIL call.</p> <p>The second hexadecimal token is the address of DFPARMS.</p> |
| ENTER DSNECP31 ESTAIRTN R0=, SDWACWT= token1 token2 | <p>The subtask abended and CSECT DSNECP31's ESTAI routine has been entered.</p> <p>The first hexadecimal token is register 0.</p> <p>The second hexadecimal token is SDWACWT.</p> |

| Message text | Explanation |
|---|--|
| R0=12 R1=token1 token2 | The ESTAI routine was entered with no SDWA. The first hexadecimal token is register 1. The second hexadecimal token is not used. |
| R0=12, X06 ABEND R0=, R1= token1 token2 | An 806 system ABEND occurred in the subtask and no SDWA was passed to the ESTAI routine. A typographic error probably occurred on the subcommand entered. The first hexadecimal token is register 0. The second hexadecimal token is register 1. |
| SDWA COMPLETION CODE, PSW ADDRESS SDWAABCC=, SDWANXT1= token1 token2 | The subtask abended and a SDWA was passed to CSECT DSNECP31's ESTAI routine. The first hexadecimal token is SDWAABCC. The second hexadecimal token is SDWANXT1. |
| R0<>12 SDWAPSTI=token1 token2 | The subtask ended with an 806 system ABEND and a SDWA was presented to CSECT DSNECP31's ESTAI routine. The first hexadecimal token is SDWAPSTI. The second hexadecimal token is not used. |
| END DSNECP31 ESTAIRTN ZRETCODE=, R14SAVE= token1 token2 | The ESTAI routine is returning to ABTERM processing. The first hexadecimal token is ZRETCODE. The second hexadecimal token is R14SAVE. |
| CSECT: DSNECP40 | |
| Message text | Explanation |
| ENTER DSNECP40 CIBPTR=token1 token2 | This message announces entry into CSECT DSNECP40, which parses the BIND subcommand. The first hexadecimal token is the CIB address. The second hexadecimal token is not used. |
| AFTER IKJPARS, R15=, PPLPTR= token1 token2 | This message is displayed after the call to IKJPARS has taken place. The first hexadecimal token is register 15. The second hexadecimal token is the address of the Parse Parameter List. |

| Message text | Explanation |
|---|---|
| PLAN NAME LOOPTOP CHAR: <i>char</i> ZI=, ZPARMLEN= <i>token1 token2</i> | <p>This loop executes once for each character in the PLAN name. That character is printed in the message.</p> <p>The first hexadecimal token is ZI, the loop index.</p> <p>The second hexadecimal token is ZPARMLEN, the length of the PLAN.</p> |
| PLAN : <i>planname</i> ZPARMLEN=, ZPDEBIN= <i>token1 token2</i> | <p>This message shows the plan parameter, <i>planname</i>, from the BIND subcommand.</p> <p>The first hexadecimal token is ZPARMLEN, the length of the plan name.</p> <p>The second hexadecimal token is ZPDEBIN, a parameter presence indicator.</p> |
| MEMBER : <i>memname</i> ZPARMLEN=, ZPTR= <i>token1 token2</i> | <p>This message shows the member parameter, <i>memname</i>, from the BIND subcommand.</p> <p>The first hexadecimal token is ZPARMLEN, the length of the member name.</p> <p>The second hexadecimal token is the PDE address.</p> |
| NO LIBRARY PARM ENTERED ZPDEBIND=, ZPTR= <i>token1 token2</i> | <p>This message indicates that the user did not enter a library parameter.</p> <p>The first hexadecimal token is ZPDEBIN, a parameter presence indicator.</p> <p>The second hexadecimal token is the address of the LIBRARY parameter's PDE.</p> |
| LIBRARY : <i>libname</i> ZPARMLEN=, SIBBPDSN= <i>token1 token2</i> | <p>This message shows the library parameter, <i>libname</i>, from the BIND subcommand.</p> <p>The first hexadecimal token is ZPARMLEN, the name length.</p> <p>The second hexadecimal token is SIBBPDSN, the address of the name.</p> |
| PASSWORD : <i>password</i> ZPWDLEN=, ZPWDPTR= <i>token1 token2</i> | <p>This message shows the password parameter from the BIND subcommand.</p> <p>The first hexadecimal token is ZPWDLEN, the length of the password.</p> <p>The second hexadecimal token is ZPWDPTR, a pointer to the password.</p> |
| ACTION : <i>action</i> ZPARMLEN=, ZPTR= <i>token1 token2</i> | <p>This message shows the action parameter from the BIND subcommand.</p> <p>The first hexadecimal token is ZPARMLEN, the length of the parameter.</p> <p>The second hexadecimal token is ZPTR.</p> |

| Message text | Explanation |
|---|--|
| RETAIN : <i>retparm</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | This message shows the retain parameter, <i>retparm</i> , from the BIND subcommand. The first hexadecimal token is ZPDEBIN. The second hexadecimal token is ZPTR. |
| VALIDATE : <i>valparm</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | This message shows the validate parameter, <i>valparm</i> , from the BIND subcommand. The first hexadecimal token is ZPDEBIN. The second hexadecimal token is ZPTR. |
| ACQUIRE : <i>acqparm</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | This message shows the acquire parameter, <i>acqparm</i> , from the BIND subcommand. The first hexadecimal token is ZPDEBIN. The second hexadecimal token is ZPTR. |
| RELEASE : <i>relparm</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | This message shows the release parameter, <i>relparm</i> , from the BIND subcommand. The first hexadecimal token is ZPDEBIN. The second hexadecimal token is ZPTR. |
| ISOLATE : <i>isoparm</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | This message shows the isolate parameter, <i>isoparm</i> , from the BIND subcommand. The first hexadecimal token is ZPDEBIN. The second hexadecimal token is ZPTR. |
| FLAG : <i>flagval</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | This message shows the flag parameter, <i>flagval</i> , from the BIND subcommand. The first hexadecimal token is ZPDEBIN. The second hexadecimal token is ZPTR. |
| LEAVE DSNECP40 R15=, SIBRCODE= <i>token1 token2</i> | CSECT DSNECP40 has completed processing. Control will return to the calling CSECT, DSNECP19. The first hexadecimal token is register 15. The second hexadecimal token is SIBRCODE. |

CSECT: DSNECP41

| Message text | Explanation |
|---|--|
| ENTER DSNECP41 ADDR(SIBRBIND)=, R6= <i>token1 token2</i> | This message announces entry into CSECT DSNECP41, which parses the REBIND subcommand. The first hexadecimal token is the address of the BIND area of the SIB. The second hexadecimal token is R6, the CIB address. |

| Message text | Explanation |
|---|--|
| AFTER IKJPARS, R15= <i>token1 token2</i> | <p>This message is displayed after the call to IKJPARS has taken place.</p> <p>The first hexadecimal token is register 15.</p> <p>The second hexadecimal token is not used.</p> |
| PLAN : <i>planname</i> ZPARMLEN=, ZPTR= <i>token1 token2</i> | <p>This message shows the plan parameter, plan name, from the REBIND subcommand.</p> <p>The first hexadecimal token is ZPARMLEN, the length of the plan name.</p> <p>The second hexadecimal token is ZPTR, the PDE address for the plan parameter.</p> |
| PLAN NAME LOOPTOP ZJ=, SIBRBPSZ= <i>token1 token2</i> | <p>This loop executes once per plan name.</p> <p>The first hexadecimal token is ZJ, the loop index.</p> <p>The second hexadecimal token is SIBRBPSZ, the size for the coming FREEMAIN.</p> |
| FOUND AN ASTERISK ZPARMLEN=, ZPTR= <i>token1 token2</i> | <p>The plan name was an asterisk, indicating 'all' plan names.</p> <p>The hexadecimal tokens are not used.</p> |
| LITTLE LOOPTOP ZI=, ZPARMLEN= <i>token1 token2</i> | <p>This loop executes once for each character in a plan name.</p> <p>The first hexadecimal token is ZI.</p> <p>The second hexadecimal token is ZPARMLEN.</p> |
| PUT PARM INTO LIST ZPARMLEN=<i>token1 token2</i> | <p>A plan name was just added to the plan name list.</p> <p>The first hexadecimal token is ZPARMLEN.</p> <p>The second hexadecimal token is zero.</p> |
| VALIDATE : <i>valparm</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | <p>This message shows the validate parameter, valparm, from the REBIND subcommand.</p> <p>The first hexadecimal token is ZPDEBIN.</p> <p>The second hexadecimal token is ZPTR.</p> |
| ACQUIRE : <i>acqparm</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | <p>This message shows the acquire parameter, acqparm, from the REBIND subcommand.</p> <p>The first hexadecimal token is ZPDEBIN.</p> <p>The second hexadecimal token is ZPTR.</p> |
| RELEASE : <i>relparm</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | <p>This message shows the release parameter, relparm, from the REBIND subcommand.</p> <p>The first hexadecimal token is ZPDEBIN.</p> <p>The second hexadecimal token is ZPTR.</p> |

| Message text | Explanation |
|--|--|
| ISOLATE : <i>isoparm</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | <p>This message shows the isolate parameter, <i>isoparm</i>, from the REBIND subcommand.</p> <p>The first hexadecimal token is ZPDEBIN.</p> <p>The second hexadecimal token is ZPTR.</p> |
| EXPLAIN : NO ZPTR=, ZPARMLN= <i>token1 token2</i> | <p>This message shows that the NO EXPLAIN parameter was detected in the REBIND subcommand.</p> <p>The first hexadecimal token is ZPTR.</p> <p>The second hexadecimal token is ZPARMLN.</p> |
| LEAVE DSNECP41 SIBRCODE=, SIBRBSW= <i>token1 token2</i> | <p>CSECT DSNECP41 has just completed. Control will return to the calling CSECT, DSNECP19.</p> <p>The first hexadecimal token is SIBRCODE, the return code from the REBIND. The second hexadecimal token is SIBRBSW, the REBIND switches.</p> |
| CSECT: DSNECP42 | |
| Message text | Explanation |
| ENTER DSNECP42, SIBFREE= <i>token1 token2</i> | <p>This message announces entry into CSECT DSNECP42, which parses the FREE subcommand.</p> <p>The first hexadecimal token is the address of the FREE area in the SIB.</p> <p>The second hexadecimal token is not used.</p> |
| AFTER IKJPARS, R15= <i>token1 token2</i> | <p>This message is displayed after the call to IKJPARS has taken place.</p> <p>The first hexadecimal token is register 15, the return code from the parse.</p> <p>The second hexadecimal token is zero.</p> |
| BEFORE MEMBER LOOP <i>token1 token2</i> | <p>This message appears before the loop that processes the planid list that the user entered.</p> <p>The hexadecimal tokens are not used.</p> |
| PLAN : <i>planname</i> <i>token1 token2</i> | <p>This message shows the plan parameter, <i>planname</i>, from the FREE subcommand. This message appears once for each PLAN to be freed.</p> <p>The first hexadecimal token is ZPARMLN, the length of the parameter.</p> <p>The second hexadecimal token is zero.</p> |

| Message text | Explanation |
|--|---|
| GETMAIN SUCCESSFUL, ZGMPTR= token1 token2 | <p>The GETMAIN for the area to hold the compacted plan name list was successful.</p> <p>The first hexadecimal token is ZGMPTR, the address of the area just obtained.</p> <p>The second hexadecimal token is not used.</p> |
| PLAN NAME LOOPTOP ZJ=, SIBFRBPS= token1 token2 | <p>This loop executes once per plan name.</p> <p>The first hexadecimal token is ZJ, the loop index.</p> <p>The second hexadecimal token is SIBFRBPS, the size of the previous GETMAIN.</p> |
| FOUND AN ASTERISK token1 token2 | <p>An asterisk was entered in the plan name list.</p> <p>The hexadecimal tokens are not used.</p> |
| LITTLE LOOPTOP ZI=, ZPARMLEN token1 token2 | <p>This loop executes once per character in each plan name.</p> <p>The first hexadecimal token is ZI, the loop index.</p> <p>The second hexadecimal token is ZPARMLEN, the length of the plan name.</p> |
| PUT PARM INTO LIST token1 token2 | <p>A plan name was just added to the plan name list.</p> <p>The hexadecimal tokens are not used.</p> |
| FLAG : flagval ZPDEBIN token1 token2 | <p>This message shows the flag parameter, flagval, from the FREE subcommand.</p> <p>The first hexadecimal token is ZPDEBIN, a parameter presence indicator.</p> <p>The second hexadecimal token is not used.</p> |
| CALL DSNECP30, SIBFREE=, SIBRCODE= token1 token2 | <p>CSECT DSNECP42 is about to call CSECT DSNECP30 to invoke Db2 to do the FREE processing.</p> <p>The first hexadecimal token is the address of the FREE area in the SIB.</p> <p>The second hexadecimal token is SIBRCODE, the FREE return code so far.</p> |
| AFTER DSNECP30, SIBFREE=, SIBRCODE= token1 token2 | <p>CSECT DSNECP42 just regained control after calling CSECT DSNECP30.</p> <p>The first hexadecimal token is the address of SIBFREE.</p> <p>The second hexadecimal token is SIBRCODE.</p> |

| Message text | Explanation |
|---|---|
| LEAVE DSNECP42, SIBRCODE=, SIBFRBSW= <i>token1 token2</i> | CSECT DSNECP42 has completed. Control will return to the calling CSECT, DSNECP19. The first hexadecimal token is SIBRCODE, the FREE return code. The second hexadecimal token is SIBFRBSW. |
| CSECT: DSNECP44 | |
| Message text | Explanation |
| ENTER DSNECP44, CIPPTR= <i>token1 token2</i> | This message announces entry into CSECT DSNECP44, which parses the BIND PACKAGE subcommand. The first hexadecimal token is CIBPTR. The second hexadecimal token is not used. |
| AFTER IKJPARS, R15= <i>token1 token2</i> | This message is displayed after the call to IKJPARS has taken place. The first hexadecimal token is register 15, the return code from the parse. The second hexadecimal token is zero. |
| MEMBER : memname ZPARMLEN=, ZPDEPTR= <i>token1 token2</i> | This message shows the member parameter, memname, from the BIND PACKAGE subcommand. The first hexadecimal token is ZPARMLEN, the length of the parameter. The second hexadecimal token is the PDE address. |
| NO LIBRARY PARM ENTERED ZPDEBIND=, ZPTR= <i>token1 token2</i> | This message indicates that the user did not enter a library parameter. The first hexadecimal token is ZPDEBIN, a parameter presence indicator. The second hexadecimal token is the address of the LIBRARY parameter's PDE. |
| LIBRARY : libname ZPARMLEN=, SIBBPDSN= <i>token1 token2</i> | This message shows the library parameter, libname, from the BIND PACKAGE subcommand. The first hexadecimal token is ZPARMLEN, the name length. The second hexadecimal token is SIBBPDSN, the address of the name. |

| Message text | Explanation |
|---|---|
| PASSWORD : <i>password</i> ZPWDLEN=, ZPWDPTR= <i>token1 token2</i> | <p>This message shows the password parameter from the BIND PACKAGE subcommand.</p> <p>The first hexadecimal token is ZPWDLEN, the length of the password.</p> <p>The second hexadecimal token is ZPWDPTR, a pointer to the password.</p> |
| COPYVER : <i>version</i> ZPARMLEN= <i>token1</i> | <p>This message shows the COPYVER parameter from the BIND PACKAGE subcommand.</p> <p>The first hexadecimal token is ZPARMLEN, the length of the COPYVER parameter.</p> |
| OPTIONS : <i>optionsval</i> ZPDEBIN=, ZPDEPTR= <i>token1 token2</i> ZPARMLEN= <i>token1</i> | <p>This message shows the OPTIONS parameter from the BIND PACKAGE subcommand.</p> <p>The first hexadecimal token is ZPDEBIN, a parameter presence indicator.</p> <p>The second hexadecimal token is the address of the OPTIONS parameter's PDE.</p> |
| ACTION : <i>action</i> ZPARMLEN=, ZPTR= <i>token1</i> <i>token2</i> | <p>This message shows the action parameter from the BIND PACKAGE subcommand.</p> <p>The first hexadecimal token is ZPARMLEN, the length of the parameter.</p> <p>The second hexadecimal token is ZPTR.</p> |
| REPLVER : <i>version</i> ZPARMLEN= <i>token1</i> | <p>This message shows the REPLVER parameter from the BIND PACKAGE subcommand.</p> <p>The first hexadecimal token is ZPARMLEN, the length of the REPLVER parameter.</p> |
| VALIDATE : <i>valparm</i> ZPDEBIN=, ZPTR= <i>token1</i> <i>token2</i> | <p>This message shows the validate parameter, valparm, from the BIND PACKAGE subcommand.</p> <p>The first hexadecimal token is ZPDEBIN.</p> <p>The second hexadecimal token is ZPTR.</p> |
| RELEASE : <i>relparm</i> ZPDEBIN=, ZPTR= <i>token1</i> <i>token2</i> | <p>This message shows the RELEASE parameter, relparm, from the BIND PACKAGE subcommand.</p> <p>The first hexadecimal token is ZPDEBIN.</p> <p>The second hexadecimal token is ZPTR.</p> |
| SQLERROR : <i>sqlerract</i> ZPDEBIN=, ZPTR= <i>token1</i> <i>token2</i> | <p>This message shows the SQLERROR parameter, sqlerract, from the BIND PACKAGE subcommand.</p> <p>The first hexadecimal token is ZPDEBIN.</p> <p>The second hexadecimal token is ZPTR.</p> |

| Message text | Explanation |
|--|---|
| ISOLATION : <i>isoparm</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | This message shows the ISOLATION parameter, <i>isoparm</i> , from the BIND PACKAGE subcommand. The first hexadecimal token is ZPDEBIN. The second hexadecimal token is ZPTR. |
| FLAG : <i>flagval</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | This message shows the FLAG parameter, <i>flagval</i> , from the BIND PACKAGE subcommand. The first hexadecimal token is ZPDEBIN. The second hexadecimal token is ZPTR. |
| EXPLAIN : <i>explval</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | This message shows the EXPLAIN parameter, <i>explval</i> , from the BIND PACKAGE subcommand. The first hexadecimal token is ZPDEBIN. The second hexadecimal token is ZPTR. |
| CURRENTDATA : <i>cdval</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | This message shows the CURRENTDATA parameter, <i>cdval</i> , from the BIND PACKAGE subcommand. The first hexadecimal token is ZPDEBIN. The second hexadecimal token is ZPTR. |
| DYNAMICRULES: <i>dynval</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | This message shows the DYNAMICRULES parameter, <i>dynval</i> , from the BIND PACKAGE subcommand. The first hexadecimal token is ZPDEBIN. The second hexadecimal token is ZPTR. |
| DBPROTOCOL: <i>dbprotval</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | This message shows the DBPROTOCOL parameter, <i>dbprotval</i> , from the BIND PACKAGE subcommand. The first hexadecimal token is ZPDEBIN. The second hexadecimal token is ZPTR. |
| DEFER: <i>defer</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | This message shows the DEFER parameter, <i>defer</i> , from the BIND PACKAGE subcommand. The first hexadecimal token is ZPDEBIN. The second hexadecimal token is ZPTR. |
| NODEFER: <i>nodefer</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | This message shows the NODEFER parameter, <i>nodefer</i> , from the BIND PACKAGE subcommand. The first hexadecimal token is ZPDEBIN. The second hexadecimal token is ZPTR. |
| REOPT: <i>reopt</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | This message shows the REOPT parameter, <i>reopt</i> , from the BIND PACKAGE subcommand. The first hexadecimal token is ZPDEBIN. The second hexadecimal token is ZPTR. |

| Message text | Explanation |
|--|---|
| NOREOPT: <i>noreopt</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | <p>This message shows the NOREOPT parameter, <i>noreopt</i>, from the BIND PACKAGE subcommand.</p> <p>The first hexadecimal token is ZPDEBIN.</p> <p>The second hexadecimal token is ZPTR.</p> |
| KEEPDYNAMIC: <i>keepdynval</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | <p>This message shows the KEEPDYNAMIC parameter, <i>keepdynval</i>, from the BIND PACKAGE subcommand.</p> <p>The first hexadecimal token is ZPDEBIN.</p> <p>The second hexadecimal token is ZPTR.</p> |
| NO PATH PARAMETER ENTERED: ZPDEBIN=, ZPTR= <i>token1 token2</i> | <p>This message is displayed if the PATH is not specified for the BIND PACKAGE subcommand.</p> <p>The first hexadecimal token is ZPDEBIN.</p> <p>The second hexadecimal token is ZPTR.</p> |
| PATH : ZPARMLEN=, PATHPTR= <i>token1 token2</i> | <p>This message displays the schema names in the PATH parameter for the BIND PACKAGE subcommand.</p> <p>The first hexadecimal token is ZPARMLEN.</p> <p>The second hexadecimal token is the pointer to the PATH list.</p> |
| PATH (# SCHEMA NAMES): SIBBPATHN <i>token1</i> | <p>This message displays the number of schema names in the PATH parameter for the BIND PACKAGE subcommand.</p> <p>The first hexadecimal token is the number of schemas in the PATH list.</p> |
| OWNER : <i>ownerid</i> ZPARMLEN= <i>token1</i> | <p>This message shows the OWNER parameter, <i>ownerid</i>, from the BIND PACKAGE subcommand.</p> <p>The first hexadecimal token is ZPARMLEN.</p> |
| QUALIFIER : <i>qualid</i> ZPARMLEN= <i>token1</i> | <p>This message shows the QUALIFIER parameter, <i>qualid</i>, from the BIND PACKAGE subcommand.</p> <p>The first hexadecimal token is ZPARMLEN.</p> |
| DEGREE : <i>degreeprm</i> ZPARMLEN=, ZPTR= <i>token1 token2</i> | <p>This message shows the DEGREE parameter, <i>degreeprm</i>, from the BIND PACKAGE subcommand.</p> <p>The first hexadecimal token is ZPARMLEN, the length of the DEGREE parameter value.</p> <p>The second hexadecimal token is PDE address.</p> |

| Message text | Explanation |
|---|--|
| SIB RELEASE DEPENDENCY MARK IS: <i>token1</i> | This message appears before exit from DSNECP44. The first hexadecimal token is the release dependency character for the BIND command set by this CSECT. |
| LEAVE DSNECP44, R15=, SIBRCODE= <i>token1 token2</i> | CSECT DSNECP44 has completed. Control will return to the calling CSECT, DSNECP19. The first hexadecimal token is the value in register 15. The second hexadecimal token is SIBRCODE, the BIND return code. |
| CSECT: DSNECP45 | |
| Message text | Explanation |
| ENTER DSNECP45, SIBRBIND= <i>token1 token2</i> | This message announces entry into CSECT DSNECP45, which parses the REBIND PACKAGE subcommand. The first hexadecimal token is CIBPTR. The second hexadecimal token is not used. |
| AFTER IKJPARS, R15= <i>token1 token2</i> | This message is displayed after the call to IKJPARS has taken place. The first hexadecimal token is register 15, the return code from the parse. The second hexadecimal token is zero. |
| VALIDATE : <i>valparm</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | This message shows the validate parameter, <i>valparm</i> , from the REBIND PACKAGE subcommand. The first hexadecimal token is ZPDEBIN. The second hexadecimal token is ZPTR. |
| RELEASE : <i>relparm</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | This message shows the RELEASE parameter, <i>relparm</i> , from the REBIND PACKAGE subcommand. The first hexadecimal token is ZPDEBIN. The second hexadecimal token is ZPTR. |
| ISOLATE : <i>isoparm</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | This message shows the ISOLATION parameter, <i>isoparm</i> , from the REBIND PACKAGE subcommand. The first hexadecimal token is ZPDEBIN. The second hexadecimal token is ZPTR. |

| Message text | Explanation |
|--|---|
| FLAG : <i>flagval</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | This message shows the FLAG parameter, <i>flagval</i> , from the REBIND PACKAGE subcommand. The first hexadecimal token is ZPDEBIN. The second hexadecimal token is ZPTR. |
| EXPLAIN : <i>explval</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | This message shows the EXPLAIN parameter, <i>explval</i> , from the REBIND PACKAGE subcommand. The first hexadecimal token is ZPDEBIN. The second hexadecimal token is ZPTR. |
| DYNAMICRULES: <i>dynval</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | This message shows the DYNAMICRULES parameter, <i>dynval</i> , from the REBIND PACKAGE subcommand. The first hexadecimal token is ZPDEBIN. The second hexadecimal token is ZPTR. |
| DBPROTOCOL: <i>dbprotval</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | This message shows the DBPROTOCOL parameter, <i>dbprotval</i> , from the REBIND PACKAGE subcommand. The first hexadecimal token is ZPDEBIN. The second hexadecimal token is ZPTR. |
| CURRENTDATA : <i>cdval</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | This message shows the CURRENTDATA parameter, <i>cdval</i> , from the REBIND PACKAGE subcommand. The first hexadecimal token is ZPDEBIN. The second hexadecimal token is ZPTR. |
| DEFER: <i>defer</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | This message shows the DEFER parameter, <i>defer</i> , from the REBIND PACKAGE subcommand. The first hexadecimal token is ZPDEBIN. The second hexadecimal token is ZPTR. |
| NODEFER: <i>nodefer</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | This message shows the NODEFER parameter, <i>nodefer</i> , from the REBIND PACKAGE subcommand. The first hexadecimal token is ZPDEBIN. The second hexadecimal token is ZPTR. |
| REOPT: <i>reopt</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | This message shows the REOPT parameter, <i>reopt</i> , from the REBIND PACKAGE subcommand. The first hexadecimal token is ZPDEBIN. The second hexadecimal token is ZPTR. |

| Message text | Explanation |
|--|--|
| NOREOPT: <i>noreopt</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | This message shows the NOREOPT parameter, <i>noreopt</i> , from the REBIND PACKAGE subcommand. The first hexadecimal token is ZPDEBIN. The second hexadecimal token is ZPTR. |
| KEEPDYNAMIC: <i>keepdynval</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | This message shows the KEEPDYNAMIC parameter, <i>keepdynval</i> , from the REBIND PACKAGE subcommand. The first hexadecimal token is ZPDEBIN. The second hexadecimal token is ZPTR. |
| NO PATH PARAMETER ENTERED: ZPDEBIN=, ZPTR= <i>token1 token2</i> | This message is displayed if the PATH is not specified for the REBIND PACKAGE subcommand. The first hexadecimal token is ZPDEBIN. The second hexadecimal token is ZPTR. |
| PATH : ZPARMLEN=, PATHPTR= <i>token1 token2</i> | This message displays the schema names in the PATH parameter for the REBIND PACKAGE subcommand. The first hexadecimal token is ZPARMLEN. The second hexadecimal token is the pointer to the PATH list. |
| PATH (# SCHEMA NAMES): SIBBPATHN <i>token1</i> | This message displays the number of schema names in the PATH parameter for the REBIND PACKAGE subcommand. The first hexadecimal token is the number of schemas in the PATH list. |
| PATHDEFAULT: ZPDEBIN, SIBRPATHP <i>token1 token2</i> | This message displays the PATHDEFAULT parameter for the REBIND PACKAGE subcommand. The first hexadecimal token is ZPTR. The second hexadecimal token is a pointer to the PATH list. |
| OWNER : <i>ownerid</i> ZPARMLEN= <i>token1</i> | This message shows the OWNER parameter, <i>ownerid</i> , from the REBIND PACKAGE subcommand. The first hexadecimal token is ZPARMLEN. |
| QUALIFIER : <i>qualid</i> ZPARMLEN= <i>token1</i> | This message shows the QUALIFIER parameter, <i>qualid</i> , from the REBIND PACKAGE subcommand. The first hexadecimal token is ZPARMLEN. |

| Message text | Explanation |
|---|--|
| BEFORE CALL DSNECP30 SIBRBIND=, SIBRCODE= <i>token1 token2</i> | <p>CSECT DSNECP45 is about to call CSECT DSNECP30 to make the Db2 call to perform the REBIND PACKAGE.</p> <p>The first hexadecimal token is SIBRCODE.</p> <p>The second hexadecimal token is SIBRCODE, the highest return code so far.</p> |
| AFTER CALL DSNECP30, SIBRBIND=, SIBRCODE= <i>token1 token2</i> | <p>CSECT DSNECP45 just regained control after calling CSECT DSNECP30.</p> <p>The first hexadecimal token is the address of SIBRBIND.</p> <p>The second hexadecimal token is SIBRCODE, the highest return code so far.</p> |
| AFTER PARSE <i>token1</i> | <p>This message after parsing of the REBIND PACKAGE subcommand is complete.</p> <p>The first hexadecimal token is the release dependency character for the BIND command set by this CSECT.</p> |
| SIB RELEASE DEPENDENCY MARK IS: <i>token1</i> | <p>This message appears before exit from DSNECP45.</p> <p>The first hexadecimal token is the release dependency character for the BIND command set by this CSECT.</p> |
| LEAVE DSNECP45, R15=, SIBRCODE= <i>token1 token2</i> | <p>CSECT DSNECP45 has completed. Control will return to the calling CSECT, DSNECP19.</p> <p>The first hexadecimal token is the value in register 15.</p> <p>The second hexadecimal token is SIBRCODE, the BIND return code.</p> |
| CSECT: DSNECP46 | |
| Message text | Explanation |
| ENTER DSNECP46, SIBFREE PACKAGE= <i>token1 token2</i> | <p>This message announces entry into CSECT DSNECP46, which parses the FREE PACKAGE subcommand.</p> <p>The first hexadecimal token is the address of the FREE PACKAGE area in the SIB.</p> <p>The second hexadecimal token is not used.</p> |
| AFTER IKJPARS, R15= <i>token1 token2</i> | <p>This message is displayed after the call to IKJPARS has taken place.</p> <p>The first hexadecimal token is register 15, the return code from the parse.</p> |

| Message text | Explanation |
|---|---|
| FLAG : <i>flagval</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | <p>This message shows the FLAG parameter, <i>flagval</i>, from the FREE PACKAGE subcommand.</p> <p>The first hexadecimal token is ZPDEBIN.</p> <p>The second hexadecimal token is ZPTR.</p> |
| BEFORE CALL DSNECP30, SIBFREE PACKAGE=, SIBRCODE= <i>token1 token2</i> | <p>CSECT DSNECP46 is about to call CSECT DSNECP30 to invoke Db2 to do the FREE PACKAGE processing.</p> <p>The first hexadecimal token is the address of the FREE PACKAGE area in the SIB.</p> <p>The second hexadecimal token is SIBRCODE, the FREE PACKAGE return code so far.</p> |
| AFTER CALL DSNECP30, SIBFREE PACKAGE=, SIBRCODE= <i>token1 token2</i> | <p>CSECT DSNECP46 just regained control after calling CSECT DSNECP30.</p> <p>The first hexadecimal token is the address of SIBFREE PACKAGE.</p> <p>The second hexadecimal token is SIBRCODE.</p> |
| LEAVE DSNECP46, SIBRCODE=, SIBFRBSW= <i>token1 token2</i> | <p>CSECT DSNECP46 has completed. Control will return to the calling CSECT, DSNECP19.</p> <p>The first hexadecimal token is SIBRCODE, the FREE PACKAGE return code.</p> <p>The second hexadecimal token is SIBFRBSW.</p> |
| CSECT: DSNECP47 | |
| Message text | Explanation |
| ENTER DSNECP47, SIBRBIND= <i>token1 token2</i> | <p>This message announces entry into CSECT DSNECP47, which parses the REBIND TRIGGER PACKAGE subcommand.</p> <p>The first hexadecimal token is CIBPTR.</p> <p>The second hexadecimal token is not used.</p> |
| AFTER IKJPARS, R15= <i>token1 token2</i> | <p>This message is displayed after the call to IKJPARS has taken place.</p> <p>The first hexadecimal token is register 15, the return code from the parse.</p> <p>The second hexadecimal token is zero.</p> |
| TRIGGER : <i>trigname</i> ZPDEBIN=, ZPTR= <i>token1 token2</i> | <p>This message shows the TRIGGER parameter, <i>trigname</i>, from the REBIND TRIGGER PACKAGE subcommand.</p> <p>The first hexadecimal token is ZPDEBIN.</p> <p>The second hexadecimal token is ZPTR.</p> |

| Message text | Explanation |
|--|--|
| RELEASE : relparm ZPDEBIN=, ZPTR= token1 token2 | <p>This message shows the RELEASE parameter, relparm, from the REBIND TRIGGER PACKAGE subcommand.</p> <p>The first hexadecimal token is ZPDEBIN.</p> <p>The second hexadecimal token is ZPTR.</p> |
| ISOLATION : isoparm ZPDEBIN=, ZPTR= token1 token2 | <p>This message shows the ISOLATION parameter, isoparm, from the REBIND TRIGGER PACKAGE subcommand.</p> <p>The first hexadecimal token is ZPDEBIN.</p> <p>The second hexadecimal token is ZPTR.</p> |
| FLAG : flagval ZPDEBIN=, ZPTR= token1 token2 | <p>This message shows the FLAG parameter, flagval, from the REBIND TRIGGER PACKAGE subcommand.</p> <p>The first hexadecimal token is ZPDEBIN.</p> <p>The second hexadecimal token is ZPTR.</p> |
| EXPLAIN : explval ZPDEBIN=, ZPTR= token1 token2 | <p>This message shows the EXPLAIN parameter, explval, from the REBIND TRIGGER PACKAGE subcommand.</p> <p>The first hexadecimal token is ZPDEBIN.</p> <p>The second hexadecimal token is ZPTR.</p> |
| CURRENTDATA : cdval ZPDEBIN=, ZPTR= token1 token2 | <p>This message shows the CURRENTDATA parameter, cdval, from the REBIND TRIGGER PACKAGE subcommand.</p> <p>The first hexadecimal token is ZPDEBIN.</p> <p>The second hexadecimal token is ZPTR.</p> |
| BEFORE CALL DSNECP30 SIBRBIND=, SIBRCODE= token1 token2 | <p>CSECT DSNECP47 is about to call CSECT DSNECP30 to make the Db2 call to perform the REBIND TRIGGER PACKAGE.</p> <p>The first hexadecimal token is SIBRCODE.</p> <p>The second hexadecimal token is SIBRCODE, the highest return code so far.</p> |
| AFTER CALL DSNECP30, SIBRBIND=, SIBRCODE= token1 token2 | <p>CSECT DSNECP47 just regained control after calling CSECT DSNECP30.</p> <p>The first hexadecimal token is the address of SIBRBIND.</p> <p>The second hexadecimal token is SIBRCODE, the highest return code so far.</p> |

| Message text | Explanation |
|--|--|
| AFTER PARSE <i>token1</i> | <p>This message after parsing of the REBIND TRIGGER PACKAGE subcommand is complete.</p> <p>The first hexadecimal token is the release dependency character for the BIND command set by this CSECT.</p> |
| SIB RELEASE DEPENDENCY MARK IS: <i>token1</i> | <p>This message appears before exit from DSNECP47.</p> <p>The first hexadecimal token is the release dependency character for the BIND command set by this CSECT.</p> |
| LEAVE DSNECP47, R15=, SIBRCODE= <i>token1 token2</i> | <p>CSECT DSNECP47 has completed. Control will return to the calling CSECT, DSNECP19.</p> <p>The first hexadecimal token is the value in register 15.</p> <p>The second hexadecimal token is SIBRCODE, the BIND return code.</p> |
| CSECT: DSNECP50 | |
| Message text | Explanation |
| ENTER DSNECP50 ZPTR=, SIBRCODE= <i>token1 token2</i> | <p>This message announces entry into CSECT DSNECP50, which validates the BIND PLAN parameters.</p> <p>The first hexadecimal token is the ZPTR, the pointer to the parameter list.</p> <p>The second hexadecimal token is SIBRCODE, the highest return code so far.</p> |
| OWNER : <i>ownerid</i> ZPARMLEN= <i>token1</i> | <p>This message shows the OWNER parameter, <i>ownerid</i>, from the BIND PLAN subcommand.</p> <p>The first hexadecimal token is ZPARMLEN.</p> |
| QUALIFIER : <i>qualid</i> ZPARMLEN= <i>token1</i> | <p>This message shows the QUALIFIER parameter, <i>qualid</i>, from the BIND PLAN subcommand.</p> <p>The first hexadecimal token is ZPARMLEN.</p> |
| OPTHINT : <i>qoptprm</i> ZPARMLEN=, ZPDEPTR= <i>token1 token2</i> | <p>This message shows the OPTHINT parameter, <i>qoptprt</i>, from the BIND PLAN subcommand.</p> <p>The first hexadecimal token is ZPARMLEN.</p> <p>The second hexadecimal token is the address of the OPTHINT parameter's PDE.</p> |

| Message text | Explanation |
|--|--|
| PKLIST : <i>pklist</i> ZPARMLEN=, ZPARAM_PTR= <i>token1 token2</i> | <p>This message shows an entry in the PKLIST parameter, <i>pklist</i>, from the BIND PLAN subcommand.</p> <p>The first hexadecimal token is ZPARMLEN.</p> <p>The second hexadecimal token is ZPARAM_PTR, the pointer to the current entry in the package list.</p> |
| # IN PKLIST : SIBBPKLN= <i>token1</i> | <p>This message shows the number of entries in the package list.</p> <p>The first hexadecimal token is SIBBPKLN, the number of entries in the package list.</p> |
| CURRSERV : <i>currentserver</i> ZPARMLEN=, ZPDEPTR=, <i>token1 token2</i> | <p>This message shows the CURRENTSERVER parameter, <i>currentserver</i>, from the BIND PLAN subcommand.</p> <p>The first hexadecimal token is ZPARMLEN.</p> <p>The second hexadecimal token is the address of the CURRENTSERVER parameter's PDE.</p> |
| DEGREE : <i>degree</i> ZPARMLEN=, ZPDEPTR=, <i>token1 token2</i> | <p>This message shows the DEGREE parameter, <i>degree</i>, from the BIND PLAN subcommand.</p> <p>The first hexadecimal token is ZPARMLEN.</p> <p>The second hexadecimal token is the address of the DEGREE parameter's PDE.</p> |
| ENABLE : <i>enableparm</i> ZPARMLEN= <i>token1</i> | <p>This message shows the ENABLE parameter, <i>enableparm</i>, from the BIND PLAN subcommand.</p> <p>The first hexadecimal token is ZPARMLEN.</p> |
| DISABLE : <i>disableparm</i> ZPARMLEN= <i>token1</i> | <p>This message shows the DISABLE parameter, <i>disableparm</i>, from the BIND PLAN subcommand.</p> <p>The first hexadecimal token is ZPARMLEN.</p> |
| CICSCON : <i>cicsconnid</i> ZPARMLEN= <i>token1</i> | <p>This message shows the CICS connection ID subparameter, <i>cicsconnid</i>, of the ENABLE(CICS(<i>cicsconnid</i>)) parameter from the BIND PLAN subcommand.</p> <p>The first hexadecimal token is ZPARMLEN.</p> |
| IMSBCON : <i>imsid</i> ZPARMLEN= <i>token1</i> | <p>This message shows the IMS BMP IMSID subparameter, <i>imsid</i>, of the ENABLE(IMSBMP(<i>imsid</i>)) parameter from the BIND PLAN subcommand.</p> <p>The first hexadecimal token is ZPARMLEN.</p> |

| Message text | Explanation |
|---|--|
| IMSMCON : <i>imsid</i> ZPARMLEN= <i>token1</i> | This message shows the IMS MPP IMSID subparameter, <i>imsid</i> , of the ENABLE(IMSMPP(<i>imsid</i>)) parameter from the BIND PLAN subcommand. The first hexadecimal token is ZPARMLEN. |
| DLIBATC : <i>connection-name</i> ZPARMLEN= <i>token1</i> | This message shows the connection ID subparameter, <i>connection-name</i> , of the ENABLE(DLIBATCH(<i>connection-name</i>)) parameter from the BIND PLAN subcommand. The first hexadecimal token is ZPARMLEN. |
| LEAVE DSNECP50 SIBRCODE= <i>token1</i> | CSECT DSNECP50 has completed processing. Control will return to the calling CSECT, DSNECP40. The first hexadecimal token is SIBRCODE, the highest return code. |
| CSECT: DSNECP51 | |
| Message text | Explanation |
| ENTER DSNECP51 CIBPTR= <i>token1</i> | This message announces entry into CSECT DSNECP51, which validates the REBIND PLAN parameters. The first hexadecimal token is the CIB address. |
| NUM OF PLAN : SIBRBPN= <i>token1</i> | This message shows the number of plans to rebind. The first hexadecimal token, SIBRBPN, is the number of plans to rebind. |
| PLAN NAME: ZPARMLEN=, ZJ= <i>token1 token2</i> | This message shows the name of a plan to rebind. The first hexadecimal token, ZPARMLEN, is the length of the plan name. The second hexadecimal token, ZJ, is the number of the plan name in the list of plans to rebind. |
| FOUND AN ASTERISK ZPARMLEN=, ZPDEPTR= <i>token1 token2</i> | This message indicates that the list of plans to rebind contains an asterisk. The first hexadecimal token, ZPARMLEN, is 1. The second hexadecimal token, ZPDEPTR, is the pointer to the PDE for the PLAN parameter. |
| PUT PARM INTO LIST: <i>planid</i> ZPARMLEN= <i>token1</i> | This message shows the ID of a plan in the list of plans to rebind. The first hexadecimal token, ZPARMLEN, is the length of the plan name. |

| Message text | Explanation |
|--|--|
| OWNER : <i>ownerid</i> ZPARMLEN= <i>token1</i> | This message shows the OWNER parameter, ownerid, from the REBIND PLAN subcommand. The first hexadecimal token is ZPARMLEN. |
| QUALIFIER : <i>qualid</i> ZPARMLEN= <i>token1</i> | This message shows the QUALIFIER parameter, qualid, from the REBIND PLAN subcommand. The first hexadecimal token is ZPARMLEN. |
| PKLIST : <i>pklist</i> ZPARMLEN=, ZPARAM_PTR= <i>token1 token2</i> | This message shows an entry in the PKLIST parameter, pklist, from the REBIND PLAN subcommand. The first hexadecimal token is ZPARMLEN. The second hexadecimal token is ZPARAM_PTR, the pointer to the current entry in the package list. |
| # IN PKLIST : SIBBPKN= <i>token1</i> | This message shows the number of entries in the package list. The first hexadecimal token is SIBBPKN, the number of entries in the package list. |
| CURRSERV : <i>currentserver</i> ZPARMLEN=, ZPDEPTR=, <i>token1 token2</i> | This message shows the CURRENTSERVER parameter, currentserver, from the REBIND PLAN subcommand. The first hexadecimal token is ZPARMLEN. The second hexadecimal token is the address of the CURRENTSERVER parameter's PDE. |
| DEGREE : <i>degree</i> ZPARMLEN=, ZPDEPTR=, <i>token1 token2</i> | This message shows the DEGREE parameter, degree, from the REBIND PLAN subcommand. The first hexadecimal token is ZPARMLEN. The second hexadecimal token is the address of the DEGREE parameter's PDE. |
| ENABLE : <i>enableparm</i> ZPARMLEN= <i>token1</i> | This message shows the ENABLE parameter, enableparm, from the REBIND PLAN subcommand. The first hexadecimal token is ZPARMLEN. |
| DISABLE : <i>disableparm</i> ZPARMLEN= <i>token1</i> | This message shows the DISABLE parameter, disableparm, from the REBIND PLAN subcommand. The first hexadecimal token is ZPARMLEN. |
| CICS CON NUM: SIBRCNUM= <i>token1</i> | This message shows the number of connection ID entries for the ENABLE(CICS(cicsconnid)) parameter from the REBIND PLAN subcommand. The first hexadecimal token is ZPARMLEN. |

| Message text | Explanation |
|--|---|
| IMSBMP CON NUM: SIBRINUM= token1 | <p>This message shows the number of IMSBMP entries for the ENABLE(IMSBMP(imsid)) parameter from the REBIND PLAN subcommand.</p> <p>The first hexadecimal token is SIBRINUM, the number of IMSBMP entries.</p> |
| IMSBMP: imsid ZPARMLEN= token1 | <p>This message shows the IMS BMP IMSID subparameter, imsid, of the ENABLE(IMSBMP(imsid)) parameter from the REBIND PLAN subcommand.</p> <p>The first hexadecimal token is ZPARMLEN.</p> |
| IMSMPP CON NUM: SIBRIMUM= token1 | <p>This message shows the number of IMSMPP entries for the ENABLE(IMSMPP(imsid)) parameter from the REBIND PLAN subcommand.</p> <p>The first hexadecimal token is SIBRIMUM, the number of IMSMPP entries.</p> |
| IMSMPP: imsid ZPARMLEN= token1 | <p>This message shows the IMS MPP IMSID subparameter, imsid, of the ENABLE(IMSMPP(imsid)) parameter from the REBIND PLAN subcommand.</p> <p>The first hexadecimal token is ZPARMLEN.</p> |
| DLIBATCH CON NUM: SIBRDNUM= token1 | <p>This message shows the number of DLIBATCH entries for the ENABLE(DLIBATCH(connection-name)) parameter from the REBIND PLAN subcommand.</p> <p>The first hexadecimal token is SIBRDNUM, the number of DLIBATCH entries.</p> |
| DLIBATC: connection-name ZPARMLEN= token1 | <p>This message shows the connection ID subparameter, connection-name, of the ENABLE(DLIBATCH(connection-name)) parameter from the REBIND PLAN subcommand.</p> <p>The first hexadecimal token is ZPARMLEN.</p> |
| LEAVE DSNECP51 SIBRCODE= token1 | <p>CSECT DSNECP51 has completed processing. Control will return to the calling CSECT, DSNECP41.</p> <p>The first hexadecimal token is SIBRCODE, the highest return code.</p> |
| CSECT: DSNECP54 | |
| Message text | Explanation |
| ENTER DSNECP54 CIBPTR= token1 | <p>This message announces entry into CSECT DSNECP54, which validates the BIND PACKAGE parameters.</p> <p>The first hexadecimal token is the CIBPTR.</p> |

| Message text | Explanation |
|--|---|
| PACKAGE: <i>package</i> ZPARMLEN=, ZPDEPTR= <i>token1 token2</i> | This message shows the PACKAGE parameter, package, from the BIND PACKAGE subcommand. The first hexadecimal token is ZPARMLEN. The second hexadecimal token is the pointer to the PDE for the PACKAGE parameter. |
| COPY ID: <i>package</i> ZPARMLEN=, ZPDEPTR= <i>token1 token2</i> | IF the COPY parameter is specified, this message shows the name of the package to copy. The first hexadecimal token is ZPARMLEN. The second hexadecimal token is the pointer to the PDE for the COPY parameter. |
| COPYVER: <i>version</i> ZPARMLEN= <i>token1</i> | This message shows the COPYVER parameter, copyver, from the BIND PACKAGE subcommand. The first hexadecimal token is ZPARMLEN. |
| OPTIONS : <i>optionsval</i> ZPARMLEN= <i>token1</i> | This message shows the OPTIONS parameter from the BIND PACKAGE subcommand. The first hexadecimal token is ZPARMLEN. |
| REPLVER : <i>version</i> ZPARMLEN= <i>token1</i> | This message shows the REPLVER parameter, version, from the BIND PACKAGE subcommand. The first hexadecimal token is ZPARMLEN. |
| OWNER : <i>ownerid</i> ZPARMLEN= <i>token1</i> | This message shows the OWNER parameter, ownerid, from the BIND PACKAGE subcommand. The first hexadecimal token is ZPARMLEN. |
| QUALIFIER : <i>qualid</i> ZPARMLEN= <i>token1</i> | This message shows the QUALIFIER parameter, qualid, from the BIND PACKAGE subcommand. The first hexadecimal token is ZPARMLEN. |
| OPTHINT : <i>qoptprm</i> ZPARMLEN= <i>token1</i> | This message shows the OPTHINT parameter, qoptprt, from the BIND PACKAGE subcommand. The first hexadecimal token is ZPARMLEN. The second hexadecimal token is the address of the OPTHINT parameter's PDE. |
| DEGREE : <i>degree</i> ZPARMLEN=, ZPDEPTR= <i>token1 token2</i> | This message shows the DEGREE parameter, degree, from the BIND PACKAGE subcommand. The first hexadecimal token is ZPARMLEN. The second hexadecimal token is the address of the DEGREE parameter's PDE. |

| Message text | Explanation |
|--|---|
| ENABLE : <i>enableparm</i> ZPARMLEN= <i>token1</i> | <p>This message shows the ENABLE parameter, enableparm, from the BIND PACKAGE subcommand.</p> <p>The first hexadecimal token is ZPARMLEN.</p> |
| DISABLE : <i>disableparm</i> ZPARMLEN= <i>token1</i> | <p>This message shows the DISABLE parameter, disableparm, from the BIND PACKAGE subcommand.</p> <p>The first hexadecimal token is ZPARMLEN.</p> |
| CICSCON : <i>cicsconnid</i> ZPARMLEN= <i>token1</i> | <p>This message shows the CICS connection ID subparameter, cicsconnid, of the ENABLE(CICS(cicsconnid)) parameter from the BIND PACKAGE subcommand.</p> <p>The first hexadecimal token is ZPARMLEN.</p> |
| IMSBCON : <i>imsid</i> ZPARMLEN= <i>token1</i> | <p>This message shows the IMS BMP IMSID subparameter, imsid, of the ENABLE(IMSBMP(imsid)) parameter from the BIND PACKAGE subcommand.</p> <p>The first hexadecimal token is ZPARMLEN.</p> |
| IMSMCON : <i>imsid</i> ZPARMLEN= <i>token1</i> | <p>This message shows the IMS MPP IMSID subparameter, imsid, of the ENABLE(IMSMPP(imsid)) parameter from the BIND PACKAGE subcommand.</p> <p>The first hexadecimal token is ZPARMLEN.</p> |
| DLIBATC : <i>connection-name</i> ZPARMLEN= <i>token1</i> | <p>This message shows the connection ID subparameter, connection-name, of the ENABLE(DLIBATCH(connection-name)) parameter from the BIND PACKAGE subcommand.</p> <p>The first hexadecimal token is ZPARMLEN.</p> |
| REMOTE : <i>connection-name</i> ZPARMLEN=, ZPDEPTR= <i>token1</i> <i>token2</i> | <p>This message shows the connection ID subparameter, connection-name, of the ENABLE(REMOTE(location-name)) or ENABLE(REMOTE(<luname>)) parameter from the BIND PACKAGE subcommand.</p> <p>The first hexadecimal token is ZPARMLEN.</p> <p>The first hexadecimal token is the pointer to the PDE for ENABLE(REMOTE(location-name)) or ENABLE(REMOTE(<luname>)).</p> |
| LEAVE DSNECP54 SIBRCODE= <i>token1</i> | <p>CSECT DSNECP54 has completed processing. Control will return to the calling CSECT, DSNECP44.</p> <p>The first hexadecimal token is SIBRCODE, the highest return code.</p> |

CSECT: DSNECP55

| Message text | Explanation |
|--|---|
| ENTER DSNECP55 CIBPTR= <i>token1</i> | This message announces entry into CSECT DSNECP55, which validates the REBIND PACKAGE parameters. The first hexadecimal token is CIBPTR. |
| OWNER : <i>ownerid</i> ZPARMLEN= <i>token1</i> | This message shows the OWNER parameter, <i>ownerid</i> , from the REBIND PACKAGE subcommand. The first hexadecimal token is ZPARMLEN. |
| QUALIFIER : <i>qualid</i> ZPARMLEN= <i>token1</i> | This message shows the QUALIFIER parameter, <i>qualid</i> , from the REBIND PACKAGE subcommand. The first hexadecimal token is ZPARMLEN. |
| OPTHINT : <i>qoptprm</i> ZPARMLEN=, ZPDEPTR= <i>token1 token2</i> | This message shows the OPTHINT parameter, <i>qoptprt</i> , from the REBIND PACKAGE subcommand. The first hexadecimal token is ZPARMLEN. The second hexadecimal token is the address of the OPTHINT parameter's PDE. |
| DEGREE : <i>degree</i> ZPARMLEN=, ZPDEPTR= <i>token1 token2</i> | This message shows the DEGREE parameter, <i>degree</i> , from the REBIND PACKAGE subcommand. The first hexadecimal token is ZPARMLEN. The second hexadecimal token is the address of the DEGREE parameter's PDE. |
| ENABLE : <i>enableparm</i> ZPARMLEN= <i>token1</i> | This message shows the ENABLE parameter, <i>enableparm</i> , from the REBIND PACKAGE subcommand. The first hexadecimal token is ZPARMLEN. |
| DISABLE : <i>disableparm</i> ZPARMLEN= <i>token1</i> | This message shows the DISABLE parameter, <i>disableparm</i> , from the REBIND PACKAGE subcommand. The first hexadecimal token is ZPARMLEN. |
| CICS CON NUM: SIBRCNUM= <i>token1</i> | This message shows the number of connection ID entries for the ENABLE(CICS(<i>cicsconnid</i>)) parameter from the REBIND PACKAGE subcommand. The first hexadecimal token is ZPARMLEN. |
| PUT PARM INTO LIST ZPARMLEN=<i>token1</i> | This message is issued when a connection ID is added to the CICS connection ID list. The first hexadecimal token is ZPARMLEN. |

| Message text | Explanation |
|--|---|
| IMSBMP CON NUM: SIBRINUM= <i>token1</i> | <p>This message shows the number of IMSBMP entries for the ENABLE(IMSBMP(imsid)) parameter from the REBIND PACKAGE subcommand.</p> <p>The first hexadecimal token is SIBRINUM, the number of IMSBMP entries.</p> |
| IMSBMP: <i>imsid</i> ZPARMLEN= <i>token1</i> | <p>This message shows the IMS BMP IMSID subparameter, imsid, of the ENABLE(IMSBMP(imsid)) parameter from the REBIND PACKAGE subcommand.</p> <p>The first hexadecimal token is ZPARMLEN.</p> |
| IMSMPP CON NUM: SIBRIMUM= <i>token1</i> | <p>This message shows the number of IMSMPP entries for the ENABLE(IMSMPP(imsid)) parameter from the REBIND PACKAGE subcommand.</p> <p>The first hexadecimal token is SIBRIMUM, the number of IMSMPP entries.</p> |
| IMSMPP: <i>imsid</i> ZPARMLEN= <i>token1</i> | <p>This message shows the IMS MPP IMSID subparameter, imsid, of the ENABLE(IMSMPP(imsid)) parameter from the REBIND PACKAGE subcommand.</p> <p>The first hexadecimal token is ZPARMLEN.</p> |
| DLIBATCH CON NUM: SIBRDNUM= <i>token1</i> | <p>This message shows the number of DLIBATCH entries for the ENABLE(DLIBATCH(connection-name)) parameter from the REBIND PACKAGE subcommand.</p> <p>The first hexadecimal token is SIBRDNUM, the number of DLIBATCH entries.</p> |
| DLIBATC: <i>connection-name</i> ZPARMLEN= <i>token1</i> | <p>This message shows the connection ID subparameter, connection-name, of the ENABLE(DLIBATCH(connection-name)) parameter from the REBIND PACKAGE subcommand.</p> <p>The first hexadecimal token is ZPARMLEN.</p> |
| REMOTE CON NUM: SIBRSNUM=, SIBRSNUM <i>token1</i> | <p>This message shows the number of REMOTE entries for the ENABLE(REMOTE(location or <luname>)) parameter from the REBIND PACKAGE subcommand.</p> <p>The first hexadecimal token is SIBRSNUM, the number of REMOTE entries.</p> |

| Message text | Explanation |
|--|--|
| REMOTE:location or <luname> ZPARMLEN=, ZPDEPTR= token1 token2 | <p>This message shows the connection ID subparameter, connection-name, of the ENABLE(REMOTE(location or <luname>)) parameter from the REBIND PACKAGE subcommand.</p> <p>The first hexadecimal token is ZPARMLEN.</p> <p>The second hexadecimal token is the pointer to the PDE for ENABLE(REMOTE(location or <luname>)).</p> |
| LEAVE DSNECP55 SIBRCODE= token1 | <p>CSECT DSNECP55 has completed processing. Control will return to the calling CSECT, DSNECP45.</p> <p>The first hexadecimal token is SIBRCODE, the highest return code.</p> |
| CSECT: DSNECP60 | |
| Message text | Explanation |
| ENTER DSNECP60, CIBPTR=, ADDR(SIBDCL)= token1 token2 | <p>This message announces entry into CSECT DSNECP60, where a SQL declaration will be written.</p> <p>The first hexadecimal token is the address of the CIB.</p> <p>The second hexadecimal token is the address of the DCLGEN area in the SIB.</p> |
| WROTE "EXEC SQL DCL", SIBDOFF= token1 token2 | <p>This message indicates that CSECT DSNECP60 has begun writing the very first part of the SQL declaration.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is not used.</p> |
| WROTE FIRST LINE, SIBDOFF= token1 token2 | <p>The first line of SQL output has been written.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is not used.</p> |
| BEGIN BIG LOOP, SIBDOFF=, ZCOL= token1 token2 | <p>Execution is now at the top of the main loop in this CSECT.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is ZCOL, the column number being processed.</p> |

| Message text | Explanation |
|---|--|
| BEFORE DSNECP67 <i>sqlname</i> SIBDOFF=, ZCOL= <i>token1 token2</i> | <p>CSECT DSNECP60 is about to write a field specification. The name of the field is <i>sqlname</i>.</p> <p>The first hexadecimal token is SIBDOFF, the current offset into the output buffer.</p> <p>The second hexadecimal token is ZCOL, the number of the column being processed.</p> |
| BEFORE LABEL <i>sqlname</i> SQLNAMEL(ZCOL+SQLD)=, SQLD= <i>token1 token2</i> | <p>DSNECP61 is about to write a label. The label is <i>sqlname</i> (which can contain unprintable characters).</p> <p>The first hexadecimal token is the length of the label.</p> <p>The second hexadecimal token is the number of columns in the table.</p> |
| BEFORE FIELD SPECIFICATION, SIBDOFF=, ZCOL= <i>token1 token2</i> | <p>DCLGEN is about to write a field type specification.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is ZCOL, the column number being processed.</p> |
| WROTE NOT NULL, SIBDOFF=, ZCOL= <i>token1 token2</i> | <p>The string 'NOT NULL' was just written to the output buffer.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is ZCOL, the column number being processed.</p> |
| DID NOT WRITE NOT NULL, SIBDOFF=, ZCOL= <i>token1 token2</i> | <p>The string 'NOT NULL' was not written to the output buffer.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is ZCOL, the column number being processed.</p> |
| LOOP BOTTOM, SIBDOFF=, ZCOL= <i>token1 token2</i> | <p>CSECT DSNECP60 is now executing at the bottom of its main loop.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is ZCOL, the column number being processed.</p> |

| Message text | Explanation |
|---|--|
| LEAVE DSNECP60, SIBDFLAG=, SIBDOFF= <i>token1 token2</i> | CSECT DSNECP60 has completed. Control will return to the calling CSECT, DSNECP14. The first hexadecimal token is SIBDFLAG, the DCLGEN flag byte. The second hexadecimal token is SIBDOFF, the offset into the output buffer. |
| FOUND NON-VANILLA CHARACTER IN zbuf <i>token1 token2</i> | CSECT DSNECP60 found a character that is not an upper case alphanumeric as it scanned an identifier. The identifier is zbuf. The hexadecimal tokens are zero. |
| VANILLA IDENTIFIER: zbuf <i>token1 token2</i> | CSECT DSNECP60 found a character that is an upper case alphanumeric as it scanned an identifier. The identifier is zbuf. The hexadecimal tokens are zero. |
| CSECT: DSNECP61 | |
| Message text | Explanation |
| ENTER DSNECP61, CIBPTR=, ADDR(SIBDCL)= <i>token1 token2</i> | This message announces entry into CSECT DSNECP61, which writes a COBOL declaration. The first hexadecimal token is the address of the CIB. The second hexadecimal token is the address of the DCLGEN area in the SIB. |
| BEFORE WRITE LINE OF ASTERISKS, SIBDOFF= <i>token1 token2</i> | This message indicates that CSECT DSNECP61 is about to write a line of decorative asterisks. The first hexadecimal token is SIBDOFF, the offset into the output buffer. The second hexadecimal token is not used. |
| WROTE FIRST COMMENT, SIBDOFF= <i>token1 token2</i> | The first comment has been written. The first hexadecimal token is SIBDOFF, the offset into the output buffer. The second hexadecimal token is not used. |
| WROTE LEVEL 1 LINE, SIBDOFF= <i>token1 token2</i> | The first line of SQL output has been written. The first hexadecimal token is SIBDOFF, the offset into the output buffer. The second hexadecimal token is not used. |

| Message text | Explanation |
|---|--|
| BEGIN BIG LOOP, SIBDOFF=, ZCOL= <i>token1 token2</i> | <p>Execution is now at the top of the main loop in CSECT DSNECP61.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is ZCOL, the number of the column being processed.</p> |
| BEFORE FIELD SPECIFICATION, SIBDOFF=, ZCOL= <i>token1 token2</i> | <p>CSECT DSNECP61 is about to write a field specification.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is ZCOL, the number of the column being processed.</p> |
| BEFORE LABEL <i>sqlname</i> SQLNAMEL(ZCOL+SQLD)=, SQLD= <i>token1 token2</i> | <p>CSECT DSNECP61 is about to write a column label.</p> <p>The first hexadecimal token is the length of the label.</p> <p>The second hexadecimal token is SQLD, the number of the columns in the table be processed.</p> |
| BEGIN LEVEL 3, SIBDOFF=, ZCOL= <i>token1 token2</i> | <p>CSECT DSNECP61 is now beginning to write the third level of the declaration.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is ZCOL, the number of the column being processed.</p> |
| AFTER LEVEL 3 COMMENT 1, SIBDOFF=, SQLD= <i>token1 token2</i> | <p>The first of the level-3 comments was just written to the output buffer.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is SQLD, the number of the columns in the table.</p> |
| AFTER LEVEL 3 CODE 1, SIBDOFF=, ZCOL= <i>token1 token2</i> | <p>The first of the level 3 code was just written to the output buffer.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is ZCOL, the number of the column being processed.</p> |
| AFTER LEVEL 3 COMMENT 2, SIBDOFF=, ZCOL= <i>token1 token2</i> | <p>The second of the level-3 comments was just written to the output buffer.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is ZCOL, the number of the column being processed.</p> |

| Message text | Explanation |
|---|--|
| AFTER LEVEL 3 COMMENT 2, LOOP BOTTOM SIBDOFF= ,ZCOL= token1 token2 | <p>The second of the level-3 comments was just written to the output buffer. Execution is at the bottom of the main loop.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is ZCOL, the number of the column being processed.</p> |
| LEAVE DSNECP61, SIBDFLAG=, ZCOL= token1 token2 | <p>CSECT DSNECP61 has just completed. Control will return to the calling CSECT, DSNECP14.</p> <p>The first hexadecimal token is SIBDFLAG, the DCLGEN flag area.</p> <p>The second hexadecimal token is ZCOL, the number of the column being processed.</p> |
| ENTER GETNAME, SIBDOFF= token1 token2 | <p>The GETNAME procedure was just entered. This procedure puts the field names into the output buffer.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is not used.</p> |
| LEAVE GETNAME, SIBDOFF= token1 token2 | <p>The GETNAME procedure was just exited. This procedure put the field names into the output buffer.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is not used.</p> |
| ENTER COMMENT, SIBDOFF= token1 token2 | <p>Procedure COMMENT was just entered. This procedure initializes ZBIGBUF and writes a comment.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is not used.</p> |
| LEAVE COMMENT, SIBDOFF= token1 token2 | <p>The COMMENT procedure was just exited. This procedure put the comment characters into ZBIGBUF and set the offset into that buffer.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is not used.</p> |

CSECT: DSNECP62

| Message text | Explanation |
|---|--|
| ENTER DSNECP62, CIBPTR=, ADDR(SIBDCL)= token1 token2 | <p>This message announces entry into CSECT DSNECP62, where a PLI declaration will be written.</p> <p>The first hexadecimal token is the CIB address.</p> <p>The second hexadecimal token is the address of the DCLGEN area of the subcommand information block (SIB) area.</p> |
| WROTE FIRST COMMENT, SIBDOFF= token1 token2 | <p>This message indicates that the CSECT just wrote the first comment from this CSECT.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is not used.</p> |
| WROTE FIRST LINE, SIBDOFF= token1 token2 | <p>The first line of PLI output has been written.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is not used.</p> |
| BIG LOOP TOP, SIBDOFF=, ZCOL= token1 token2 | <p>Execution is now at the top of the main loop in CSECT DSNECP62.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is ZCOL, the column number being processed.</p> |
| BEFORE DSNECP67, SQLNAME(ZCOL):fieldname SIBDOFF=, ZCOL= token1 token2 | <p>CSECT DSNECP62 is about to write a field name by calling DSNECP67 with a request to put it into the main output buffer.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is ZCOL, the column number being processed.</p> |
| BEFORE FIELD SPECIFICATION SIBDOFF=, SIBDNAML= token1 token2 | <p>CSECT DSNECP62 is about to write a field type specification.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is SIBDNAML, the length of the column name being processed.</p> |
| LABEL WAS YES ZLBCOL=, ZLABLEN= token1 token2 | <p>The user specified a LABEL parameter of YES.</p> <p>The first hexadecimal token is ZCOL + SQLD, the number of the SQLVAR element containing the label information.</p> <p>The second hexadecimal token is ZLABLEN, the length of the label being processed.</p> |

| Message text | Explanation |
|--|--|
| BBUF zbigbuf ZCOL=token1 token2 | <p>The main output buffer, ZBIGBUF, is echoed in this message.</p> <p>The first hexadecimal token is ZCOL, the number of the current column being processed.</p> <p>The second hexadecimal token is not used.</p> |
| LOOP BOTTOM, SIBDOFF=, ZCOL= token1 token2 | <p>The bottom of the main loop in CSECT DSNECP62 has just been reached.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is ZCOL, the column number being processed.</p> |
| END DSNECP62, SIBDFLAG=, ZCOL= token1 token2 | <p>CSECT DSNECP62 has just completed. Control will return to the calling CSECT, DSNECP14.</p> <p>The first hexadecimal token is SIBDFLAG, the DCLGEN flag area.</p> <p>The second hexadecimal token is ZCOL, the column number being processed.</p> |
| CSECT: DSNECP63 | |
| Message text | Explanation |
| ENTER DSNECP63, CIBPTR=, ADDR(SIBDCL)= token1 token2 | <p>This message announces entry into CSECT DSNECP63, where a C declaration will be written.</p> <p>The first hexadecimal token is the CIB address.</p> <p>The second hexadecimal token is the address of the DCLGEN area of the subcommand information block (SIB) area.</p> |
| BEFORE DSNECP67, ZBUF= zbuf SIBDOFF=, ZCOL= token1 token2 | <p>CSECT DSNECP63 is about to write a field specification. The name of the field is zbuf.</p> <p>The first hexadecimal token is SIBDOFF, the current offset into the output buffer.</p> <p>The second hexadecimal token is ZCOL, the number of the column being processed.</p> |
| WROTE FIRST COMMENT, SIBDOFF= token1 token2 | <p>This message indicates that the CSECT just wrote the first comment from this CSECT.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is not used.</p> |

| Message text | Explanation |
|---|---|
| WROTE FIRST LINE, SIBDOFF= <i>token1 token2</i> | <p>The first line of C output has been written.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is not used.</p> |
| BIG LOOP TOP, SIBDOFF=, ZCOL= <i>token1 token2</i> | <p>Execution is now at the top of the main loop in CSECT DSNECP63.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is ZCOL, the column number being processed.</p> |
| FOUND AN INTEGER, SQLNAME(ZCOL)=<i>colname</i> SIBDOFF=, SIBDNAML= <i>token1 token2</i> | <p>The table column with name <i>colname</i> has the INTEGER data type.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is SIBDNAML, the length of the column name.</p> |
| FOUND A SMALL INTEGER, SQLNAME(ZCOL)=<i>colname</i> SIBDOFF=, SIBDNAML= <i>token1 token2</i> | <p>The table column with name <i>colname</i> has the SMALLINT data type.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is SIBDNAML, the length of the column name.</p> |
| FOUND A ROWID. SQLNAME(ZCOL)=<i>colname</i> SIBDOFF=, SIBDNAML= <i>token1 token2</i> | <p>The table column with name <i>colname</i> has the ROWID data type.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is SIBDNAML, the length of the column name.</p> |
| FOUND A BLOB, CLOB, OR DBCLOB, SQLNAME(ZCOL)=<i>colname</i> SIBDOFF=, SIBDNAML= <i>token1 token2</i> | <p>The table column with name <i>colname</i> has the ROWID data type.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is SIBDNAML, the length of the column name.</p> |
| FOUND BLOB, CLOB, OR DBCLOB. SQLNAME(ZCOL)=<i>colname</i> SIBDOFF=, SIBDNAML= <i>token1 token2</i> | <p>The table column with name <i>colname</i> has the BLOB, CLOB, or DBCLOB data type.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is SIBDNAML, the length of the column name.</p> |

| Message text | Explanation |
|---|---|
| FOUND CHAR,DATE,TIME, OR TS. SQLNAME(ZCOL)=colname SIBDOFF=, SIBDNAML= token1 token2 | <p>The table column with name colname has the CHAR, DATE, TIME, or TIMESTAMP data type.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is SIBDNAML, the length of the column name.</p> |
| FOUND A VARCHAR, SQLNAME(ZCOL)=colname SIBDOFF=, SIBDNAML= token1 token2 | <p>The table column with name colname has the VARCHAR data type.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is SIBDNAML, the length of the column name.</p> |
| FOUND GRAPHIC. SQLNAME(ZCOL)=colname SIBDOFF=, SIBDNAML= token1 token2 | <p>The table column with name colname has the GRAPHIC data type.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is SIBDNAML, the length of the column name.</p> |
| FOUND A VARGRAPHIC, SQLNAME(ZCOL)=colname SIBDOFF=, SIBDNAML= token1 token2 | <p>The table column with name colname has the VARGRAPHIC data type.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is SIBDNAML, the length of the column name.</p> |
| FOUND A FLOAT, SQLNAME(ZCOL)=colname SIBDOFF=, SIBDNAML= token1 token2 | <p>The table column with name colname has the FLOAT data type.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is SIBDNAML, the length of the column name.</p> |
| LOOP BOTTOM, SIBDOFF=, ZCOL= token1 token2 | <p>The bottom of the main loop in CSECT DSNECP63 has just been reached.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is ZCOL, the column number being processed.</p> |
| WROTE INDICATOR VAR ARRAY, SIBDOFF=, token1 | <p>DCLGEN wrote an indicator variable array to the output buffer.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> |

| Message text | Explanation |
|---|---|
| ENTER PROCEDURE LABELER. FIELDNAME=colname SIBDOFF=, ZBEGCOM= token1 token2 | <p>Entering the subroutine that is called if the user specified LABEL(YES) on the DCLGEN subcommand.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is ZBEGCOM, the column in an output record where comments begin.</p> |
| LABEL WAS YES ZLABCOL=, ZLABLEN= <i>token1 token2</i> | <p>The user specified a LABEL parameter of YES.</p> <p>The first hexadecimal token is ZCOL + SQLD, the number of the SQLVAR element containing the label information.</p> <p>The second hexadecimal token is ZLABLEN, the length of the label being processed.</p> |
| BBUF zbigbuf ZCOL= <i>token1 token2</i> | <p>The main output buffer, ZBIGBUF, is echoed in this message.</p> <p>The first hexadecimal token is ZCOL, the number of the current column being processed.</p> <p>The second hexadecimal token is not used.</p> |
| LEAVE PROCEDURE LABELER. FIELDNAME=colname SIBDOFF=, ZBEGCOM= token1 token2 | <p>Leaving the subroutine that is called if the user specified LABEL(YES) on the DCLGEN subcommand.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is ZBEGCOM, the column in an output record where comments begin.</p> |
| ENTER PROCEDURE FIELDNAM. FIELDNAME=colname SIBDOFF=, ZBEGCOM= token1 token2 | <p>Entering the subroutine that generates a host variable name that corresponds to colname.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is ZBEGCOM, the column in an output record where comments begin.</p> |
| LEAVE PROCEDURE FIELDNAM. FIELDNAME=colname SIBDOFF=, ZBEGCOM= token1 token2 | <p>Leaving the subroutine that generates a host variable name that corresponds to colname.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is ZBEGCOM, the column in an output record where comments begin.</p> |

| Message text | Explanation |
|--|--|
| ENTER PROCEDURE COMMENT. FIELDNAME=colname SIBDOFF=, ZBEGCOM= token1 token2 | <p>Entering the subroutine that writes a comment that contains colname.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is ZBEGCOM, the column in an output record where comments begin.</p> |
| LEAVE PROCEDURE COMMENT. FIELDNAME=colname SIBDOFF=, ZBEGCOM= token1 token2 | <p>Leaving the subroutine that writes a comment that contains colname. that corresponds to colname.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is ZBEGCOM, the column in an output record where comments begin.</p> |
| ENTER PROCEDURE CHKDB. | <p>Entering the subroutine that checks for DBCS characters in a column name.</p> |
| SI FOUND AT ZI= token1 | <p>The column name contains a DBCS shift-in character at position ZI.</p> <p>The first hexadecimal token, ZI, contains the position in the column name of the shift-in character.</p> |
| LEAVE PROCEDURE CHKDB. | <p>Leaving the subroutine that checks for DBCS characters in a column name.</p> |
| END DSNECP63, SIBDFLAG=, ZCOL= token1 token2 | <p>CSECT DSNECP63 has just completed. Control will return to the calling CSECT, DSNECP14.</p> <p>The first hexadecimal token is SIBDFLAG, the DCLGEN flag area.</p> <p>The second hexadecimal token is ZCOL, the column number being processed.</p> |
| CSECT: DSNECP66 | |
| Message text | Explanation |
| ENTER DSNECP66, CIBPTR=, ADDR(SIBDCL)= token1 token2 | <p>This message announces entry into CSECT DSNECP66, which assists CSECT DSNECP68 to make the DCLGEN SQL call.</p> <p>The first hexadecimal token is the CIB address.</p> <p>The second hexadecimal token is the address of the DCLGEN area in the SIB.</p> |

| Message text | Explanation |
|---|---|
| BEFORE FIRST GETMAIN, SIBDSQDS= token1 token2 | <p>This message indicates that CSECT DSNECP66 is about to do the GETMAIN for SQL DSECT.</p> <p>The first hexadecimal token is SIBDSQDS, the size of the first getmained area.</p> <p>The second hexadecimal token is not used.</p> |
| GETMAIN SUCCESSFUL, R1= token1 token2 | <p>The GETMAIN for the area needed by the precompiler was successful.</p> <p>The first hexadecimal token is the address of the getmained area.</p> <p>The second hexadecimal token is not used.</p> |
| AFTER PREPARE, SQLCODE= token1 token2 | <p>The PREPARE is complete.</p> <p>The first hexadecimal token is SQLCODE.</p> <p>The second hexadecimal token is not used.</p> |
| BEFORE DESCRIBE1, SQLN=, SIBDSQLD= token1 token2 | <p>CSECT DSNECP66 is about to do its first DESCRIBE, so it can determine how large to make the real SQLDA for the second DESCRIBE.</p> <p>The first hexadecimal token is SQLN, the estimated number of columns.</p> <p>The second hexadecimal token is the SQLDA address.</p> |
| BEFORE SQLDA GETMAIN, SIBDSQLS=, SQLD= token1 token2 | <p>DCLGEN is about to getmain the area for the actual SQLDA.</p> <p>The first hexadecimal token is the size of the area to getmain.</p> <p>The second hexadecimal token is the actual number of columns in the table.</p> |
| REAL SQLDA SIBDSQLD=, SIBDSQLS= token1 token2 | <p>The area for the final SQLDA has been obtained.</p> <p>The first hexadecimal token is SIBDSQLD, the address of the SQLDA.</p> <p>The second hexadecimal token is SIBDSQLS, the size of the SQLDA.</p> |
| AFTER GETMAIN, SIBDSQLD= token1 token2 | <p>The GETMAIN for the real SQLDA is complete.</p> <p>The first hexadecimal token is SIBDSQLD, the SQLDA address.</p> <p>The second hexadecimal token is not used.</p> |
| BEFORE DESCRIBE2 SIBDSQLD=, SQLN= token1 token2 | <p>The main DESCRIBE is about to happen.</p> <p>The first hexadecimal token is the SQLDA address.</p> <p>The second hexadecimal token is the estimated number of columns.</p> |

| Message text | Explanation |
|--|--|
| AFTER DESCRIBE2, SQLD=, SQLCODE= <i>token1 token2</i> | <p>The final DESCRIBE is complete.</p> <p>The first hexadecimal token is the number of columns in the table.</p> <p>The second hexadecimal token is SQLCODE, the return code from the DESCRIBE.</p> |
| BEFORE FREEMAIN SIBDSECT=, SIBDSQDS= <i>token1 token2</i> | <p>The SQLDSECT area is about to be freed.</p> <p>The first hexadecimal token is the address of the area to free.</p> <p>The second hexadecimal token is the size of the area to free.</p> |
| LEAVE DSNECP66, SIBDFLAG=, SQLD= <i>token1 token2</i> | <p>CSECT DSNECP66 has completed. Control will return to the calling CSECT, DSNECP14. DCLGEN SQL calls are now complete.</p> <p>The first hexadecimal token is the DCLGEN flag area.</p> <p>The second hexadecimal token is the actual number of columns in the table.</p> |
| ENTER ZCLEANER, ZPLACE=, ZSIZE= <i>token1 token2</i> | <p>CSECT DSNECP66 is about to zero a getmained area.</p> <p>The first hexadecimal token is the address to clear.</p> <p>The second hexadecimal token is the size of the area.</p> |
| CLEAR LOOPTOP, ZPLACE=, ZSIZE= <i>token1 token2</i> | <p>The ZCLEANER procedure is at the top of its clearing loop. It is about to clear 255 bytes.</p> <p>The first hexadecimal token is the address of the area that is currently being cleared.</p> <p>The second hexadecimal token is the size of the remaining area to clear.</p> |
| LEAVE ZCLEANER, ZPLACE=, ZSIZE= <i>token1 token2</i> | <p>The ZCLEANER procedure is complete. The getmained area is zeroed.</p> <p>The first hexadecimal token is meaningless.</p> <p>The second hexadecimal token should be zero or negative.</p> |
| ENTER CHEKCODE, SQLCODE= <i>token1 token2</i> | <p>CSECT DSNECP66 is about to translate a non-zero SQLCODE into an error message.</p> <p>The first hexadecimal token is SQLCODE.</p> |

| Message text | Explanation |
|---|---|
| BEFORE CALLING DSNTIAR, ADDR(ZMSGBUF),SIBDSQLC <i>token1 token2</i> | <p>DSNTIAR is the routine that translates SQLCODEs into error messages.</p> <p>The first hexadecimal token is the address of the message buffer.</p> <p>The second hexadecimal token is the address of the SQLCA.</p> |
| AFTER CALLING DSNTIAR, ADDR(ZMSGBUF)=, ZR15= <i>token1 token2</i> | <p>Control has returned from DSNTIAR.</p> <p>The first hexadecimal token is the address of the buffer from DSNTIAR.</p> <p>The second hexadecimal token is ZR15, the return code from DSNTIAR.</p> |
| LEAVE CHEKCODE, SQLCODE= <i>token1 token2</i> | <p>The CHEKCODE procedure is complete. The SQLCODE has been evaluated and any necessary error messages have been issued.</p> <p>The first hexadecimal token is the SQLCODE.</p> <p>The second hexadecimal token is not used.</p> |
| CSECT: DSNECP67 | |
| Message text | Explanation |
| ENTER DSNECP67, ZACTION="xxx", SIBDOFF=, ZHEXNUM= <i>token1 token2</i> | <p>This message announces entry into CSECT DSNECP67, where various DCLGEN utility functions are performed. xxx is a code indicating which function is to be performed.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is ZHEXNUM, a number to be translated to EBCDIC.</p> |
| ZEBCDIC=ebcdic , ZCVESIG= <i>token1 token2</i> | <p>The variable 'ebcdic' is the converted number.</p> <p>The first hexadecimal token is used for the EDMK instruction.</p> |
| ZERO ENTERED, ZCVESIG= <i>token1 token2</i> | <p>The number to be converted was a zero.</p> <p>The first hexadecimal token is used by the EDMK instruction.</p> <p>The second hexadecimal token is zero.</p> |
| NON-ZERO ENTERED, OLD SIBDOFF= <i>token1 token2</i> | <p>The number to be converted was not a zero.</p> <p>The first hexadecimal token is SIBDOFF, the non-updated offset into the output buffer.</p> <p>The second hexadecimal token is not used.</p> |

| Message text | Explanation |
|--|--|
| ZBIGBUF: <i>zbigbuf</i> SIBDOFF= <i>token1 token2</i> | <p>This is the current contents of the main output buffer. This buffer can appear truncated because of length restrictions on trace message tokens.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is zero.</p> |
| PUT: SIBDOFF=, SIBDCBA= <i>token1 token2</i> | <p>CSECT DSNECP67 is about to PUT ZBIGBUF to the output data set.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is the DCB address, SIBDCBA.</p> |
| LEAVE WRITEBUF, SIBDOFF= <i>token1 token2</i> | <p>CSECT DSNECP67 just wrote a line to the output data set.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is not used.</p> |
| ZBIGBUF: <i>zbigbuf</i> SIBDOFF=<i>token1 token2</i> | <p>The variable '<i>zbigbuf</i>' is the new ZBIGBUF.</p> <p>The first hexadecimal token is SIBDOFF, the offset into the output buffer.</p> <p>The second hexadecimal token is not used.</p> |
| LEAVE DSNECP67, SIBDFLAG=, SIBDOFF= <i>token1 token2</i> | <p>CSECT DSNECP67 has just completed processing. Control will return to the calling CSECT.</p> <p>The first hexadecimal token is the DCLGEN flag area.</p> <p>The second hexadecimal token is SIBDOFF, the offset into the output buffer.</p> |
| CSECT: DSNECP68 | |
| Message text | Explanation |
| ENTER DSNECP68, CIBPTR=, ADDR(SIBDCL)= <i>token1 token2</i> | <p>This message announces entry into CSECT DSNECP68, where the DCLGEN SQL calls will be made.</p> <p>The first hexadecimal token is the CIB address.</p> <p>The second hexadecimal token is the address of the DCLGEN area of the SIB.</p> |

| Message text | Explanation |
|---|--|
| AFTER WHENEVER SQL ERROR, SIBDWHAT= <i>token1 token2</i> | This message indicates that the WHENEVER SQL ERROR statement was just issued. The first hexadecimal token is SIBDWHAT, the code for what to do next. The second hexadecimal token is zero. |
| R2(SIBDSQLC)=, R3(SIBDSQLD)= <i>token1 token2</i> | The code is now setting up base registers. The first hexadecimal token is the SQLCA address. The second hexadecimal token is the SQLDA address. |
| R4(SIBDSECT)= <i>token1 token2</i> | This message displays the address of the precompiler's SQLDSECT area. The first hexadecimal token is the address of the precompiler's SQLDSECT area. The second hexadecimal token is not used. |
| BEFORE GET SIBDSQDS(LEN OF SQLDSECT) <i>token1 token2</i> | Execution is just before the GETMAIN (which occurs in DSNECP66) for the precompiler area. The hexadecimal tokens are not used. |
| AFTER GET SIBDSQDS, SIZE= <i>token1 token2</i> | The previous GETMAIN just completed. The first hexadecimal token is the size of the area getmained. The second hexadecimal token is not used. |
| zsqlstmt <i>token1 token2</i> | This is the SQL statement, including the substituted table name, that is about to be prepared. The hexadecimal tokens are not used. |
| AFTER PREPARE <i>token1 token2</i> | The PREPARE has completed. The hexadecimal tokens are not used. |
| BEFORE DESCRIBE, R3= <i>token1 token2</i> | A DESCRIBE is about to be issued. The first hexadecimal token is register 3, the SQLDA address. The second hexadecimal token is not used. |
| AFTER DESCRIBE <i>token1 token2</i> | A DESCRIBE has completed. The hexadecimal tokens are not used. |

CSECT: DSNECP69

| Message text | Explanation |
|--|---|
| ENTER DSNECP69, ADDR(SIBDCL)= token1 token2 | <p>This message announces entry into CSECT DSNECP69, which performs dynamic allocation services for DCLGEN.</p> <p>The first hexadecimal token is the address of the DCLGEN area in the SIB.</p> <p>The second hexadecimal token is not used.</p> |
| SIBRCODE=, SIBDFLAG= token1 token2 | <p>The contents of the named variables are displayed by this message.</p> <p>The first hexadecimal token is the DCLGEN return code so far.</p> <p>The second hexadecimal token is the DCLGEN flag area.</p> |
| END INIT, ADDR(ZOHDAIR)=, ADDR(ZS99RB)= token1 token2 | <p>Initialization has completed.</p> <p>The first hexadecimal token is the address of ZOHDAIR.</p> <p>The second hexadecimal token is the address of ZS99RB.</p> |
| DALLOC CALLED, R14=, SIBDFLAG= token1 token2 | <p>CSECT DSNECP69 has started to deallocate the output data set.</p> <p>The first hexadecimal token is register 14.</p> <p>The second hexadecimal token is SIBDFLAG, the DCLGEN flag area.</p> |
| DALLOC TU, ZTUNDX=, ZPTR=, DSNAME token1 token2 | <p>The deallocation text unit has been built.</p> <p>The first hexadecimal token is ZTUNDX, the index into the text unit list.</p> <p>The second hexadecimal token is ZPTR, the pointer to the current TU.</p> |
| S99TUPAR(1:S99TULNG) token1 token2 | <p>This message prints the dsname being processed.</p> <p>The hexadecimal tokens are not used.</p> |
| AFTER DYNALLOC, S99RSC=, R15= token1 token2 | <p>This message appears just after the DYNALLOC SVC.</p> <p>The first hexadecimal token is S99RSC, the reason code from dynamic deallocation.</p> <p>The second hexadecimal token is register 15, the return code from dynamic deallocation.</p> |

| Message text | Explanation |
|---|--|
| DALLOC ENDED, SIBDFLAG=, token1 token2 | <p>The deallocation portion of CSECT DSNECP69 has completed.</p> <p>The first hexadecimal token is the DCLGEN flag area.</p> <p>The second hexadecimal token is zero.</p> |
| EXIT DSNECP69, SIBRCODE=, SIBDFLAG= token1 token2 | <p>CSECT DSNECP69 is about to return to caller.</p> <p>The first hexadecimal token is the DCLGEN return code, so far.</p> <p>The second hexadecimal token is the DCLGEN flag area.</p> |
| BEFORE CALL TO DAIRFAIL, R15=, S99RSC= token1 token2 | <p>DAIRFAIL is about to be invoked.</p> <p>The first hexadecimal token is ZR15.</p> <p>The second hexadecimal token is S99RSC.</p> |
| AFTER DAIRFAIL, R15=, R1= token1 token2 | <p>This message appears after the DAIRFAIL call.</p> <p>The first hexadecimal token is register 15, the return code from the DAIRFAIL call.</p> <p>The second hexadecimal token is register 1, the address of DFPARMS.</p> |
| ZOPEN CALLED, R14=, SIBDCBA= token1 token2 | <p>The OPEN subroutine has been called.</p> <p>The first hexadecimal token is register 14.</p> <p>The second hexadecimal token is SIBDCBA.</p> |
| SIBRCODE=, SIBDFLAG= token1 token2 | <p>Processing continues in the OPEN routine.</p> <p>The first hexadecimal token is SIBRCODE.</p> <p>The second hexadecimal token is the DCLGEN flag area.</p> |
| LEAVE ZOPEN, SIBRCODE=, SIBDFLAG= token1 token2 | <p>The OPEN function has completed.</p> <p>The first hexadecimal token is SIBRCODE, the DCLGEN return code so far.</p> <p>The second hexadecimal token is the DCLGEN flag area.</p> |
| ZCLOSE CALLED, R14=, SIBDFLAG= token1 token2 | <p>The CLOSE subroutine has been called.</p> <p>The first hexadecimal token is register 14.</p> <p>The second hexadecimal token is the DCLGEN flag area.</p> |

| Message text | Explanation |
|---|---|
| LEAVE ZCLOSE, SIBDCBA=, SIBDFLAG= <i>token1 token2</i> | <p>The CLOSE function has completed.</p> <p>The first hexadecimal token is SIBDCBA, the DCB address.</p> <p>The second hexadecimal token is the DCLGEN flag area.</p> |
| ZALLOC CALLED, R14=, ZUSEMBR= <i>token1 token2</i> | <p>The dynamic allocation subroutine has been called.</p> <p>The first hexadecimal token is register 14.</p> <p>The second hexadecimal token is ZUSEMBR.</p> |
| 1ST TU, ZTUNDX=, ZPTR=, DSNAME= <i>token1 token2</i> | <p>The dsname text unit has been built.</p> <p>The first hexadecimal token is ZTUNDX, an index into the text unit list.</p> <p>The second hexadecimal token is ZPTR, the address of the current TU.</p> |
| dsname <i>token1 token2</i> | <p>This message prints the data set name.</p> <p>The hexadecimal tokens are not used.</p> |
| 2ND TU, ZTUNDX=, ZPTR= <i>token1 token2</i> | <p>The return DDNAME text unit has been built.</p> <p>The first hexadecimal token is ZTUNDX, an index into the text unit list.</p> <p>The second hexadecimal token is ZPTR, the address of the current TU.</p> |
| 3RD TU, ZTUNDX=, ZPTR= <i>token1 token2</i> | <p>The return DSORG text unit has been built.</p> <p>The first hexadecimal token is ZTUNDX, an index into the text unit list.</p> <p>The second hexadecimal token is ZPTR, the address of the current TU.</p> |
| 4TH TU, ZTUNDX=, ZPTR= <i>token1 token2</i> | <p>The data set status text unit has been built.</p> <p>The first hexadecimal token is ZTUNDX, an index into the text unit list.</p> <p>The second hexadecimal token is ZPTR, the address of the current TU.</p> |
| 5TH TU, ZTUNDX=, ZPTR= <i>token1 token2</i> | <p>The normal disposition text unit has been built.</p> <p>The first hexadecimal token is ZTUNDX, an index into the text unit list.</p> <p>The second hexadecimal token is ZPTR, the address of the current TU.</p> |

| Message text | Explanation |
|---|--|
| 6TH TU, ZTUNDX=, ZPTR= <i>token1 token2</i> | <p>The conditional disposition text unit has been built.</p> <p>The first hexadecimal token is ZTUNDX, an index into the text unit list.</p> <p>The second hexadecimal token is ZPTR, the address of the current TU.</p> |
| MBR TU, ZTUNDX=, ZPTR= <i>token1 token2</i> | <p>The member text unit has been built.</p> <p>The first hexadecimal token is ZTUNDX, an index into the text unit list.</p> <p>The second hexadecimal token is ZPTR, the address of the current TU.</p> |
| PSWD TU, ZTUNDX=, ZPTR= <i>token1 token2</i> | <p>The password text unit has been built.</p> <p>The first hexadecimal token is ZTUNDX, an index into the text unit list.</p> <p>The second hexadecimal token is ZPTR, the address of the current TU.</p> |
| BEFORE DYNALLOC, S99RBPTR=, ZPTR= <i>token1 token2</i> | <p>CSECT DSNECP69 is about to issue the DYNALLOC SVC.</p> <p>The first hexadecimal token is S99RBPTR.</p> <p>The second hexadecimal token is ZPTR.</p> |
| AFTER DYNALLOC, S99RSC=, R15= <i>token1 token2</i> | <p>This message appears after the DYNALLOC SVC.</p> <p>The first hexadecimal token is S99RSC, the reason code.</p> <p>The second hexadecimal token is register 15, the return code.</p> |
| LEAVE ZALLOC, SIBDFLAG= <i>token1 token2</i> | <p>The dynamic allocation subroutine has completed.</p> <p>The first hexadecimal token is the DCLGEN flag area.</p> <p>The second hexadecimal token is zero.</p> |
| CSECT: DSNETRAP | |
| Message text | Explanation |
| BEFORE SQL CALL=====FRB,R1== <i>token1 token2</i> | <p>Control is about to pass from the TSO attach package to Db2 to process a SQL call.</p> <p>The first hexadecimal token is the FRB address.</p> <p>The second hexadecimal token is register 1.</p> |

| Message text | Explanation |
|---|---|
| AFTER SQL CALL=====RC1,FBACK== token1 token2 | Control has just returned to the TSO attach facility after Db2 processed a SQL call. The first hexadecimal token is FRBRC1. The second hexadecimal token is FRBFBACK. |

PSPI

Call attachment facility trace messages

An application programmer that uses the call attachment facility (CAF) to write an application that attaches to Db2 can choose whether the messages are displayed or not. They display if the DSNTRACE *ddname* is allocated during CAF execution.

PSPI

The following table describes the following information:

- Message number
- Name of the CSECT that caused the message to be written.
- TCB address of the task currently running.
- A series of tokens and token-explanations unique to each message.

| Message number | Message text | Explanation |
|----------------|---|--|
| DSNA800I | DSNACA00 TCB= <i>address</i> ENTERED DSNACA00 ACTION= <i>action</i> R1= <i>address</i> CABPTR= <i>address</i> CABFLAG1= <i>flag1</i> CABFLAG2= <i>flag2</i> | The call attachment facility received a call attachment facility request to perform the 'action' function. The parameter list begins at location 'address'. CABPTR specifies the location of the call attachment facility control block (the CAB). CABFLAG1 and CABFLAG2 are the current flag bytes. This message is issued by the following module: DSNACA00 |
| DSNA801I | DSNACA00 TCB= <i>address</i> LEAVE DSNACA00 CABFLAG1= <i>flag1</i> CABFLAG2= <i>flag2</i> CABPTR= <i>address</i> | The call attachment facility issues this message following completion of a call attachment facility request. CABFLAG1 and CABFLAG2 are the flag bytes and CABPTR specifies the location of the call attachment facility control block. This message is issued by the following module: DSNACA00 |

| Message number | Message text | Explanation |
|----------------|--|---|
| DSNA805I | DSNACA00 TCB= <i>address</i> CONNECT REQUEST SSID= <i>ssid</i> SECBP= <i>ecb address</i> TECBP= <i>ecb address</i> RIBW= <i>address</i> | The call attachment facility received a CONNECT request with the specified parameter values. SSID is the subsystem identifier, SECBP is the Db2 start-up ECB address, TECBP is the Db2 termination ECB address, and RIBW is the address of the fullword that will be set to contain the address of the release information block (RIB) after the CONNECT completes. This message is issued by the following module: DSNACA00 |
| DSNA806I | DSNACA00 TCB= <i>address</i> DISCONNECT REQUEST | The call attachment facility received a DISCONNECT request for the specified TCB. This message is issued by the following module: DSNACA00 |
| DSNA807I | DSNACA00 TCB= <i>address</i> OPEN REQUEST SSID= <i>ssid</i> PLAN= <i>plan name</i> | The call attachment facility received an OPEN request with the specified SSID and PLAN values. This message is issued by the following module: DSNACA00 |
| DSNA808I | DSNACA00 TCB= <i>address</i> CLOSE REQUEST TRMOP= <i>terminate option</i> | The call attachment facility received a CLOSE request with the specified termination option. This message is issued by the following module: DSNACA00 |
| DSNA810I | DSNACA00 TCB= <i>address</i> BEFORE function ===== | The call attachment facility issues this message before a 'function' request to Db2. This is the last call attachment facility action performed before passing control to Db2 for processing. If this is the last call attachment facility message before an abend, then the abend very probably occurred in the Db2 code proper, rather than in the call attachment facility code. This message is issued by the following module: DSNACA00, DSNACA70 |

| Message number | Message text | Explanation |
|-----------------------|---|--|
| DSNA811I | DSNACA00 TCB= <i>address</i> AFTER <i>function</i> ===== | The call attachment facility issues this message when Db2 returns control after a 'function' request. This message is issued by the following modules: DSNACA00, DSNACA70 |
| DSNA813I | <i>csectname</i> TCB= <i>address</i> FRBP= <i>address</i> RAL= <i>data</i> RALE= <i>data</i> FVLE= <i>data</i> PARM= <i>data</i> PCNT= <i>data</i> | This message, along with message DSNA814I, displays the contents of the FRB control block before or after a call to Db2. This message is issued by the following modules: DSNACA00, DSNACA70 |
| DSNA814I | <i>csectname</i> TCB= <i>address</i> RC1= <i>retcode</i> RC2= <i>reascode</i> FBACK= <i>feedback</i> RHPC= <i>data</i> QUAL= <i>data</i> RSV1= <i>data</i> | This message, along with message DSNA813I, displays the contents of the FRB control block before or after a call to Db2. This message is issued by the following modules: DSNACA00, DSNACA70 |
| DSNA815I | DSNACA70 TCB= <i>address</i> FDBKEIB=EIB- <i>address</i> SSID= <i>ssid</i> GATT= <i>group-attach-name</i> GRPN= <i>group-name</i> MBRN= <i>member-name</i> SGATT= <i>subgroup-name</i> | This message displays the address and contents of the environment information control block (EIB). Db2 generates this message when the EIB address is not 0. |
| DSNA821I | DSNACA00 TCB= <i>address</i> IMPLICIT CONNECTION REQUEST | The call attachment facility received an SQL request before establishing the Db2 connection. This results in an implicit connection to Db2 using defaults for the application plan name and, if no CONNECT was issued, the Db2 subsystem identifier. This message is issued by the following module: DSNACA00 |
| DSNA822I | DSNACA00 TCB= <i>address</i> TRANSLATE REQUEST SQLCA= <i>address</i> CABFRC1= <i>retcode</i> CABFRC2= <i>reascode</i> FRBRHPC= <i>frbrhpc</i> | The call attachment facility received a TRANSLATE request. This message identifies the location of the SQLCA whose SQLCODE field will be set and the return codes that will be translated. The translate will not be performed if FRBRHPC is zero. |

| Message number | Message text | Explanation |
|----------------|---|---|
| DSNA824I | DSNACA00 TCB= <i>address</i> CAF READY FOR NEW CONNECTION SERVICE REQUESTS | <p>This message indicates that the previous Db2 connection has been terminated, certain clean-up processing has occurred, and that the call attachment facility is now ready to process additional connection requests.</p> <p>This message is issued by the following module: DSNACA70. It is accompanied by return code 4 in register 15 and code reason code 00C10824 in register 0.</p> |
| DSNA825I | DSNACA70 TCB= <i>address</i> CABSSID= <i>ssid</i> CABREL= <i>release</i> CABDECPP= <i>address</i> EIBPARM=EIB- <i>address</i> | This message lists the current values in DSNHDECP, which is link edited with the call attachment facility main load module, DSNACAF. These values are used as defaults on implicit connection requests. |
| DSNA826I | DSNACA00 TCB= <i>address</i> BEFORE VALIDITY CHECKING THE <i>type</i> ECB AT ADDRESS <i>location</i> | This trace message indicates that a check of the 'type' ECB (either start-up or shutdown), is about to be done to determine if the address identified by 'location' is addressable. If an OC4 abend occurs immediately following this message, then an invalid ECB address was passed on a CONNECT request. |
| DSNA827I | DSNACA00 TCB= <i>address</i> AFTER VALIDITY CHECKING THE ECB WHICH CONTAINS ' <i>data</i> ' | <p>This trace message indicates that the identified ECB is addressable and contains the specified data value. Appearance of this message is not a guarantee that the ECB address that was passed is valid; it has only passed preliminary tests. Deleting this ECB without Db2 knowing makes the ECB invalid.</p> <p>This message is issued by the following module: DSNACA00</p> |

| Message number | Message text | Explanation |
|-----------------------|--|--|
| DSNA828I | DSNACA00 TCB= <i>address</i> RIB ADDRESS= <i>ribaddr</i> RIBPTR ADDR= <i>ribptr</i> | This trace message lists the address of the Db2 release information block (<i>ribaddr</i>) and the address of the fullword that points to the RIB (<i>ribptr</i>) after connection attempts that were made to a Db2 subsystem identifier that existed. This message is issued by the following module: DSNACA70 |
| DSNA829I | DSNACA00 TCB= <i>address</i> R1(FRB)= <i>frbaddr</i> SQLDSECT(FRBPARAM)= <i>dsectaddr</i> RDI= <i>rdiaddr</i> SQLCA= <i>sqlcaaddr</i> CABFLAG2= <i>flag2</i> | This trace message lists the addresses of the FRB, the SQLDSECT, the RDI and the SQLCA. The CABFLAG2 flag byte is useful for learning the AMODE of DSNACA00's caller, the Language Interface (DSNALI). This message is issued by the following module: DSNACA00 |
| DSNA830I | DSNACA00 TCB= <i>address</i> SQLCODE= <i>sqlcode</i> CABFLAG1= <i>flag1</i> CABFLAG2= <i>flag2</i> | This trace message lists the SQLCODE and the two call attachment flag bytes. This message is issued by the following module: DSNACA00 |

| Message number | Message text | Explanation |
|----------------|--|--|
| DSNA831I | DSNACA70 RELEASE LEVELS NOT COMPATIBLE. CAF= <i>release_level</i> , DB2= <i>release_level</i> | <p>The call attachment facility (CAF) has detected a release level incompatibility between Db2 and itself. The current release levels of CAF code and the Db2 subsystem load modules are indicated as a string of three numeric characters, one for each of the following items:</p> <ul style="list-style-type: none"> • Version • Release • Modification level <p>The connection to the Db2 subsystem is not successful. A return code of 8, and a reason code of 00C10831 are returned to the caller's application program.</p> <p>If you used JCL or a TSO logon procedure to invoke CAF, check that the correct Db2 libraries are defined in your JOBLIB and STEPLIB allocations. See your system programmer if you invoke CAF by using procedures that are supplied by your system programmer.</p> <p>If you are unable to determine the problem, you can trace CAF execution by defining a DSNTTRACE DD data set.</p> <p>Ensure that coexistence is supported on both release levels (CAF and Db2). If coexistence is supported, ensure that the lower-level release has the correct SPE level code support that is required to coexist with the current higher level release.</p> |

PSPI

SPUFI trace messages

The SPUFI trace messages, issued by the TSO attachment facility, have unique message numbers.

PSPI

The following SPUFI trace messages are listed in message number order.

| Message text | Explanation |
|---|--|
| DSNESH85 MODULE LIST INCOMPLETE. <i>module address</i> NOT DEFINED | <p>An internal Db2 error occurred. The tracing subsystem is disabled.</p> <p>This message is issued by the following CSECT: DSNESH85</p> |
| <i>csectname</i> ENTERED | <p>The CSECT specified in the message was entered.</p> <p>This message is issued by the following CSECTs: DSNESH21, DSNESH30, DSNESH32, DSNESH81</p> |
| <i>csectname</i> EXITED | <p>Control left the CSECT that is named in the message. This CSECT is about to return control to the CSECT that called it.</p> <p>This message is issued by the following CSECTs: DSNESH21, DSNESH30, DSNESH32, DSNESH81</p> |
| <i>csectname</i> STARTING INITIALIZATION | <p>The CSECT named in the message is starting its initialization processing. Events such as GETMAINs and variable initialization will be taking place.</p> <p>This message is issued by the following CSECT: DSNESH32</p> |
| <i>csectname</i> STARTING WRAPUP | <p>The CSECT specified in the message is starting its cleanup processing. Events such as FREEMAINs will be taking place.</p> <p>This message is issued by the following CSECT: DSNESH32</p> |
| <i>csectname</i> BEFORE GETMAIN | <p>The CSECT specified in the message is about to perform a GETMAIN of the area that holds the YCOLWID array.</p> <p>This message is issued by the following CSECTs: DSNESH32, DSNESH21</p> |
| <i>csectname</i> AFTER GETMAIN | <p>The CSECT specified in the message has completed a GETMAIN of the area that holds the YCOLWID array.</p> <p>This message is issued by the following CSECTs: DSNESH32, DSNESH21</p> |
| <i>csectname</i> BEFORE MAIN LOOP | <p>The CSECT specified in the message is about to begin running the CSECT's main loop.</p> <p>This message is issued by the following CSECTs: DSNESH30, DSNESH32, DSNESH81</p> |
| <i>csectname</i> LOOPTOP | <p>This message is issued after the main loop in the referred to CSECT specified in the message begins running.</p> <p>This message is issued by the following CSECTs: DSNESH21, DSNESH30, DSNESH32, DSNESH81</p> |

| Message text | Explanation |
|---|--|
| <i>csectname</i> TYPE IS INTEGER | <p>The CSECT specified in the message has detected a data type of integer.</p> <p>This message is issued by the following CSECT: DSNЕСM32</p> |
| <i>csectname</i> TYPE IS SMALL INTEGER | <p>The CSECT specified in the message has detected a data type of small integer.</p> <p>This message is issued by the following CSECT: DSNЕСM32</p> |
| <i>csectname</i> TYPE IS DECIMAL | <p>The CSECT specified in the message has detected a data type of decimal.</p> <p>This message is issued by the following CSECTS: DSNЕСM30, DSNЕСM32</p> |
| <i>csectname</i> TYPE IS FLOAT | <p>The CSECT specified in the message has detected a data type of float.</p> <p>This message is issued by the following CSECTS: DSNЕСM30, DSNЕСM32</p> |
| <i>csectname</i> TYPE IS CHARACTER | <p>The CSECT specified in the message has detected a data type of character.</p> <p>This message is issued by the following CSECTS: DSNЕСM30, DSNЕСM32</p> |
| <i>csectname</i> TYPE IS VARCHAR | <p>The CSECT specified in the message has detected a data type of VARCHAR.</p> <p>This message is issued by the following CSECT: DSNЕСM32</p> |
| <i>csectname</i> TYPE IS LONG VARCHAR | <p>The CSECT specified in the message has detected a data type of LONG VARCHAR.</p> <p>This message is issued by the following CSECT: DSNЕСM32</p> |
| <i>csectname</i> CHECKING FOR COLUMN HEADER TRUNCATION | <p>The CSECT specified in the message has started the section of code that checks for column headers that will not fit in the space allowed for the column (as specified on SPUFI panel 2).</p> <p>This message is issued by the following CSECT: DSNЕСM32</p> |
| <i>csectname</i> COLUMN HEADER TRUNCATION OCCURRED | <p>The CSECT specified in the message has discovered a column header that will not fit into the space allotted for it on panel 2. This will cause a message to be written to the output data set.</p> <p>This message is issued by the following CSECT: DSNЕСM32</p> |

| Message text | Explanation |
|--|---|
| <i>csectname</i> BEFORE WRITING GUTTER CHARACTER | <p>The CSECT specified in the message is about to write the string of characters that appears between columns of data in SPUFI output.</p> <p>This message is issued by the following CSECT: DSNESM32</p> |
| <i>csectname</i> FOUND A QUOTE | <p>The CSECT specified in the message is scanning the SQL statement in the input data set and has just encountered a quotation ("). This quotation indicates the beginning or end of a quoted string.</p> <p>This message is issued by the following CSECT: DSNESM21</p> |
| <i>csectname</i> FOUND A DBCS SO | <p>The CSECT specified in the message is scanning the SQL statement in the input data set and has just encountered a shift out (SO) character. This indicates the beginning of a DBCS string or substring.</p> <p>This message is issued by the following CSECT: DSNESM21</p> |
| <i>csectname</i> FOUND A DBCS SI | <p>The CSECT specified in the message is scanning the SQL statement in the input data set and has just encountered a shift in (SI) character. This indicates the end of a DBCS string or substring.</p> <p>This message is issued by the following CSECT: DSNESM21</p> |
| <i>csectname</i> NOT END OF FILE YET | <p>The CSECT specified in the message has not yet discovered an end-of-file condition. It will continue to look for SQL input.</p> <p>This message is issued by the following CSECT: DSNESM21</p> |
| <i>csectname</i> BEGINNING FIRST TIME PROCESSING | <p>The CSECT specified in the message is now starting its initialization section. This section performs a GETMAIN and does variable initialization.</p> <p>This message is issued by the following CSECT: DSNESM21</p> |
| <i>csectname</i> COMPLETING FIRST TIME PROCESSING | <p>The CSECT specified in the message has finished its initialization processing.</p> <p>This message is issued by the following CSECT: DSNESM21</p> |

| Message text | Explanation |
|--|--|
| <i>csectname</i> ECHOED A RECORD TO OUTPUT DATA SET | <p>The CSECT specified in the message has written a line of SQL from the input data set to the output data set. The SQL statement is echoed for documentation purposes.</p> <p>This message is issued by the following CSECT: DSNESM21</p> |
| <i>csectname</i> FOUND AN APOSTROPHE | <p>The CSECT specified in the message is scanning the SQL statement in the input data set and has just encountered an apostrophe. This indicates the beginning or end of a quoted string.</p> <p>This message is issued by the following CSECT: DSNESM21</p> |
| <i>csectname</i> FOUND A SEMICOLON | <p>The CSECT specified in the message was scanning the SQL input from the SPUFI input data set and found a semicolon. This indicates the end of an SQL statement.</p> <p>This message is issued by the following CSECT: DSNESM21</p> |
| <i>csectname</i> BEFORE MOVING SQL TO IBUF | <p>The CSECT specified in the message is about to move part of an SQL statement (from the SPUFI input data set) into the statement buffer (IBUF).</p> <p>This message is issued by the following CSECT: DSNESM21</p> |
| <i>csectname</i> AFTER MOVING SQL TO IBUF | <p>The CSECT specified in the message has moved part of an SQL statement (from the SPUFI input data set) into the statement buffer (IBUF).</p> <p>This message is issued by the following CSECT: DSNESM21</p> |
| <i>csectname</i> NULL DATA FOUND | <p>The CSECT specified in the message has discovered a null data item. Null strings are about to be written to the SPUFI output data set.</p> <p>This message is issued by the following CSECT: DSNESM30</p> |
| <i>csectname</i> TYPE IS INTEGER OR SMALL INTEGER | <p>The CSECT specified in the message has discovered a data item of type integer or small integer.</p> <p>This message is issued by the following CSECT: DSNESM30</p> |

| Message text | Explanation |
|--|--|
| <i>csectname</i> BEFORE CONVERSION | <p>The CSECT specified in the message is about to perform a data conversion of either decimal, integer, or small integer to EBCDIC. This is so the data can be written to the output data set.</p> <p>This message is issued by the following CSECT: DSNЕСM30</p> |
| <i>csectname</i> AFTER CONVERSION | <p>The named CSECT just performed a data conversion of either decimal, integer, or small integer to EBCDIC. This is so the data can be written to the output data set.</p> <p>This message is issued by the following CSECT: DSNЕСM30</p> |
| <i>csectname</i> TRUNCATION OCCURRED | <p>The CSECT specified in the message has discovered a data item that will not fit in the space allotted for it on the second SPUFI panel. A message will be written to the output data set to indicate that this has happened. Character-type data will be truncated and the remainder will be written. Numeric data will not be written; rather, it will show as all asterisks.</p> <p>This message is issued by the following CSECT: DSNЕСM30</p> |
| <i>csectname</i> NO TRUNCATION OCCURRED | <p>The CSECT specified in the message has discovered a data item that fits in the space allotted for it on the second SPUFI panel. No truncation has occurred.</p> <p>This message is issued by the following CSECT: DSNЕСM30</p> |
| <i>csectname</i> BEFORE CALLING DSNЕСM34 | <p>The CSECT specified in the message is about to call CSECT DSNЕСM34 to convert a floating-point number to EBCDIC.</p> <p>This message is issued by the following CSECT: DSNЕСM30</p> |
| <i>csectname</i> AFTER CALLING DSNЕСM34 | <p>The CSECT specified in the message has called CSECT DSNЕСM34 to convert a floating-point number to EBCDIC.</p> <p>This message is issued by the following CSECT: DSNЕСM30</p> |
| <i>csectname</i> TYPE IS CHAR, VARCHAR, OR LVARCHAR | <p>The CSECT specified in the message has detected a data item of type character, VARCHAR, or LONG VARCHAR.</p> <p>This message is issued by the following CSECT: DSNЕСM30</p> |

| Message text | Explanation |
|--|--|
| <i>csectname</i> LEAVE YSTUFBUF WITH RBUFU NOW = offset | <p>The CSECT specified in the message has inserted a string of characters into the row buffer so that it can be written to the output data set. The next available free position in the output buffer (RBUF) is now offset.</p> <p>This message is issued by the following CSECT: DSNESM30</p> |
| <i>csectname</i> TYPE IS GRAPHIC | <p>The CSECT specified in the message has detected a data type of GRAPHIC.</p> <p>This message is issued by the following CSECT: DSNESM32</p> |
| <i>csectname</i> TYPE IS VARGRAPHIC OR LONG VARGRAPHIC | <p>The CSECT specified in the message has detected a data type of VARGRAPHIC or LONG VARGRAPHIC.</p> <p>This message is issued by the following CSECT: DSNESM32</p> |
| <i>csectname</i><i>csectname</i> - START COLUMN LOOP WITH SQLD = colcount | <p>The program is about to enter a loop that will cycle once for each column looking for truncated data or column names.</p> <p>This message is issued by the following CSECT: DSNESM40</p> |
| <i>csectname</i> - BEGIN YBLDL SUBROUTINE | <p>CSECT DSNE55 has entered the YBLDL internal subroutine.</p> <p>This message is issued by the following CSECT: DSNESM55</p> |
| <i>csectname</i> - END YBLDL SUBROUTINE - OPEN oresult; BLDL bresult | <p>The BLDL has just been completed with an open return code of "oresult " and a BLDL return code of "bresult."</p> <p>This message is issued by the following CSECT: DSNESM55</p> |
| <i>modid</i> - END EXECUTION returncode | <p>The CSECT <i>csectname</i> has returned to the caller. The "returncode " in the message is its return code.</p> <p>This message is issued by the following CSECTs: DSNESM20, DSNE52</p> |
| <i>csectname</i> - EXTRACTED DAIRFAIL TEXT DISPLAYED dftext2 | <p>This message will accompany message DSNE582. The "dftext2" in the message is the second-level error explanation from DAIRFAIL.</p> |
| DSNESM55 - BEFORE CLOSE OF DATA SET | <p>DSNESM55 is about to close a data set.</p> <p>This message is issued by the following CSECT: DSNESM55</p> |

| Message text | Explanation |
|---|---|
| DSNESM55 - AFTER CLOSE OF DATA SET | <p>DSNESM55 has closed a data set.</p> <p>This message is issued by the following CSECT: DSNESM55</p> |
| DSNESM55 - END OF MODULE EXECUTION - RCODE = <i>returncode</i> | <p>DSNESM55 has terminated.</p> <p>This message is issued by the following CSECT: DSNESM55 The "returncode " in the message is its return code.</p> |
| DSNESM55 - END OPEN SUBROUTINE | <p>DSNESM55 has completed the internal OPEN subroutine.</p> <p>This message is issued by the following CSECT: DSNESM55</p> |
| DSNESM55 - BEGIN OPEN SUBROUTINE | <p>The CSECT DSNESM55 is attempting to OPEN a data set.</p> <p>This message is issued by the following CSECT: DSNESM55</p> |
| DSNESM55 - END DEALLOCATION SUBROUTINE | <p>The deallocation subroutine of CSECT DSNESM55 has completed.</p> <p>This message is issued by the following CSECT: DSNESM55</p> |
| DSNESM55 - AFTER DYNALLOC DEALLOCATION REQUEST | <p>The CSECT DSNESM55 has issued a DYNALLOC request to deallocate a data set.</p> <p>This message is issued by the following CSECT: DSNESM55</p> |
| DSNESM55 - BEFORE DYNALLOC DEALLOCATION REQUEST | <p>The CSECT DSNESM55 is about to request deallocation of a data set.</p> <p>This message is issued by the following CSECT: DSNESM55</p> |
| DSNESM55 - BEFORE DEALLOCATION SUBROUTINE | <p>The CSECT DSNESM55's deallocation subroutine has been entered.</p> <p>This message is issued by the following CSECT: DSNESM55</p> |
| DSNESM55 - END OF ALLOCATION SUBROUTINE | <p>The CSECT DSNESM55's deallocation subroutine has completed.</p> <p>This message is issued by the following CSECT: DSNESM55</p> |
| DSNESM55 - AFTER DYNALLOC ALLOCATION REQUEST | <p>The CSECT DSNESM55 has issued a DYNALLOC request.</p> <p>This message is issued by the following CSECT: DSNESM55</p> |

| Message text | Explanation |
|--|--|
| DSNESM55 - BEFORE DYNALLOC ALLOCATION REQUEST | <p>The CSECT DSNESM55 is about to issue a DYNALLOC request.</p> <p>This message is issued by the following CSECT: DSNESM55</p> |
| DSNESM55 - BEFORE ALLOCATION SUBROUTINE | <p>The CSECT DSNESM55's allocation subroutine has been entered.</p> <p>This message is issued by the following CSECT: DSNESM55</p> |
| DSNESM55 - AFTER INITIALIZATION | <p>Initialization for the current call to CSECT DSNESM55 has completed.</p> <p>This message is issued by the following CSECT: DSNESM55</p> |
| <i>modid</i> - BEGIN EXECUTION | <p>The CSECT "csectname" has been entered.</p> <p>This message is issued by the following CSECTS: DSNESM20, DSNESM22</p> |
| CALL PRH. RALE=<i>rale#</i>, FVLE=<i>fvle#</i>, QUAL=<i>qualifier</i> | <p>CSECT DSNESM70 is about to transfer program control to DSNAPRH, the application program request handler, for connection to Db2. The RALE, FVLE, and QUAL are parameters that are passed to the APRH.</p> <p>This message is issued by the following CSECT: DSNESM70</p> <p>A TSO trace message is issued.</p> |
| CONNECTION TO SUBSYSTEM <i>subsystem-id</i> SUCCESSFUL | <p>SPUFI has successfully completed all the necessary protocols to access SQL processing functions. SPUFI is about to open the user-specified input and output data sets.</p> <p>This message is issued by the following CSECT: DSNESM00</p> <p>A TSO trace message is issued.</p> |
| END OF SQL INPUT AND OUTPUT PROCESSING | <p>SPUFI SQL execution function has just completed. The SQL statements in the input data set have been processed, and the results have been written to the output data set. Messages DSNE802 and DSNE805 together bracket the section of SPUFI code that is executed whenever the EXECUTE option is YES and the Db2 subsystem is available.</p> <p>This message is issued by the following CSECT: DSNESM00</p> <p>A TSO trace message is issued.</p> |

| Message text | Explanation |
|---|--|
| <i>csectname</i> JUST READ IN ONE SQL STATEMENT | <p>A SQL statement from the input data set is now in the SPUFI input buffer. The processing of this statement by Db2 SQL-dependent code is to begin immediately.</p> <p>This message is issued by the following CSECT: DSNESH20</p> <p>A TSO trace message is issued.</p> |
| CSECT <i>name</i> JUST COMPLETED PROCESSING ONE SQL STATEMENT | <p>The processing of the SQL statement (which is still in the SPUFI input buffer) by Db2 SQL dependent code has ended. The execution results have been written to the output data set. Messages DSNE810 and DSNE811 together bracket the parts of SPUFI code that are dependent on the protocols and data structures used by the Db2 SQL processing functions.</p> <p>This message is issued by the following CSECT: DSNESH20</p> <p>A TSO trace message is issued.</p> |
| <i>csectname</i> ABOUT TO WRITE A LINE OF OUTPUT | <p>The current content of the SPUFI output buffer has been formatted and is about to be written to the output data set.</p> <p>This message is issued by the following CSECT: DSNESH22</p> <p>A TSO trace message is issued.</p> |
| <i>csectname</i> - BEGIN EXECUTION - SQL REQUEST = <i>request-type</i> | <p>Program control has been given to the CSECT specified in the message. The "request-type" in the message is the Db2 request that is to be issued by DSNESH68. The meaning of the request-types (PREPARE, DESCRIBE, EXECUTE, OPEN, CLOSE, FETCH, COMMIT, ROLLBACK) correlate exactly with SQL statements with the same names. PRESCRIB, however, correlates with the composite SQL statement PREPARE INTO.</p> <p>This message is issued by the following CSECT: DSNESH68</p> <p>A TSO trace message is issued.</p> |
| <i>csectname</i> - END OF MODULE EXECUTION - SQLCODE = <i>sqlcode</i> | <p>Program control has been returned to the CSECT specified in the message from Db2. The SQLCODE in the SQLCA is displayed in hexadecimal.</p> <p>This message is issued by the following CSECT: DSNESH68</p> <p>A TSO trace message is issued.</p> |

Chapter 8. Collecting diagnostic data

To diagnose Db2 problems you might need to collect symptom descriptions, traces, logs, and dump data sets.

Before you begin

Prior to a problem occurring, you can set up your z/OS environment to collect diagnostic data. Collecting data before you open a support case with IBM Support can help expedite the resolution of the problem.

About this task

The type of data that is needed to diagnose a problem depends on the problem that is being investigated. Sometimes you cannot solve a problem with only a description of its symptoms. In such cases, you need to collect diagnostic data. Collecting data starts with answering questions like these to narrow down the list of system components that are involved:

- With which components are errors associated?
- Do the symptoms match any known problems? If so, has a fix or workaround been published?
- Can the problem be identified and resolved without a code fix?
- Under what conditions does the problem occur?

When you are researching the cause of your problem, you need to describe your problem and collect diagnostic data.

Procedure

1. Write a clear description of the problem and any steps that are required to re-create the problem.
2. Identify and record all messages and codes that were issued as a result of the problem.
3. Search the Db2 information for troubleshooting topics about your problem.
4. Search to check whether a fix or solution already exists for your problem.

See [Chapter 2, “Searching for known problems and solutions for Db2 for z/OS,” on page 3](#). For a list of fixes available for Db2 for z/OS, see the [IBM Support page for Db2 for z/OS](#).

5. Use the recommended procedures to diagnose the problem and to determine your next actions.

What to do next

If you cannot find documented procedures to solve your problem, you can contact IBM Support. You might be asked to collect logs, data sets, and dumps that can help determine the source of your problem. For certain problem symptoms, or for problems in a specific component, you might need to collect additional data. Follow the instructions to collect diagnostic data for problems of your type.

Related tasks

[Requesting product support for problems in Db2 for z/OS \(Troubleshooting problems in Db2\)](#)

Setting up the z/OS environment to collect diagnostic data

To collect diagnostic data efficiently, you need to set up your z/OS environment correctly.

About this task

The diagnostic data that is needed to diagnose Db2 problems might originate from several components in the z/OS environment. To ensure that the diagnostic data is available when problems occur, set up your environment to capture this data.

Maximizing the size of the z/OS system trace table

The system trace table is valuable in diagnosing various problems, such as abends, loops, and performance issues. If you increase the size of the trace table, you can collect system trace information for a longer time period.

Before you begin

System trace tables reside in fixed storage on each processor. Ensure that your system has enough real storage to support an increase in system trace table size. Before changing the size of the system trace table, you can view the current trace table settings by using the following DISPLAY TRACE z/OS operator command:

```
DISPLAY TRACE
```

About this task

The default size and maximum size of the system trace table depend on the z/OS version. Consider the amount of fixed storage that is available on your system when setting the size of the system trace. A commonly used trace table value is 5 M.

Procedure

Update the COMMNDxx member of the SYS1.PARMLIB partitioned data set to issue the TRACE command during z/OS initialization.

For example, the following z/OS command sets the system trace table size to 5 M:

```
TRACE ST,5M
```

Related concepts

[System trace \(MVS Diagnosis: Tools and Service Aids\)](#)

[Increasing the size of the z/OS system trace \(MVS Diagnosis: Tools and Service Aids\)](#)

Related reference

[z/OS DISPLAY TRACE command \(MVS System Commands\)](#)

[z/OS TRACE command \(MVS System Commands\)](#)

Related information

[z/OS COMMNDxx \(commands automatically issued at initialization\) \(MVS Initialization and Tuning Reference\)](#)

Increasing the size of the master trace table

The master trace maintains a wraparound table of the most recently issued operator messages. At the time of failure, these messages provide a view of external events. You might want to increase the size of the master trace table so that it includes trace messages for a longer time period.

About this task

The master trace table is embedded in dumps that have the TRT option. The default size of the master trace table is 24 KB, which is enough space for approximately 336 messages. A size of 500 KB is enough space for approximately 7000 messages.

Procedure

To increase the size of the master trace table to 500 KB, use one of the following approaches::

- Update the SCHEDxx member of the SYS1.PARMLIB partitioned data set to issue the following command during z/OS initialization:

```
MT SIZE(500K)
```

- Enter the following z/OS trace command:

```
TRACE MT,500K
```

Related concepts

[Master trace \(MVS Diagnosis: Tools and Service Aids\)](#)

Related reference

[z/OS TRACE command \(MVS System Commands\)](#)

Related information

[z/OS SCHEDxx member \(MVS Initialization and Tuning Reference\)](#)

Increasing the storage capacity for an SVC dump

By increasing the MAXSPACE setting, you can avoid a partial dump and capture the entire internal supervisor call (SVC) dump.

Before you begin

Ensure that the local page data sets are large enough to contain their normal peak load plus additional SVC dumps. Estimate the amount of storage that is needed by adding the total amount of global storage that is in use (at peak usage) in all regions. You can use an online monitor program to determine the storage that is used at peak time. If the MAXSPACE size setting is not large enough, partial dumps might be produced without all the necessary diagnostic data.

About this task

The default maximum virtual space that is allocated is 500 MB. The typical size of SDUMP MAXSPACE settings for large Db2 subsystems is 16000 MB.

Procedure

Update the COMMNDxx member of the SYS1.PARMLIB partitioned data set to issue the following CHNGDUMP command to set MAXSPACE to the appropriate size during z/OS initialization:

```
CHNGDUMP SET,SDUMP,MAXSPACE=16000M
```

Related concepts

[SVC Dump \(MVS Diagnosis: Tools and Service Aids\)](#)

Related reference

[z/OS CHNGDUMP command \(MVS System Commands\)](#)

Related information

[z/OS COMMNDxx \(commands automatically issued at initialization\) \(MVS Initialization and Tuning Reference\)](#)

Ensuring that dump data sets are available and automatically updated

To avoid partial supervisor call (SVC) dumps, you can ensure that automatic dump data set allocation is in place.

Before you begin

Ensure that the assigned storage class for dump data sets has enough disk space to avoid partial dumps and to capture the entire internal supervisor call (SVC) dump. You might need to increase the amount of virtual storage to capture volatile virtual storage data, capture summary dump data, and capture component-specific data.

Procedure

Use the COMMNDxx member of the SYS1.PARMLIB partitioned data set to issue the appropriate DUMPDS commands during z/OS initialization.

Example

The following commands create a name pattern, add data sets, and activate automatic allocation of dump data sets:

```
DUMPDS NAME=&SYSNAME. .&JOBNAME. .T&HHMMSS. .S&SEQ.
DUMPDS ADD,DSN=ALL
DUMPDS ALLOC=ACTIVE
```

Related reference

[z/OS DUMPDS command \(MVS System Commands\)](#)

Related information

[z/OS COMMNDxx \(commands automatically issued at initialization\) \(MVS Initialization and Tuning Reference\)](#)

Setting up a customized Db2 SVC dump

You can add customized parameters to the IEADMCxx member of SYS1.PARMLIB to simplify the operator interface to create Db2 supervisor call (SVC) dumps.

About this task

You can minimize the number of DUMP commands you enter by setting up a customized PARMLIB member before the error event.

Procedure

Create an IEADMCxx member of the SYS1.PARMLIB partitioned data set that contains parameters for a customized Db2 SVC dump on several subsystems. In the following example, member IEADMCD1 specifies customized options that are to be invoked by the DUMP command:

```
/*
THIS GETS INVOKED VIA CONSOLE COMMAND:
DUMP COMM=(...),PARMLIB=D1
*/
TITLE=('DUMP OF DSNT1 GROUP')
JOBNAME=(XCFAS,DT1*),
SDATA=(XESDATA,COUPLE,PSA,LPA,RGN,CSA,SUM,TRT),
REMOTE=(SYSLIST=(STLABBB('XCFAS','DT11*'),
STLABB6('XCFAS','DT12*'),
STLABBC('XCFAS','DT13*'),
STLABB7('XCFAS','DT14*')),SDATA),
END
```

In this example, wildcard characters are used to specify multiple parameter values to create dumps on multiple subsystems. The wildcard character * (asterisk) indicates multiple characters, and the wildcard character ? (question mark) indicates a single character.

Related concepts

[SVC Dump \(MVS Diagnosis: Tools and Service Aids\)](#)

[Dump suppression \(MVS Diagnosis: Tools and Service Aids\)](#)

Related tasks

[Requesting Db2 SVC dumps \(Collecting data\)](#)

Related reference

[z/OS DUMP command \(MVS System Commands\)](#)

[z/OS DUMP command wildcards \(MVS System Commands\)](#)

Related information

[z/OS IEADMCxx member \(MVS Initialization and Tuning Reference\)](#)

Setting up Db2 SLIP traps for a data sharing environment

In a data sharing environment, you can set up a PARMLIB member to create serviceability level indication processing (SLIP) traps. SLIP traps create dumps that can help you diagnose hangs or other problems.

About this task

In data sharing environments, a hang or problem on one Db2 member might be due to a problem that originates from another member. To find the problem, you need to create serviceability level indication processing (SLIP) trap supervisor call (SVC) dumps on the members involved. Ensure that dump parameters specify all appropriate address spaces on each system.

Procedure

Create an IEASLPxx member of the SYS1.PARMLIB partitioned data set containing parameters for a SLIP trap. In the following example, member IEASLPD1 specifies customized options for Db2 SVC dumps on several subsystems.

```
SLIP SET,IF,N=(IEAVEDS0,00,FF),ID=DB21,ML=1,
JOBLIST=(DT44*),A=SVCD,
SDATA=(COUPLE,CSA,GRSQ,LPA,LSQA,PSA,RGN,SQA,SUM,SWA,TRT,XESDATA),
REMOTE=(SYSLIST=(STLAB21,STLAB22,STLABBB,STLABBC),
JOBLIST,SDATA),END
```

In this example, wildcard characters are used to specify multiple parameter values. The wildcard character * (asterisk) indicates multiple characters, and the wildcard character ? (question mark) indicates a single character.

Related tasks

[Requesting data sharing environment Db2 SLIP traps \(Collecting data\)](#)

Related reference

[z/OS SLIP command \(MVS System Commands\)](#)

[z/OS SLIP command SET parameters \(MVS System Commands\)](#)

Related information

[z/OS IEASLPxx \(SLIP commands\) \(MVS Initialization and Tuning Reference\)](#)

Preserving standard diagnostic documentation

If an error occurs, you can preserve diagnostic data that might be helpful in problem diagnosis. You can create operating procedures to preserve commonly needed diagnostic information.

About this task

Due to the interactions between address spaces, you might sometimes need to obtain a dump of associated address spaces so that you can diagnose problems. To ensure that information that is needed to diagnose the problem is available, consider the following factors when requesting dumps of data:

If you request a dump of less data, the dump process might complete more quickly and capture diagnostic data as it existed at the time of the error. Thus more control block information might be synchronized. Conversely, if you request a complete address space dump, it might contain more data areas, but data areas might have changed since the error event occurred.

Consider these recommendations when deciding which data areas to include in the dump data set:

- If a slow-moving performance problem is being diagnosed, request a complete address space dump.
- If Db2 is in a hard-wait situation, request a complete address space dump.

- If you can narrow the problem to specific components, request a dump of only the pertinent address space and data areas.
- If you cannot narrow the problem to a specific component, request a dump of all related address spaces with minimal data for each address space.

Procedure

To collect and preserve diagnostic data, complete the following tasks.

Preserving the z/OS console (SYSLOG)

The z/OS console log (SYSLOG) contains messages about the state of the operating system. Because these messages might indicate the source of the problem, retaining the SYSLOG data set is recommended.

About this task

The SYSLOG data set contains system-level messages and commands about the state of the z/OS system. The SYSLOG data set provides symptom data about failures of components of the operating system. The SYSLOG data set for each LPAR that is involved needs to be preserved. The SYSLOG is a binary data set, but it can be viewed using the Spool Display and Search Facility (SDSF). Retaining four hours of SYSLOG data is enough for 90% of problem situations.

Procedure

At the time of an error event, copy the SYSLOG data set to preserve its contents.

Related concepts

[SYSLOG records \(MVS Diagnosis: Tools and Service Aids\)](#)

Related reference

[z/OS SDSF Operation and Customization](#)

Preserving LOGREC data

The log record (LOGREC) data contains messages about hardware and software errors. Therefore, retaining the LOGREC data is recommended.

About this task

The LOGREC data contains information about hardware and software errors across the z/OS system. LOGREC data provides symptom data about failures of components. You can use LOGREC data with corresponding dump data sets. The LOGREC data is written to a log stream or the data set SYS1.LOGREC. One way to read LOGREC data is to use the Environmental Record, Editing, and Printing program (EREP). The LOGREC entries that are recorded near the time of abend might provide valuable historical information about the events leading up to the error. Retaining four hours of data is enough for 90% of problem situations.

LOGREC data tracks the recovery of Db2 abends and provides evidence of FRR (functional recovery routines) percolations and retries.

Procedure

At the time of an error event, copy the LOGREC data set to preserve its contents.

Related concepts

[Recording LOGREC error records \(MVS Diagnosis: Tools and Service Aids\)](#)

Preserving the JES job logs for key Db2 address spaces

The JES job log from key Db2 address spaces contains relevant job-related messages. Therefore, retaining the JES job log is recommended.

About this task

The JES job log contains job-related messages that can help you diagnose certain problems that might occur. Therefore, retaining the JES job log from key Db2 address spaces is recommended. You can view JES job log information using the System Display and Search Facility (SDSF).

Procedure

To view relevant messages for Db2 address spaces, save the JES job logs for:

- *ssnmDBM1* address space
- *ssnmMSTR* address space
- *ssnmDIST* address space if distributed data facility (DDF) is in use
- *ssnmWLM* address space if Db2 stored procedures run in WLM-managed address spaces

Related tasks

[Setting up a WLM application environment for stored procedures during installation \(Db2 Installation and Migration\)](#)

Related reference

[z/OS SDSF Operation and Customization](#)

Retention of Db2 dump data sets

To ensure that you have diagnostic data available if a problem occurs, you need to retain SYS1.DUMP data sets and dump data sets that were dynamically allocated.

In the Db2 subsystem, a problem might not have a clear origin. Dump data sets are a representation of virtual storage at the time that an error occurs. Therefore, you need to retain all dump data sets at the time of failure regardless of the subsystem of origin. Specifically, you need to retain SYS1.DUMPxx and all dynamically allocated dump data sets. Multiple dump data sets might be created. You might need to examine all of the information in these dumps by using the Interactive Program Control System (IPCS).

Retention of original dumps for redacted buffer pool data

Db2 marks buffer pool data in the dump as `sensitive=yes` for data privacy. You can optionally redact the buffer pool data before you send the dump to IBM Support for analysis. However, always retain the original dump until all problem analysis concludes, in case IBM Support requests specific information from the redacted data. For more information see [Redacting buffer pool data in SVC dumps for data privacy \(Troubleshooting problems in Db2\)](#).

Related reference

[Introduction to IPCS \(MVS IPCS User's Guide\)](#)

Retention of Db2 logs

Db2 log data sets are often helpful when you diagnose problems. Therefore, retaining Db2 log data sets is recommended.

To ensure that you have all of the information that might be needed to solve the problem, retain approximately 4 hours worth of Db2 log data prior to the point at which the failure occurred. This log data is especially important for Db2 problems that involve data inconsistency and restart. For data inconsistency, log data for all updates might be needed from the last time the page was valid, which could be a few days. In this case, IBM Support might provide you with a Db2 DSN1LOGP utility job to limit the output.

Related reference

[DSN1LOGP \(Db2 Utilities\)](#)

Requesting Db2 SVC dumps

You can use the system console DUMP command to request a Db2 supervisor call (SVC) dump.

Before you begin

Set up a customized member in the SYS1.PARMLIB partitioned data set before issuing the DUMP command. All jobs must be active; otherwise, Db2 does not produce a dump on the system on which you enter the dump command. If you do not specify a JOBNAME for a remote system, a dump of the entire DUMPSERV address space might be generated. Also ensure that your system is not suppressing the generation of SVC dumps.

Procedure

Issue the DUMP command from the system console.

For example, to use the IEADMCD1 member of the SYS1.PARMLIB partitioned data set, issue the following DUMP command:

```
DUMP TITLE=(DUMP OF DB2 production),PARMLIB=(D1)
```

What to do next

Db2 marks buffer pool data in the dump as `sensitive=yes` for data privacy. You can optionally redact the buffer pool data before you send the dump to IBM Support for analysis. However, always retain the original dump until all problem analysis concludes, in case IBM Support requests specific information from the redacted data. For more information see [Redacting buffer pool data in SVC dumps for data privacy \(Troubleshooting problems in Db2\)](#).

Related concepts

[SVC Dump \(MVS Diagnosis: Tools and Service Aids\)](#)

[Dump suppression \(MVS Diagnosis: Tools and Service Aids\)](#)

Related tasks

[Setting up a customized Db2 SVC dump \(Collecting data\)](#)

Related information

[z/OS IEADMCDxx member \(MVS Initialization and Tuning Reference\)](#)

Requesting data sharing environment Db2 SLIP traps

In a Db2 data sharing environment, a hang or problem on one Db2 member might originate in a separate member. To find the problem, you need to create a serviceability level indication processing (SLIP) trap to catch error events and to create supervisor call (SVC) dumps.

Before you begin

Be sure to set up the IEASLPxx member in the SYS1.PARMLIB partitioned data set. Before activating the SLIP trap, disable all existing program event recordings (PER) SLIP traps for each system image in the data sharing group. For example, use the following command, where *trap-id* identifies the SLIP trap:

```
ROUTE *ALL,SLIP MOD,DISABLE,ID=trap-id
```

Procedure

1. To activate the SLIP trap using the IEASLPD1 member, enter the following system command:

```
SET SLIP=D1
```

2. To invoke the associated supervisor call (SVC) dump for trap identifier DB21, enter the system command:

```
SLIP MOD,ENABLE, ID=DB21
```

Results

When the conditions that you defined in the SLIP trap occur, matching dumps are produced based on the wildcard specification in the IEASLPxx member.

Related tasks

[Setting up Db2 SLIP traps for a data sharing environment \(Collecting data\)](#)

Related reference

[z/OS SLIP command \(MVS System Commands\)](#)

Related information

[z/OS IEASLPxx \(SLIP commands\) \(MVS Initialization and Tuning Reference\)](#)

Collecting service SQL documentation

Service SQL documentation is the standard format for query problem analysis documentation, for use by IBM Support. It includes the data definition statements (DDL), catalog statistics, EXPLAIN records, and other data that describes a Db2 for z/OS environment.

Before you begin

- Ensure that the latest Db2 maintenance is applied.
- You must have appropriate Db2 administrative authority.
- If you want to report an access path performance problem and you have not already done so, complete the following tasks first:
 1. Investigate the problem, as described in [Investigating access path problems \(Db2 Performance\)](#).
 2. If you are unable to resolve the problem, collect diagnostic data, as described in [Collecting data for access path performance problems \(Collecting data\)](#).

About this task

The service SQL documentation includes information that describes the following aspects of the environment for Db2 for z/OS SQL statements:

- Objects such as tables, indexes and views that are referenced by the SQL statements
- Catalog statistics the Db2 to select access paths for the SQL statements
- EXPLAIN table information that describes access paths for the SQL statements
- Db2 module details
- Subsystem parameter settings

IBM Support can use this information to re-create and analyze the performance problems.

Procedure

To generate problem analysis documentation for a Db2 environments:

1. If you have not already done so, run EXPLAIN statements to populate the EXPLAIN table records for the SQL statement or workload.
2. Generate the data definition statements, EXPLAIN table records, and catalog statistics that describe the Db2 environment for the SQL statement or workload.

You can use the following approaches to call the ADMIN_INFO_SQL stored procedure:

- Run the DSNTEJ6I sample job that is supplied with Db2 in the *prefix.SDSNSAMP* data set. DSNTEJ6I calls DSNADMSB which executes SYSPROC.ADMIN_INFO_SQL. The job prologue contains detailed instructions for customizing the job. For more information, see [DSNTEJ6I \(Db2 Programming samples\)](#).
- Use the *capture query environment* feature of tools such as [IBM Db2 Administration Foundation for z/OS](#) and [IBM Db2 for z/OS Developer Extension](#)

Tip: Always select the **Parallelism** option, even if your system does not use parallelism. This option ensures that important statistics are included in the Service SQL data.

The output from ADMIN_INFO_SQL is intended primarily for the use of IBM Support. The format and content of the output might change at any time.

What to do next

For information about submitting the service SQL data to IBM Support, see [Requesting product support for problems in Db2 for z/OS \(Troubleshooting problems in Db2\)](#).

Related tasks

[Investigating access path problems \(Db2 Performance\)](#)

[Collecting data for CCSID problems \(Collecting data\)](#)

Related reference

[DSNADMSB \(Db2 Utilities\)](#)

[ADMIN_INFO_SQL stored procedure \(Db2 SQL\)](#)

Collecting data for specific types of Db2 problems

Depending on the type of Db2 problem that you are diagnosing, you might collect different types of diagnostic data.

Before you begin

If you contact IBM Support regarding a Db2 problem, you might be asked to collect logs, data sets, and dumps that can help determine the source of your problem. Before a problem occurs, set up your z/OS environment to collect diagnostic data. Collecting data before you open a support case can help expedite the problem resolution process.

About this task

Review the following topics to determine the type of diagnostic data that you might collect for your problem.

What to do next

If you cannot find documented procedures to correct the problem, contact IBM Support by following the instructions in [Requesting product support for problems in Db2 for z/OS \(Troubleshooting problems in Db2\)](#).

Related tasks

[Setting up the z/OS environment to collect diagnostic data \(Collecting data\)](#)

Collecting data for general performance problems

For performance problems that are not related to access path issues, collect a description of the problem symptoms, SMF data, RMF data, and a supervisor call (SVC) dump.

Before you begin

To diagnose performance problems, you must have the appropriate authorization to collect data from different z/OS components.

Procedure

To collect diagnostic data for general performance problems:

1. Write a description of the problem and include answers to these questions:
 - a) What are the names of packages, user IDs, and subsystems that are experiencing performance problems?
 - b) If a job abended, what were the abend codes?
 - c) Has anything recently changed in the environment that caused the problem?
 - d) How do you reproduce the problem?
2. Collect SMF type 100-102 trace records by issuing Db2 START TRACE commands for both a successful and an unsuccessful run.
For example, start both accounting and statistics traces:

GUIP

```
-START TRACE(ACCTG) CLASS(1,2,3,7,8) DEST(SMF)
-START TRACE(STAT) CLASS(1,3,4,5,6) DEST(SMF)
```

GUIP

Separate accounting data from other transactions to prevent accumulation. In a Resource Recovery Services (RRS) or distributed data facility (DDF) attach environment, set the ACCUMACC subsystem parameter to NO.

3. Collect RMF Monitor I and RMF Monitor II SMF type 70-79 trace records by issuing RMF commands for both a successful and unsuccessful run:
 - a) Start RMF Monitor I:

```
MODIFY RMF, START ZZ
```

- b) Start RMF Monitor II:

```
MODIFY RMF, START session-id
```

In this example, *session-id* is two alphanumeric characters (not ZZ) to start RMF Monitor II.

4. Issue the z/OS DUMP command to dump Db2 associated address spaces:

```
DUMP COMM=(title),
JOBNAME=(ssnmIRLM,ssnmMSTR,ssnmDBM1,ssnmDIST),
SDATA=(RGN,CSA,SQA,LPA,LSQA,SWA,PSA,ALLNUC,XESDATA,TRT,GRSQ,SUM),END
```

where *title* is the title of the dump data set and *ssnm* is the Db2 subsystem name.

5. If IBM Support asks you to do so, issue this command:

PSPI

```
DISPLAY DATABASE(database-name) SPACENAM(space-name) PART(part-num) LRSN
```

PSPI

This command is used for diagnosis of performance problems that are related to over-locking or inefficient space reuse.

6. Collect service SQL documentation that describes the environment for the SQL statement, as described in [Collecting service SQL documentation \(Troubleshooting problems in Db2\)](#).

Related concepts

[SVC Dump \(MVS Diagnosis: Tools and Service Aids\)](#)

Related tasks

[Requesting Db2 SVC dumps \(Collecting data\)](#)

[Requesting data sharing environment Db2 SLIP traps \(Collecting data\)](#)

Related reference

[z/OS RMF User's Guide](#)

[-START TRACE command \(Db2\) \(Db2 Commands\)](#)

[DDF/RRSAF ACCUM field \(ACCUMACC subsystem parameter\) \(Db2 Installation and Migration\)](#)

[STATISTICS TIME field \(STATIME subsystem parameter\) \(Db2 Installation and Migration\)](#)

Collecting data for access path performance problems

For performance problems that are related to access path issues, you can collect a description of the problem symptoms, the output of the EXPLAIN statement for the query, related data definition statements, and catalog statistics.

Before you begin

- Ensure that the data is well organized and that complete, accurate, and current statistics are available for relevant database objects. For more information, see [Maintaining data organization and statistics \(Db2 Performance\)](#).
- Try to resolve the access path problem by using the statistics advisor feature of free tools such as Data Server Manager or your own analysis of the statistics, as described in [Investigating access path problems \(Db2 Performance\)](#).
- To collect data to diagnose performance problems, you need the appropriate Db2 administrative authority.
- Ensure that the latest Db2 maintenance is applied.

About this task

Most access path performance issues and access path performance regressions can be resolved by ensuring that a complete, current, and accurate set of statistics from the RUNSTATS utility is available to Db2. Include the basic statistics that are needed for all database objects, and selectivity statistics that support the particular SQL statement.

If you contact IBM Support, you can provide information to help diagnose your access path problems.

Tip: Enhanced query tuning capabilities that can help you with this task are available in IBM Db2 Query Workload Tuner for z/OS and IBM Db2 Administration Foundation for z/OS.

Procedure

To collect access path diagnostic data to send to IBM Support:

1. Generate an EXPLAIN report of the query during a time period when the query performed poorly. For example, issue the following EXPLAIN statement. Replace *query-number* with the PLAN_TABLE rows for the query, and replace *problem-SQL-statement* with the SQL statement.

```
EXPLAIN PLAN SET QUERYNO = query-number FOR
problem-SQL-statement;
```

You can issue the following SQL statement to create a report that describes the access path for the SQL statement:

```
SELECT *
FROM PLAN_TABLE
WHERE QUERYNO = query-number
ORDER BY TIMESTAMP, QUERYNO, QBLOCKNO, PLANNO, MIXOPSEQ;
```

2. Collect service SQL documentation that describes the Db2 environment for the SQL statement or workload, as described in [Collecting service SQL documentation \(Troubleshooting problems in Db2\)](#). If possible, collect the service SQL documentation for both before and after the access path problem occurred.

Related concepts

[Investigating SQL performance by using EXPLAIN \(Db2 Performance\)](#)

Related tasks

[Managing and preventing access path change \(Db2 Performance\)](#)

Related reference

[EXPLAIN statement \(Db2 SQL\)](#)

[RUNSTATS \(Db2 Utilities\)](#)

Collecting data for data access problems

You can collect diagnostic data to diagnose data access problems.

About this task

Collect the following types of information:

- Supervisor call (SVC) dumps
- Service SQL documentation, as described in [Collecting service SQL documentation \(Troubleshooting problems in Db2\)](#).
- Other system data sets that are needed to resolve Db2 data access problems.

Collecting data for incorrect output from an SQL statement

If the results of an SQL query are unexpected, collect a copy of the SQL statement, the related DDL and catalog statistics, and the EXPLAIN output to diagnose the problem.

Before you begin

To collect data to diagnose incorrect SQL results, you need the appropriate Db2 administrative authority.

Procedure

To collect data for incorrect output from an SQL statement:

1. Obtain a copy of the SQL statement to investigate.
2. Write a detailed description of the SQL statement results. Describe whether the results returned contain unexpected rows, too few rows, or too many rows. Also, describe any recent changes to the Db2 subsystem such as Db2 migrations, maintenance, application code updates, or access path changes.
3. Generate an EXPLAIN report of the query.
For example, use the following EXPLAIN statement. Replace *query-number* with the row identifier in the PLAN_TABLE and *problem-SQL-statement* with the SQL statement.

```
EXPLAIN PLAN SET QUERYNO = query-number FOR
{problem-SQL-statement}
SELECT *
```

```
FROM PLAN_TABLE
WHERE QUERYNO = query-number
ORDER BY TIMESTAMP, QUERYNO, QBLOCKNO, PLANNO, MIXOPSEQ;
```

4. Collect service SQL documentation that describes the environment for the SQL statement, as described in [Collecting service SQL documentation \(Troubleshooting problems in Db2\)](#).

Related reference

[EXPLAIN statement \(Db2 SQL\)](#)

Collecting data when an abend occurs for an SQL query

If an abend occurs during an SQL query, collect the SQL statement, EXPLAIN report, DDL, supervisor call (SVC) dump, LOGREC, and SYSLOG diagnostic data.

Before you begin

To diagnose query abend problems, you must have the appropriate authorization to collect data from different z/OS components.

Procedure

Collect the following data to diagnose an abend for an SQL query:

- A copy of the SQL statement to investigate
- Service SQL documentation that describes the environment for the SQL statement, as described in [Collecting service SQL documentation \(Troubleshooting problems in Db2\)](#)
- The LOGREC, which might contain messages during the time of the error
- The SYSLOG data set messages from the *ssnn*MSTR address space
- Any supervisor call (SVC) dump that is generated by the abend

Related concepts

[SVC Dump \(MVS Diagnosis: Tools and Service Aids\)](#)

[Dump suppression \(MVS Diagnosis: Tools and Service Aids\)](#)

Related tasks

[Preserving LOGREC data \(Collecting data\)](#)

[Preserving the z/OS console \(SYSLOG\) \(Collecting data\)](#)

[Setting up a customized Db2 SVC dump \(Collecting data\)](#)

[Requesting Db2 SVC dumps \(Collecting data\)](#)

Related reference

[EXPLAIN statement \(Db2 SQL\)](#)

Collecting data for stored procedure problems

If Db2 returns SQLCODE -440, collect data for diagnosing stored procedure not found conditions.

Before you begin

To diagnose performance problems, you must have the appropriate authorization to collect data from different z/OS components.

Procedure

Collect the following data to diagnose stored procedure problems:

- A detailed problem description which includes:
 - A description of the stored procedure with a list of source code programming languages

- A copy of source code of the stored procedure
- The CREATE PROCEDURE SQL statement
- The WLM PROC startup JCL
- For SQLCODE -440 errors, include:
 - The value of CURRENT PATH special register
 - The BIND PACKAGE command for the stored procedure
 - The data types of parameters from the CALL statement to the stored procedure

Related tasks

[Troubleshooting Db2 stored procedure problems \(Db2 Administration Guide\)](#)

Related reference

[CREATE PROCEDURE statement \(overview\) \(Db2 SQL\)](#)

[WLM PROC NAME field \(Db2 Installation and Migration\)](#)

[CURRENT PATH special register \(Db2 SQL\)](#)

Related information

[-440 \(Db2 Codes\)](#)

Collecting data for authorization problems

If Db2 returns SQLCODE -551, which indicates an operation was denied, and you think it is valid, collect data for diagnosing authorization problems.

Before you begin

To diagnose authorization problems, you must have the appropriate authorization to collect data from different z/OS components.

About this task

The type of diagnosis data that is collected is different for systems that use Db2 internal security than external security.

Procedure

Collect the following data to diagnose authorization problems:

- Service SQL information for the SQL statement to investigate, as described in [Collecting service SQL documentation \(Troubleshooting problems in Db2\)](#).
- The complete SQLCA that is returned
- The Db2 audit trace class 1 and 7 and Db2 performance trace class 22 from the time of failure
- The source code for any Db2 exits (Sign-on, Collection, or Authorization) that are in effect
- The bind parameters for the failing package

Collecting data for CCSID problems

If CCSID settings are not correct, Db2 might display an incorrect character or encounter a CCSID conversion error. If Db2 returns an unexpected character from a query, message DSNT552I, SQLCODE -330, or SQLCODE -332, collect data for diagnosing CCSID problems.

Procedure

To collect data to diagnose CCSID problems:

1. If you receive an unexpected display character or message DSNT552I, collect the application default CCSID setting that is stored in DSNHDECP.

2. If you receive an unexpected SQLCODE, such as SQLCODE -330 or SQLCODE -332, generate a DSN1SDMP data set.
3. Collect the bind encoding CCSID from the catalog.
For example, issue the following select statements, where:

- *collection-id* is the id of the collection
- *package-name* is the name of the package
- *plan-name* is the name of the plan

```
SELECT ENCODING_CCSID FROM SYSIBM.SYSPACKAGE WHERE COLLID=collection-id
AND NAME=package-name;
SELECT ENCODING_CCSID FROM SYSIBM.SYSPLAN WHERE NAME=plan-name;
```

4. Collect the object CCSID settings from the catalog for any involved databases, table spaces, columns, and stored procedures.
For example, issue the following select statements, where:

- *database-name* is the name of the database
- *tablespace-name* is the name of the table space
- *column-name* is the name of the column
- *table-name* is the name of the table
- *routine-name* is the name of the routine
- *schema-name* is the name of the schema

```
SELECT SBCS_CCSID FROM SYSIBM.SYSDATABASE WHERE NAME=database-name;
SELECT SBCS_CCSID FROM SYSIBM.SYSTABLESPACE WHERE NAME=tablespace-name
AND DBNAME=database-name;
SELECT CCSID FROM SYSIBM.SYSCOLUMNS WHERE NAME=column-name
AND TBNAME=table-name;
SELECT CCSID FROM SYSIBM.SYSPARMS WHERE NAME=routine-name
AND SCHEMA=schema-name;
```

5. Collect the hexadecimal representation of the column data in error.
For example, issue the following select statement, where:

- *column-in-error* is the name of the column with the invalid character
- *user-table* is the name of the user table with the invalid character

```
SELECT column-in-error, HEX(column-in-error) FROM user-table
```

6. If you are using a local terminal emulator program, find the CCSID setting.
The Personal Communications (PCOM) CCSID setting is in the **"Personal Parameters"** menu **"Host Code Page"** field. For other terminal emulator programs, consult related documentation for its CCSID or code page setting.
7. If data is being passed from a distributed system, generate a client performance trace.
For example, issue the following -START TRACE command, where *auth-id* is the authorization ID needed to start the trace command:

```
-START TRACE(PERFM) CLASS(1,2,32) DEST(GTF) TDATA(COR,TRA) IFCID(180) AUTHID(auth-id)
```

Related concepts

[Application defaults parameters \(Db2 Installation and Migration\)](#)

Related tasks

[Debugging CCSID and Unicode problems \(Db2 Internationalization Guide\)](#)

[Collecting data for an unexpected SQLCODE \(Collecting data\)](#)

Related reference

[Character conversion terminology \(Db2 Internationalization Guide\)](#)

[-START TRACE command \(Db2\) \(Db2 Commands\)](#)

Related information

[DSNT552I \(Db2 Messages\)](#)

[-330 \(Db2 Codes\)](#)

[-332 \(Db2 Codes\)](#)

Collecting data for corruption and inconsistency problems

If you encounter corrupted or inconsistent data problems, collect diagnostic data from a time period when the problem occurred. DSNI and DSNT messages usually indicate that the Db2 data manager subcomponent encountered a problem.

About this task

Diagnosing data corruption problems and inconsistencies between indexes and data require IBM Software Support expertise. A common reason for these errors is the incorrect use of DSN1COPY to replace or restore Db2 objects. If the underlying data is consistent, a rebuild of the index might resolve the problem.

Procedure

To collect diagnostic data if DSNI and DSNT messages indicate a corruption or inconsistency problem:

1. Collect Db2 archive log data sets from all Db2 members to the last point of consistency. Also, collect the corresponding consistent REORG or IMAGE COPY data sets.
2. If data is corrupted, collect the DSN1COPY of the table space or index.
3. If an index is inconsistent, run CHECK INDEX or CHECK LOB.
4. Collect the SYSLOG, which might contain messages that indicate the source of the problem.
5. Collect the LOGREC, which might contain messages that indicate the source of the problem.
6. Collect the JES job log for the system services address space (*ssnmMSTR*).
7. Collect any supervisor call (SVC) dump generated with abend code 04E and reason code 00C9xxxx for your Db2 environment.

Collecting data for IBM Db2 Analytics Accelerator for z/OS problems

When you encounter a IBM Db2 Analytics Accelerator for z/OS problem, report it directly to Db2 for z/OS Technical Support, who will work with you to diagnose and resolve the problem.

Before you begin

Before contacting Db2 for z/OS Technical Support team to report a problem:

- Be prepared to provide the following documentation about the problem:
 - A detailed description of the problem
 - An indication of whether the problem is associated with a new query or with an existing query that was changed in some way (for example, data volumes or query structures)
 - Analytics Accelerator DEFAULT trace output, if you are reporting a failure with IBM Db2 Analytics Accelerator for z/OS
 - Relevant documents, such as Db2 logs, Db2 dumps, JES logs
 - Additional trace output that could be relevant to the investigation
- Set up remote access to your accelerator environment to enable the support representative to access your environment remotely, if necessary for further diagnosis purposes. Remote access to the accelerator environment goes through Db2 for z/OS using the SSH (Secure Shell) command interface.

About this task

You do not need to perform problem source identification (PSI) to determine if the issue associated with software or hardware components in your accelerator environment. When you encounter a problem in your accelerator environment, report the issue directly to for Db2 for z/OS Technical Support.

Procedure

- In response to requests by the support team that is working on your problem, provide additional information or documentation.

Examples of information that might be requested include:

- Timing of query runs
- Details of the previous runs (prior to the problem)
- Any queries that are involved
- Any DDL involved
- Any storage dump that was encountered
- Screen shots that could help the investigation
- Status of the statistics

Using information about the problem, the Db2 for z/OS Technical Support team provides the preliminary problem diagnosis and determines the source of the reported problem. If needed, the support team will engage with other appropriate IBM Technical Support teams for further problem determination.

Due to the number of products that are involved in most accelerator environments, you might need to work with IBM Technical Support representatives from different groups in order to resolve the problem efficiently.

- Provide remote access to your accelerator environment to the IBM Technical Support representative, if needed.

Generally, the accelerator environment acts as an appliance and can be accessed only by IBM Technical Support representatives to whom the user has explicitly granted authority to remotely access and control the system. The IBM Technical Support team can generate the service password to access the system based on the serial number of the system. The user does not have the root password to access the accelerator environment.

Related tasks

[Enabling Db2 for IBM Db2 Analytics Accelerator for z/OS \(Db2 Performance\)](#)

[Monitoring the use of accelerators for Db2 for z/OS queries \(Db2 Performance\)](#)

Related reference

[Reference information for working with accelerators \(Db2 Performance\)](#)

Collecting data for application programming problems

You can collect diagnostic data to diagnose problems with the interface between Db2 and your application program.

Collecting data for an unexpected SQLCODE

If Db2 returns an unexpected SQLCODE, collect the SQLCA and DSN1SDMP dump data set that was triggered by the SQLCODE.

Before you begin

To diagnose unexpected SQLCODE problems, you must have the appropriate authorization to collect data from different z/OS components.

Procedure

Collect the following diagnostic data to diagnose unexpected SQLCODE problems:

- Service SQL information for the SQL statement, as described in [Collecting service SQL documentation \(Troubleshooting problems in Db2\)](#).
- A copy of the complete SQLCA that was returned to your application
- The DSN1SDMP dump data set that was triggered by the SQLCODE. DSN1SDMP *SDMPIN DD* control statements are commonly supplied by IBM Software Support

Related reference

[The included SQLCA \(Db2 SQL\)](#)

[DSN1SDMP \(Db2 Utilities\)](#)

Collecting data for Db2 coprocessor or Db2 precompiler problems

When you diagnose Db2 coprocessor or Db2 precompiler problems, collect generated output listings and diagnostic data.

About this task

Db2 coprocessor and Db2 precompiler problems might surface during the preparation or running of your application.

Tip: The Db2 coprocessor is the recommended method for processing SQL statements in application programs. Compared to the Db2 precompiler, the Db2 coprocessor has fewer restrictions on SQL programs, and more fully supports the latest SQL and programming language enhancements. See [Processing SQL statements by using the Db2 coprocessor \(Db2 Application programming and SQL\)](#).

Procedure

1. Collect the following diagnostic data that is generated when you prepare your application with the Db2 coprocessor or Db2 precompiler:
 - A simplified version of your application source code with minimal dependency on included members
 - Any source code that is included by your application from a host language COPY or include statement
 - A source listing output with the SOURCE option
 - Any diagnostic messages that are produced by the precompiler or coprocessor with your application
 - The cross-reference listing with the XREF option
2. Collect the following diagnostic data that is generated when you run your application:
 - Service SQL information for the SQL statement in error, as described in [Collecting service SQL documentation \(Troubleshooting problems in Db2\)](#).
 - The SQLCODE and any generated dump data sets
 - The DDCS trace

Related concepts

[Output from the Db2 coprocessor \(Db2 Application programming and SQL\)](#)

[Output from the Db2 precompiler \(Db2 Application programming and SQL\)](#)

Related tasks

[Processing SQL statements for program preparation \(Db2 Application programming and SQL\)](#)

Related reference

[Descriptions of SQL processing options \(Db2 Application programming and SQL\)](#)

Collecting data for XML problems

If you encounter problems with Db2 XML data, collect some initial diagnostic information about your environment.

Before you begin

To diagnose Db2 XML data problems, you must have the appropriate authorization to collect data from different z/OS components.

Procedure

To collect diagnostic data for Db2 XML data problems:

1. Collect a copy of the SQL statement, related SQL data manipulation statements and catalog statistics, and the EXPLAIN output to diagnose incorrect output from an SQL statement, as described in [Collecting service SQL documentation \(Troubleshooting problems in Db2\)](#).
2. Capture a supervisor call (SVC) dump for your Db2 application environment. Include all Db2 address spaces in the JOBLIST of your DUMP command.
3. Unload your XML data by using the DSNTIAUL program.

Related concepts

[SVC Dump \(MVS Diagnosis: Tools and Service Aids\)](#)

Related tasks

[Collecting data for incorrect output from an SQL statement \(Collecting data\)](#)

[Requesting Db2 SVC dumps \(Collecting data\)](#)

[Requesting data sharing environment Db2 SLIP traps \(Collecting data\)](#)

Related reference

[DSNTIAUL sample program \(Db2 Application programming and SQL\)](#)

Collecting data for operational problems

You can collect diagnostic data to diagnose problems that you encounter managing Db2 operations.

Collecting data for problems with DFSMS VSAM data sets

If you encounter problems with DFSMS VSAM data set manipulation, you can collect initial diagnostic information about your environment. DSNP messages usually indicate that the Db2 data space manager subcomponent encountered a problem.

Before you begin

To diagnose problems, you must have the appropriate authorization to collect data from different z/OS components. DFSMS VSAM error messages often accompany the Db2 errors and can help in problem determination.

Procedure

To collect diagnostic data if DSNP messages indicate a problem with DFSMS VSAM data set manipulation:

1. Collect the SYSLOG, which might contain messages that indicate the source of the problem.
2. Collect the LOGREC, which might contain messages that indicate the source of the problem.
3. Collect the JES job log for the system services address space (*ssnmMSTR*) and any allied agent address space.
4. Collect any supervisor call (SVC) dump that was generated with abend code 04E and reason code 00D7xxxx for your Db2 environment.
5. For a data set extend failure that is identified by message DSNP007I, collect the following data:

- a) Collect IDCAMS LISTCAT extent information for the target linear data set.
- b) Collect Db2 Statistics CLASS 3 trace with IFCID 0258 with information about the data set extend request.

Related concepts

[SVC Dump \(MVS Diagnosis: Tools and Service Aids\)](#)

Related tasks

[Requesting Db2 SVC dumps \(Collecting data\)](#)

[Requesting data sharing environment Db2 SLIP traps \(Collecting data\)](#)

[Preserving the z/OS console \(SYSLOG\) \(Collecting data\)](#)

[Preserving LOGREC data \(Collecting data\)](#)

[Preserving the JES job logs for key Db2 address spaces \(Collecting data\)](#)

Collecting data for problems with deadlocks and timeout failures

If you encounter locking problems, which are indicated by 00C90088 abends for deadlocks or 00C9008E failures for timeouts, collect diagnostic information about your environment. DSNT50xx messages usually indicate that the Db2 data manager subcomponent encountered a problem.

Before you begin

To diagnose problems, you must have the appropriate authorization to collect data from different z/OS components.

Procedure

To collect diagnostic data if DSNT50xx messages indicate locking failures:

1. Collect the JES job log for the system services address space (*ssnmMSTR*).
2. Collect SMF 100–101 records. Statistics Class 3 and Performance Class 6 records contain information about deadlocks and timeouts. IBM Support might request Performance Class 7 detailed lock information in some situations.

Related tasks

[Preserving the JES job logs for key Db2 address spaces \(Collecting data\)](#)

[Preserving LOGREC data \(Collecting data\)](#)

Collecting data for data set access problems

If you encounter data set open, allocation, reading, writing, or synchronization problems, collect diagnostic data from a time period when the problem occurred. DSNB messages usually indicate that the Db2 buffer manager subcomponent encountered a problem.

Before you begin

To diagnose problems, you must have the appropriate authorization to collect data from different z/OS components.

About this task

Diagnosing data set access problems requires IBM Support expertise. Often, DFSMS VSAM errors are involved.

Procedure

To collect diagnostic data if DSNB messages indicate a corruption or inconsistency problem:

1. Collect Db2 archive log data sets from all Db2 members to the last point of consistency. Also, collect the corresponding consistent REORG or IMAGE COPY data sets.

2. Collect the SYSLOG, which might contain messages that indicate the source of the problem.
3. Collect the LOGREC, which might contain messages that indicate the source of the problem.
4. Collect the JES job log for the system services address space (*ssnmMSTR*).
5. Collect any supervisor call (SVC) dump that was generated with abend code 04E and reason code 00C2xxxx for your Db2 environment.

Related concepts

[SVC Dump \(MVS Diagnosis: Tools and Service Aids\)](#)

Related tasks

[Requesting Db2 SVC dumps \(Collecting data\)](#)

[Requesting data sharing environment Db2 SLIP traps \(Collecting data\)](#)

[Preserving the z/OS console \(SYSLOG\) \(Collecting data\)](#)

[Preserving LOGREC data \(Collecting data\)](#)

[Preserving the JES job logs for key Db2 address spaces \(Collecting data\)](#)

Collecting data for storage abends

If you encounter storage errors, collect diagnostic data from a time period when the problem occurred. DSNS messages usually indicate the Db2 storage manager subcomponent encountered a problem.

Before you begin

To diagnose problems, you must have the appropriate authorization to collect data from different z/OS components.

About this task

Diagnosing storage management problems requires IBM Support expertise.

Procedure

To collect diagnostic data if DSNS messages indicate a storage management problem:

1. Collect any supervisor call (SVC) dump that was generated with abend code 878 and reason code 00E20003 for your Db2 environment. If a storage leak exists, multiple dumps might be needed.
2. Start a Statistics Class 1 or 6 trace to collect IFCID 0225 records. Collect SMF type 100 records from startup until the storage abend.

Related concepts

[SVC Dump \(MVS Diagnosis: Tools and Service Aids\)](#)

Related reference

[z/OS MVS System Code 878](#)

[z/OS SMF Records](#)

Collecting data for resource limit facility problems

If SQL queries are not properly governed by the resource limit facility (RLF), then collect RLF table data, and generate dumps to diagnose the problem.

About this task

Introductory concepts

[The resource limit facility \(Introduction to Db2 for z/OS\)](#)

Procedure

To collect data to diagnose resource limit facility problems:

1. Collect service SQL information for the SQL statement to investigate, as described in [Collecting service SQL documentation \(Troubleshooting problems in Db2\)](#)
2. Collect the complete contents of the active RLF tables.
For example, use the following SELECT statements, where xx are the two-character identifiers that you specified on the name of the active RLF tables:

```
SELECT * FROM DSNRLSTxx  
SELECT * FROM DSNRLMTxx
```

3. If a dynamic SQL statement is not being limited by RLF, then generate a Db2 supervisor call (SVC) dump when you expect the limit is exceeded.
4. If an SQL statement is limited improperly, generate a DSN1SDMP that is triggered by SQLCODE -905.

Related tasks

[Requesting Db2 SVC dumps \(Collecting data\)](#)

Related reference

[Resource limit facility tables \(Db2 Performance\)](#)

[DSN1SDMP \(Db2 Utilities\)](#)

Collecting data for problems with the EDM pool

If Db2 returns message DSNT500I with SQLCODE -904 and reason code 00C90089, collect data for diagnosing EDM pool full conditions. EDM pool storage shortages might occur if there are too many concurrent instances or too many result sets of a stored procedure.

Before you begin

To diagnose problems, you must have the appropriate authorization to collect data from different z/OS components.

Procedure

To collect data to diagnose problems with EDM pool full conditions:

1. Collect the JES job log for the system services address space (*ssnmMSTR*).
2. Collect the LOGREC, which might contain messages that indicate the source of the problem.
3. Collect the SYSLOG data set, which might contain messages that indicate the source of the problem.
4. Generate a supervisor call (SVC) dump on the next occurrence of an EDM pool full condition, and enable an internal EDM trace as part of the generated dump.

To do that, start a performance trace for IFCID 133 and IFCID 134 by issuing this command:

```
-START TRACE PERFM CLASS(30) IFCID(133,134)
```

Related concepts

[SVC Dump \(MVS Diagnosis: Tools and Service Aids\)](#)

Related tasks

[Preserving the JES job logs for key Db2 address spaces \(Collecting data\)](#)

[Preserving LOGREC data \(Collecting data\)](#)

[Preserving the z/OS console \(SYSLOG\) \(Collecting data\)](#)

Related reference

[-START TRACE command \(Db2\) \(Db2 Commands\)](#)

Collecting data for distributed data facility problems

You can collect diagnostic data to diagnose distributed data facility (DDF) problems.

Collecting data for a problem between a DRDA requester and server

Db2 trace data can be used to diagnose problems with the distributed data facility (DDF).

Before you begin

To diagnose DDF problems, you must have the appropriate authorization to collect data from different z/OS components.

Procedure

Issue the appropriate START TRACE command for your situation:

- Start the Db2 performance trace:

```
-START TRACE (PERFM) CLASS(16) IFCID(165,180,184)
```

- For SQL problems, add class 3 to the performance trace:

```
-START TRACE (PERFM) CLASS(3)
```

- For DRDA exceptions, start statistics trace with class 4:

```
-START TRACE (STAT) CLASS(4)
```

- For authorization issues, start an audit trace with class 1 and 7:

```
-START TRACE (AUDIT) CLASS(1,7)
```

Related reference

[-START TRACE command \(Db2\) \(Db2 Commands\)](#)

Collecting data for a DDF abend or DRDA exception

DDF abends and DRDA exceptions are indicated by DSNL messages. Information in messages DSNL032I and DSNL027I might indicate the reasons for the abend or exception. Collect generated dumps, trace records, and standard diagnostic information about Db2 and VTAM to diagnose the DDF abends and DRDA exceptions.

Before you begin

To diagnose DDF problems, you must have the appropriate authorization to collect data from different z/OS components.

Procedure

To collect diagnostic data for DDF abends and DRDA exceptions:

1. Collect any supervisor call (SVC) dumps that were generated at the time of the abend.
2. Collect the SYSLOG, which might contain messages that indicate the source of the problem. Specifically, collect DSNL messages.
3. Collect the JES lob log, which might contain messages that indicate the source of the problem. Specifically, collect DSNL messages.
4. Collect the LOGREC, which might contain messages that indicate the source of the problem.
5. Collect SMF Type 102 data with IFCID 0191 and 0192 from a Db2 statistics trace. Information in message DSNL032I indicates the sequence number of the trace record to capture.
6. To capture trace records for exceptional conditions while Db2 is running:

```
-START TRACE(STAT) CLASS(4)
```

Related concepts

[SVC Dump \(MVS Diagnosis: Tools and Service Aids\)](#)

Related tasks

[Requesting Db2 SVC dumps \(Collecting data\)](#)

[Preserving the JES job logs for key Db2 address spaces \(Collecting data\)](#)

[Preserving LOGREC data \(Collecting data\)](#)

[Preserving the z/OS console \(SYSLOG\) \(Collecting data\)](#)

Related reference

[-START TRACE command \(Db2\) \(Db2 Commands\)](#)

Collecting data for a DDF hang

DRDA requesters excessively waiting for Db2 might indicate that DDF is hanging. Collect diagnostic data that is similar to general Db2 hangs and information about your distributed environment.

Before you begin

To diagnose DDF problems, you must have the appropriate authorization to collect data from different z/OS components.

Procedure

To collect data to diagnose DDF hangs:

1. Display the suspended Db2 threads before any dumps are generated by running the DISPLAY THREAD command:

```
-DISPLAY THREAD(*) SERVICE(WAIT)
```

2. Capture a supervisor call (SVC) dump for your Db2 environment. Include the IRLM, VTAM, and TCP/IP address spaces in the JOBLIST of your DUMP command.
3. Collect the SYSLOG, which might contain messages that indicate the source of the problem.
4. Collect the LOGREC, which might contain messages that indicate the source of the problem.

Related concepts

[SVC Dump \(MVS Diagnosis: Tools and Service Aids\)](#)

Related tasks

[Collecting data for a Db2 hang \(Collecting data\)](#)

[Requesting Db2 SVC dumps \(Collecting data\)](#)

[Preserving the JES job logs for key Db2 address spaces \(Collecting data\)](#)

[Preserving LOGREC data \(Collecting data\)](#)

[Preserving the z/OS console \(SYSLOG\) \(Collecting data\)](#)

Collecting data for Db2 utility problems

Db2 utilities perform work in the *ssnmDBM1* and utility address spaces. The type of data that you collect to diagnose utility problems depends on the utility that you are running and the address space where it is running.

About this task

Db2 utilities can run as batch jobs or they can be run by any of the following Db2-supplied stored procedures:

- DSNUTILS
- DSNUTILU
- DSNUTILV

Db2 utilities use z/OS multitasking and perform work in both the database services address space (*ssnmDBM1*) and utility address spaces. In general, utility diagnostic information is needed from either the database services address space (*ssnmDBM1*) or the utility address space.

Related concepts

[WLM management of stored procedures \(Db2 Installation and Migration\)](#)

Related reference

[DSNUTILU stored procedure \(Db2 SQL\)](#)

[DSNUTILV stored procedure \(Db2 SQL\)](#)

Collecting data for Db2 utility address spaces

To diagnose Db2 utility problems, collect Db2 utility address space and z/OS system diagnostic data.

Before you begin

To diagnose problems, you must have the appropriate authorization to collect data from different z/OS components.

About this task

The following types of problems typically originate from the Db2 utility address space:

- Virtual storage availability issues that are identified by z/OS system code 878 with reason code 10
- Issues that are related to DFSORT SORT and MERGE functions of the utility
- Utility data set access issues
- Template code errors from the Db2 Utility parser

Procedure

Collect the following data to diagnose Db2 utility address space problems:

- The utility job log
- LOGREC data
- SVC dump data sets that were generated by the error
- The SYSLOG if a hardware-related issue is suspected
- DFSORT diagnostic data that is written to the SORTDIAG DD statement. To write to SORTDIAG , add the following JCL to the utility job:

```
//SORTDIAG DD DUMMY
```

What to do next

Depending on the problem, more information might be requested by IBM Support. This information might include:

- DDL for the table spaces, tables, and indexes
- SMF trace data
- Supervisor call (SVC) dump data sets that were generated by SLIP traps
- Db2 diagnose reports, which can be generated by running the following command:

```
DIAGNOSE DISPLAY MEPL (BEPL) report
```

Related concepts

[Recording LOGREC error records \(MVS Diagnosis: Tools and Service Aids\)](#)

[SVC Dump \(MVS Diagnosis: Tools and Service Aids\)](#)

[SYSLOG records \(MVS Diagnosis: Tools and Service Aids\)](#)

Related reference

[z/OS MVS System Code 878](#)

[z/OS MVS System Management Facilities \(SMF\)](#)

[DIAGNOSE \(Db2 Utilities\)](#)

Related information

[DFSORT Application Programming Guide](#)

Collecting data for *ssnmDBM1* utility problems

To diagnose Db2 utility problems, collect specific data from the database services address space (*ssnmDBM1*).

About this task

Problems with Db2 utilities in the database services address space (*ssnmDBM1*) can involve a broad range of components:

- z/OS
- DFSMS
- Db2 catalog
- Third-party software
- Hardware

Procedure

Collect the following data to diagnose problems with utilities in the database services address space (*ssnmDBM1*):

- A description of the problem situation
- The utility job log output
- LOGREC data
- Generated SVC dump data sets
- *ssnmDBM1* job log output
- *ssnmMSTR* job log output
- SYSLOG data
- Db2 catalog information from SYSIBM.SYSTABLEPART columns IPREFIX, AVGWLEN, and PAGESAVE

What to do next

Depending on the problem, more information might be requested by IBM Support. This information might include:

- Db2 diagnose reports
- Db2 internal trace reports from SMF or GTF
- CHECK INDEX, REPAIR DBD, DSN1COPY, and DSN1PRINT reports

Related concepts

[Recording LOGREC error records \(MVS Diagnosis: Tools and Service Aids\)](#)

[SVC Dump \(MVS Diagnosis: Tools and Service Aids\)](#)

[SYSLOG records \(MVS Diagnosis: Tools and Service Aids\)](#)

Related reference

[z/OS MVS System Management Facilities \(SMF\)](#)

[DIAGNOSE \(Db2 Utilities\)](#)

Collecting data for failures after a point-in-time recovery

If you have an inaccessible or incorrect data in a table space after a point-in-time recovery, collect diagnostic information about your table space.

About this task

If a table space becomes inaccessible or incorrect output results

1. After you run REORG to complete recovery to a point in time
2. Before materialization of pending definition changes
3. Check the integrity of your subsystem to diagnose the cause

Procedure

To collect data to diagnose table space failures after a point-in-time recovery:

1. Save the output from the failed REORG TABLESPACE job.
2. Run REPAIR DBD DIAGNOSE to ensure that the catalog is consistent with the directory.
3. If you ran REPORT RECOVERY before you ran the point-in-time recovery, save the REPORT RECOVERY output.
4. Examine the RECOVER output from the point-in-time recovery for error messages. Save that output. Correct the errors, and run the RECOVER job again.
5. If the RECOVER completed successfully when you reran it in the previous step, run REORG TABLESPACE again to complete the point-in-time recovery.
6. Examine the REORG TABLESPACE output from the previous step for error messages. Save that output. Correct the errors, and run the REORG TABLESPACE job again.
7. If the REORG TABLESPACE job completed successfully when you reran it in the previous step, run REPORT RECOVERY to determine whether the appropriate records were inserted into the SYSIBM.SYSCOPY table by REORG TABLESPACE.
8. Issue a `SELECT * FROM SYSIBM.SYSPENDINGDDL`, statement, and save the results.
9. Run DSN1PRNT to dump the first few data pages of the table space, and validate the contents of the data rows. Save the output.
10. Submit all saved output to IBM Software Support.

Related reference

[RECOVER \(Db2 Utilities\)](#)

[REORG TABLESPACE \(Db2 Utilities\)](#)

[REPORT \(Db2 Utilities\)](#)

Collecting data for IRLM problems

You can collect diagnostic data to diagnose internal resource lock manager (IRLM) problems. This information can include supervisor call (SVC) dumps and other system data sets that are needed to resolve Db2 locking problems.

Collecting data for IRLM child-lock delays

Issue the MODIFY command to collect diagnostic data when internal resource lock manager (IRLM) processes are delayed during an operation that involves child-lock propagation.

Before you begin

To diagnose IRLM problems, you must have the appropriate authorization to collect data from different z/OS components.

About this task

In a data sharing environment, when child-lock propagation exceeds a time threshold, modify settings so that a supervisor call (SVC) dump is created for the associated address spaces.

Procedure

To collect IRLM diagnostic trace information, use the following command in the z/OS console:

```
MODIFY irlmproc,DIAG,DELAY
```

Where *irlmproc* is the IRLM procedure name as identified in the IEFSSNxx member of the SYS1.PARMLIB data set.

Results

The next time that the delay occurs, IRLM initiates dumps of the IRLM and Db2 address spaces to the SYS1.DUMPxx data set.

Related tasks

[Preserving the z/OS console \(SYSLOG\) \(Collecting data\)](#)

Related reference

[MODIFY irlmproc,DIAG command \(z/OS IRLM\) \(Db2 Commands\)](#)

Related information

[IEFSSNxx \(subsystem definitions\) - keyword parameter form \(MVS Initialization and Tuning Reference\)](#)

Collecting data for IRLM P-lock delays

Issue the MODIFY command to collect diagnostic data when the internal resource lock manager (IRLM) processes are delayed during an operation that involves physical lock (P-lock) negotiation.

Before you begin

To diagnose IRLM problems, you must have the appropriate authorization to collect data from different z/OS components.

About this task

In a data sharing environment, when P-lock negotiation exceeds a time threshold, modify settings so that a supervisor call (SVC) dump is created for the associated address spaces. Message DSNT501I might be issued to the z/OS console (SYSLOG), and SQLCODE -904 with reason code 00C20255 might be returned to the application, indicating a P-lock cannot be obtained.

Procedure

To collect IRLM diagnostic trace information, use the following command in the z/OS console:

```
MODIFY irlmproc,DIAG,PLOCK
```

Where *irlmproc* is the IRLM procedure name as identified in the IEFSSNxx member of the SYS1.PARMLIB data set.

Results

The next time that the delay occurs, IRLM initiates dumps of the IRLM and Db2 address spaces to the SYS1.DUMPxx data set.

Related tasks

[Preserving the z/OS console \(SYSLOG\) \(Collecting data\)](#)

Related reference

[MODIFY irlmproc,DIAG command \(z/OS IRLM\) \(Db2 Commands\)](#)

Related information

[IEFSSNxx \(subsystem definitions\) - keyword parameter form \(MVS Initialization and Tuning Reference\)](#)

[00C20255 \(Db2 Codes\)](#)

[DSNT501I \(Db2 Messages\)](#)

Collecting data for IRLM hangs

Db2 uses the IRLM subsystem to control the locking of data. To diagnose IRLM waits and hangs, collect IRLM supervisor call (SVC) dumps, SYSLOG, and LOGREC.

Before you begin

To diagnose IRLM problems, you must have the appropriate authorization to collect data from different z/OS components.

About this task

The following messages might be displayed on the z/OS console (SYSLOG):

- DXR167E indicates that IRLM detected a hang. Diagnostic information must be collected before IRLM no longer detects delayed processes.
- IXL040E with reference to the IRLM address space in CONNECTOR NAME, JOBNAME, and ASID message tokens indicates that z/OS detected a possible hang with IRLM.

Procedure

To collect IRLM diagnostic information when a message indicates an IRLM hang:

1. Capture the IRLM SVC dump for your Db2 environment by setting up a SLIP trap.

- If you are not in a data sharing environment, set up and enable a SLIP trap to capture associated SVC dumps:

```
SLIP SET,MSGID=message-id,ACTION=SVCD,ID=trapid,  
JOBLIST=(irlmproc,ssnmMSTR,ssnmDBM1),  
SDATA=(XESDATA,COUPLE,PSA,LSQA,LPA,GRSQ,RGN,CSA,SQA,SUM,TRT,ALLNUC),END
```

Where:

- *message-id* is the message that indicates the hang, (for example DXR167E or IXL040E)
- *trapid* is the trap identifier
- *irlmproc*, *ssnmMSTR*, *ssnmDBM1* are the address spaces to be dumped
- If you are in a data sharing environment, set up and enable an SLIP trap to capture associated SVC dumps:

```
SLIP SET,MSGID=message-id,ACTION=SVCD,ID=trapid,  
JOBLIST=(XCF*,irlmproc,ssnmMSTR,ssnmDBM1),  
SDATA=(XESDATA,COUPLE,PSA,LSQA,LPA,GRSQ,RGN,CSA,SQA,SUM,TRT,ALLNUC),  
REMOTE=(JOBLIST,SDATA),END
```

Where:

- *message-id* is the message that indicates the hang, (for example DXR167E or IXL040E)
 - *trapid* is the trap identifier
 - XCF* is the value that you specify for the Cross System Coupling Facility name using wildcards
 - *irlmproc*, *ssnmMSTR*, *ssnmDBM1* are the address spaces to be dumped
2. Collect any abend dumps with JOBNAME *ssnmIRLM* for ABEND=S026, REASON=08118001.

3. Collect the SYSLOG, which might contain messages that indicate the source of the problem.
4. Collect the LOGREC, which might contain messages that indicate the source of the problem.

Related tasks

[Preserving LOGREC data \(Collecting data\)](#)

[Preserving the z/OS console \(SYSLOG\) \(Collecting data\)](#)

Related reference

[z/OS SLIP command \(MVS System Commands\)](#)

[z/OS SLIP command SET parameters \(MVS System Commands\)](#)

[Message IXL040E \(MVS System Messages\)](#)

Related information

[DXR167E \(IRLM messages and codes\)](#)

Collecting data for a Db2 hang

If Db2 seems to hang while processing, you can collect initial diagnostic information about your environment.

Before you begin

To diagnose Db2 hang problems, you must have the appropriate authorization to collect data from different z/OS components.

Procedure

To collect diagnostic data for Db2 hang problems:

1. Issue a DISPLAY THREAD command to display threads in a long-term suspend state. Refer to [“Db2 commands for troubleshooting” on page 280](#) for information about the options of the DISPLAY THREAD command to diagnose thread waits.
2. Capture a supervisor call (SVC) dump for your Db2 environment. Include all Db2 address spaces in the JOBLIST of your DUMP command. Also, include the address space of any jobs in a long-term suspend state that is waiting for Db2. In a data sharing environment, create a SLIP trap to capture address spaces from remote Db2 members.
3. Collect the SYSLOG, which might contain messages that indicate the source of the problem. Ensure this log contains information about the oldest waiting job.
4. Collect the LOGREC, which might contain messages that indicate the source of the problem. Ensure this log contains information about the oldest waiting job.
5. Collect the JES job logs for key Db2 address spaces.

Related concepts

[SVC Dump \(MVS Diagnosis: Tools and Service Aids\)](#)

Related tasks

[Requesting Db2 SVC dumps \(Collecting data\)](#)

[Requesting data sharing environment Db2 SLIP traps \(Collecting data\)](#)

[Preserving the z/OS console \(SYSLOG\) \(Collecting data\)](#)

[Preserving LOGREC data \(Collecting data\)](#)

[Preserving the JES job logs for key Db2 address spaces \(Collecting data\)](#)

Collecting data for SQL data definition language statement errors

If an error occurs processing an SQL data definition language statement such as CREATE or ALTER, collect the job output and all data definition language statements that are needed to re-create the objects that are involved in your Db2 environment.

Before you begin

To diagnose data definition language statement errors, you must have the appropriate authorization to collect data from different z/OS components.

Procedure

Collect the following data to diagnose data definition language statement errors:

- A copy of the job output with the failing SQL statement, and service SQL information for the statement, as described in [Collecting service SQL documentation \(Troubleshooting problems in Db2\)](#).
- A copy of the complete SQLCA returned to your application
- Data definition language statements to re-create the query environment.
- A DSN1COPY of the catalog data if requested by IBM Support
- Output of the REPAIR DBD TEST or REPAIR DBD DIAGNOSE utility if requested by IBM Support

Related tasks

[Repairing DBDs \(Db2 Utilities\)](#)

Related reference

[The included SQLCA \(Db2 SQL\)](#)

Collecting data for RDS problems

To analyze a Db2 RDS problem, IBM Support might request several types of data.

PSPI

In a situation with both a "good" and "bad" scenario, include data from both.

When the data is requested as "hardcopy", print the data on paper. When the data is requested as "softcopy", put the data on a cartridge or tape. For the DDL and DML the preferred format is fixed block, LRECL 80 or 132.

Class 1 data - always needed

A complete copy of the problem query

It is important that this information is an exact, unmodified copy of the query that is causing the problem. If there are host variables in the query, they should be left in the query. Include a hardcopy and a softcopy.

Values that are used for any host variables in the query

Include a hardcopy.

Service SQL documentation for the query

The service SQL documentation includes DDL statements for relevant objects, EXPLAIN records for the SQL statement, and relevant catalog statistics in a standard format for analysis by IBM software support. For instructions, see [Collecting service SQL documentation \(Troubleshooting problems in Db2\)](#).

Buffer pool sizes (specified in IFCID 0202)

Include a hardcopy.

CPU model

Include a hardcopy.

A count of the number of rows that are returned from the query

Include a hardcopy.

DBRMs for static applications

Include a softcopy.

Class 2 data - might be needed

Performance summary report. The preferred report is the Db2 PM accounting summary report. However, if Db2 PM is not available, a summary report from a different performance monitor can be used. Include a hardcopy.

Class 3 data - might be needed

A complete, unformatted storage memory dump at a specified module or at the failure point. An IBM support person can provide more details about how to generate a storage memory dump.

Class 4 data - might be needed

Performance trace of the problem query. The following start command is the most common type of trace. An IBM support person might ask that different trace classes be turned on.

```
-START TRACE(PERFM) CLASS(1,2,3,6,8,9,10,32) IFCID(135,136,186)
  RMID(*) DEST(GTF) TDATA(COR,CPU) AUTHID(AUTHID)
  PLANID(PLANID)
```

DEST(SMF) can be used. The trace must be unformatted. Include a softcopy.

Class 5 data - might be needed

A DSN1COPY of all table spaces and index spaces that are involved in the query, including DDL for all tables that are defined in the table spaces and all DBID, PSID, and OBID information. Include a softcopy.

PSPI

Collecting data for a cached dynamic statement that blocks another statement with reason code 00E70081

A data definition (DDL) statement can fail because a statement in the dynamic statement cache is also using an object that the data definition statement is modifying. You can collect trace data to determine the cached statement that is causing the problem.

About this task

If you receive SQLCODE -904 with reason 00E70081 and resource type 00001202 from DSNXIDMH during the execution of a data DDL statement, the DDL statement references an object that is also referenced by a prepared dynamic SQL statement that is currently stored in the dynamic statement cache and in use by an application.

The resource name is the statement identifier for the prepared statement in the dynamic statement cache. If the application is running in a data sharing environment, the resource name has the form *member-name.statement-name*, where member-name identifies data sharing member.

The DDL statement causes Db2 to mark the blocking statement invalid, so the blocking statement is not returned by EXPLAIN STMTCACHE ALL, which only returns valid cached statements. However, even though it is marked invalid, the cached statement can continue to block the DDL statement because metadata for an application remains active until the transaction completes.

Procedure

You can use one of the following methods to obtain information about the cached statement that is using the object that the DDL statement is modifying:

- Use an IFI READS for IFCID 316 to obtain information about the cached statement.

This option is best because it provides real-time diagnostic information, and cached dynamic statement usage can change after a short time. The IFI READS retrieves all the cached statements and reports them. Using a monitoring application or tool is the better option, especially if the statements that are causing SQLCODE -904 are in the cache for a short time.

For more information about writing a monitoring application, see [Monitoring the dynamic statement cache with READS calls \(Db2 Performance\)](#).

Also, you can use monitor programs such as IBM OMEGAMON for Db2 Performance Expert on z/OS to dump the cache contents. See [Viewing the SQL statements in the dynamic SQL cache \(Tivoli OMEGAMON XE for Db2 Performance Expert on z/OS\)](#).

For a complete description of the fields in an IFCID 316 record, see the DSNWMSGs file.

- Generate dumps by using the DSN1SDMP utility and a SLIP command on other data sharing members, by completing the following steps:

- a) Under the guidance of IBM Support, you can run the DSN1SDMP utility to generate a dump when SQLCODE -904 with reason code 00E70081 occurs.

For example, you can run the following DSN1SDMP job on the member where you execute the DDL statement:

```
//SDMPIN DD * START TRACE=P CLASS(32) IFCID(340) DEST(SMF) TDATA(TRA)
AFTER(1)
FOR(1)
ACTION(ABENDTER(00E60194))
SELECT P4,00
* COMPARE FOR 340 IFCID
DR,04,X'0154'
* POSITION TO SECOND SECTION (1ST DATA SECTION)
P4,08
* COMPARE QW0340_MODNAME to DSNXIDMH
DR,6,C'DSNXIDMH'
* COMPARE SQLCODE FOR -904
DR,16,X'FFFFFF78'
/*
```

- b) Because the resource might be used by statements on other members, also set a SLIP trap on the dump to be generated by DSN1SDMP with the REMOTE option to take dumps on all data sharing members.

For example, you can issue the following SLIP SET command on the same member where you set the DSN1SDMP. Change *ssnm* to the Db2 subsystem name.

```
SLIP SET, ID=IDMH, C=04E, DATA=(15R, EQ, 00E60194), ACTION=SVCD,
JOBLIST=(XCFAS, ssnmIRLM, ssnmMSTR, ssnmDBM1),
SDATA=(XESDATA, COUPLE, PSA, LSQA, LPA, GRSQ, SWA, RGN, CSA, SQA, SUM, TRT, ALLNUC),
ML=1, REMOTE=(JOBLIST, SDATA, DSPNAME), END
```

Because Db2 only stores information such as the statement text and number of current users of the statement, you must identify the application or thread that is using the statement separately.

Related tasks

[Monitoring the dynamic statement cache with READS calls \(Db2 Performance\)](#)

Related reference

[DSN1SDMP \(Db2 Utilities\)](#)

[Viewing the SQL statements in the dynamic SQL cache \(Tivoli OMEGAMON XE for Db2 Performance Expert on z/OS\)](#)

[z/OS SLIP command \(MVS System Commands\)](#)

Related information

[00E70081 \(Db2 Codes\)](#)

-904 (Db2 Codes)

Chapter 9. Preparing to transfer data sets with program objects to IBM

When IBM Support asks you to provide Db2 data sets that contain program objects, you can either pack a copy of the contents and send them, or use the AMAPDUPL utility to prepare and send the contents.

About this task

IBM Support might ask you to provide them with Db2 partitioned data sets that contain program objects, such as the *prefix.SDSNLOAD* data set. You cannot use a pack utility directly on those data sets.

You can use either of the following methods to pack and transfer a partitioned data set that contains program objects:

- Use the procedure described below to pack a copy of the data set, and use FTP to transfer it IBM Support.
- Use the AMAPDUPL utility to prepare and send the contents of the data set to IBM Support. When you use AMAPDUPL, you do not need to pack the contents of the data set. See the [AMAPDUPL](#) documentation for usage information and examples.

Procedure

To pack the contents of a partitioned data set that contains program objects and transfer the contents to IBM Support, follow these steps.

1. Use the IEBCOPY or DFDSS utility to copy the contents of the data set to a sequential data set.
2. Use a utility such as the [AMATERSE](#) utility to pack the data set copy that you created in step “1” on [page 503](#).
3. FTP the packed data set to IBM Support.

Example: Pack the contents of the DSN1310.SDSNLOAD data set

Use JCL similar to this example to copy the contents of the DSN1310.SDSNLOAD data set to sequential data set DSN1310.SDSNLOAD.IEBCOPY.

```
//JOBSTEP EXEC PGM=IEBCOPY
//SYSOUT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//IN1 DD DSN=DSN1310.SDSNLOAD,DISP=SHR
//OUT1 DD DSN=DSN1310.SDSNLOAD.IEBCOPY,
// DISP=(NEW,CATLG,DELETE),
// SPACE=(CYL,(900,300),RLSE),
// RECFM=FB,LRECL=80,DSORG=PS
//SYSUT3 DD UNIT=VIO,SPACE=(TRK,(10,10))
//SYSUT4 DD UNIT=VIO,SPACE=(TRK,(10,10))
//SYSIN DD *
COPY INDD=IN1,OUTDD=OUT1
/*
```

Suppose that you are sending data for case number TS123456789 to IBM Support. Use JCL similar to this example to pack the contents of the DSN1310.SDSNLOAD.IEBCOPY data set before you transfer it to IBM Support.

```
//STEPNAME EXEC PGM=AMATERSE,PARM='PACK'
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DISP=SHR,DSN=DSN1310.SDSNLOAD.IEBCOPY
//SYSUT2 DD DSN=TS123456.789.SDSNLOAD.IEBCOPY.TRS,
// SPACE=(TRK,(1,1000),RLSE),
// UNIT=3390,DISP=(NEW,CATLG,DELETE)
```

Related information

[AMATERSE: Pack and unpack a data set](#)

[AMAPDUPL: Problem Documentation Upload Utility](#)

Chapter 10. Requesting product support for problems in Db2 for z/OS

If you are unable to successfully troubleshoot a problem in Db2 for z/OS and you cannot find an existing fix, you might need to contact IBM Support for assistance.

Before you begin

Before contacting IBM Support, check whether a fix or solution already exists for your problem. See Chapter 2, “Searching for known problems and solutions for Db2 for z/OS,” on page 3. For a list of fixes available for Db2 for z/OS, see the [IBM Support page for Db2 for z/OS](#).

Also, your company must have an active IBM Support subscription and support contract, and you must be authorized to submit problems to IBM Support. For information about the types of available support, see [Support Portfolio](#).

Procedure

To contact IBM Support about Db2 problems:

1. Define the problem, gather background information, and determine the severity of the problem.
2. Collect appropriate diagnostic data for the type of problem that you are reporting. You can collect the diagnostic data manually or automatically, depending on the data.

Diagnostic data can help reduce the time for resolving your Db2 problem. For example, having access to relevant messages, error codes, log data, trace output, or dump output can speed the resolution process. For more information, see [Collecting diagnostic data \(Collecting data\)](#).

3. Contact IBM Support by opening a case at <https://www.ibm.com/mysupport/s/>.

You can expect to provide the following information:

- Your customer number
 - Current service level (PTF list and list of APAR fixes applied)
 - Processor number (serial—model)
 - Keyword string that is used to search the IBM Support database
4. Send the diagnostic data to IBM Support, as described in [“Sending diagnostic data to IBM”](#) on page 505.
 5. If directed by IBM Support, download data from IBM Support as described in [“Receiving support information from IBM”](#) on page 506.

What to do next

If IBM Support determines that an *authorized program analysis report* (APAR) is necessary to resolve your problem, you might need to apply the PTF for the APAR. For more information, see [“Getting fixes”](#) on page 51.

Sending diagnostic data to IBM

For IBM Support to diagnose or identify a problem, you might need to provide diagnostic data and other information from your system.

Before you begin

If you have not already done so, you must create an IBM Support *File Transfer ID* with your IBM ID. For details, see [Support File Transfer Details](#). Only secure transfer protocols, such as SFTP, HTTPS, MTFTP are supported.

Procedure

1. Collect diagnostic data for the type of problem that you are reporting or as requested by IBM Support, as described in [Collecting diagnostic data \(Collecting data\)](#).
2. If the diagnostic data includes SVC dumps captured on IBM z15 or later processors, you can redact buffer pool data in the dump. For more information, see [Redacting buffer pool data in SVC dumps for data privacy \(Troubleshooting problems in Db2\)](#).
3. Compress the data and name the files according to the naming convention, as described in [ECuRep: Prepare data](#).
4. Use one of the following secure methods to submit the data to IBM Support:
 - If you use IBM Blue[®] Diamond, see [Blue Diamond Help: Secure FTP](#).
 - Otherwise, use a secure method to submit the data to the Enhanced Customer Data Repository (ECuRep), as described in [ECuRep: Send data](#).

For JCL examples, see [JCL examples for PDUU](#). When you modify the JCL examples for your environment, apply the following changes:

- For USERID, specify the Secure File Transfer ID that you created in the [transfer ID generation app](#).
- For PASSWORD, specify the password that was generated with the Secure File transfer ID.
- Specify the server that is closest to your physical location:

| Region | TARGET_SYS= |
|--------------|--------------------------|
| Americas | testcase.boulder.ibm.com |
| Asia Pacific | ftp.ap.ecurep.ibm.com |
| Europe | ftps.ecurep.ibm.com |

- Specify DIRECTORY=/toibm/im.
- Do not specify a CIPHER_KEY value. Separate encryption of the files is unnecessary because secure transfer methods are always used.

Tip: Server-side values are case sensitive.

Receiving support information from IBM

IBM Support might ask you to download diagnostic tools or other files to help solve your problem.

Before you begin

If you have not already done so, you must create an IBM Support *File Transfer ID* with your IBM ID. For details, see [Support File Transfer Details](#). Only secure transfer protocols, such as SFTP, HTTPS, MTFTP are supported.

About this task

IBM Support might ask you to download diagnostic tools or other files.

Procedure

To download files from IBM Support:

1. Connect to the server by using the information that IBM Support provides.
2. Change to the generated directory name that IBM Support provides.

```
cd fromibm/generated-directory-name
```

3. Enable binary mode for your session.

```
binary
```

4. Use the **get** command to download the file that IBM Support specified.

```
get filename.extension
```

5. End the session.

```
quit
```

Example

The following sample job downloads sample files from IBM:

```
//FTPGETIT EXEC PGM=FTP,PARM='testcase.boulder.ibm.com (EXIT'  
//SYSFTPD DD DISP=SHR,DSN=tcpip.ftps.data.testcase For Secure FTP  
//OUTPUT DD SYSOUT=*  
//AI12345 DD DISP=(,CATLG),UNIT=SYSDA,DSN=&SYSUID..FIX.AI12345,  
// DSORG=PS,RECFM=FB,BLKSIZE=3120,LRECL=80,SPACE=(CYL,(1,1))  
//AI56789 DD DISP=(,CATLG),UNIT=SYSDA,DSN=&SYSUID..FIX.AI56789,  
// DSORG=PS,RECFM=FB,BLKSIZE=3120,LRECL=80,SPACE=(CYL,(1,1))  
//*  
//* NOTE: Clear sequence numbers in columns 73-80.  
//* FTP Server-side values are Case-Sensitive.  
//*  
//INPUT DD *  
your-user-ID  
your-password  
cd /fromibm/generated-directory-name/  
bin  
get AI12345.HDBAA10 //DD:AI12345  
get AI56789.HDBAA10 //DD:AI56789  
quit  
/*
```


Information resources for Db2 for z/OS and related products

You can find the online product documentation for Db2 13 for z/OS and related products in IBM Documentation.

For all online product documentation for Db2 13 for z/OS, see [IBM Documentation \(https://www.ibm.com/docs/en/db2-for-zos/13\)](https://www.ibm.com/docs/en/db2-for-zos/13).

For other PDF manuals, see PDF format manuals for Db2 13 for z/OS (https://www.ibm.com/docs/en/SSEPEK_13.0.0/home/src/tpc/db2z_pdfmanuals.html).

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

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, MD-NC119
Armonk, NY 10504-1785 US*

For license inquiries regarding double-byte (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*

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 jurisdictions 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.

Automated (Machine) Translation: The original version of IBM technical content, including product documentation, is the English version of this content. If you have any questions or concerns about the translated content, please refer to the English version. IBM disclaims any liability for any damages or losses of any kind caused by the use of automatically (machine) translated content. To provide feedback on the translated content, see the note at the top of the equivalent page in the translated online HTML version of the content.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

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:

*IBM Director of Licensing IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785 US*

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 document 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.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains 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.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as shown below:

© (your company name) (year).

Portions of this code are derived from IBM Corp. Sample Programs.

© Copyright IBM Corp. (enter the year or years).

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

This information is intended to help you diagnose problems in a Db2 13 for z/OS environment. This information also documents General-use Programming Interface and Associated Guidance Information and Product-sensitive Programming Interface and Associated Guidance Information provided by Db2 13 for z/OS.

General-use Programming Interface and Associated Guidance Information

General-use Programming Interfaces allow the customer to write programs that obtain the services of Db2 13 for z/OS.

General-use Programming Interface and Associated Guidance Information is identified where it occurs by the following markings:

 General-use Programming Interface and Associated Guidance Information... 

Product-sensitive Programming Interface and Associated Guidance Information

Product-sensitive Programming Interfaces allow the customer installation to perform tasks such as diagnosing, modifying, monitoring, repairing, tailoring, or tuning of this IBM software product. Use of such interfaces creates dependencies on the detailed design or implementation of the IBM software product. Product-sensitive Programming Interfaces should be used only for these specialized purposes. Because of their dependencies on detailed design and implementation, it is to be expected that programs written to such interfaces may need to be changed in order to run with new product releases or versions, or as a result of service.

Product-sensitive Programming Interface and Associated Guidance Information is identified where it occurs by the following markings:

Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com)® are trademarks or registered marks 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 on the web at "Copyright and trademark information" at: <http://www.ibm.com/legal/copytrade.shtml>.

Linux® is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions:

Applicability: These terms and conditions are in addition to any terms of use for the IBM website.

Personal use: You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use: You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights: Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Privacy policy considerations

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Statement at <http://www.ibm.com/privacy>.

Glossary

The glossary is available in IBM Documentation

For definitions of Db2 for z/OS terms, see [Db2 glossary \(Db2 Glossary\)](#).

Index

Numerics

00C9009F abend [175](#)
00C90101 abend [35](#), [346](#)
00C90102 abend [35](#), [346](#)
00C90105 abend [35](#)
00C9010x abend [335](#), [343](#)
00C9011x abend [344](#), [346](#)
00C902xx abend [35](#), [346](#)

A

abend

00C90101 [35](#), [346](#)
00C90102 [35](#), [338](#)
00C90105 [35](#)
00C9010x [335](#), [343](#)
00C9011x [344](#), [346](#)
00C902xx [35](#), [335](#), [343](#), [346](#)
AEY9 [62](#)
ASP7 [62](#)
backward log recovery [98](#)
CICS
 abnormal termination [62](#)
 scenario [66](#)
 waits [62](#)
current status rebuild [85](#)
DXR122E [53](#)
forward log recovery [93](#)
IMS
 procedure [58](#)
 scenario [60](#), [61](#)
 U3047 [60](#), [61](#)
 U3051 [60](#), [61](#)
IRLM
 scenario [53](#)
log
 damage [80](#)
 lost information [103](#)
log initialization phase [83](#)
page problems [101](#)
restarting [83](#)
SQLCODE -923 [67](#)
VVDS (VSAM volume data set)
 destroyed [120](#)
 out of space [120](#)

ABEND

subcommand of DSN [283](#)

ABENDx keyword [14](#)

access method services

ALTER command [121](#)
commands
 DEFINE [101](#)
 IMPORT [101](#)
 REPRO [78](#)

damaged bootstrap data set (BSDS)
 deleting [76](#)

access method services (*continued*)

damaged bootstrap data set (BSDS) (*continued*)
 renaming [76](#)
damaged BSDS (bootstrap data set)
 deleting [76](#)
 renaming [76](#)
table spaces
 re-creating [101](#)

access path performance problems

collecting diagnostic data [478](#)
Db2 performance [478](#), [480](#), [481](#), [483](#), [487–491](#)

accessibility

keyboard [x](#)
shortcut keys [x](#)

ACFTAP option

buffer trace formatting [311](#)
VTAM IO trace [311](#)

active log data sets

stopping
 effects [72](#)

active logs

gaps
 creating [105](#)
 out-of-space conditions [69](#)
 recovering [69](#)
 troubleshooting [69](#)

address space

home [196](#), [281](#)
primary [196](#), [281](#)
secondary [196](#), [281](#)

ALTER command

access method services [121](#)

analyze dumps [181](#)

analyze traces [238](#)

analyzing inconsistencies using SVC dump [338](#)

application changes

backing out
 with quiesce point [56](#)

application errors

backing out
 without a quiesce point [57](#)

application program

monitor trace [239](#)

application programs

errors [56](#)
recovery procedures
 CICS [62](#)
 IMS [60](#), [61](#)
running
 error recovery [56](#)

archive log data sets

recovering [73](#)

ASCB (address space control block, z/OS)

in SVC dumps [197](#)

ASCE (address space control element)

captured in SVC dump [198](#)

ASCE (address space control element) (*continued*)
finding in dump [210](#)
ASCESCOM field [210](#)
ASID (address space identifier)
in MEPL [211](#)
in SYS1.LOGREC [226](#)
authority
problems [276](#)

B

backward log recovery phase
failures
recovering [98](#)
batch utility MEPL [213](#)
BM (buffer manager)
finding in a dump [343](#)
BMEPL
finding in dump [213](#)
illustration [213](#)
BSDS (bootstrap data set)
dual recovery [78](#)
failure symptoms [80](#)
recovery procedures [76](#)
recovery scenarios [101](#)
restoring
from archive logs [78](#)
single recovery [78](#)
buffer trace formatting
ACFTAP option [311](#)

C

CAB (call attachment facility control block)
traces [253](#)
CAF (call attachment facility)
finding trace table [253](#)
interpreting trace messages [253](#)
producing trace messages [252](#)
trace [252](#)
call attachment facility
trace messages [452](#)
catalog tables, DB2
consistency between [276](#)
diagnosis [276](#)
catalogs
Db2
recovery procedures [119](#)
CCSID problems
DB2 unexpected results [481](#)
change number of sessions (CNOS) [130](#)
CHECK option (DSN1COPY) [175](#), [344](#)
CICS
operating
terminates AEY9 [67](#)
recovery procedures
application failures [62](#)
attachment facility failures [66](#)
CICS not operational [62](#)
DB2 connection failures [63](#)
indoubt units of recovery [64](#)
CNOS (change number of sessions)
failures [130](#)

cold start
bypassing the damaged log [81](#)
special situations [103](#)
collecting data
service SQL [475](#)
collecting diagnostic data
accelerator problems [483](#)
address spaces, key [473](#)
data definition language errors [498](#)
Db2 attach problems
programming interface [484](#)
Db2 data manager [487](#)
Db2 data space manager [486](#)
Db2 dump data sets [473](#)
Db2 hangs [497](#)
Db2 logs [473](#)
Db2 packing data sets [503](#)
Db2 performance
access path performance problems [478](#), [480](#), [481](#),
[483](#), [487–491](#)
general performance problems [477](#)
Db2 problems
DB2 utilities [491](#)
IRLM [494](#)
performance [479](#), [486](#), [489](#)
Db2 unexpected results
CCSID problems [481](#)
Db2 utilities
database services address space (*ssnmDBM1*) [493](#)
failure after point-in-time recovery [494](#)
utility address space [492](#)
Db2 XML [486](#)
dump data sets, automatic updates [469](#)
dump data sets, DB2 [473](#)
factors to consider
component-specific problems [471](#)
performance, slow [471](#)
wait situation [471](#)
incorrect SQL results [479](#)
IRLM problems
child-lock delay problems [494](#)
hang problems [496](#)
P-lock delay problems [495](#)
JES job logs [473](#)
LOGREC data [472](#)
logs, DB2 [473](#)
Managing Db2 operations
resource limit facility problems [488](#)
master trace, increasing storage capacity [468](#)
overview [467](#)
preserving [471](#)
setting up z/OS [467](#)
SLIP traps for data sharing environments [471](#)
SLIP traps, requesting [474](#)
SQL abend [480](#)
SQLCODE
unexpected SQLCODE [484](#)
SVC dumps, increasing storage capacity [469](#)
SVC dumps, requesting [474](#)
SVC dumps, setting up customized [470](#)
SYS1.LOGREC data [472](#)
SYSLOG [472](#)
z/OS console (SYSLOG) [472](#)
z/OS console (SYSLOG), preserving [472](#)

- collecting diagnostic data (*continued*)
 - z/OS system trace, maximizing [468](#)
- communications failure
 - scenarios [163](#)
- component identifier keyword [10](#)
- conditional restart
 - control record
 - backward log recovery failures [99](#)
 - current status rebuild failures [91](#)
 - forward log recovery failures [97](#)
 - log initialization failures [91](#)
 - excessive loss of active log data [104](#)
 - total loss of log [103](#)
- CONN-ID [335](#)
- connections
 - IDs
 - identifying a unit of recovery [56](#)
- consistency groups [157](#)
- control status events
 - IMS attachment facility [257](#)
 - trace
 - IMS table contents [257](#)
- conversion error [276](#)
- CORR-ID [335](#)
- correlation IDs
 - CICS [64](#)
 - duplicates [64](#)
- CSECT
 - keyword [35](#)
 - name, finding in dump [212](#)
 - SVC dump [14](#), [27](#)
- CT (cursor table)
 - content [215](#)
 - finding in dump [215](#), [216](#)
- CTERQUAL [202](#), [226](#)
- CTRACE
 - description [268](#)
 - example [269](#)
- CTRDSP field [218](#)
- current status rebuild
 - failure recovery [83](#)

D

- damaged data
 - data page [339](#)
 - index page [342](#)
 - resolving data and index inconsistencies [343](#)
- DASD
 - problems [276](#)
- data
 - inconsistencies
 - resolving [107](#)
 - limiting access to [178](#)
 - restoring [133](#)
 - unavailable [175](#)
- data inconsistency problems
 - causes [174](#)
 - description [174](#)
 - resolving with REBUILD INDEX [343](#)
 - resolving with RECOVER [343](#)
 - symptoms
 - 00C9011x abend [344](#)
 - symptoms and actions [174](#)
- data inconsistency problems (*continued*)
 - types [174](#)
- data mirroring
 - recovery [156](#), [158](#)
- data page
 - analyzing [339](#)
 - illustration [339](#)
- data page header (PGHEAD) [339](#)
- data sets
 - adding [121](#), [124](#)
 - extending [121](#)
- data sharing
 - environment [285](#)
 - problem diagnosis
 - hangs [285](#)
 - inconsistent data [287](#)
 - IRLM delays [286](#)
 - timeouts, deadlocks [286](#)
- data sharing environments, SLIP traps for [471](#), [474](#)
- database name [343](#)
- database services address space (*ssnmDBM1*)
 - Db2 utilities [493](#)
- databases
 - access threads
 - failures [128](#)
 - security failure [131](#)
 - recovering
 - failure scenarios [111](#)
- Db2 catalog
 - recovery procedure [119](#)
- Db2 commands [280](#)
- DB2 dump data sets
 - preserving [473](#)
- Db2 logs
 - preserving [473](#)
- Db2 performance
 - Db2 problems [479](#), [486](#), [489](#)
- Db2 programming interface
 - Db2 attach problems [484](#)
- DB2 subsystem
 - restarting
 - log truncation [86](#)
 - resolving inconsistencies [92](#)
 - starting [94](#)
 - termination scenario [67](#)
- Db2-managed data sets
 - enlarging [123](#)
- DBD (database descriptor)
 - analyzing [349](#)
 - hierarchic structure [349](#)
 - inconsistencies, resolving [344](#)
 - locating in dump [346](#)
 - OBDDMAP [349](#)
 - obtaining dumps of [345](#)
- DDF (distributed data facility)
 - failures
 - recovering [126](#)
- DDF error messages
 - analyzing [294](#)
- DEFINE command
 - access method services
 - re-creating table spaces [101](#)
- DELETE command
 - access method services [101](#)

DFSMShsm (Data Facility Hierarchical Storage Manager)
 FRBACKUP PREPARE command [161](#)

diagnosis with utilities [278](#)

diagnostic area in SVC dump [335](#)

diagnostic data
 address spaces, key [473](#)
 collecting [467](#)
 data definition language errors [498](#)
 Db2 attach problems
 programming interface [484](#)
 Db2 dump data sets [473](#)
 Db2 hangs [497](#)
 Db2 logs [473](#)
 Db2 operations
 resource limit facility problems [488](#)
 Db2 packing data sets [503](#)
 Db2 performance
 access path performance problems [478](#), [480](#), [481](#),
[483](#), [487–491](#)
 general performance problems [477](#)
 Db2 problems
 DB2 utilities [491](#)
 IRLM [494](#)
 performance [479](#), [486](#), [489](#)
 DB2 problems
 accelerator [483](#)
 Db2 unexpected results
 CCSID problems [481](#)
 Db2 utilities
 database services address space (ssnmDBM1) [493](#)
 point-in-time recovery [494](#)
 utility address space [492](#)
 Db2 XML [486](#)
 deadlock failures [487](#)
 DFSMS VSAM [486](#)
 dump data sets, automatic updates [469](#)
 dump data sets, DB2 [473](#)
 factors to consider
 component-specific problems [471](#)
 performance, slow [471](#)
 wait situation [471](#)
 incorrect SQL results [479](#)
 IRLM problems
 child-lock delay problems [494](#)
 hang problems [496](#)
 P-lock delay problems [495](#)
 JES job logs [473](#)
 LOGREC data [472](#)
 logs, DB2 [473](#)
 master trace, increasing storage capacity [468](#)
 preserving [471](#)
 setting up z/OS [467](#)
 SLIP traps for data sharing environments [471](#)
 SLIP traps, requesting [474](#)
 SQL abend [480](#)
 SQLCODE
 unexpected SQLCODE [484](#)
 SVC dumps, increasing storage capacity [469](#)
 SVC dumps, requesting [474](#)
 SVC dumps, setting up customized [470](#)
 SYS1.LOGREC data [472](#)
 SYSLOG [472](#)
 SYSLOG, preserving [472](#)
 z/OS console (SYSLOG) [472](#)

diagnostic data (*continued*)
 z/OS system trace, maximizing [468](#)

diagnostic documentation
 service SQL [475](#)

diagnostics
 DRDA exceptions [318](#)

directory
 order of recovery
 I/O errors [119](#)

disability [x](#)

disaster recovery
 archive logs [133](#), [138](#)
 data mirroring [156](#)
 image copies [133](#), [138](#)
 rolling disaster [156](#)
 scenarios [132](#)
 system-level backups [132](#)
 tracker sites [147](#)

DISPLAY DATABASE(*) RESTRICT [178](#)

DISPLAY OASN command
 IMS [58](#)

DISPLAY SPUFI [282](#)

DISPLAY THREAD command
 TYPE (INDOUBT) option [64](#)

distributed data facility
 "hangs" during processing [298](#)
 commands
 CANCEL DDF THREAD [298](#)
 D NET,BFRUSE [304](#)
 D NET,ROUTE [301](#)
 DISPLAY THREAD DETAIL [298](#)
 DISPLAY THREAD(*) DETAIL [301](#)
 VTAM TERMINATE [298](#)

DDF dumps [293](#)
 DDF serviceability trace [293](#)

diagnosing
 failures [293](#)
 loops [303](#)
 waits [293](#)

distributed SQL application flow
 TCP/IP connection [294](#)
 VTAM connection [293](#)

error messages on the z/OS console
[294](#)

error unique to DDF [293](#)
 gathering diagnostic information [293](#)
 global trace [248](#)

looping
 VTAM internal trace [303](#)

problem determination procedures [299](#)

SNA
 pacing [293](#)
 storage shortage [304](#)
 terminating a session [298](#)

VTAM
 buffer trace [310](#)
 common service area [299](#)
 internal trace [314](#)
 paging [301](#)
 path information units (PIU) [308](#)
 request unit (RU) [309](#)
 request/response header [308](#)
 return codes [294](#)
 session identifier [298](#)

- distributed data facility (*continued*)
 - VTAM (*continued*)
 - traces [307](#)
 - transmission header [308](#)
- Distributed data facility
 - diagnosing
 - waits [25](#)
- distributed environment
 - restart conditions [107](#)
 - restarting
 - DB2 subsystem [107](#)
- distributed relational database architecture (DRDA)
 - Db2 reason codes [317](#)
 - DDM model [318](#)
 - diagnosis
 - database server [317](#)
 - requester [317](#)
 - distributed data interchange services RDTA structure [323](#)
 - distributed data interchange services ZEDAstructure [324](#)
 - early descriptor LID [319](#)
 - Early descriptors [318](#)
 - exception condition [316](#)
 - exception event notification [316](#)
 - FDOCA
 - descriptors, local identifier (LID) [318](#)
 - model [318](#)
 - late descriptors [319](#)
 - object descriptors [318](#)
 - statistics class 4 [318](#)
 - trace records
 - IFCID 0191 [317](#), [324](#)
 - IFCID 0192 [317](#), [330](#)
 - IFCID 0193 [317](#), [330](#)
 - IFCID 0194 [317](#)
 - IFCID 0195 [317](#)
- distributed two-phase commit
 - alerts [331](#)
 - error conditions [331](#)
 - statistics class 4 [331](#)
 - trace records
 - IFCID 0203 [331](#)
- down-level detection
 - controlling [116](#)
 - DSNTIPN panel [116](#)
- down-level page sets
 - recovering [116](#)
- DSN command processor [264](#)
- DSN subcommmands
 - ABEND [283](#)
- DSN1 value [226](#)
- DSN1COPY dump
 - using to resolve inconsistency [338](#)
- DSN1COPY utility
 - CHECK option [178](#)
 - log RBA
 - resetting [110](#)
- DSN1LOGP utility
 - examples [88](#)
 - limitations [110](#)
 - lost work
 - showing [80](#)
 - output [90](#)
- DSN1PRNT dump
 - using to resolve inconsistency [337](#), [338](#)
- DSN2 value [226](#)
- DSN3AUFR
 - VRA data recorded by [237](#)
- DSN9SCN9
 - VRA data recorded by [226](#)
- DSNDB07 database
 - data sets
 - extending [125](#)
- DSNGDxxx module
 - SVC dump [345](#)
- DSNLFRCV
 - VRA data recorded by [226](#)
- DSNSDWA [226](#)
- DSNTFRCV
 - VRA data recorded by [226](#)
- DSNTIPN panel
 - LEVEL UPDATE FREQ field [116](#)
- DSNTRACE [264](#)
- DSNTRACE messages [252](#)
- DSNU259I [344](#)
- DSNU500I [344](#)
- DSNV086E [18](#)
- DSNWDMP
 - gathering information from (SVC) [196](#)
 - option keywords [183](#)
 - volatile data [183](#)
- DSNWDSDM
 - VRA data recorded by [226](#)
- DSNWMMSG file [252](#)
- DSNWSDDWA
 - VRA data recorded by [229](#)
- dump
 - Db2-issued SVC
 - analyzing [196](#)
 - DBDs, of
 - obtaining [345](#)
 - DSN1COPY (using to resolve inconsistency) [337](#), [338](#)
 - DSN1PRNT (using to resolve inconsistency) [337](#), [338](#)
 - finding
 - ASCE [210](#)
 - BMEPL [213](#)
 - CSECT name [212](#)
 - CT [215](#)
 - EB [207](#)
 - load module name [211](#)
 - MEPL [209](#)
 - save area trace [208](#)
 - SCOM [210](#)
 - SQL statement [218](#)
 - SQLCA [216](#)
 - TCB summary [206](#)
 - VRA report [205](#)
 - IRLM data set [284](#)
 - IRLM SVC [274](#)
 - Print Dump Index [203](#)
 - REPAIR (using to resolve inconsistency) [337](#), [338](#)
 - sample
 - RTM2WA summary [225](#)
 - SQLCA, unformatted [217](#)
 - SVC title [199](#)
 - SYSUDUMP [224](#)
 - SUMDUMP Dump Index [203](#)

- dump (*continued*)
 - SVC [196](#)
 - SYS1.LOGREC data [182](#)
 - TCB summary section [182](#)
 - title format, Db2-issued SVC
 - data manager variation [200](#)
 - distributed title variation [200](#)
 - format for database access threads [199](#)
 - format for threads with no remote activity [199](#)
 - format for threads with remote activity [199](#)
 - variation with PSW and ASID [200](#)
 - trace code in SVC dump
 - finding in SVC dump [338](#)
 - types [181](#)
- dump data sets
 - automatic updates [469](#)
 - preserving [473](#)
- dump title
 - IRLM-issued [202](#)
- dynamic statement cache
 - problem diagnosis [499](#)

E

- EB (execution block)
 - address of failing EB [245](#)
 - finding in dump [207](#)
- EBHASCE field [210](#)
- EBPASCE field [210](#)
- EBRMVT field [219](#)
- EDM pool space
 - diagnosing [274](#)
- EDM pool space shortage [276](#)
- EID (event identifier)
 - description [361](#)
 - SDSNSAMP(DSNWEIDS) [361](#)
- EID descriptions [361](#), [362](#)
- error qualifier [202](#)
- ESTAE
 - established by IMS [260](#)
- event identifier (EID) [361](#)

F

- failure symptoms
 - abend
 - log problems [98](#)
 - restart failure [93](#)
 - BSDS (bootstrap data set) [80](#)
 - CICS
 - attachment abends [63](#)
 - task abends [66](#)
 - waits [62](#)
 - logs
 - lost information [103](#)
 - messages
 - DFH2206 [62](#)
 - DFS555 [60](#), [61](#)
 - DSNB207I [111](#)
 - DSNJ 101
 - DSNJ001I [77](#)
 - DSNJ004I [71](#)
 - DSNJ100 [101](#)

- failure symptoms (*continued*)
 - messages (*continued*)
 - DSNJ103I [73](#)
 - DSNJ105I [70](#)
 - DSNJ106I [72](#)
 - DSNJ107 [101](#)
 - DSNJ114I [74](#)
 - DSNM002I [58](#)
 - DSNM005I [59](#)
 - DSNM3201I [62](#)
 - DSNP007I [121](#)
 - DSNP012I [120](#)
 - DSNU086I [118](#), [119](#)
 - processing failure [53](#)
 - subsystem termination [67](#)
- file page set
 - data page
 - illustration [339](#)
- forward log recovery
 - failures [93](#)
 - scenarios [93](#)
- FRR (functional recovery routine)
 - keyword [40](#)

G

- general performance problems
 - Db2 performance [477](#)
- general-use programming information, described [512](#)
- global trace
 - for distributed data facility [248](#)
 - starting [240](#)
- GRANT statement
 - catalog table format [276](#)
- GTF (generalized trace facility)
 - global trace data [248](#)
- GUPI symbols [512](#)

I

- I/O errors
 - archive log data sets
 - recovering [75](#)
 - catalog [119](#)
 - directory [119](#)
 - table spaces [118](#)
- IEPL (initialization entry point list)
 - in MEPL [209](#)
- IFCEREP1 service aid [226](#), [284](#)
- IFCID
 - 0160 [310](#)
 - 0161 [310](#)
 - 0191 [317](#), [324](#)
 - 0192 [317](#), [330](#)
 - 0193 [317](#), [330](#)
 - 0194 [317](#)
 - 0195 [317](#)
 - 0203 [331](#)
- IFCID (instrumentation facility component identifier)
 - 0330 [69](#)
- IFCID trace
 - MELP [211](#)
- IMPORT command

IMPORT command (*continued*)
access method services [101](#)

IMS

loops [58](#)
recovery procedures [57–59](#)
waits [58](#)

IMS attachment facility
trace [254](#)

IMS commands
DISPLAY OASN [58](#)

inconsistent data
data page [339](#)
identifying [88](#)
index page [342](#)
resolving data and index inconsistencies [343](#)

inconsistent data problems
causes [174](#)
description [174](#)
symptoms
00C9011x abend [344](#)
symptoms and actions [174](#)
types [174](#)

inconsistent DBD problems
resolving [344](#)

inconsistent page [175](#)

INCORROUT keyword [35, 42](#)

indefinite wait condition
recovering [131](#)

index
problems [276](#)

index name
finding in a dump [343](#)
finding in VRA [343](#)
finding in VRA report [343](#)

index page
analyzing [342](#)

index-based partitions
redefining [121](#)

indoubt threads
resolving [162](#)

indoubt units of recovery
CICS [64](#)
IMS [58](#)

integrated catalog facility catalog
VVDS (VSAM volume data set) failure
recovering [120](#)

IPCS service aid [181](#)

IRLM

child-lock delay problems [494](#)
Db2 problems [494](#)
hang problems [496](#)
P-lock delay problems [495](#)

IRLM (internal resource lock manager)
failure [53](#)
recovery procedures [53](#)

IRLM (Internal resource lock manager)
trace, IRLM-DB2 [268](#)

IRLM (Internal Resource Lock Manager)
dump data set [284](#)
function level [284](#)
service aids [284](#)

IRLM SVC dump support [274](#)

J

JES job logs, preserving [473](#)

K

keyword

ABENDx [14](#)
component identifier [10](#)
CSECT [35](#)
dependency [49](#)
flowchart [4](#)
INCORROUT [35, 42](#)
load module [38](#)
LOOP keyword [19](#)
MSGx [33, 49](#)
PERFM [35](#)
recovery routine [40](#)
release identifier [12](#)
search process [3](#)
searching [50](#)
SQLCODE [48](#)
string, building [3](#)
structured, examples [199](#)
VRA [41](#)
WAIT keyword [19](#)

keyword formats

free format [4](#)
structured database cross-reference [6](#)
structured database format [6](#)

L

limiting access to data [178](#)

links

non-IBM Web sites
[513](#)

load module

keyword [38](#)
name, finding in dump [211](#)

LOBs

invalid
recovering [117](#)

log

IMS [259](#)
ISPF [266](#)
log initialization phase
failure recovery [83](#)
LOGDATA [182](#)
logical unit of work
identifier in SYS1.LOGREC [226](#)

LOGREC data, preserving [472](#)

logs

dual
minimizing restart efforts [101](#)
excessive loss [103](#)
failure
symptoms [80](#)
total loss [103](#)
recovery procedures
active logs [69](#)
archive logs [73](#)
recovery scenario [101](#)

logs (*continued*)
truncation [88](#)
logs, DB2
preserving [473](#)
LOOP diagnosing
during distributed processing [19](#)
lost work
identifying [88](#)

M

master trace
increasing storage capacity for [468](#)
MAXSPACE
SVC dump setting [469](#)
media failures
recovering [152](#)
MELP
IFCID trace [211](#)
MEPL (module entry point list)
finding in dump [209](#)
illustration [210](#)
in SVC dump [198](#)
message by identifier
DFS3602I [59](#)
DFS554I [60, 61](#)
DFS555A [60, 61](#)
DFS555I [60, 61](#)
DSN1150I [99](#)
DSN1151I [57](#)
DSN1157I [88, 99](#)
DSN1160I [88, 99](#)
DSN1162I [57, 88, 99](#)
DSN1213I [106](#)
DSN2001I [64](#)
DSN2025I [67](#)
DSN2034I [64](#)
DSN2035I [64](#)
DSN2036I [64](#)
DSN3100I [67](#)
DSN3104I [67](#)
DSN3201I [62](#)
DSNB204I [111](#)
DSNB207I [111](#)
DSNB232I [116](#)
DSNJ001I [80, 83](#)
DSNJ003I [78](#)
DSNJ004I [71](#)
DSNJ007I [86, 94](#)
DSNJ012I [86, 94](#)
DSNJ100I [76, 77, 80, 101](#)
DSNJ103I [73, 86, 94](#)
DSNJ104I [73, 86, 94](#)
DSNJ105I [70](#)
DSNJ106I [72, 86, 94](#)
DSNJ107I [76, 80, 101](#)
DSNJ108I [76](#)
DSNJ110E [69](#)
DSNJ111E [69](#)
DSNJ113E [86, 94, 100](#)
DSNJ114I [74](#)
DSNJ115I [73](#)
DSNJ119I [80](#)
DSNJ119I [101](#)

message by identifier (*continued*)

DSNJ120I [76, 77](#)
DSNJ124I [72](#)
DSNJ126I [76](#)
DSNJ128I [75](#)
DSNL030I [131](#)
DSNL500I [130](#)
DSNL501I [126, 130](#)
DSNL502I [126, 130](#)
DSNL700I [127](#)
DSNL701I [127](#)
DSNL702I [127](#)
DSNL703I [127](#)
DSNL704I [127](#)
DSNL705I [127](#)
DSNM002I [58, 67](#)
DSNM004I [58](#)
DSNM005I [59](#)
DSNP001I [121](#)
DSNP007I [121](#)
DSNP012I [120](#)
DSNR002I [80](#)
DSNR003I [57, 96, 99](#)
DSNR004I [80, 83, 93](#)
DSNR005I [80, 83, 98](#)
DSNR006I [80](#)
DSNU086I [118, 119](#)
DSNU561I [125](#)
DSNU563I [125](#)
DSNV086E [67](#)
DSNV401I [64](#)
DSNV406I [64](#)
DSNV408I [64](#)
DSNV414I [64](#)
DSNV415I [64](#)
DXR122E [53](#)
EDC3009I [120](#)
IEC161I [111](#)
MODIFY command
function level [284](#)
IRLM [284](#)
order of issue [29](#)
MSGx keyword [33, 49](#)
multiple address space
data sets [196](#)

N

network ID (NID)
CICS [64](#)
IMS [58](#)
NID (network ID)
CICS [64](#)
IMS [58](#)

O

OBD (object descriptor)
finding (given OBID) [349](#)
OBDDMAP
analyzing OBD structure [349](#)
originating sequence number (OASN)
indoubt units of recovery [58](#)

P

- page number
 - determining from DSNI message [343](#)
 - finding in a dump [343](#)
 - finding in VRA [343](#)
 - finding in VRA report [343](#)
- page set name
 - determining from DSNI message [343](#)
 - finding in a dump [343](#)
 - finding in VRA [343](#)
 - finding in VRA report [343](#)
- parallelism problems
 - IFCID [290](#)
- partitions
 - index-controlled
 - redefining [121](#)
 - redefining
 - index-based partitioning [124](#)
 - table-controlled
 - redefining [121](#)
- PC linkage
 - description [281](#)
 - index, finding in dump [209](#)
- percolation
 - finding [229](#)
 - multiple for single error [229](#)
 - recovery termination manager [229](#)
- PERFM keyword [35](#)
- PGHEAD (page header) [339](#)
- point-in-time recovery
 - Db2 subsystem [113](#)
 - Db2 utilities [494](#)
- Print Dump Index
 - dump index illustration [203](#)
 - dump index usage [203](#)
 - illustration [203](#)
 - usage [203](#), [205](#)
 - using to find TCB summary [206](#)
- print dumps [181](#), [182](#)
- print log map utility
 - before fall back [101](#)
- print traces [238](#)
- problem analysis documentation
 - service SQL [475](#)
- problem determination
 - analyzing 00C90102 abends [338](#)
 - analyzing 00C9010x or 00C902xx abends [335](#)
 - analyzing a DBD [349](#)
 - analyzing symptoms of data inconsistency [178](#)
 - dump analysis [196](#)
 - symptoms of data inconsistency [175](#)
- product-sensitive programming information, described [512](#)
- programming interface information, described [512](#)
- PSPI symbols [512](#)
- PSW
 - as index into MEPL [211](#)
 - in dumps [196](#)

Q

- QMF-related failures [66](#)
- query parallelism [288](#)

R

- RDA (RDS control area)
 - in SVC dump [218](#)
- RDASPPT1 field [218](#)
- RDS (relational data system)
 - problem diagnosis [498](#)
- REBUILD INDEX utility
 - using to resolve inconsistency [343](#)
- RECORDING MAX field
 - panel DSNTIPA
 - preventing frequent BSDS wrapping [100](#)
- RECOVER INDOUBT command
 - free locked resources [64](#)
- RECOVER TABLESPACE utility
 - modified data
 - recovering [101](#)
- RECOVER utility
 - failure [276](#)
 - options
 - TOLOGPOINT [56](#)
 - recovery cycle [150](#)
 - removing inconsistencies [344](#)
 - using to resolve inconsistency [343](#)
- recovery
 - application changes
 - backing out [56](#)
 - backward log recovery failures [98](#)
 - BSDS (bootstrap data set) [78](#)
 - communications failure [163](#)
 - Db2 outages
 - cold start [168](#)
 - Db2 subsystem [105](#), [113](#)
 - DDF (distributed data facility) failures [126](#)
 - disk failures [54](#)
 - down-level page sets [116](#)
 - heuristic decisions
 - correcting [173](#)
 - IMS outage with cold start
 - scenario [167](#)
 - IMS-related failures
 - during indoubt resolution [59](#)
 - indoubt units of recovery [58](#)
 - inconsistent data
 - resolving [97](#)
 - indoubt threads [162](#)
 - indoubt units of recovery
 - CICS [64](#)
 - integrated catalog facility catalog
 - VVDS failure [120](#)
 - invalid LOBs [117](#)
 - logs
 - truncating [86](#)
 - lost status information [91](#)
 - point in time [113](#)
 - procedures [53](#)
 - scenarios [168](#)
 - table spaces [118](#)
 - temporary resource failures [68](#)
- recovery cycle
 - RECOVER utility [150](#)
- RECOVERY option
 - REPORT utility [56](#)
- recovery procedures

- recovery procedures (*continued*)
 - application program errors [56](#)
 - CICS-related failures
 - application failures [62](#)
 - attachment facility failures [66](#)
 - indoubt units of recovery [64](#)
 - not operational [62](#)
 - Db2-related failures
 - active log failures [69](#)
 - archive log failures [73](#)
 - BSDS (bootstrap data set) [76](#)
 - catalog or directory I/O errors [119](#)
 - database failures [111](#)
 - subsystem termination [67](#)
 - table space I/O errors [118](#)
 - IMS-related failures
 - application failures [60](#), [61](#)
 - control region failures [58](#)
 - integrated catalog facility catalog
 - VVDS failure [121](#)
 - IRLM failures [53](#)
 - out-of-disk-space [121](#)
 - QMF-related failures [66](#)
 - restart [80](#)
 - VTAM ACB OPEN failures [129](#)
 - z/OS failures [53](#)
 - z/OS power failures [53](#)
- recovery routine keyword [40](#)
- recovery scenarios
 - Db2 cold start [171](#)
 - failures
 - current status rebuild phase [83](#)
 - log initialization phase [83](#)
 - heuristic decisions
 - making [165](#)
- referential constraint
 - violation
 - recovering [125](#)
- release level keyword [12](#)
- remote logical units
 - failures [130](#)
- REPAIR DBD [344](#)
- REPAIR dump
 - using to resolve inconsistency [337](#), [338](#)
- REPAIR utility
 - Analyzing a repair [347](#)
 - inconsistent data
 - resolving [110](#)
 - using to resolve inconsistency [342](#)
- REPORT utility
 - options
 - RECOVERY [56](#)
 - TABLESPACESET [56](#)
- REPRO command
 - access method services [78](#)
- resolving inconsistencies
 - using REBUILD INDEX utility [343](#)
 - using RECOVER utility [343](#)
 - using REPAIR utility [342](#)
- resource limit facility problems
 - Db2 operations [488](#)
- restart
 - backward log recovery
- restart (*continued*)
 - backward log recovery (*continued*)
 - failure during [98](#)
 - BSDS (bootstrap data set) problems [101](#)
 - cold start situations [103](#)
 - conditional
 - excessive loss of active log data [104](#)
 - total loss of log [103](#)
 - current status rebuild phase
 - failure recovery [83](#)
 - forward log recovery phase
 - failure during [93](#)
 - inconsistencies
 - resolving [107](#)
 - log data set problems [101](#)
 - log initialization phase
 - failure recovery [83](#)
- restart processing
 - limiting [94](#)
- RESTORE SYSTEM
 - recovery cycle
 - establishing [148](#)
- RMFT (resource manager function table)
 - captured in SVC dump [198](#)
 - unformatted trace table [219](#)
- RMFTRUSE [219](#)
- RMVT (resource manager vector table)
 - summary portion of dump [198](#)
 - unformatted trace table [219](#)
- rolling disaster [156](#)
- RRSAF
 - finding trace table [262](#)
 - interpreting trace messages [262](#)
 - starting trace [261](#)
 - trace [261](#)
- RTM2WA
 - finding in dump [206](#)
 - illustration [206](#)
 - summary in SYSABEND or SYSUDUMP [253](#)
 - TCB summary section [206](#)
- RTM2WA summary (sample) [225](#)
- RUNSTATS function [276](#)

S

- save area trace [208](#)
- SCOM (subsystem communications block)
 - finding in dump [210](#)
 - in SVC dump [198](#)
- SCOMMEPL field [210](#)
- SDSNSAMP(DSNWEIDS) [361](#)
- SDUMP
 - services [197](#)
- SDWA (system diagnostic work area)
 - finding in a dump [204](#), [343](#)
 - functional recovery routines [197](#)
- SDWACOMU field [226](#)
- SDWAEC1 field [226](#)
- search process [3](#)
- service aid
 - IPCS [181](#)
- service SQL [475](#)
- shortcut keys
 - keyboard [x](#)

- SKB (stack storage block)
 - locating for failing agent [221](#)
- SLIP traps for data sharing environments [471](#), [474](#)
- SMF (System Management Facilities)
 - global trace data [248](#)
- SPA (space block structure)
 - in SVC dump [218](#)
- space name
 - determining from DSNJ message [343](#)
- SPASQLTX field [218](#)
- SPUFI
 - diagnostic panels, activating [282](#)
 - trace messages [457](#)
- SQL (structured query language)
 - finding statements in dump [218](#)
- SQLCA (SQL communication area)
 - content [217](#)
 - finding in dump [216](#)
 - in SVC dump [198](#)
- SQLCODE keyword [48](#)
- SQLDA
 - binary XML data [277](#)
- stack storage block (SKB) [221](#)
- stack storage for failing agent, locating [221](#)
- stand-alone log services (DSNJSLR)
 - change log inventory (DSNJU003) [275](#)
 - print log map (DSNJU004) [275](#)
- starting
 - Db2 [94](#)
- statistics class 4 [318](#), [331](#)
- Stop Trace command
 - description [243](#)
- stored procedures
 - SQLCODE -430 [174](#)
 - troubleshooting [174](#)
- SUMLSTA [198](#)
- SVC dump
 - diagnostic area [335](#)
 - indexes [203](#)
 - save area trace report section [208](#)
 - summary portion [198](#)
 - title [199](#)
 - using to find trace code [338](#)
 - using to resolve inconsistency [338](#)
- SVC dumps
 - increasing storage capacity for [469](#)
 - MAXSPACE setting [469](#)
 - requesting [474](#)
 - setting up customized [470](#)
- symptoms of data inconsistency problems
 - 00C9011x abend [344](#)
- syntax diagram
 - how to read [xi](#)
- SYS1.LOGREC
 - entry [284](#)
 - IRLM [284](#)
 - percolation analysis [226](#)
 - sample [226](#)
- SYS1.LOGREC data set [67](#)
- SYS1.LOGREC data, preserving [472](#)
- SYSABEND dump [253](#)
- SYSIBM.SYSCOLUMNS catalog table [276](#)
- SYSIBM.SYSCOPY catalog table [276](#)
- SYSIBM.SYSINDEXES catalog table [276](#)

- SYSIBM.SYSPACKAGE catalog table [276](#)
- SYSIBM.SYSPLANDEP catalog table [276](#)
- SYSIBM.SYSSTOGROUP catalog table [276](#)
- SYSIBM.SYSTABLE catalog table [276](#)
- SYSIBM.SYSTABLESPACE catalog table [276](#)
- SYSIBM.SYSUSAGE catalog table [276](#)
- SYSLOG, preserving [472](#)
- system trace, maximizing [468](#)
- system-level backups
 - disaster recovery [132](#)
- SYSUDUMP [253](#)
- SYSUDUMP dump sample [224](#)

T

- TAB (trace anchor block)
 - in unformatted trace table [219](#)
- table spaces
 - recovering [118](#)
 - restoring [109](#)
- table-based partitions
 - redefining [121](#)
- TABLESPACESET option
 - REPORT utility [56](#)
- takeover site
 - setting [153](#), [154](#)
- TCB (task control block)
 - summary, finding in dump [206](#)
- TCP/IP
 - failure recovery [130](#)
 - gethostbyaddr call failure [297](#)
 - listener not started [297](#)
 - startup problems [297](#)
- terminating
 - Db2
 - scenarios [67](#)
 - title of SVC dump [199](#)
- trace
 - activating DSN command processor [264](#)
 - activating SPUFI [266](#)
 - activating TSO attachment CLIST [265](#)
 - CAF trace [252](#)
 - classes for global trace [248](#)
 - code
 - EID (event identifier) [361](#)
 - contents of unformatted table [246](#)
 - Db2 global [239](#)
 - Db2 vs. z/OS [248](#)
 - Db2-IRLM [268](#)
 - DSNTRACE [264](#)
 - finding CAF table [253](#)
 - finding Db2 table [219](#)
 - finding formatted table [219](#)
 - finding IMS table [259](#)
 - finding RRSF table [262](#)
 - finding unformatted table [219](#)
 - formatting, how to [219](#)
 - IMS attachment facility [254](#), [256](#)
 - IMS log entries [259](#)
 - interpreting CAF messages [253](#)
 - interpreting formatted table [245](#)
 - interpreting GTF data [248](#)
 - interpreting RRSF messages [262](#)
 - interpreting SMF data [248](#)

- trace (*continued*)
 - producing DSNTRACE messages [252](#)
 - RRSAF trace [261](#)
 - save area [208](#)
 - SPUFI [266](#)
 - starting RRSAP trace [261](#)
 - statistics class 4 records [331](#)
 - table, header fields [219](#)
 - TSO attachment CLIST [265](#)
 - TSO attachment facility [263](#), [264](#)
 - when to use Db2 table [245](#)
 - z/OS [197](#), [275](#)
- trace field descriptions [252](#)
- trace messages
 - call attachment facility [452](#)
 - TSO attachment facility [363](#)
- tracker site
 - characteristics [147](#)
 - converting
 - takeover site [153](#), [154](#)
 - disaster recovery [147](#)
 - maintaining [153](#)
 - recovering
 - RECOVER utility [154](#)
 - RESTORE SYSTEM utility [153](#)
 - recovery cycle
 - RESTORE SYSTEM utility [148](#)
 - setting up [148](#)
- troubleshooting
 - QMF-related failures [66](#)
 - stored procedures [174](#)
- troubleshooting with utilities [278](#)
- truncation
 - active logs [88](#)
- TSO attachment facility
 - SPUFI diagnostic panels [282](#)
 - trace messages [363](#)
- type code
 - finding in VRA report [343](#)

U

- unavailable data [175](#)
- unexpected SQLCODE
 - SQLCODE [484](#)
- units of recovery
 - indoubt
 - CICS [64](#)
 - IMS [58](#)
- user-defined data sets
 - extending [122](#)
 - volumes
 - adding [122](#)
- user-managed data sets
 - enlarging [123](#)
- utilities
 - for diagnosis and troubleshooting [278](#)
- utilities, DB2
 - Db2 problems [491](#)
- utility address space, DB2
 - Db2 utilities [492](#)
- utility jobs
 - running
 - recovery procedures [145](#)

V

- VRA (variable recording area)
 - Db2-unique data [205](#)
 - diagnostic information report [205](#)
 - finding in a dump [343](#)
 - key descriptions [226](#)
 - obtaining data [205](#)
 - percolation analysis [226](#)
 - recording subcomponents [234](#)
 - starting point in SDWA [207](#)
- VRA keyword [41](#)
- VRARRK14 [343](#)
- VRARRK15 [343](#)
- VRARRK5 [337](#)
- VRARRK6 [339](#)
- VSAM (virtual storage access method)
 - VSAM volume data set (VVDS)
 - recovering [120](#)
- VTAM
 - buffer trace [310](#)
 - failures
 - recovering [128](#)
 - internal trace [314](#)
 - path information units (PIU) [308](#)
 - request unit (RU) [309](#)
 - request/response header [308](#)
 - traces [307](#)
 - transmission header [308](#)
- VTAM ACB OPEN)
 - failure [129](#)
- VTAM IO trace
 - ACFTAP option [311](#)
- VTAM)
 - recovery procedures [129](#)
- VVDS (VSAM volume data set)
 - recovering [120](#)

W

- WAIT keyword
 - hangs during distributed processing [19](#)
- WAL (IMS-work area list key
 - 7)
 - trace description [259](#)
- WAU (IMS-work area user key
 - 8)
 - trace description [259](#)
- work file databases
 - data sets
 - enlarging [125](#)
 - enlarging [121](#)
 - extending [125](#)

X

- XRC (Extended Remote Copy) [161](#)

Z

- z.OS console, preserving [472](#)
- z/OS
 - power failure

z/OS *(continued)*

power failure *(continued)*

recovering [53](#)

z/OS abend

IEC030I [75](#)

IEC031I [75](#)

IEC032I [75](#)

z/OS system trace, maximizing [468](#)



Product Number: 5698-DB2
5698-DBV