IBM Migration Utility for z/OS
Version 5 Release 1

Generating and Parsing XML and JSON Documents

SC27-9083-00

Note

Before using this information and the product it supports, be sure to read the general information under “Notices” on page 31.

First Edition (October 2020)

This edition applies to IBM Migration Utility for z/OS, Version 5 Release 1, Program Number 5698-MG5 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.

The information in this manual was furnished by Transiom Software, Inc.

Contents

About this manual..... v
Who should use this manual..... v
Structure of this manual..... v
Syntax notation..... v
How to send your comments to IBM®..... vii
Email feedback template..... vii
If you have a technical problem..... vii
Generating XML documents..... 9
Using the XML option on the REPORT statement..... 9
Syntax..... 9
Parameters..... 9
Special rules..... 9
Sample XML programs..... 10
Program example..... 10
Using %XML macro for parsing XML files..... 11
XML macro: coding rules..... 11
Format 1 processing logic..... 12
Format 2 processing logic..... 12
Special Registers..... 12
The content of XML-EVENT..... 14
XML PARSE example..... 15
Publishing XML documents and reports to z/OS Server..... 17
To publish a report or XML document to z/OS UNIX..... 17
Defining UNIX files in the JCL..... 17
Generating JSON documents..... 19
Using the JSON option on the REPORT statement..... 19
Syntax..... 19
Parameters..... 19
Special rules..... 19
Using the %JSON macro for parsing JSON files or hard coded JSON Script..... 20
JSON macro: coding rules..... 20
Format 1: parsing a JSON string..... 20
Format 1: processing logic..... 21
Special JSON registers..... 21
JCL (JOBs) for running JSON programs..... 22
Building data structure in Easytrieve Plus program for JSON PARSE..... 22
Running JCMUJUT0 Utility..... 22
Publishing JSON documents and reports to a z/OS Server..... 23
To publish a report or JSON document to z/OS UNIX..... 23
Defining UNIX files in the JCL..... 24
Program examples and error messages..... 25
Messages..... 25
Program examples..... 26
Example 1: Generate JSON Script using REPORT JSON option..... 26
Example 2: PARSE field information from JSON Script file using JSON PARSE..... 27
Example 3: Extract field information from hard coded JSON Script file in the program source.. 28
Example 4: PARSE field information from JSON Script file that contains arrays using JSON PARSE EXIT option..... 29

Notices..... 31

Trademarks..... 32

Index..... 33

About this manual

This manual describes how to use the XML and JSON statements in Easytrieve Plus programs supported by IBM Migration Utility for z/OS licensed program, hereafter referred to as Migration Utility.

Who should use this manual

This manual is for anyone who wants to use XML and JSON statements in Easytrieve Plus programs supported by Migration Utility.

Structure of this manual

"Generating XML documents" on page 9 describes how to produce XML reports using PRINTER file(s), how to parse an XML buffer or file, and how to publish XML documents and XML reports to z/OS server,

"Generating JSON documents" on page 19 describes the REPORT option, the JSON macro, how to publish JSON reports, and gives program examples and where to find error message details.

Syntax notation

Throughout this book, syntax descriptions use the structure defined below.

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

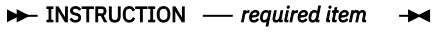
  The ►►—>>- symbol indicates the beginning of a statement.

  The ——► - -> symbol indicates that the statement syntax is continued on the next line.

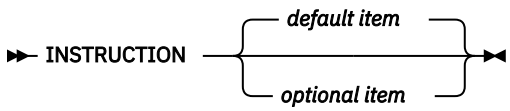
  The ►——>- - symbol indicates that a statement is continued from the previous line.

  The —►◄- ->< indicates the end of a statement.
- **Keywords** appear in uppercase letters (for example, ASPACE) or upper and lower case (for example, PATHFile). They must be spelled exactly as shown. Lower case letters are optional (for example, you could enter the PATHFile keyword as PATHF, PATHFI, PATHFIL or PATHFILE).

  Variables appear in all lowercase letters in a special typeface (for example, *integer*). They represent user-supplied names or values.
- If punctuation marks, parentheses, or such symbols are shown, they must be entered as part of the syntax.
- Required items appear on the horizontal line (the main path).

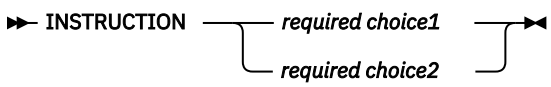


- Optional items appear below the main path. If the item is optional and is the default, the item appears above the main path.

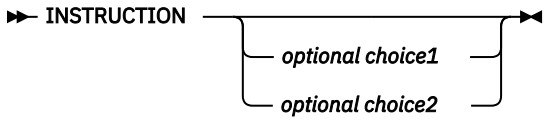


- When you can choose from two or more items, they appear vertically in a stack.

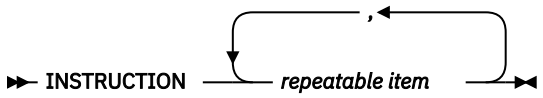
  If you **must** choose one of the items, one item of the stack appears on the main path.



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

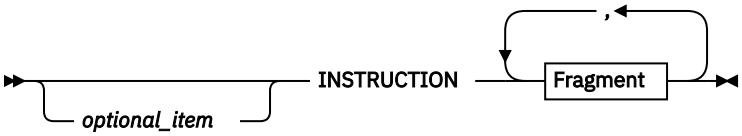


- An arrow returning to the left above the main line indicates an item that can be repeated. When the repeat arrow contains a separator character, such as a comma, you must separate items with the separator character.

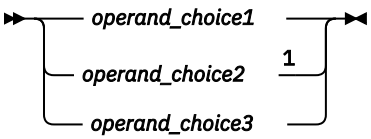


A repeat arrow above a stack indicates that you can make more than one choice from the stacked items, or repeat a single choice.

All of these elements can be combined together into one diagram. For example:



Fragment



Notes:

[1] *operand_choice2* and *operand_choice3* must not be specified together.

optional_item
Is an optional item, and when you code the command, you may code the item or not.

INSTRUCTION
This key word must be specified and coded as shown.

Fragment
This item is a required operand. You can see the choices for the operand in the fragment of the syntax diagram shown below `Fragment:` at the bottom of the diagram. The return loop in the main diagram shows that the operand can also be repeated. That is, more than one choice can be specified, with each choice separated by a comma. The note at the bottom of the syntax diagram indicates a restriction on the choice.

For example, here are some acceptable commands:

```
INSTRUCTION operand_choice1
optional_item INSTRUCTION operand_choice_1, operand_choice_2
```

And some not acceptable commands:

```
optional_item operand_choice_2
  - It misses out INSTRUCTION
INSTRUCTION
```

- It doesn't supply an operand after the key word
`INSTRUCTION operand_choice2, operand_choice3`
- It breaks the restriction mentioned in the note

How to send your comments to IBM®

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

Use one of the following methods to send your comments:

1. Use the feedback link at the bottom of Knowledge Center.
2. Use the feedback template below and send us an email at "mhvrcfs@us.ibm.com"
3. Mail the comments to the following address:

   IBM Corporation
   Attention: MHVRCFS Reader's Comments
   Department H6MA, Building 707
   2455 South Road
   Poughkeepsie, NY 12601-5400
   US

Email feedback template

Please cut and paste the template below into your email. Then fill in the required information.

- My name:
- My Company, University or Institution:
- The URL of the topic or web page you are commenting on:
- The text of your comment

If you are willing to talk to us about your comment, please feel free to include a phone number and the best time to reach you.

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

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

If you have a technical problem

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

- Contact your IBM service representative
- Call IBM technical support
- Visit the IBM support portal at https://www.ibm.com/support/home/.

If you have a technical problem

Generating XML documents

Using the XML option on the REPORT statement

The REPORT statement XML option provides users with a capability to generate XML reports.

Syntax

```
REPORT PRINTER &REPORT. . . . XML (version="&vers" encoding="&encode"
standalone="&option")
```

Parameters

&REPORT

REPORT file DDname as defined by the FILE statement. The &REPORT file must be defined as a printer file or a printer file of type SERVER.

&vers

XML Version. The default is 1.0

&encode

Encoding. The default is ibm-1140

&option

YES or NO. The default is YES.

Special rules

The XML document is written to the specified printer file.

To publish the XML document to z/OS UNIX, define the PRINTER file as a SERVER file and provide printer file DD statement in the JCL. The DCB information in the JCL is not needed.

Define example:

```
FILE REPORT1 PRINTER (132) SERVER
```

Note: See “Publishing XML documents and reports to z/OS Server” on page 17 for details.

When &REPORT is defined as a SERVER, the XML document is written as a Variable Blocked (VB) file with the maximum record length of 4096 bytes.

- The print control character is omitted.
- The attributes are automatically forced by IMU.
- The DCB information provided in the JCL is ignored.

When &REPORT is **not** defined as a SERVER, the XML document is written as a standard report file.

All REPORT statement spacing and positioning options such as NOADJUST, COL, SKIP, ETC., serve no purpose and are ignored regardless of the document destination.

The XML document is generated from fields defined on the CONTROL and LINE statements. A hierarchical structured is constructed from the CONTROL and the LINE fields. The CONTROL fields become group (parent) items and the LINE fields become the lowest level (child) elements.

- The TITLE statements are ignored.
- The Control break totals are ignored.

If the SUMMARY option is specified, the lowest level elements become the fields that would have been printed for the lowest level control break.

The SUMFILE option is supported as in a non-XML environment.

The tags are generated from the field headings. Multiple heading columns are connected by IMU with the underscore (_) character automatically.

Example: HEADING BALANCE ('CURRENT' 'BALANCE') is generated as:

```
<CURRENT_BALANCE>&value</CURRENT_BALANCE>
```

The XML document is written using PRINT statement in the Activity section and optionally DISPLAY statements in the report exits.

The DISPLAY statement fields are not formatted as XML Document elements.

The DISPLAY is written exactly as for a non-XML report, however if writing to a SERVER, the print control characters are ignored.

In the XML report, the report SEQUENCE statement functions as it does with a non-XML report. A temporary spool file is created of all fields; the spool file is sorted and the document is printed.

Report exits are invoked as for non-XML reports. However, the ENDPAGE exit is ignored because there is no page concept in an XML document.

If a PRINTER FILE is specified and defined with a MODIFY EXIT, each physical record being written to the printer file is passed to the exit. This is the same as for non-XML reports.

Sample XML programs

To experiment with XML documents, use sample JCLs provided in SYS1.SFSYJCLS as listed below. Each job has a documentation section to help you customize it for your needs.

JCMUXML0
Create and publish XML documents to a z/OS Server.

JCMUXML1
Create XML documents to a flat file.

JCMUXML2
Parse XML document created by JCMUXML0 or JCMUXML1.

Program example

Generate an XML report of WAGE and RATE with CONTROL breaks by COMPANY and OFFICER number.

FILEIN content:

```
10001BBBB0550000010500
10001CCCC0560000010500
20003CCCC0445000011000
20003DDDD0478000010500
```

Easytrieve Plus Program:

```
* EASYTRAN: IOMODE DYNAM
* EASYTRAN: DEBUG (LIST ESPI-FULL)
* END-EASYTRAN
FILE REPORT1 PRINTER SERVER
FILE FILEIN F (80)
COMPANY * 5 N HEADING ('COMPANY')
OFFICER * 4 A HEADING ('OFFICER')
WAGE * 8 N 2 HEADING ('WAGE')
RATE * 5 N 3 HEADING ('RATE')
JOB INPUT FILEIN
PRINT RPT1
GOTO JOB
REPORT RPT1 PRINTER REPORT1 XML
SEQUENCE COMPANY OFFICER
```

```
CONTROL COMPANY OFFICER
HEADING COMPANY ('COMPANY' 'NUMBER')
HEADING OFFICER ('OFFICER' 'NUMBER')
LINE 1 WAGE RATE
```

Produced XML document:

```
<?xml version="1.0" encoding="ibm-1140" standalone="yes" ?>
<RPT1_REPORT>
      <COMPANY COMPANY="10001">
         <OFFICER OFFICER="BBBB">
            <RPT1>
               <WAGE> 55,000.00 </WAGE>
               <RATE> 10.500 </RATE>
            </RPT1>
         </OFFICER>
         <OFFICER OFFICER="CCCC">
            <RPT1>
               <WAGE> 56,000.00 </WAGE>
               <RATE> 10.500 </RATE>
            </RPT1>
         </OFFICER>
      </COMPANY>
      <COMPANY COMPANY="20003">
         <OFFICER OFFICER="CCCC">
            <RPT1>
               <WAGE> 44,500.00 </WAGE>
               <RATE> 11.000 </RATE>
            </RPT1>
         </OFFICER>
         <OFFICER OFFICER="DDDD">
            <RPT1>
               <WAGE> 47,800.00 </WAGE>
               <RATE> 10.500 </RATE>
            </RPT1>
         </OFFICER>
      </COMPANY>
</RPT1_REPORT>
```

Using %XML macro for parsing XML files

The XML macro generates logic for parsing XML documents using the XML PARSE statement supported by z/OS COBOL. Users who intend to use this macro must be fully familiar with the XML document syntax and the XML document structure.

Note:

Describing the format and structure of XML documents is beyond the scope of this manual. Users not familiar with XML should learn from the publicly available XML publications. XML PARSE, provided by COBOL, is documented in the *Enterprise COBOL for z/OS Programming Guide*, including the use of the national and non-national characters.

XML macro: coding rules

Purpose: Parse an XML format document into individual elements using the COBOL XML PARSE statement.

Format 1 - parse XML string

```
%XML PARSE &object EVENT EXIT &proc
```

Where:

&object
File name that contains XML format document.

OR

A field name that contains XML format document.

&proc
PROC name in the Activity section that handles XML events.

Format 2 - debug XML PARSE errors

```
%XML DEBUG &text
```

Where:

&text
XML-TEXT use this register for non-national characters

XML-NTEXT use this register for national characters

Format 1 processing logic

This macro allows the user to parse/extract elements values from XML documents.

When &object is a file name, the file is assumed to be in XML format.

When &object is referenced in the XML macro, the &object file cannot be used for any other purpose in the same JOB.

The following outlines the parsing steps:

1. The &object file is read to calculate the document size in bytes.
2. A buffer is dynamically allocated for the calculated size.
3. The &object file is loaded into the allocated buffer.
4. The COBOL XML parser is invoked to parse the buffer.
5. The &proc procedure is invoked for each event.

When &object is a field/buffer, the value in the field is assumed to be in XML format. The following outlines the parsing steps:

1. The COBOL XML parser is invoked to parse the buffer.
2. The &proc procedure is invoked for each event.

The user is responsible for recognizing events in the event &proc PROC and capturing appropriate values for each element.

Format 2 processing logic

Use this format for debugging (displaying XML parsing events). It is coded in combination with the Format 1 event exit. That is, the FORMAT 1 statement is required with &proc pointing to the proc that contains the %XML DEBUG macro. Refer to "XML PARSE example" on page 15 to get an idea of how it works.

Special Registers

Special registers allocated by the XML parser.

These registers are available in the EVENT &proc Procedure to aid the programmer in identifying and extracting XML elements.

XML-EVENT
Alphanumeric field of 30 characters in length.

This register contains the event name in process.

For meaning, refer to "The content of XML-EVENT" on page 14.

The following events are available:

- XML-TEXTATTRIBUTE-CHARACTER
- ATTRIBUTE-CHARACTERS
- ATTRIBUTE-NAME
- ATTRIBUTE-NATIONAL-CHARACTER
- COMMENT

- CONTENT-CHARACTER
- CONTENT-CHARACTERS
- CONTENT-NATIONAL-CHARACTER
- DOCUMENT-TYPE-DECLARATION
- ENCODING-DECLARATION
- END-OF-CDATA-SECTION
- END-OF-DOCUMENT
- END-OF-ELEMENT
- EXCEPTION
- PROCESSING-INSTRUCTION-DATA
- PROCESSING-INSTRUCTION-TARGET
- STANDALONE-DECLARATION
- START-OF-CDATA-SECTION
- START-OF-DOCUMENT
- START-OF-ELEMENT
- UNKNOWN-REFERENCE-IN-ATTRIBUTE
- UNKNOWN-REFERENCE-IN-CONTENT
- VERSION-INFORMATION

XML-TEXT
Variable length field

Contains the value described by the event.

XML-NTEXT
Variable length National Character field

Contains the value described by the event.

XML-CODE
Completion code

Zero means a good completion. Any other value signals exception.

Important: Additional special registers for XML macro internal use. Do not allocate any field names that conflict with the following register names:

XML-DOC-SEQ
Full word binary field contains the XML macro invocation sequence.

XML-DOC-SIZE
Memory size required to hold the XML file content when &object is XML file.

XML-DOC-HEADER
Alphanumeric field of 7 bytes for internal use in the parsing logic.

XML-DOC-TEXT1
Variable length dynamically allocated buffer when `&object` is XML file.

XML-DOC-MEMP1
Pointer to the dynamically allocated buffer above when `&object` is XML file.

XML-DOC-SIZE1
A four (4) byte integer used by the `XML DEBUG` option. It contains the text size in the `XML-TEXT` for the event `EXCEPTION`.

XML-DOC-SIZ21
A four (4) byte integer used internally when `&object` is an XML file.

XML-DOC-DISP1
A four (4) byte integer used internally when `&object` is an XML file.

The content of XML-EVENT

When an event occurs during XML parsing, the XML parser places the appropriate event name shown below into the XML-EVENT special register. The event is passed to the EVENT exit, and the text that corresponds to the event is provided in either the XML-TEXT or the XML-NTEXT special register.

ATTRIBUTE-CHARACTER
Occurs in attribute values for the predefined symbols references '&', ''', '>', '<', and '"'. See XML specification for details about predefined entities.

ATTRIBUTE-CHARACTERS
Occurs for each fragment of an attribute value. XML text contains the fragment. An attribute value normally consists only of a single string, even if it is split across lines. The attribute value might consist of multiple events, however.

ATTRIBUTE-NAME
Occurs for each attribute in an element start tag or empty element tag, after a valid name is recognized. XML text contains the attribute name.

ATTRIBUTE-NATIONAL-CHARACTER
Occurs in attribute values for numeric character references (Unicode code points or "scalar values") of the form '&#dd..;' or '&#hh..;', where d and h represent decimal and hexadecimal digits, respectively. If the scalar value of the national character is greater than 65,535 (NX'FFFF'), XML-NTEXT contains two encoding units (a surrogate pair) and has a length of 4 bytes. This pair of encoding units represents a single character. Do not create characters that are not valid by splitting this pair. (See the related reference below about code-page-sensitive characters for information about coding the number sign (#).)

COMMENT
Signals comment in the XML document. XML text contains the data between the opening and closing comment delimiters, '<!–' and '-->'. (See the related reference below about code-page-sensitive characters for information about coding the exclamation point (!).)

CONTENT-CHARACTER
Occurs in element content for the predefined entity references '&', ''', '>', '<', and '"'. See XML specification for details about predefined entities.

CONTENT-CHARACTERS
This event represents the principal part of an XML document: the character data between element start and end tags. XML text contains this data, which usually consists only of a single string even if it is split across lines. If the content of an element includes any references or other elements, the complete content might consist of several events. The parser also uses the CONTENT-CHARACTERS event to pass the text of CDATA sections to your program.

CONTENT-NATIONAL-CHARACTER
Occurs in element content for numeric character references (Unicode code points or "scalar values") of the form '&#dd..;' or '&#hh..;', where d and h represent decimal and hexadecimal digits, respectively. If the scalar value of the national character is greater than 65,535 (NX'FFFF'), XML-NTEXT contains two encoding units (a surrogate pair) and has a length of 4 bytes. This pair of encoding units represents a single character. Do not create characters that are not valid by splitting this pair. (See the related reference below about code-page-sensitive characters for information about coding the number sign (#).)

DOCUMENT-TYPE-DECLARATION
Signals a document type declaration. Document type declarations begin with the character sequence '<!DOCTYPE' and end with a right angle bracket ('>') character; See XML specification for XML programming rules that can be in between. (Also see the related reference below about code-page-sensitive characters for information about coding the exclamation point (!).) For this event, XML text contains the entire declaration, including the opening and closing character sequences. This is the only event for which XML text includes the delimiters.

ENCODING-DECLARATION
Signals the optional encoding declaration located within the XML declaration for XML. XML text contains the encoding value.

END-OF-CDATA-SECTION
Occurs when the parser recognizes the end of a CDATA section. (See the related reference below about code-page-sensitive characters for information about coding the right square bracket (]).)

END-OF-DOCUMENT
Signals that document parsing has completed.

END-OF-ELEMENT
Occurs one time for each element end-tag or empty element tag when the parser recognizes the closing angle bracket of the tag. XML text contains the element name.

EXCEPTION
Signals that error in processing the XML document is detected. For encoding conflict exceptions, which are signaled before parsing begins, XML-TEXT (for XML documents in an alphanumeric data item) or XML-NTEXT (for XML documents in a national data item) either is zero length or contains only the encoding declaration value from the document.

PROCESSING-INSTRUCTION-DATA
Signifies the data that follows the PI target, up to but not including the PI closing character sequence, '?>'. XML text contains the PI data, which includes trailing, but not leading, white-space characters.

PROCESSING-INSTRUCTION-TARGET
Occurs when the parser recognizes the name that follows the opening character sequence, '<?', of a processing instruction (PI). PIs allow XML documents to contain special instructions for applications.

STANDALONE-DECLARATION
XML declaration for the optional standalone= parameter. XML text contains the standalone value.

START-OF-CDATA-SECTION
Start of a CDATA section. CDATA sections begin with the string '<![CDATA[' and end with the string ']]>'. Such sections are used to "escape" blocks of text that contain characters that would otherwise be recognized as XML markup. XML text always contains the opening character sequence '<![CDATA['. The parser passes the content of a CDATA section between these delimiters as a single CONTENT-CHARACTERS event. (See the related reference below about code-page-sensitive characters for information about coding the exclamation point (!) and left square bracket ([).)

START-OF-DOCUMENT
The beginning of the parsing of the document. XML text is the entire document, including any line-control characters such as LF (Line Feed) or NL (New Line).

START-OF-ELEMENT
Start element tag or empty element tag. XML text is set to the element name.

UNKNOWN-REFERENCE-IN-ATTRIBUTE
Attribute values for entity references other than the five predefined entity references, as shown for ATTRIBUTE-CHARACTER above.

UNKNOWN-REFERENCE-IN-CONTENT
Element content for entity references other than the predefined entity references, as shown for CONTENT-CHARACTER above.

VERSION-INFORMATION
XML declaration for the version information. XML text contains the version value. An XML declaration is XML text that specifies the version of XML that is used and the encoding of the document.

XML PARSE example

JCMUXML2 job located in SYS1.SFSYJCLS demonstrates the use of %XML PARSE. The input to this job is the output XML document created by JCMUXML0 or JCMUXML1 also located in SYS1.SFSYJCLS.

The source for the program below is in the JCMUXML2 job.

```
*****************************************************************
* XML parser demo program.                                      *
*****************************************************************
* EASYTRAN: IOMODE DYNAM
* EASYTRAN: CAPS=OFF
* EASYTRAN: DEBUG (LIST COBOL ESPI-FULL)
```

```
* END-EASYTRAN

FILE FILEIN V (4096)
FILE FILEOUT F (80)
COMPANY * 2 A HEADING ('COMPANY')
BRANCH * 3 A HEADING ('BRANCH')
OFFICER * 4 A HEADING ('OFFICER')
WAGE * 8 N 2 HEADING ('WAGE')
RATE * 5 N 3 HEADING ('RATE')

JOB INPUT NULL
INITIALIZE FILEOUT
*
* Note: change A00-PARSE-XML below to A00-TRACE-XML to trace errors.
%XML PARSE FILEIN +
      EVENT EXIT A00-PARSE-XML

IF XML-CODE NE ZERO
   DISPLAY 'Error in XML document - JOB terminated'
   RETURN-CODE = 16
   STOP EXECUTE
END-IF
STOP

* This paragraph traces XML events. Use it to find errors.
A00-TRACE-XML. PROC
   %XML DEBUG XML-TEXT
END-PROC

* This paragraph extracts useful fields found in the XML document.
A00-PARSE-XML. PROC
   DEFINE CURRENT-ELEMENT W 30 A
   DEFINE CURRENT-ATTRIBUTE W 30 A
   CASE XML-EVENT
        WHEN 'ATTRIBUTE-NAME'
           MOVE XML-TEXT TO CURRENT-ATTRIBUTE
        WHEN 'ATTRIBUTE-CHARACTERS'
           CASE CURRENT-ATTRIBUTE
              WHEN 'COMPANY'
*               DISPLAY 'COMPANY = ' XML-TEXT
                COMPANY = XML-TEXT
              WHEN 'BRANCH'
*               DISPLAY 'BRANCH = ' XML-TEXT
                BRANCH = XML-TEXT
              WHEN 'OFFICER'
*               DISPLAY 'OFFICER = ' XML-TEXT
                OFFICER = XML-TEXT
           END-CASE
           MOVE SPACES TO CURRENT-ATTRIBUTE

        WHEN 'START-OF-ELEMENT'
          MOVE XML-TEXT TO CURRENT-ELEMENT

        WHEN 'CONTENT-CHARACTERS'
          CASE CURRENT-ELEMENT
             WHEN 'OFFICER_NUMBER'
*              DISPLAY 'OFFICER = ' XML-TEXT
               OFFICER = XML-TEXT
             WHEN 'WAGE'
*              DISPLAY 'WAGE = ' XML-TEXT
               WAGE = FUNCTION numval-c(XML-TEXT)
* when positive signed number, OR zone to F's '
               IF WAGE GE ZERO
                  WAGE = WAGE OR '00000000'
               END-IF
             WHEN 'RATE'
*              DISPLAY 'RATE = ' XML-TEXT
               RATE = FUNCTION numval-c(XML-TEXT)
* when positive signed number, OR zone to F's '
               IF RATE GE ZERO
                  RATE = RATE OR '00000'
               END-IF
               PUT FILEOUT
               INITIALIZE WAGE
               INITIALIZE RATE
          END-CASE

       WHEN 'END-OF-ELEMENT'
          MOVE SPACES TO CURRENT-ELEMENT
       OTHERWISE
          CONTINUE
```

```
   END-CASE
END-PROC
```

Publishing XML documents and reports to z/OS Server

Follow these instructions to publish a report or XML document to z/OS UNIX.

To publish a report or XML document to z/OS UNIX

Procedure

1. Define a PRINTER file as a SERVER in your **Easytrieve Plus/IMU** program.
2. Use the defined file as PRINTER on the REPORT statement.
3. Code the printer file DD statement in the JCL as a new flat file. The DCB information in the JCL is not needed.
4. Define UNIX files in the JCL as described in “Defining UNIX files in the JCL” on page 17.

Example

Example of printer definition in the program:

```
FILE REPORT1 PRINTER (132) SERVER
```

Example of a REPORT statement:

```
REPORT RPT1 PRINTER REPORT1 XML
```

Example of printer DD statement in the JCL:

```
//REPORT1 DD DSN=&REPORT1,
// DISP=(NEW,CATLG,CATLG),
// SPACE=(TRK,(15,5),RLSE)
```

When REPORT is creating an XML document, the XML document is written as a Variable Blocked (VB) file with the maximum record length of 4096 bytes. The print control character is ignored. The DCB information in the JCL is ignored.

When REPORT is creating a standard report, the report is written as a standard printer file with file attributes as per the PRINTER file definition in the program and/or the DCB information in the JCL.

Defining UNIX files in the JCL

To use the z/OS UNIX environment the z/OS Internet server must be activated on the z/OS system. A root directory on the UNIX system must be established for each user. For more information on the UNIX environment requirements, consult your z/OS System administrator.

The JCMUXML0 job in the SYS1.SFSYJCLS IMU library demonstrates the requirements for publishing documents to the z/OS Server.

UNIX files are handled by the **Fsyunix1 Migration Utility** program. This program is dynamically loaded at the end of the job for each document. Fsyunix1 invokes BPXBATCH which performs the publishing to z/OS UNIX.

Code the DDnames as shown below when you want to write documents directly into an HFS (UNIX Directory) on the z/OS UNIX system.

Note: UNIX is case-sensitive; that is, commands, directories, and file names must by typed exactly as shown.

Migration Utility checks the JCL for the FJUNIX0 DDname. If FJUNIX0 exists, Migration Utility assumes that the documents are being written directly into the z/OS UNIX System.

The following DDnames are required when writing documents directly into the z/OS UNIX system.

Note: In the examples shown, assume that the root directory is `/u/migutil/user01`.

FJCONFG
The UNIX system configuration file used to determine the code set of each file type (ASCII or EBCDIC).

Migration Utility uses the code set for the file types found in the `httpd.conf` file. In this way, the XML documents are always in sync with the UNIX standards on your z/OS system. See your UNIX system administrator for the location of the `httpd.conf` file.

Example:

```
//FJCONFG DD PATHOPTS=(ORDONLY),
// PATH='/u/vagen1/httpd.conf'
```

FJDMAP0
Log of directories and files created on the UNIX system. This is a standard SYSOUT file.

FJUNIX0
The output directory on the z/OS UNIX System where files are to be written. All documents are written to this `FJUNIX0 DDname`. Note that **PATH=** must point to your root directory.

Example:

```
//FJUNIX0 DD PATHOPTS=(ORDONLY),
// PATH='/u/migutil/user01'
```

The printer DDname specified on the REPORT statement becomes the directory name, followed by the report sequence number (that is, the first report in the program would be `r001`) and the report DDname becomes a filename with the extension XML.

The directory is created as a new directory. If the directory already exists, the job is abnormally terminated.

Example: Assuming **PATH='/u/migutil/user01'**, for printer `DDname REPORT1`, the following UNIX commands are issued by IMU:

```
REPLACE /u/migutil/user01/REPORT1
SH rm -r /u/migutil/user01/REPORT1
SH mkdir /u/migutil/user01/REPORT1
SH mkdir /u/migutil/user01/REPORT1/r001
ALLOC FJUNIX1 PATH=/u/migutil/user01/REPORT1/r001/REPORT1.XML
```

To reuse a directory, execute the BPXBATCH program before the application step as follows:

```
//BPXBATCH EXEC PGM=BPXBATCH,
// PARM='SH rm -r /u/migutil/user01/htmlfil1'
```

CAUTION: The above statements delete the specified directory and all sub-directories within it. It will not give you a second chance.

STDOUT
The BPXBATCH program `stdout` file. This is an optional file. Point **PATH=** to your own directory.

Example:

```
//STDOUT DD PATH='/u/migutil/user01/fsyunix1.out',
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=(SIRWXU,SIXGRP)
```

STDERR
The BPXBATCH program `stderr` file. This is an optional file. Point **PATH=** to your own directory.

Example:

```
//STDERR DD PATH='/u/migutil/user01/fsyunix1.err',
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=(SIRWXU,SIXGRP)
```

Generating JSON documents

Using the JSON option on the REPORT statement

The REPORT statement JSON option provides users with a capability to generate JSON documents/ reports.

Syntax

```
REPORT PRINTER &REPORT. . . . JSON [ (version="&vers")]
```

Parameters

&REPORT

REPORT file DDname as defined by the FILE statement. The &REPORT file must be defined as a printer file or a printer file of type SERVER.

&vers

JSON Version. This information is not used at this time.

Special rules

The JSON document is written to the specified printer file.

To publish the JSON document to z/OS UNIX, define the PRINTER file as a SERVER file and provide printer file DD statement in the JCL. The DCB information in the JCL is not needed.

Define example:

```
FILE REPORT1 PRINTER (132) SERVER
```

Note: See “Publishing JSON documents and reports to a z/OS Server” on page 23 for details.

When &REPORT is defined as a SERVER, the JSON document is written as a Variable Blocked (VB) file with the maximum record length of 4096 bytes.

- The print control character is omitted.
- The attributes are automatically forced by IMU.
- The DCB information provided in the JCL is ignored.

When &REPORT is **not** defined as a SERVER, the JSON document is written as a standard report file.

All REPORT statement spacing and positioning options such as NOADJUST, COL, SKIP, ETC., serve no purpose and are ignored regardless of the document destination.

The JSON document is generated from fields defined on the CONTROL and LINE statements. A hierarchical structure is constructed from the CONTROL and the LINE fields. The CONTROL fields become group (parent) items and the LINE fields become the lowest level (child) elements.

- The TITLE statements are ignored.
- The Control break totals are ignored.

If the SUMMARY option is specified, the lowest level elements become the fields that would have been printed for the lowest level control break.

The SUMFILE option is supported as in a non-JSON environment.

The tags (field names) in the JSON Script are the field names specified on report LINE statement.

The JSON document is written using PRINT statement in the Activity section and optionally DISPLAY statements in the report exits.

The DISPLAY statement fields are not formatted as JSON Document elements, however.

The DISPLAY is written exactly as for a non-JSON report, however if writing to a SERVER, the print control characters are ignored.

In JSON report, the report SEQUENCE statement functions as it does with a non-JSON report. A temporary spool file is created of all fields; the spool file is sorted and the document is printed.

Report exits are invoked as for non-JSON reports. However, the ENDPAGE exit is ignored because there is no page concept in an JSON document.

If a PRINTER FILE is specified and defined with a MODIFY EXIT, each physical record being written to the printer file is passed to the exit. This is the same as for non-JSON reports.

Using the %JSON macro for parsing JSON files or hard coded JSON Script

The JSON macro generates logic for parsing JSON documents using the JSON PARSE statement supported by z/OS COBOL 6.3 and later versions. Users who intend to use this macro must be fully familiar with the JSON document syntax and the JSON document structure.

Note:

Describing the format and structure of JSON documents is beyond the scope of this manual. Users not familiar with JSON should learn from the publicly available JSON publications. JSON PARSE, provided by COBOL, is documented in the *Enterprise COBOL for z/OS Programming Guide*, including the use of the national and non-national characters.

JSON macro: coding rules

Purpose: Parse an JSON format document into individual elements using the COBOL JSON PARSE statement.

Format 1: parsing a JSON string

```
%JSON PARSE &object INTO &target +
        [WITH DETAIL] +
        [NAME OF &name is &json_name +
                  .
                  .
                   &name_n is &json_name_n] +
        [SUPPRESS  &supp_name +
                  .
                  .
                   &supp_name_n] +
        [EXIT &exit] +
        END-JSON
```

Where:

&object
File name that contains JSON format document.

The file organization must be a sequential or VSAM Sequential file.

OR

A defined W/S field name that contains JSON format document.

&target
The target group field defined as W/S field. The elementary items within the group must be defined as NATIONAL type (Refer to “Program examples” on page 26). The group field must have the same hierarchy as the JSON Script with field names exactly as found in the JSON Script.

&name_1..&name_n
Field names defined in the &target group item to be matched to the field names in the JSON Script.

&json_name . . is &json_name_n
Field names in the JSON Script to be correlated to the &name in the &target object.

&supp_name . . is &supp_name_n
Field names in the JSON Script to be suppressed (ignored).

&exit
User Exit for off-loading JSON result. This must be a PROC in the Easytrieve Plus program. In the exit, the user must write logic to offload the derived JSON data to a hard media with DISPLAY or other means. If &exit is a file, no exit is taken, the data is written to the specified file.

The default is no exit.

CAUTION: Exit is required for JSON Script that contains array.

Format 1: processing logic

This macro extracts field values located in the JSON Script into the &target group items by field name.

When &object is a file name, the file is assumed to be in JSON format. The file organization must be a sequential or VSAM Sequential file.

When &object file is referenced in JSON macro, the &object file cannot be used for any other purpose in the same Easytrieve Plus JOB.

The following outlines the parsing steps when &object is a file:

1. The &object file is read to calculate the document size in bytes.
2. A buffer is dynamically allocated for the calculated size.
3. The &object file is loaded into the allocated buffer in ASCII format.
4. The COBOL JSON parser is invoked to parse the buffer.
5. The &target is populated by matching the field names in the &target definition to the field names in JSON Script.

When &object is a group field/buffer, the value in the group field is assumed to be in JSON format. The following outlines the parsing steps:

1. The COBOL JSON parser is invoked to parse the buffer (&object).
2. A buffer is dynamically allocated for the calculated &object size.
3. The &object is loaded into the allocated buffer in ASCII format.
4. The COBOL JSON parser is invoked to parse the buffer.
5. The &target is populated by matching the field names in the &target definition to the field names in the JSON Script.

Special JSON registers

Special registers allocated by the JSON parser are those required by COBOL. These register names cannot be defined in the user program. They are reserved for COBOL internal use. The registers are documented in the *Enterprise COBOL for z/OS, 6.3 Language Reference* and the *Enterprise COBOL for z/OS, 6.3 Programming Guide*.

Use JSON-CODE and JSON-STATUS to check JSON PARSE statement for a successful completion.

Refer to *Enterprise COBOL for z/OS, 6.3 Programming Guide* for additional information.

- JSON-CODE
- JSON-STATUS

The following registers are reserved for COBOL internal use:

- JSON-EVENT

- JSON-INFORMATION
- JSON-NAMESPACE
- JSON-NAMESPACE-PREFIX
- JSON-NNAMESPACE
- JSON-NNAMESPACE-PREFIX
- JSON-NTEXT
- JSON-SCHEMA
- JSON-TEXT
- JSON-DOC-SEQ
- JSON-DOC-SIZE
- JSON-DOC-HEADER

JCL (JOBs) for running JSON programs

The following JCL (JOBs) are provided in SYS1.SFSYJCLS library. Copy and customize these JOBs for your needs.

JCMUJSN0.jcl
Create JSON documents to a flat file on z/OS

JCMUJSN1.jcl
Create and publish JSON documents to z/OS Unix Server

JCMUJSN2.jcl
Parse JSON document created by JCMUJSON0

JCMUJSN3.jcl
Parse JSON document hard coded in the program

Building data structure in Easytrieve Plus program for JSON PARSE

JSON PARSE statement requires that a data structure is defined in W or S memory locations for offloading parsed JSON Script elements (fields). The fields in the defined target must be defined in the same hierarchy as in the JSON Script.

The field names in the defined data structure must match the same field names as found in the JSON Script. This is a challenging problem as JSON Scripts are hard to decipher.

To combat this problem, IMU provides JCMUJUT0 （FSYJSNU0）utility that generates the data structure from the JSON Script files.

The utility generates Easytrieve Plus layout that maps JSON Script field names and hierarchy. The utility produces an output file that can be customized and copied into your Easytrieve Plus program (hard copy or as a macro).

Running JCMUJUT0 Utility

Make a copy of JCMUJUT0 job and customize it for your environment. JCMUJUT0 is located in SYS1.SFSYJCLS IMU product library.

```
PARM= statement syntax
PARM= (PUNCH,&arg2,&arg3,&arg4,&arg5)
    Where:
        &arg1        PUNCH.  This is a required option (do not change it).
        &arg2        DECIMAL=PERIOD or DECIMAL=COMMAD
        &arg3        Currency symbol. Use one character such as ‘$’.
        &arg4        CR=CR or CR=”-“
        &arg5        DR= or DR=DR
```

The default values are set for the USA standards.

```
PARM='PUNCH,DECIMAL=PERIOD,CURRENCY=$,CR=CR,DR='
```

Required files:

JSONFIL	This is your JSON Script file that you intend to parse. VB files are accepted only. The file can be a flat file or a PDS/PDSE member.
JSONOUT	This is the output layout in Easytrieve Plus macro format. The file is fixed length (LRECL=80). The file can be a flat file or a PDS/PDSE member.

Note: You can input one file at a time.

Note 1: The generated layout may need customizing. Inspect generated layout field names, field lengths and field masks to make sure that the layout accommodates your needs.

Note 2: The field names in the generated layout are those found in the JSON Script file. The field names must conform to the COBOL field naming conventions. JSON Script file may contain invalid field names. To combat this situation, FSYJSNU0 utility extracts field names up to the first space encountered. For example, “COMPAY 55” in JSON Script is generated as “COMPAY”. This is convenient for JSON Scripts with multiple nodes such as those generated by Migration Utility REPORT JSON statement option where CONTROL fields become JSON Script nodes.

Note 3: All field names in the generated layout must be unique. The script file must contain unique field names except those in an array format. JSON arrays are enclosed in square brackets […].

Note 4: When multiple arrays exist, the node that contains array is generated in the layout with OCCURS 1. JSON PARSE logic breaks up input JSON Script into multiple segments. Each segment contains one array to match the layout. Process loops until the entire JSON Script is processed. JSON PARSE must include an EXIT to be used for offloading extracted data.

Example:

Assuming that JSONFIL is pointing to JSON Script as generated in the “Program examples” on page 26 'Example 1: Generate JSON Script using REPORT &report JSON option’.

The generated layout is:

```
MACRO
* FSYJSNU0: JSON utility generated layout 09/27/2020, TIME: 16.43.59.
MACRO
* FSYJSNU0: JSON utility generated layout 09/28/2020, TIME: 05.19.04.
*-------------------------------------------------------------------*
DEFINE RPT1           W 034 A
DEFINE    CTL-COMPANY    RPT1           +0000 034 NATIONAL
DEFINE       CTL-BRANCH     CTL-COMPANY    +0000 034  +
                            NATIONAL OCCURS 00001
DEFINE          COMPANY        CTL-BRANCH     +0000 002 NATIONAL
DEFINE          BRANCH         CTL-BRANCH     +0002 003 NATIONAL
DEFINE          OFFICER        CTL-BRANCH     +0005 004 NATIONAL
DEFINE          WAGE           CTL-BRANCH     +0009 010 NATIONAL
DEFINE          RATE           CTL-BRANCH     +0019 006 NATIONAL
DEFINE          WBONUS         CTL-BRANCH     +0025 009
NATIONAL
MEND
```

Publishing JSON documents and reports to a z/OS Server

Follow these instructions to publish a report or JSON document to z/OS UNIX.

To publish a report or JSON document to z/OS UNIX

Procedure

1. Define a PRINTER file as a SERVER in your **Easytrieve Plus/IMU** program.
2. Use the defined file as PRINTER on the REPORT statement.
3. Code the printer file DD statement in the JCL as a new flat file. The DCB information in the JCL is not needed.
4. Define UNIX files in the JCL as described in "Defining UNIX files in the JCL" on page 24.

Example

Example of printer definition in the program:

```
FILE REPORT1 PRINTER (132) SERVER
```

Example of a REPORT statement:

```
REPORT RPT1 PRINTER REPORT1 JSON
```

Example of printer DD statement in the JCL:

```
//REPORT1 DD DSN=&REPORT1,
// DISP=(NEW,CATLG,CATLG),
// SPACE=(TRK,(15,5),RLSE)
```

When REPORT is creating an JSON document, the JSON document is written as a Variable Blocked (VB) file with the maximum record length of 4096 bytes. The print control character is ignored. The DCB information in the JCL is ignored.

When REPORT is creating a standard report, the report is written as a standard printer file with file attributes as per the PRINTER file definition in the program and/or the DCB information in the JCL.

Defining UNIX files in the JCL

To use the z/OS UNIX environment the z/OS Internet server must be activated on the z/OS system. A root directory on the UNIX system must be established for each user. For more information on the UNIX environment requirements, consult your z/OS System administrator.

The JCMUJSON0 job in the SYS1.SFSYJCLS IMU library demonstrates the requirements for publishing documents to the z/OS Server.

UNIX files are handled by the **FSYUNIX1 Migration Utility** program. This program is dynamically loaded at the end of the job for each document. FSYUNIX1 invokes BPXBATCH which performs the publishing to z/OS UNIX.

Code the DDnames as shown below when you want to write documents directly into an HFS (UNIX Directory) on the z/OS UNIX system.

Note: UNIX is case-sensitive; that is, commands, directories, and file names must by typed exactly as shown.

Migration Utility checks the JCL for the FJUNIX0 DDname. If FJUNIX0 exists, Migration Utility assumes that the documents are being written directly into the z/OS UNIX System.

The following DDnames are required when writing documents directly into the z/OS UNIX system.

Note: In the examples shown, assume that the root directory is /u/migutil/userid1.

FJCONFG

The UNIX system configuration file used to determine the code set of each file type (ASCII or EBCDIC). See your UNIX system administrator about the location of the httpd.conf file.

Migration Utility uses the code set for the file types found in the httpd.conf file. In this way, the JSON documents are always in sync with the UNIX standards on your z/OS system.

Example:

```
//FJCONFG DD PATHOPTS=(ORDONLY),
// PATH==/u/vagen1/httpd.conf=
```

FJDMAP0
Log of directories and files created on the UNIX system. This is a standard SYSOUT file.

FJUNIX0
The output directory on the z/OS UNIX System where files are to be written. All documents are written to this FJUNIX0 DDname. Note that **PATH=** must point to your root directory.

Example:

```
//FJUNIX0 DD PATHOPTS=(ORDONLY),
// PATH==/u/migutil/userid1=
```

The printer DDname specified on the REPORT statement becomes the directory name, followed by the report sequence number (that is, the first report in the program would be r001) and the report DDname becomes a filename with the extension JSON.

The directory is created as a new directory. If the directory already exists, the job is abnormally terminated.

Example: Assuming **PATH==/u/migutil/userid1=**, for printer DDname REPORT1, the following UNIX commands are issued by IMU:

```
REPLACE /u/migutil/userid1/REPORT1
SH rm -r /u/migutil/userid1/REPORT1
SH mkdir /u/migutil/userid1/REPORT1
SH mkdir /u/migutil/userid1/REPORT1/r001
ALLOC FJUNIX1 PATH=/u/migutil/userid1/REPORT1/r001/REPORT1.JSON
```

To reuse a directory, execute the BPXBATCH program before the application step as follows:

```
//BPXBATCH EXEC PGM=BPXBATCH,
// PARM==SH rm -r /u/migutil/userid1/htmlfil1=
```

CAUTION: The above statements delete the specified directory and all sub-directories within it. It will not give you a second chance.

STDOUT
The BPXBATCH program stdout file. This is an optional file. Point **PATH=** to your own directory.

Example:

```
//STDOUT DD PATH==/u/migutil/userid1/fsyunix1.out=,
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=(SIRWXU,SIXGRP)
```

STDERR
The BPXBATCH program stderr file. This is an optional file. Point **PATH=** to your own directory.

Example:

```
//STDERR DD PATH==/u/migutil/userid1/fsyunix1.err=,
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=(SIRWXU,SIXGRP)
```

Program examples and error messages

Messages

IMU Translator messages are documented in the IBM Migration Utility for z/OS Version 5 Release 1 *Installation, User's Guide and Reference*.

JSON-CODE and JSON-STATUS codes are described in the *Enterprise COBOL for z/OS, 6.3 Programming Guide*.

Program examples

Example 1: Generate JSON Script using REPORT JSON option

This program demonstrates how to generate JSON Script using the REPORTStatement.

The source is in SYS1.SFSYJCLS(JCMUJSON0) job.

Easytrieve Plus Program:

```
* EASYTRAN: DEBUG (BLIST COBOL LKED ESPI-FULL)
* END-EASYTRAN
FILE REPORT1 PRINTER VB (136 0)

FILE FILEIN DISK F (80)
COMPANY     1   2 A   HEADING ('COMPANY')
BRANCH      3   3 A   HEADING ('BRANCH')
OFFICER     6   4 A   HEADING ('OFFICER')
WAGE       10  08 N 2 HEADING ('WAGE')
RATE       18  05 N 3 HEADING ('RATE') MASK 'ZZ.999'

WBONUS   W   5 P 2
WCOUNT   W   4 B

JOB INPUT FILEIN
WBONUS = (WAGE * RATE / 100)
PRINT RPT1

REPORT RPT1 JSON PRINTER REPORT1 LINESIZE 80
CONTROL  COMPANY BRANCH
TITLE 1  'EXECUTIVE BONUS DETAIL REPORT'
LINE  1   OFFICER WAGE RATE WBONUS
```

Input file

The input file FILEIN to this job is SYS1.SFSYEZTS(TESTFIL0).

Produced JSON document

```
{
"RPT1": {
      "CTL-COMPANY 10":{
         "CTL-BRANCH 001":[
           {
            "COMPANY":"10",
            "BRANCH":"001",
            "OFFICER":"AAAA",
            "WAGE":"55,000.00 ",
            "RATE":"10.500",
            "WBONUS":"5,775.00 "
           },
           {
            "COMPANY":"10",
            "BRANCH":"001",
            "OFFICER":"BBBB",
            "WAGE":"55,000.00 ",
            "RATE":"10.500",
            "WBONUS":"5,775.00 "
           },
           {
            "COMPANY":"10",
            "BRANCH":"001",
            "OFFICER":"CCCC",
            "WAGE":"55,000.00 ",
            "RATE":"10.500",
            "WBONUS":"5,775.00 "
           },
           {
            "COMPANY":"10",
            "BRANCH":"001",
            "OFFICER":"DDDD",
            "WAGE":"55,000.00 ",
            "RATE":"10.500",
            "WBONUS":"5,775.00 "
```

```
           },
           {
            "COMPANY":"10",
            "BRANCH":"001",
            "OFFICER":"EEEE",
            "WAGE":"55,000.00 ",
            "RATE":"10.500",
            "WBONUS":"5,775.00 "
           }
        ]
     }
  }
}
```

Example 2: PARSE field information from JSON Script file using JSON PARSE

This program demonstrates the use of `%JSON PARSE &file` statement.

The source is in `SYS1.SFSYJCLS(JCMUJSON2)` job.

```
* EASYTRAN: CAPS=OFF
* EASYTRAN: DEBUG (LIST BLIST COBOL ESPI-FULL)
* END-EASYTRAN

FILE FILEIN V (84)
FILLER 1 80 A

DEFINE client-data  S  133 A
DEFINE    account-num  client-data   +0 12 A JMASK ('999999999999')
DEFINE    balance      client-data  +12 10 A JMASK ('$$$$9.99CR')
DEFINE    billing-info client-data  +22 110  NATIONAL
DEFINE       name-first   billing-info    +00  20 NATIONAL
DEFINE       name-last    billing-info    +20  20 NATIONAL
DEFINE       addr-street  billing-info    +40  20 NATIONAL
DEFINE       addr-city    billing-info    +60  20 NATIONAL
DEFINE       addr-region  billing-info    +80  20 NATIONAL
DEFINE       addr-code    billing-info   +100  10 NATIONAL

DEFINE FIELDA W 5 A

JOB INPUT NULL
%JSON PARSE FILEIN INTO client-data +
      with detail +
end-json
DISPLAY '============================================================'
DISPLAY 'Error JSON-CODE=' JSON-CODE
DISPLAY 'Error JSON-STATUS=' JSON-STATUS

DISPLAY '============================================================'
DISPLAY 'Account: ' account-num
DISPLAY 'Balance: ' balance
DISPLAY 'Client Information:'
DISPLAY '    Name last: ' name-last
DISPLAY '   Name-first: ' name-first
DISPLAY ' Address:'
DISPLAY '   addr-street: '  addr-street
DISPLAY '   addr-city--: '  addr-city
DISPLAY '   addr-region: '  addr-region
DISPLAY '   addr-code--: '  addr-code
DISPLAY '============================================================'
STOP
```

Input file

The input file `FILEIN` to this job is `SYS1.SFSYDOCS(FSYJTST2)`.

Produced output

```
============================================================
Error JSON-CODE=        0
Error JSON-STATUS=        0
============================================================
Account: 123456789012
Balance:  $125.53CR
Client Information:
            Name last: Smith
           Name-first: John
         Address:
```

```
           addr-street: 12345 First Avenue
           addr-city--: New York
           addr-region: New York
           addr-code--: 10203
=======================================================
```

Example 3: Extract field information from hard coded JSON Script file in the program source

This program demonstrates the use of %JSON PARSE using defined JSON Script in the program.

The source is in SYS1.SFSYJCLS(JCMUJSON3) job.

```
* EASYTRAN: PROCESS OPTIMIZE(0)
* EASYTRAN: IOMODE DYNAM
* EASYTRAN: CAPS=OFF
* EASYTRAN: DEBUG (LIST BLIST COBOL ESPI-FULL)
* END-EASYTRAN

DEFINE jtxt-1047-client-data W  300 A  +
       VALUE '{"client-data":{ +
               "account-num":123456789012, +
               "balance":-125.53, +
               "billing-info":{ +
               "name-firstx":"John", +
               "name-lastx":"Smith", +
               "addr-street":"12345 First Avenue", +
               "addr-city":"New York", +
               "addr-region":"New York", +
               "addr-code":"10203" +
                } +
              } +
           } '

DEFINE client-data  W  133 A
DEFINE    account-num  client-data   +0 12 A JMASK ('999999999999')
DEFINE    balance      client-data  +12 10 A JMASK ('$$$$9.99CR')
DEFINE    billing-info client-data  +22 110  NATIONAL
DEFINE       name-first   billing-info    +00  20 NATIONAL
DEFINE       name-last    billing-info    +20  20 NATIONAL
DEFINE       addr-street  billing-info    +40  20 NATIONAL
DEFINE       addr-city    billing-info    +60  20 NATIONAL
DEFINE       addr-region  billing-info    +80  20 NATIONAL
DEFINE       addr-code    billing-info   +100  10 NATIONAL

JOB INPUT NULL
%JSON PARSE jtxt-1047-client-data INTO client-data +
      with detail +
      name of name-first is 'name-firstx' +
              name-last is name-lastx   +
      suppress name-last +
               name-first +
end-json
DISPLAY '======================================================='
DISPLAY 'Error JSON-CODE=' JSON-CODE
DISPLAY 'Error JSON-STATUS=' JSON-STATUS

DISPLAY '======================================================='
DISPLAY 'Account: ' account-num
DISPLAY 'Balance: ' balance
DISPLAY 'Client Information:'
DISPLAY '    Name last: ' name-last
DISPLAY '   Name-first: ' name-first
DISPLAY ' Address:'
DISPLAY '   addr-street: '  addr-street
DISPLAY '   addr-city--: '  addr-city
DISPLAY '   addr-region: '  addr-region
DISPLAY '   addr-code--: '  addr-code
DISPLAY '======================================================='
STOP
```

Produced output

```
=======================================================
Error JSON-CODE=        0
Error JSON-STATUS=        0
=======================================================
Account: 123456789012
Balance:  $125.53CR
```

```
Client Information:
    Name last: Smith
   Name-first: John
 Address:
   addr-street: 12345 First Avenue
   addr-city--: New York
   addr-region: New York
   addr-code--: 10203
=========================================================
```

Example 4: PARSE field information from JSON Script file that contains arrays using JSON PARSE EXIT option

```
*-------------------------------------------------------------------*
* JSON parser demo program for distribution. JSON PARSE using a file. *
*-------------------------------------------------------------------*
* EASYTRAN: CAPS=OFF
* EASYTRAN: DEBUG (LIST BLIST COBOL ESPI-FULL)
* END-EASYTRAN

FILE FILEIN V (84)
I-RECORD  80 A

DEFINE RPT1           W 034 A
DEFINE    CTL-COMPANY    RPT1           +0000 034 NATIONAL
DEFINE       CTL-BRANCH     CTL-COMPANY    +0000 034  +
                            NATIONAL OCCURS 00001
DEFINE          COMPANY        CTL-BRANCH     +0000 002 NATIONAL
DEFINE          BRANCH         CTL-BRANCH     +0002 003 NATIONAL
DEFINE          OFFICER        CTL-BRANCH     +0005 004 NATIONAL
DEFINE          WAGE           CTL-BRANCH     +0009 010 NATIONAL
DEFINE          RATE           CTL-BRANCH     +0019 006 NATIONAL
DEFINE          WBONUS         CTL-BRANCH     +0025 009 NATIONAL

JOB INPUT NULL
%JSON PARSE FILEIN INTO RPT1 +
      with detail +
      EXIT OFFLOAD-JSON-SCRIPT +
END-JSON

DISPLAY '========================================================='
DISPLAY 'Error JSON-CODE=' JSON-CODE
DISPLAY 'Error JSON-STATUS=' JSON-STATUS
IF (JSON-CODE NE 0)
   RETURN-CODE = JSON-CODE
END-IF
STOP

OFFLOAD-JSON-SCRIPT. PROC
   DEFINE SUB1 W 4 B
   DEFINE WMAX-OCCURS W 4 B VALUE 1
   SUB1 = 0
   DO WHILE (SUB1 LT WMAX-OCCURS)
      SUB1 = SUB1 + 1
      DISPLAY '========================================================='
      DISPLAY '   COMPANY '     COMPANY(SUB1)
      DISPLAY '   BRANCH  '     BRANCH(SUB1)
      DISPLAY '   OFFICER '     OFFICER(SUB1)
      DISPLAY '   WAGE    '     WAGE(SUB1)
      DISPLAY '   RATE    '     RATE(SUB1)
      DISPLAY '   WBONUS  '     WBONUS(SUB1)
   END-DO
END-PROC
```

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

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 Corporation
Mail Station P300
522 South Road
Poughkeepsie New York 12601-5400
U.S.A.

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.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information", http://www.ibm.com/legal/copytrade.shtml.

Other company, product, and service names may be trademarks or service marks of others.

Index

Special Characters

&exit 20
&json_name . . is &json_name_n 20
&name_1..&name_n 20
&object
    parsing JSON string 20
    parsing XML string 11
&proc
    parsing XML string 11
&proc PROC 12
&REPORT
    &encode 9
    &option 9
    &vers 9
    JSON
        &vers 19
    XML 9
&supp_name . . is &supp_name_n 20
&target
    parsing JSON string 20
&text 11
%JSON macro
    parsing JSON files 20
%XML DEBUG 12
%XML DEBUG &text 11
%XML macro
    parsing XML files 11
%XML PARSE
    SYS1.SFSYJCLS
        JCMUXML2 15
%XML PARSE &object EVENT EXIT &proc 11

B

BPXBATCH
    FSYUNIX1
        publishing JSON 24
        publishing XML 17
    STDERR
        publishing JSON 24
    STDOUT
        publishing JSON 24

C

COBOL JSON PARSE 20
COBOL XML PARSE 11
COBOL XML parser 12
COL
    JSON document 19
    XML document 9
comments on publication
    sending feedback vii
configuration
    FJCONFG
        publishing JSON 24
configuration *(continued)*
    FJCONFG *(continued)*
        publishing XML 17
CONTROL
    JSON document 19
    XML document 9

D

data structure 22
DCB
    JSON document 23
    XML document 17
DD
    JSON document 23
    XML document 17
DDnames
    publishing JSON 24
    publishing XML 17
debug
    XML PARSE errors 11
DISPLAY
    JSON document 19
    XML document 9

E

encode 9
ENDPAGE
    JSON document 19
    XML document 9
ETC
    JSON document 19
    XML document 9
EVENT 14
EVENT &proc 12
EXCEPTION 12

F

feedback
    email template vii
    sending reader comments vii
FJCONFG 17, 24
FJDMAP0
    publishing JSON 24
    publishing XML 17
FJUNIX0
    publishing JSON 24
    publishing XML 17
format notation
    description v
FSYUNIX1
    BPXBATCH
        publishing JSON 24
        publishing XML 17

H

HFS (UNIX Directory)
  publishing JSON 24
  publishing XML 17
httpd.conf
  publishing JSON 24
  publishing XML 17

J

JCL
  JSON 22, 24
  SYS1.SFSYJCLS
    JCMUJSN0.jcl 22
    JCMUJSN1.jcl 22
    JCMUJSN2.jcl 22
    JCMUJSN3.jcl 22
  XML 17
JCMUJSN0.jcl 22
JCMUJSN1.jcl 22
JCMUJSN2.jcl 22
JCMUJSN3.jcl 22
JCMUJSON0 24
JCMUJUT0 22
JCMUJUT0 (FSYJSNU0) 22
JCMUXML0 10, 15, 17
JCMUXML1 10, 15
JCMUXML2 10, 15
JSON macro
  parsing JSON files 20
JSON PARSE
  JCMUJUT0 22
  JSON-EVENT 21
  JSON-STATUS 21
  JSONFIL 22
  JSONOUT 22
json_name . . is &json_name_n 20
JSON-CODE 21, 25
JSON-DOC-HEADER 21
JSON-DOC-SEQ 21
JSON-DOC-SIZE 21
JSON-INFORMATION 21
JSON-NAMESPACE 21
JSON-NAMESPACE-PREFIX 21
JSON-NNAMESPACE 21
JSON-NNAMESPACE-PREFIX 21
JSON-NTEXT 21
JSON-SCHEMA 21
JSON-STATUS 21, 25
JSON-TEXT 21
JSONFIL 22
JSONOUT 22

L

license inquiry 31
LINE
  JSON document 19
  XML document 9
log
  FJDMAP0
    publishing JSON 24
log *(continued)*
  FJDMAP0 *(continued)*
    publishing XML 17

M

MODIFY EXIT
  JSON document 19
  XML document 9

N

name_1..&name_n 20
NOADJUST
  JSON document 19
  XML document 9
notation, description v

O

object
  parsing JSON string 20
  parsing XML string 11
option 9

P

parsing
  JSON files 20
  JSON string
    &exit 20
    &json_name . . is &json_name_n 20
    &name_1..&name_n 20
    &object 20
    &supp_name . . is &supp_name_n 20
    &target 20
  XML files 11
  XML string 11
PATH=
  publishing JSON 24
  publishing XML 17
PRINT
  JSON document 19
PRINTER
  JSON document 19, 23
  XML document 9, 17
PRINTER FILE
  JSON document 19
  XML document 9
proc
  parsing XML string 11
proc PROC 12
publishing
  JSON documents 23
  XML documents 17

R

r001
  publishing JSON 24
  publishing XML 17
railroad track format, how to read v
reader comments

reader comments *(continued)*
methods of sending feedback vii
register
JSON-DOC-HEADER 21
JSON-DOC-SEQ 21
JSON-DOC-SIZE 21
JSON-EVENT 21
JSON-INFORMATION 21
JSON-NAMESPACE 21
JSON-NAMESPACE-PREFIX 21
JSON-NNAMESPACE 21
JSON-NNAMESPACE-PREFIX 21
JSON-NTEXT 21
JSON-SCHEMA 21
JSON-TEXT 21
XML-CODE 12
XML-DOC-DISP1 12
XML-DOC-HEADER 12
XML-DOC-MEMP1 12
XML-DOC-SEQ 12
XML-DOC-SIZ21 12
XML-DOC-SIZE 12
XML-DOC-SIZE1 12
XML-DOC-TEXT1 12
XML-EVENT 12
XML-NTEXT 12
XML-TEXT 12
REPORT
&encode 9
&option 9
&vers 9
JSON
&vers 19
JSON document 23
XML 9
XML document 17
REPORT PRINTER &REPORT. . . . JSON 19
REPORT PRINTER &REPORT. . . . XML 9

S

SEQUENCE
JSON document 19
XML document 9
SERVER
JSON document 19
PRINTER
XML document 9
SKIP
JSON document 19
XML document 9
stacked items v
STDERR
publishing JSON 24
publishing XML 17
STDOUT
publishing JSON 24
publishing XML 17
SUMFILE
XML document 9
SUMMARY
JSON document 19
XML document 9
supp_name . . is &supp_name_n 20
syntax
notation, description v
SYS1.SFSYJCLS
JCMUJSN0.jcl 22
JCMUJSN1.jcl 22
JCMUJSN2.jcl 22
JCMUJSN3.jcl 22
JCMUXML0 10, 15
JCMUXML1 10, 15
JCMUXML2 10, 15
SYS1.SFSYJCLS IMU
JCMUJSON0 24
JCMUXML0 17
SYSOUT
publishing XML 17

T

target
parsing JSON string 20
technical problems
methods of resolving vii
TITLE
JSON document 19
XML document 9
Translator messages 25

U

UNIX Directory
HFS
publishing JSON 24
publishing XML 17

V

Variable Blocked
JSON document 19
XML document 9
vers
JSON version 19
XML version 9
VSAM Sequential file 21

W

W/S field 20

X

XML DEBUG 12
XML DEBUG &text 11
XML macro
parsing XML files 11
XML PARSE
SYS1.SFSYJCLS
JCMUXML2 15
XML PARSE &object EVENT EXIT &proc 11
XML-CODE 12
XML-DOC-DISP1 12
XML-DOC-HEADER 12
XML-DOC-MEMP1 12
XML-DOC-SEQ 12

XML-DOC-SIZ21 12
XML-DOC-SIZE 12
XML-DOC-SIZE1 12
XML-DOC-TEXT1 12
XML-EVENT
    ATTRIBUTE-CHARACTER 14
    ATTRIBUTE-CHARACTERS 14
    ATTRIBUTE-NAME 14
    ATTRIBUTE-NATIONAL-CHARACTER 14
    COMMENT 14
    CONTENT-CHARACTER 14
    CONTENT-CHARACTERS 14
    CONTENT-NATIONAL-CHARACTER 14
    DOCUMENT-TYPE-DECLARATION 14
    ENCODING-DECLARATION 14
    END-OF-CDATA-SECTION 14
    END-OF-DOCUMENT 14
    END-OF-ELEMENT 14
    EXCEPTION 14
    PROCESSING-INSTRUCTION-DATA 14
    PROCESSING-INSTRUCTION-TARGET 14
    STANDALONE-DECLARATION 14
    START-OF-CDATA-SECTION 14
    START-OF-DOCUMENT 14
    START-OF-ELEMENT 14
    UNKNOWN-REFERENCE-IN-ATTRIBUTE 14
    UNKNOWN-REFERENCE-IN-CONTENT 14
    VERSION-INFORMATION 14
XML-NTEXT 11, 12, 14
XML-TEXT 11, 12, 14

SC27-9083-00