UNISYS

ClearPath Enterprise Servers

WEBAPPSUPPORT
Application Programming Guide

ClearPath MCP 18.0

April 2017 3826 5286–007

Contents

Section 1. Introduction to Application Support

Section 2. WEBPCM Transaction Server to Internet Application Programming

Section 3. WEBAPPSUPPORT Library Interface

Section 4. XML Parser Administration

Section 5. Developing an XML Parser Application

Section 6. WEBAPPSUPPORT Library Interface for the XML Parser

Section 7. Using Sample Source Code for Parsing an XML Document

Section 8. Monitoring the XML Parser

Section 9. HTTP Client Applications

Figures

Tables

Section 1
Introduction to Application Support

This guide describes how to write applications that use the WEBAPPSUPPORT library for the following capabilities:

- Responding to HTTP requests received by MCP Web Transaction Server
- Processing XML documents with the XML Parser feature
- Making HTTP requests with the HTTP Client feature

This guide also explains how to install and administer the XML Parser for ClearPath MCP and provides information for writing applications that use the XML Parser.

Documentation Updates

This document contains all the information that was available at the time of publication. Changes identified after release of this document are included in problem list entry (PLE) 19153126. To obtain a copy of the PLE, contact your Unisys representative or access the current PLE from the Unisys Product Support website:

http://www.support.unisys.com/all/ple/19153126

Note: *If you are not logged into the Product Support site, you will be asked to do so.*

What's New?

New or Updated Information	Location
Modified TRACE and TRACEERRORS Commands	Section 3: WEBAPPSUPPORT Library Interface
Modified GET_XML_DOCUMENT procedure	Section 6: WEBAPPSUPPORT Library Interface for the XML Parser
Modified XML Mapping Structure	Section 6: WEBAPPSUPPORT Library Interface for the XML Parser
Modified GET_HTTP_RESPONSE_STATUS procedure	Section 9: HTTP Client Applications

MCP Web Enablement Application Programming Interfaces

Several Web enablement application programming interfaces (APIs) are available for the ClearPath MCP environment. Table 1–1 lists the primary interfaces and shows the application languages supported.

Table 1–1. MCP Web Enablement APIs and Supported Languages

API	COBOL 74	COBOL 85	ALGOL, NEWP	AB Suite, EAE
WebTS AAPI			X	
WEBPCM	X	X	X	
XMLParser		X	X	X
HTTP Client		X	X	X
Regular Expressions		X	X	X

Web Transaction Server

The Web Transaction Server is a highly scaleable, standards-based, high performance Hypertext Transfer Protocol (HTTP) Web server that runs in an extremely reliable MCP environment. It communicates with browsers using HTTP 1.1 over a TCP/IP network. It is written in native code, so it is not a portation.

The Web Transaction Server software incorporates Web server capabilities into a ClearPath enterprise server. It gives users the ability to access and distribute hypertext documents and hyperlinked multimedia information, including text, images, audio, video, and Java applets. A variety of client workstations can access the Web Transaction Server as a document repository or as a gateway to custom applications.

The Web Transaction Server provides a Web Transaction Server application programming interface (AAPI) that allows user-written applications in ALGOL, DMALGOL, DCALGOL, and NEWP to process HTTP requests from browser clients such as Microsoft Internet Explorer and Mozilla Firefox. Its purpose is to provide a high-performance, scaleable, and robust programming interface for the development of custom Web applications and application gateways.

The Web Transaction Server supports a direct application interface called AAPI, which uses a Connection Library interface. The applications must be written in ALGOL or NEWP.

Refer to the *Web Transaction Server for ClearPath MCP Administration and Programming Guide* for more information about this capability.

Figure 1–1 shows the flow of the applications receiving HTTP requests through the Web Transaction Server and then generating responses.

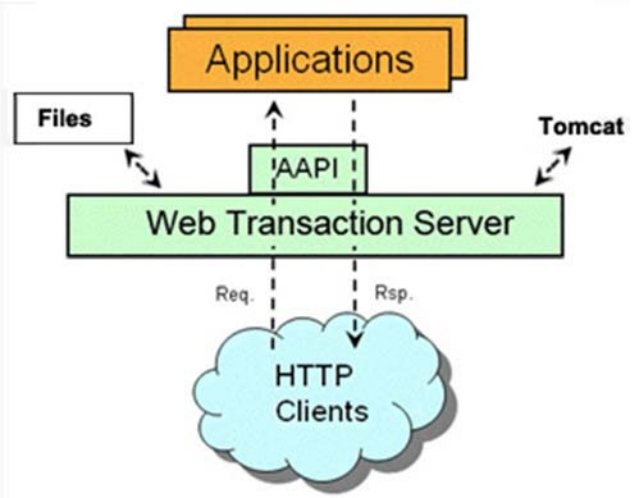

Figure 1–1. Flow of HTTP Requests/Responses using Web Transaction Server

WEBAPPSUPPORT Library APIs

The APIs of the WEBAPPSUPPORT library enable you to modernize COBOL, ALGOL, and AB Suite applications. Also, some miscellaneous capabilities are available with Web Transaction Server or one of the APIs described in this document. These miscellaneous capabilities are

- Merge Data: You can use this capability in your application to merge data into templates (HTML, XML, and so forth). See the MERGE_DATA, MERGE_FILE_AND_DATA, and MERGE_I18NFILE_AND_DATA procedures.
- UTF-8 Translation: You can use this capability in your applications to translate character sets to/from UTF-format. See the DECODE_UTF8 and ENCODE_UTF8 procedures.
- Binary64 encoding/decoding: You can use this capability in your applications to encode binary data to/from Binary64 so you can send binary data as text. See the DECODE_BINARY64 and ENCODE_BINARY64 procedures.
- Deflate/Inflate: You can use this capability in your applications to supply data in either an application array or MCP file to be compressed into or decompressed from the zlib or gzip formats, using the compression algorithm defined in the DEFLATE RFC 1951. This capability requires a Java Parser Module (JPM). See the DEFLATE_DATA and INFLATE_DATA procedures.

Figure 1–2 lists the APIs described in this document; it also includes the release in which the capability was introduced.

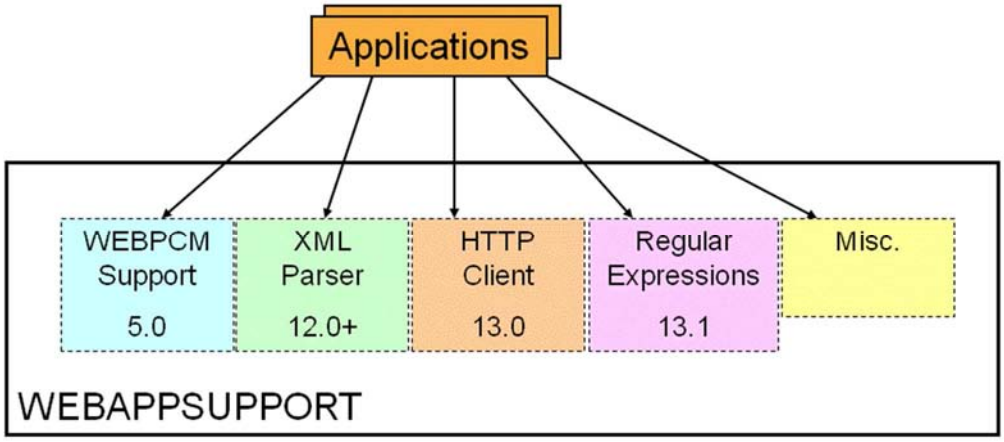

Figure 1–2. WEBAPPSUPPORT Library APIs

The subsequent topics in this section provide general overview information about the interfaces listed in Figure 1-2 except for the miscellaneous capabilities. Refer to the specific topics in the appropriate document for the miscellaneous capabilities.

WEBPCM HTTP Server Applications

WEBPCM Overview

WEBPCM is a programming environment for interfacing Transaction Server applications to Web Transaction Server so that those applications can process requests from HTTP (Web) users. WEBPCM supports programming languages such as COBOL and ALGOL.

WEBPCM comprises two software modules:

- A PCM (Protocol Converter Module) in the Custom Connect Facility (CCF) called the WEBPCM. This PCM routes requests from users using the AAPI interface in Web Transaction Server to Transaction Server applications through the CUCIPCM module of CCF.
- A support library called WEBAPPSUPPORT for accessing the HTTP request and constructing the HTTP response.

Two modules are used so that terminating CCF does not discontinue (DS) all applications linked into the WEBPCM.

Figure 1–3 provides a global view of the WEBPCM environment.

Note: *Not all CCF components are shown in this diagram; any components shown that are not WEBPCM components are those required for Web Enabler.*

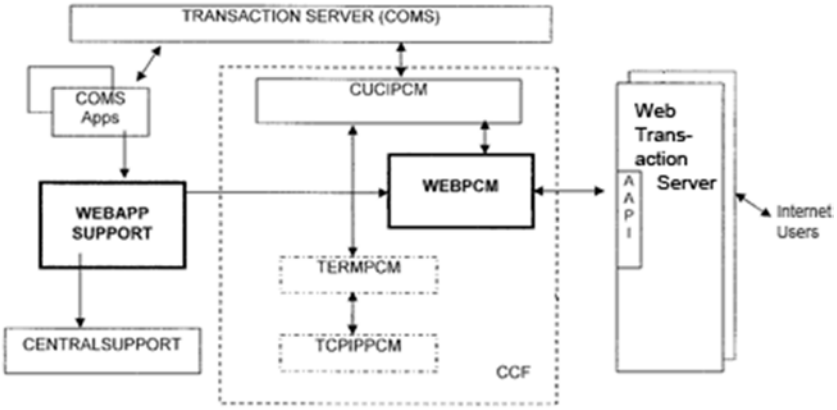

Figure 1–3. WEBPCM Environment (Global View)

As a gateway into Transaction Server, WEBPCM enables the following application types to interface to HTTP (intranet/Internet) users using normal Transaction Server mechanisms:

- Transaction Server Direct Window applications
- Transaction Server Remote File applications

An additional library, WEBAPPSUPPORT, is included with the WEBPCM for processing the Message Objects, which are messages that represent the request and the response.

Figure 1–4 provides a more detailed view of the WEBPCM environment.

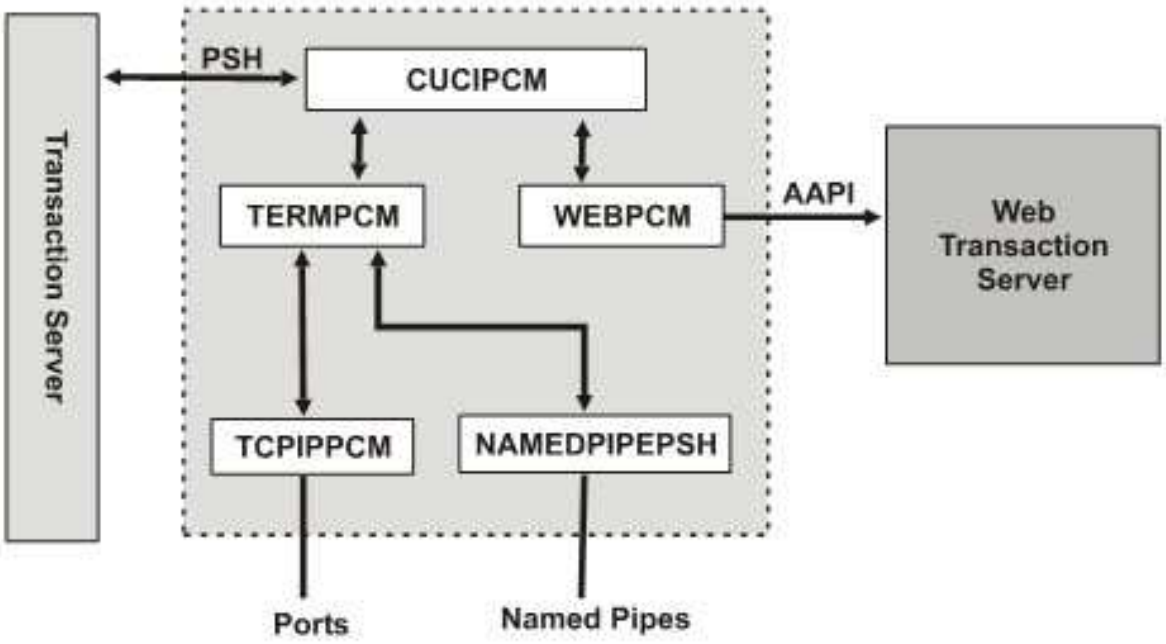

Figure 1–4. WEBPCM Environment (Detailed View)

This illustration shows some additional CCF modules that are typically used in CCF, and how the WEBPCM fits into the CCF structure.

All communication with Transaction Server is completed through the CUCIPCM, which uses the Transaction Server PSH (Protocol-Specific Handler) interface. Some PCMs, such as TCPIPPCM and NAMEDPIPEPSH, typically use terminal services provided by the TERMPCM before sending their messages to Transaction Server using CUCIPCM. The WEBPCM sends all messages directly to Transaction Server.

The WEBPCM interfaces to Web Transaction Server using the Web Transaction Server API (Application Programming Interface).

Without the WEBPCM in the Environment

Without the WEBPCM, the environment for your MCP applications to interface to the Transaction Server might appear as shown in Figure 1–5.

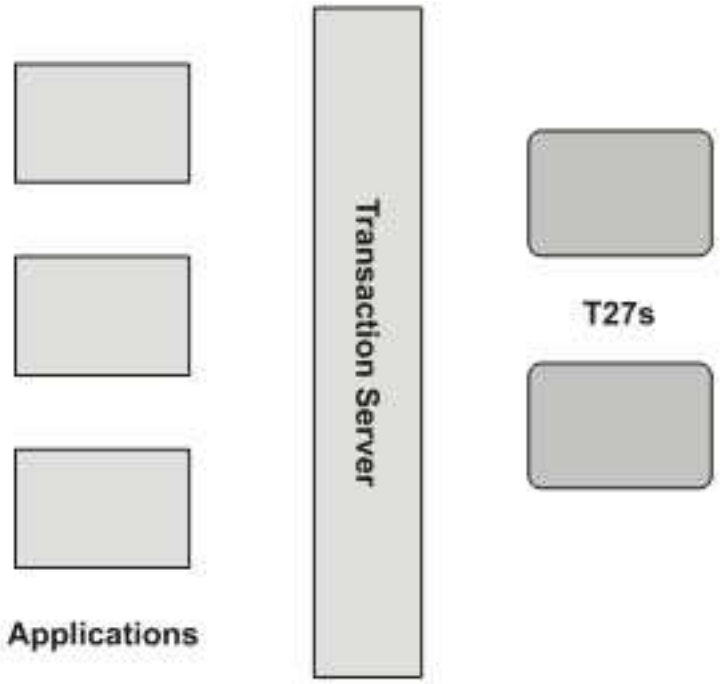

Figure 1–5. MCP Applications Without the WEBPCM

Although a number of components are not shown in this picture, the important point is that when MCP applications interface to Transaction Server, they expect the T27-compliant devices to act as the typical station.

With the WEBPCM in the Environment

With the WEBPCM in the environment, your application environment appears more like the layout shown in Figure 1–6.

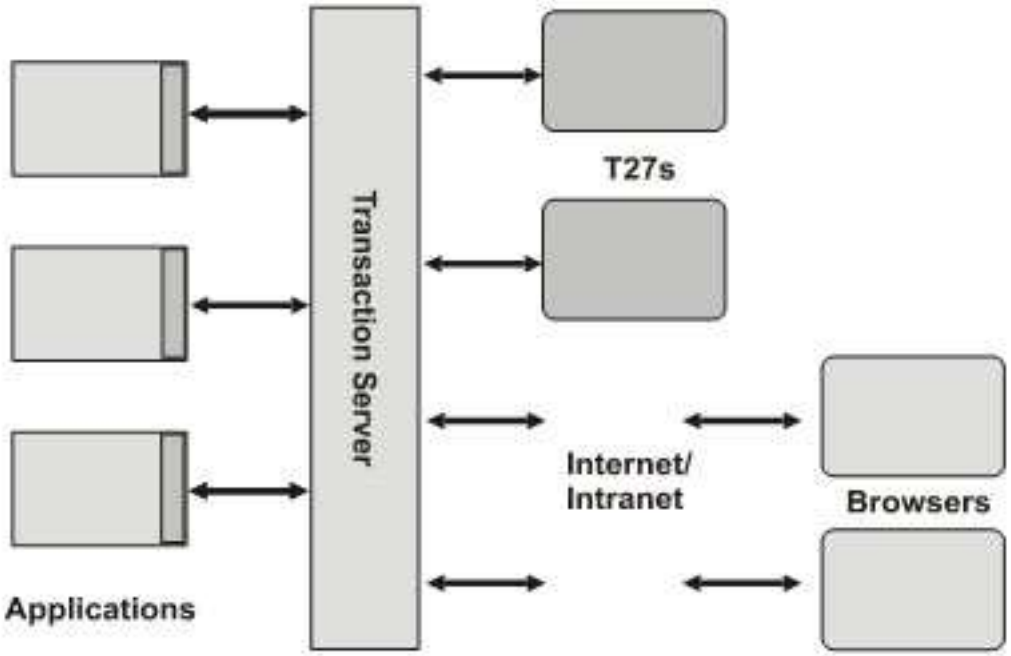

Figure 1–6. MCP Applications With the WEBPCM

In the environment shown in Figure 1–6, browser users have a normal Web page view, not a terminal emulator view. The MCP applications generate responses with graphics, links, sound, frames, and so forth.

The extra band of shading in the applications boxes shown in the figure indicates that changes are needed in the applications to interface to Web users.

This environment is a two-tier solution in which no Windows or UNIX programming or configuration is needed. The advantages of this type of environment are increased scalability, stability, and ease of maintenance.

Why Use the WEBPCM?

The WEBPCM offers you numerous benefits.

- It is easy to program.
- Because WEBPCM is processor efficient, it requires a minimal amount of processor to handle requests.
- It is scalable and capable of processing many requests simultaneously.

- WEBPCM supports Web (HTTP) access to COBOL, ALGOL, and other language Transaction Server applications. The application has full HTTP access with the ability to see all HTTP headers on input and to set any HTTP headers on output. This capability is important for controlling any HTTP caches, determining the browser level (that is, capability), and so forth.
- Most routers and firewalls allow port 80 to pass through. HTTP uses port 80 as its default port. Because opening other ports can be complicated and difficult to manage, the usage of port 80 as the default eliminates some of these problems .
- Samples are provided for the COBOL and ALGOL languages; however, other languages are also supported.
- Because the application is generating an HTML response, no further screen-scraping (mapping) is needed, which reduces system overhead/response time delays.
- The WEBPCM is included with all MCP systems, and no additional licensing fees (no per-seat licensing) are required.
- WEBPCM has no specific browser requirements. A wide range of browsers on different platforms work with applications using the WEBPCM.

Other aspects of using the WEBPCM include the following:

- The application can serve both Web users and non-Web users from the same application.
- Applications do not require major changes (modifications are minor). You can use the WEBPCM without modifying the application.
- Applications using either the Direct Window interface into Transaction Server or the Remote File interface do not need to implement a new interface to Web users (such as the direct interface into Web Transaction Server). The application can continue to use its current interface.
  - The source is easy to identify. The first 17 bytes of the input message contain text that the application can examine to determine the source. The Direct Window application can use the Transaction Server Trancode feature, which drives off of the initial text in the message.
  - Input and output processing items can be used to translate to and from Web input and output.
  - Code for parsing input and building output needs to change in the application, but core processing logic can remain the same.
- Migration is easily managed and inexpensive. Because both Web and non-Web users can use the application at the same time, migrating users to Web interfaces is more easily managed. Web client software (browsers) can be both easier and less expensive to acquire and update.
- Because the WEBPCM is a two-tier structure that does not require Windows or UNIX programming or configuration, it offers higher scalability and stability, plus reduced management.

- Standard Transaction Server features are supported, such as Agendas, Security, and Delivery Notification. This support means applications dependent on those features do not need modification.
- Transaction Server stations can be kept open for multiple transactions with each Web user.
- Character translation (ASCII to EBCDIC or other translations supported by the MCP) is provided. The Web is ASCII-based, and the WEBPCM can optionally convert Web messages from ASCII to EBCDIC or from EBCDIC to ASCII. It is designed so that applications do not have to deal with ASCII.

Using WEBPCM, you can modernize current MCP application interfaces without porting or rewriting to another platform, and you can modernize MCP application interfaces without modifying the application or with making only minor changes.

With WEBPCM, you can stay with the enterprise server advantages, and you maintain the Transaction Server interface (Direct Window or Remote File) rather than changing to a new interface.

WEBPCM supports session dialogs. The Web is generally "stateless," meaning that, after a user makes a request and receives a response, the dialog is terminated and a new dialog must be created for the next request. The WEBPCM supports several options for maintaining a session with a Web user, keeping the station open for multiple transactions with the application.

With the WEBPCM, your users can switch to using a browser as their end-user interface, which increases and facilitates user productivity because of the graphical interface. Also, the use of hypertext linkage increases ease-of-use.

By using browsers for the end-user interface, updating client software is easier and cheaper. Also, using off-the-shelf browsers enables you to eliminate or reduce expensive client terminals or emulation software.

WEBPCM Demonstrations

Demonstrations are released with the WEBPCM. You can access them through the Web Transaction Server Administration Web site. COBOL and ALGOL demonstrations, as well as instructions for manually configuring the Transaction Server, CCF, Web Transaction Server, and WEBAPPSUPPORT are included. The demonstrations show how Transaction Server applications can interface with Web users through the CCF component WEBPCM.

How the WEBPCM Works

WEBPCM uses the Web Transaction Server to interface to Web users. The Web Transaction Server provides fast, efficient, and scalable access; parses the HTTP Web requests; serves static files, such as HTML and graphic files; and optionally recognizes Server Side Include (SSI) directives for including dynamic content.

WEBPCM converts HTTP Web requests into a Message Object, which is then passed to the application. The application does not examine the Message Object, except for the first 17 bytes.

- To access the request, the application links to and calls a support library to retrieve information from the request, such as the type of browser used, any form data the user entered, check boxes selected, and so forth.
- To generate a response, the application calls the support library again to place the response text into the object and set any additional HTTP headers. It then sends or writes the object back to the station.

WEBPCM receives the message and passes output to the Web Transaction Server, which sends the response to the user.

Input Flow

Figure 1–7 shows the input flow to WEBPCM.

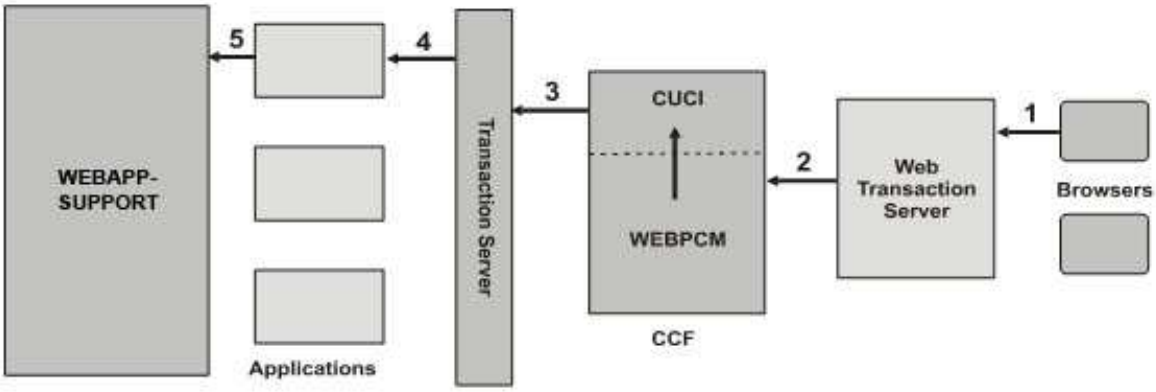

Figure 1–7. Input Flow to WEBPCM

Each step refers to a number in the preceding Figure 1–7.

1. A user at a Web browser, usually Microsoft Internet Explorer or Mozilla Firefox, clicks on a link that is addressed to a TCP/IP port on the MCP host on which the Web Transaction Server is listening. The browser builds an HTTP request and sends it to the Web server.
2. Web Transaction Server recognizes that the request is to be handled by the WEBPCM, so it signals the WEBPCM that input exists. The WEBPCM collects the information from the request and builds the Message Object. If no dialog (station) is open to the application, the WEBPCM opens the dialog.
3. The message is passed through CCF and into Transaction Server.
4. Transaction Server delivers the message to the application.
5. The application recognizes that the message came from a Web user and passes the Message Object to the WEBAPPSUPPORT library to access information in the request. The exact nature of the request is then interpreted by the application, and the request can be processed.

Output Flow

Figure 1–8 shows the output flow to WEBPCM.

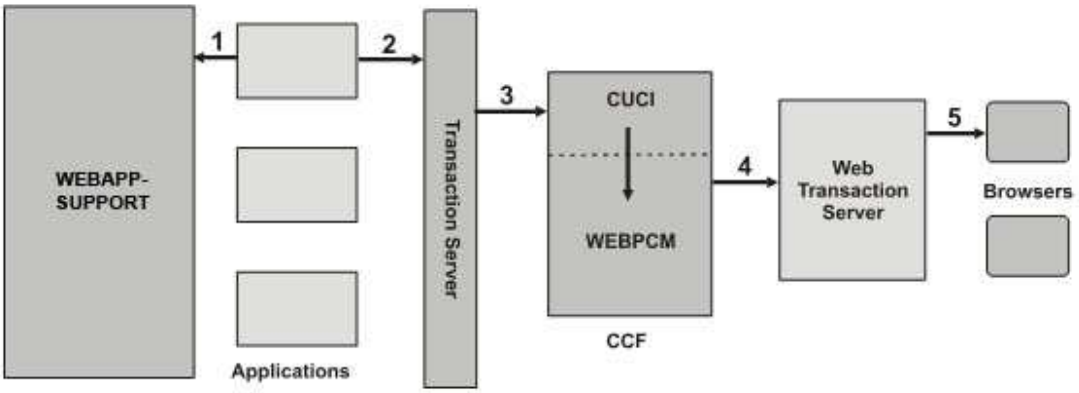

Figure 1–8. Output Flow to WEBPCM

Each step refers to a number in the preceding Figure 1–8.

1. The application builds the response by making calls into the WEBAPPSUPPORT library, which updates the Message Object with the response information. Usually the response is in HTML syntax.
2. The application writes the Message Object back to the station with a SEND or WRITE.
3. Transaction Server passes the message to CCF, and the message is routed to the WEBPCM.
4. The WEBPCM calls Web Transaction Server with the output, setting HTTP response headers and the message content.
5. Web Transaction Server sends the HTTP response to the browser, and the browser displays the response.

XML Parser

What Is XML?

Extensible Markup Language (XML) is a set of rules for defining semantic tags that organize data. XML is a recommended standard of the World Wide Web Consortium (W3C), whose website is at http://www.w3.org/.

The following is an example of a simple XML document:

```
<?xml version="1.0">

<PRODUCT>

  <NAME>Widget</NAME>

  <PARTNUM>1234</PARTNUM>
```

```
<PRICE CURRENCY="USD">7.99</PRICE>

</PRODUCT>
```

In the preceding example, note the following:

- On the first line, **?xml** indicates that this is an XML document, and **version="1.0"** identifies the level of the XML specification to which the document format conforms.
- The rest of the document contains data.
- The document contains only text characters that represent both text and numeric information.

  Storing numeric data in text, rather than binary format, makes XML very portable. Interpreting binary format on different systems can be difficult.
- The tags in the example are PRODUCT, NAME, PARTNUM, and PRICE. The user who creates an XML document, not a standards body, defines the document tags.

What is the XML Parser?

The XML Parser is an API that a COBOL85, an ALGOL, or a NEWP application can use to parse, create, transform (XSLT), or modify XML documents.

In the application you can easily include calls to XML Parser procedures rather than write the code required for XML documents. The procedures are in the WEBAPPSUPPORT library. Using these procedures, you can read or modify parts of, or an entire XML document, or create an XML document.

The XML Parser translates characters from the XML document character set to the application character set. The XML documents must be encoded in an ASCII-based character set, such as iso-8859-1 or UTF-8. An example of a supported application character set is ASERIESEBCDIC.

The XML Parser requires that a Java module supplied by Unisys be installed on a MCP Java system or on a separate system that can run Java. For example, the Java system can be a Windows or Linux system.

XML Parser Architecture

The XML Parser is implemented in two locations: the WEBAPPSUPPORT library of the CCF WEBPCM product and the Java Parser Module (JPM), which is a Java wrapper to the Apache Xerces product. Figure 1–9 shows the XML Parser architecture.

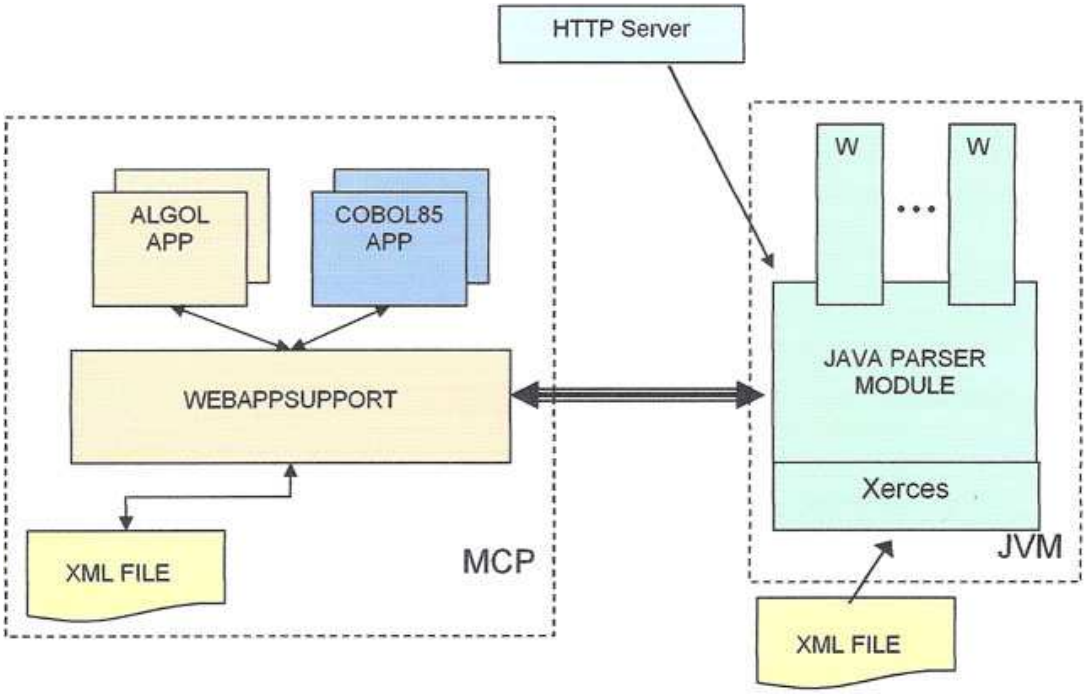

Figure 1–9. XML Parser Architecture

In Figure 1–9, an ALGOL or a COBOL85 application calls the WEBAPPSUPPORT library to access or create an XML document. The application can perform the following tasks.

- Make a current document available to the XML Parser by putting the document in any of the following places:
  - In an application array
  - In an MCP file
  - On an HTTP server
- Ask the WEBAPPSUPPORT library to create a new document and make the document available to the XML Parser

WEBAPPSUPPORT uses TCP sockets to communicate with JPM. The JPM is a Java application that wraps the Xerces XML Parser and parses XML documents. Multiple worker threads in the JPM can parse XML documents concurrently. The JPM also uses the Apache log4J package to log events.

The XML Parser returns the parsed or new document to the application or saves the document in an MCP file.

Hardware Requirements

The JPM can run on any of the following:

- An MCP Java 6.0 Java Processor
- A Microsoft Windows system running Sun JRE 6.0 or higher
- A Linux system

Software Requirements

The XML Parser requires the following software:

- Base MCP Release
- Sun Java Runtime Environment (JRE) 6.0 or 7.0
- Any application that runs on a currently supported MCP system and that is written in one of the following MCP programming languages:
  - COBOL85
  - ALGOL, all variants
  - NEWP

Major Functions

The XML Parser can perform the following:

- Parse and validate an XML document

  An application can supply any of the following to the XML Parser:
  - An XML document in an array
  - A reference to an MCP file containing an XML document
  - An HTTP URL that identifies the document location on an HTTP server

  The application can ask the XML Parser to parse and optionally validate the document.
- Provide an XML document to an application

  An application can ask the XML Parser to provide all data in an XML document sequentially (SAX-mode) or to provide specific data in an XML document (DOM-mode).
- Create or modify an XML document

  The XML Parser can create an XML document or modify a parsed XML document. The XML Parser can return the new or modified document to the application or save the document in an MCP disk file.

- Access an XML document using XPath expressions
- Transform XML documents into other documents using XSLT

  The XML Parser can transform XML documents into other XML documents or into other document types such as text.

- Monitor the following in each of the JPMs
  - Status
  - Version
  - Number of connections
  - Worker threads
  - Memory use
  - Documents parsed
- Enable applications to encrypt or decrypt XML documents (requires licensed encryption component)
- Convert XML to JavaScript Object Notation (JSON)

Standards Supported

The XML Parser supports XML standards and has a standard document structure.

XML Standards Supported

The XML Parser supports XML 1.0 standards for the following:

- Document format

  See http://www.w3.org/TR/2006/REC-xml-20060816/.

- Namespaces

  See http://www.w3.org/TR/2006/REC-xml-names-20060816/.

- XSL Transformations

  See http://www.w3.org/TR/1999/REC-xslt-19991116.html

The XML Parser supports XML Path Language (XPath) 1.0 expressions. The XML Path Language (XPath) 1.0 standard enables an application to use an expression to find data in an XML document. More information is available at http://www.w3.org/TR/xpath/.

XML Document Structure

The XML Parser supports the following document structure:

```
<document>

     +-------> <DTD>  (one only)

     +-------> <comment>

     +-------> <processing instruction>

     +-------> <element>  (one only)

                   +-----> <element>

                   +-----> <text>

                   +-----> <comment>

                   +-----> <entity reference>

                             (read-only)

                             +-----> <element>

                             +-----> <text>

                             +-----> <comment>

                             +-----> <CDATA section>

                             +-----> <processing instruction>

                             +-----> <entity reference>

                   +-----> <CDATA section>

                   +-----> <processing instruction>

                   +-----> <attribute>

                              +-------> <text>

                              +-------> <entity reference>
```

The <document> node can contain the following:

- One <DTD> node (optional)

  This node contains only a text representation of the Document Type Definition (DTD) and does not contain any child nodes.
- Any number of <comment> and <processing instruction> nodes
- One top-level <element> node> (required)

This node can contain any number of the following nodes:

– <element>

  This node can contain any nodes that the top-level <element> node can contain. An XML document can contain any number of levels of <element> nodes under other <element> nodes.

– <text>

– <comment>

– <entity reference>

– <CDATA section>

– <processing instruction>

– <attribute>

Limitations

The XML Parser has the following limitations:

- The application cannot access individual Document Type Definition (DTD) items, such as entity nodes and parameter entities.

  The application can get a document type declaration as one string and can change the document type declaration by replacing the whole DTD node.
- The XML Parser cannot handle document fragments.
- The XML Parser can parse only XML data encoded in an ASCII-based character set, not data encoded in EBCDIC-based character sets.

  If an application has XML data encoded in an EBCDIC-based character set, the application must translate the XML data into an ASCII-based character set. Unisys also recommends that the XML declaration in the XML document identify the ASCII character set.
- The maximum number of nodes in an XML document that can be parsed is not limited by a specific size. The maximum XML document size depends on the available memory in the JPM and on the MCP.
- The maximum number of XML documents that can be stored in the WEBAPPSUPPORT library is 65,536.
- The XML Parser does not support unparsed entities.

XSL Transformations (XSLT) Support

The implementation of the XSL Transformations feature satisfies the following requirements:

- It supports transformation of the XML documents using the W3C-defined rules for the XSLT 1.0 language (http://www.w3.org/TR/xslt).
- Applications can supply the stylesheet separately from the XML document or can reference the stylesheet in the XML document.

- Applications can receive the transformed document, or the transformed document can be written to an MCP file.

XML Path Language (XPath) Support

The implementation of the XPath feature supports the W3C XPath 1.0 syntax for expressions that can access an XML document. See the GET_NODE_BY_XPATH and GET_NODES_BY_XPATH procedures to use XPath.

Limitations for XPath support are as follows:

- The following node set functions are not supported: id(), namespace-uri().
- The following Boolean functions are not supported: lang().
- Variable References are not supported.
- Calculations that result in NaN (not a number), infinity, or a divide-by-zero are not supported. Positive and negative zero also are not supported.
- Only one operator can be used in a predicate or parenthesis grouping. For example, to write (1 + 2 * 3) where the multiplication should be done first, use (1 + (2 * 3)).

XML Encryption

The XML Encryption feature enables applications to encrypt and decrypt part or all of an XML document.

Applications must identify the data to be encrypted; the data can be one of the following:

- An XML document (either parsed or on disk)
- An element of a parsed XML document
- A text node of a parsed XML document
- Other data, such as a jpeg file or key file

Applications can control how encrypted data is represented in the resulting XML document—for example, whether or not the associated key information is put into the *EncryptedData* element.

When an application identifies the part of an encrypted XML document to decrypt, it gets back an updated XML document with unencrypted data. The encrypted data in an XML document can be decrypted to an application array or to an MCP file.

Unencrypted data that is being encrypted or that has been decrypted is not transmitted across networking interfaces where it could possibly be seen.

Site Requirements for XML Encryption

To use XML Encryption, the following requirements must be met at the client site:

- One or more Java Parser Modules (JPMs) to parse XML documents
- MCP Cryptography
- The XML Encryption key

Note: *If public key encryption is needed, the public keys must be stored in the Security Center database prior to the application creating objects that use the public keys.*

XML Encryption Licensing

XML Encryption is a licensed feature. Therefore, the XML Encryption key must be installed for applications to be able to call the XML Encryption interfaces. Key presence is checked

- When WEBAPPSUPPORT initiates.
- When a RESTARTXML command is processed by WEBAPPSUPPORT.

Key Management

Applications must create key objects in WEBAPPSUPPORT that are to be used for encryption or decryption. The key objects can reference existing keys stored in the Security Center database or can be symmetric keys that are supplied to WEBAPPSUPPORT for temporary storage in MCP Cryptography.

If the application delinks from WEBAPPSUPPORT, all of the created keys are discarded. The application must recreate the key objects when it relinks to WEBAPPSUPPORT.

JavaScript Object Notation (JSON) Support

The XML feature that supports the JavaScript Object Notation (JSON) enables an MCP application to work with JSON, which can be used with JavaScript applications.

Support for JSON includes these capabilities:

- Converting XML in a file or an array to JSON text
- Converting XML in WEBAPPSUPPORT memory to JSON text
- Converting comma-delimited text to JSON text

HTTP Client

What is HTTP Client?

HTTP Client is an application interface that allows MCP applications to make HTTP requests (GET, POST, PUT, and so forth) to HTTP (Web) servers and to receive responses.

HTTP Client Architecture

Figure 1–10 shows a simple representation of the HTTP Client architecture.

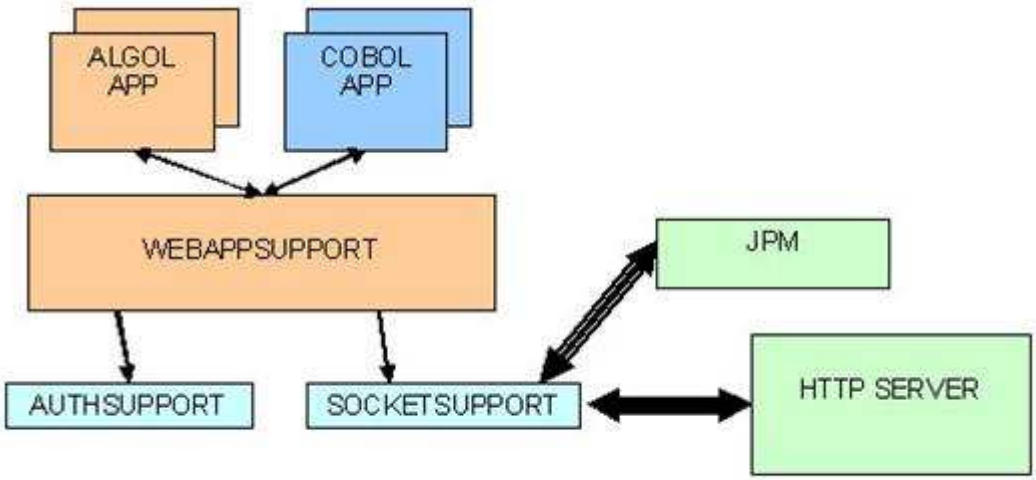

Figure 1–10. Simple Representation of HTTP Client Architecture

When applications want to send HTTP requests, the applications first link to the WEBAPPSUPPORT library. They pass a URL and a hostname or IP address. Optionally, they can specify a port and content data.

The WEBAPPSUPPORT library then opens a socket to the HTTP server, sends the HTTP request, and receives the HTTP response.

The applications can then access elements in the response.

An application can create reusable objects in WEBAPPSUPPORT that represent the HTTP server (host), the application (client), the socket, and the request-response. These objects can store special attributes such as security, cookies, and credentials. They can be reused for multiple requests. Also, multiple requests can be made over the same socket.

HTTP Client Features

- Handles the HTTP 1.0 or 1.1 protocol on behalf of the application
- Can automatically follow redirection responses sent by the server
- Supports secure sockets (https) and sending client certificates
- Can use proxy servers
- Allows content for the request to come from the application or an MCP file
- Supports automatic storing of cookies set by the server and sending in subsequent requests
- Supports HTTP Basic and NTLM authentication methods
- Supports automatic sending of credentials
- Translates EBCDIC data supplied by the application to ASCII for the request and ASCII data from the response into EBCDIC for the application
- Allows the application to set request headers and to access the response headers from the server
- Supports decompressing content compressed by the server and compressing data for sending to the server
- Supports non-WEBPCM MCP applications through the Client HTTP interface
- Supports COBOL85 as the only officially supported COBOL version

Hardware Requirements

HTTP Client runs on currently supported hardware.

Software Requirements

The HTTP Client feature requires the ClearPath MCP 13.0 (or later) release.

Standards Supported

The following are the standards supported:

- W3C RFC 2616 Hypertext Transfer Protocol – HTTP/1.1
- W3C RFC 2617 HTTP Authentication: Basic and Digest Access Authentication (Basic only)
- W3C RFC 2965 HTTP State Management Mechanism
- W3C RFC 2109 HTTP State Management Mechanism
- Netscape Corp. Persistent Client State HTTP Cookies

Regular Expressions

You can use the WEBAPPSUPPORT procedures that provide the Regular Expressions feature to enable your applications to apply expressions to data, similar to the way you can use the Perl Compatible Regular Expressions (PCRE) package. Your applications can process Regular Expressions against subject strings and then get back substrings that match.

The PCRE library is a set of functions that implements regular expression pattern matching whose syntax and semantics are as close as possible to those of the Perl 5 language. The PCRE library is a part of WEBAPPSUPPORT. For more information about PCRE, see http://www.pcre.org/.

The CCF component that implements the Regular Expressions capability is REGEXPRESSION.

Figure 1–11 illustrates this capability.

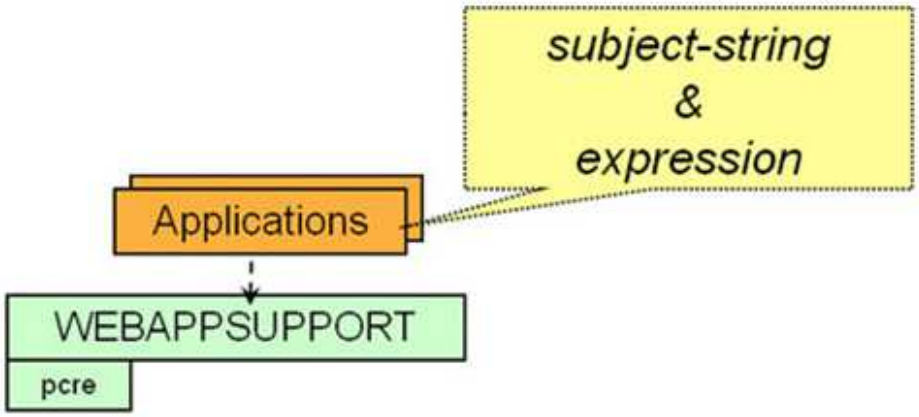

Figure 1–11. PCRE Library

Limitations

The Regular Expressions feature has the following limitations:

- The maximum length of a subject string is 15.5 MB.
- The maximum length of a pattern is 31 KB.
- PCRE callouts are not supported.
- Subject and pattern strings supplied by the application must be in UTF-8 encoding or in a character set that can be translated into either 7-bit ASCII (such as ASERIESEBCDIC) or into UCS2 (such as LATIN1EBCDIC).

Character Set Handling

Applications use Regular Expressions processing in their own character set. The patterns to be compiled and the string to match against are supplied in the character set of the application, for example ASERIESEBCDIC. The substrings returned are also in the character set of the application. The character set of the application is defined by the setting of the MLS_APPLICATON_SET parameter to the SET_TRANSLATION procedure.

The Regular Expressions feature supports the following application character sets:

- ASCII (5)
- UCS2 (85)
- Any character set that can be converted by the MCP MultiLingual System (MLS) to either ASCII or UCS2
- Latin1ISO (13)
- UTF-8 (2)

Section 2
WEBPCM Transaction Server to Internet Application Programming

Acquiring and Installing WEBPCM

The WEBPCM is released as follows:

- As a part of the Custom Connect Facility (CCF).
- With ClearPath MCP systems as part of the operating environment. It includes the WEBAPPSUPPORT library, sample application sources, and demonstrations.

The WEBPCM supports only applications that have been modified to support HTTP users. That is, applications must generate HTML instead of T27-formatted output. You cannot use WEBPCM to directly access MARC or CANDE.

WEBPCM is installed by using the standard installation tools Simple Installation (SI) or Installation Center.

Note: *The Web Transaction Server supports the WEBPCM and is required for using the WEBPCM.*

Modifying Transaction Server Applications to Serve HTTP

You can make coding changes to the application in the following ways:

- Add code to input handler to determine message source.

  The application must be modified so that when it reads a message, it looks for the message source and handles the request appropriately (assuming that non-Web requests still need to be supported).

- Add code for parsing Web input.

  The application needs an interface to the WEBAPPSUPPORT library to parse the Message Object and to build up a response in it. For ALGOL programs, an Include file is provided with the release that contains the entry points into the WEBAPPSUPPORT library plus other useful declarations.

- Add code for building a Web response instead of a non-Web response.

  Logic must be added to the application to generate a response for the Web user, usually in HTML format. The HTML can be hard-coded in the MCP application, or external HTML files can be used and data merged into them, which can reduce the need for future application updates.

  Note: *New applications can also use the WEBPCM.*

Sample sources for applications, in both COBOL and ALGOL languages, are released with the WEBPCM. You can also run online demonstrations to see how the applications work.

Note*: Code for parsing input and building output needs to change in the application, but core processing logic can remain the same.*

Using External HTML Files

Merging application data into external HTML files is a powerful feature of the WEBPCM. Figure 2–1 shows this process.

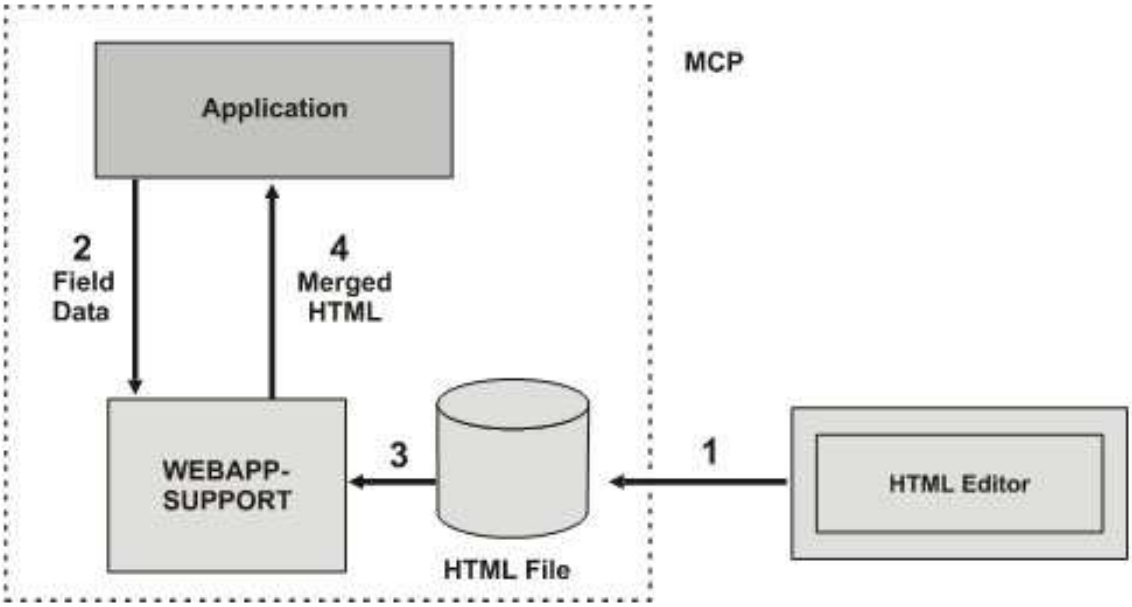

Figure 2–1. Merging Application Data Into External HTML Files

The numbers in the following steps correspond to those in Figure 2-1.

1. Use an HTML editor, such as Microsoft FrontPage or Microsoft Notepad, to create the HTML file. Then place the file on the MCP file system, perhaps using a Client Access Services share. You can also make future updates to the HTML file with the editor.
2. The application calls WEBAPPSUPPORT with data to be inserted into the HTML file. The HTML file must have tags that WEBAPPSUPPORT can recognize as locations to insert the data.

3. WEBAPPSUPPORT reads the HTML file and merges in the application data.

4. WEBAPPSUPPORT returns the merged HTML to the application. The application can then insert the HTML into a Message Object for the response. The merged HTML can also be saved (cached) by the application for subsequent requests.

Using WEBPCM without Modifying the Transaction Server Application

Rather than modifying Transaction Server applications to work with the WEBPCM, you can use processing items to achieve the same results.

- The input processing item can process the input message before the application sees it, and rebuild the message as if it came from a terminal. The input processing item program looks at any content data in the request and converts it to forms input, such as taking the text field from an HTML form and building the buffer as if it came from a T-27 terminal.
- The output processing item can examine the output from the application, build a matching HTML response, and send that response instead of the application response. The output processing item program looks at the screen that the application generated for its response, and maps (screen scrapes) that response to an HTML file.

Notes:

- *Using processing items might be more work overall (including maintenance) than directly modifying the application. You should evaluate each situation individually.*
- *Processing Items work only with Direct Window applications.*

As an alternative, consider using Web Enabler. Web Enabler can run applications in a browser without modifying the application. Refer to “Why use the WEBPCM?” in Section 1 for a list of reasons to consider using the WEBPCM. If those reasons do not satisfy your requirements, consider using Web Enabler.

Processing Item Concept

Figure 2-2 shows the work flow for a processing item.

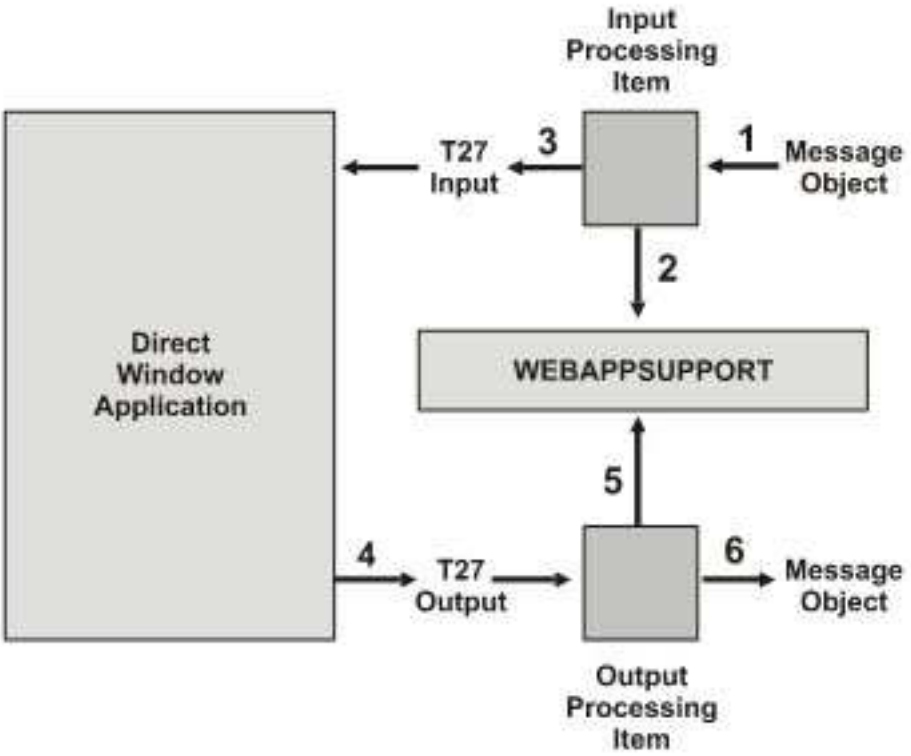

Figure 2–2. Work Flow for a Processing Item

The following steps outline the path that the data takes when you use processing items to give an existing unmodified Direct Window application access to Web users:

1. The Message Object first comes to the Input Processing Item.
2. The Input Processing Item calls WEBAPPSUPPORT to extract information from the request and convert it to input the application expects to see.
3. The Input Processing Item routes the T27-style input through Transaction Server to the application.
4. The application response is routed to the Output Processing Item.
5. The Output Processing Item examines the output, maps that output to a specific HTML, then uses WEBAPPSUPPORT to update the Message Object (saved by the Input Processing Item) with the response.
6. The Output Processing Item sends the Message Object back to the station, where the WEBPCM sends out the response.

Example: Web Enabling Existing Applications

On a Web page, the user filled out a form that created a query string with two name and value pairs, as shown in Figure 2–3.

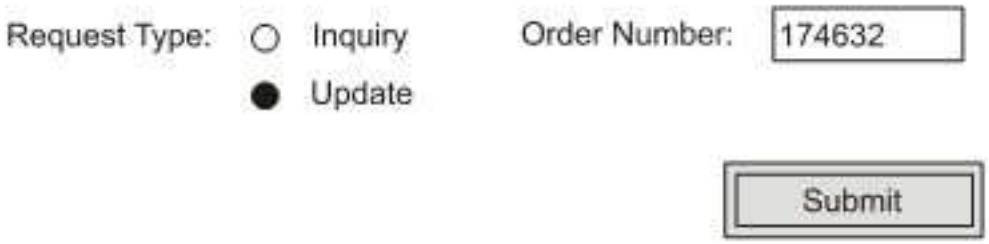

Figure 2–3. Application Handling a Query String

This example shows how an application can handle a query string that contains input data.

The HTML source for the form in this example might look like the following:

```
<CENTER><FORM ACTION="/ordertracker" METHOD=GET>
 <TABLE><TR><TD>Request Type: </TD>
 <TD WIDTH=30%><INPUT TYPE="radio" CHECKED NAME="action" VALUE="inq">
Inquiry
   <BR><INPUT TYPE="radio"     NAME="action" VALUE="upd"> Update</TD>
 <TD>Order Number: <INPUT TYPE="text" NAME="ordernum" SIZE=9>
   <BR><INPUT NAME="submit" VALUE="Submit" TYPE=SUBMIT></TD></TR>
 </TABLE></FORM></CENTER>
```

When the user clicks Submit, the HTTP request is sent to the host with the relative universal resource indicator (URI) /ordertracker, which has been configured in Web Transaction Server to map to the WEBPCM application. The request is passed to the WEBPCM to handle. The WEBPCM is configured to map the path "/ordertracker/" to the Transaction Server window ORDERTRACKER. The HTTP request is converted to a Message Object and sent to the ORDERTRACKER application.

The resulting name and value pairs as seen by the application are

Name	Value
action	inq
ordernum	174632

The ORDERTRACKER application (COBOL) might look like the following:

```
01 NAME-VALUE-BUFFER.
  03 NAME-VALUE-PAIR OCCURS 10 TIMES.
    05 QUERY-NAME  PIC X(10).
    O5 QUERY-VALUE PIC X(20).

CALL "PARSE_QUERY_STRING OF WEBAPPSUPPORT"
 USING  COMS-MESSAGE-AREA, MAX-LEN-10, MAX-LEN-20,
      NAME-VALUE-BUFFER, NUM-PAIRS
 GIVING  WEBAPP-RESULT.
IF WEB-OK
*  good result, analyze data
 IF QUERY-NAME(1) IS EQUAL TO "action"
  IF QUERY-VALUE(1) IS EQUAL TO "inq"
*    process the inquiry
* and so on
```

COMS-MESSAGE-AREA contains the Message Object.

Software Modules Necessary to Support the WEBPCM

The WEBPCM supports Transaction Server applications by implementing two software modules (released with CCF):

- WEBPCM (Web Protocol Converter Module) routes requests from users using the API in Web Transaction Server to Transaction Server applications using the CUCIPCM module of CCF. This PCM is released as code file *SYSTEM/CCF/PCM/WEB.
- WEBAPPSUPPORT is a library for accessing the HTTP request and constructing the HTTP response. This support library is released as *SYSTEM/CCF/WEBAPPSUPPORT. *SYSTEM/CCF/WEBAPPSUPPORT is created as a support library (SL command) with the function name WEBAPPSUPPORT without any special SL attributes. It has the following characteristics:
  - It is a shared-by-all library
  - It is used by Transaction Server applications to parse requests and assist with generating responses
  - It exists separately from the WEBPCM so that if CCF is terminated, the Transaction Server applications are not automatically terminated.

Figure 2–4 shows the various software modules necessary to support WEBPCM plus additional components required for the Web Enabler.

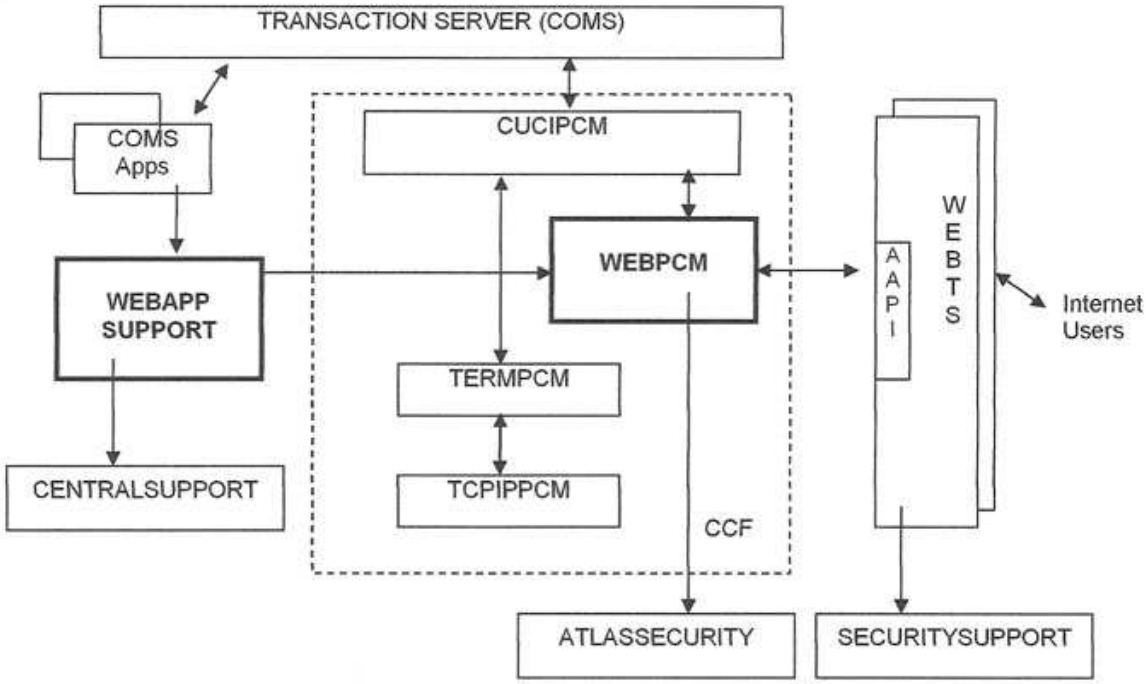

Figure 2–4. Software Modules Required to Support WEBPCM

Summary: Getting Applications to Work with the WEBPCM

Follow this process to get applications to work with the WEBPCM:

Design Web Pages

1. Prototype with an HTML editing tool; view pages in a browser.

   Start laying out your HTML pages and design the flow of screens that the browser user must follow through your Web site. You can use an HTML editing tool such as Microsoft Front Page to visually design the pages.

2. When the pages are ready, either use the HTML as an external HTML to the application (the preferred method) or insert the HTML source into the application.

Modify or Write the Application

1. Modify input and output handling routines to detect and generate Web output.
   - On input, use either the trancode (the first 17 bytes of the input message) or the station name to determine if the input message came from a browser user. Then the input message (the Message Object) is passed by the application to the WEBAPPSUPPORT library to access information from the request, such as input form field data.

- On output, the application should insert the HTML page into the response with the SET_CONTENT call to the WEBAPPSUPPORT library, along with any special output headers needed. The response is then written back to the Transaction Server station.

2. Refer to the released WEBPCM demonstration programs (accessible through the Web Transaction Server Administration Web site) for examples.
   - Examples of programs that use WEBPCM are included with the WEBPCM releases. Direct your browser to the Web Transaction Server Administration Web site, which is usually port :2488 on your MCP system, and follow the links to the WEBPCM examples. You must be a privileged user on the MCP system to access the Web Transaction Server Administration site.
   - If you used Simple Install or Installation Center to install the WEBPCM (which is installed by installing CCF and Transaction Server), the WEBPCM demonstrations should be preconfigured in Transaction Server and CCF. If not, the WEBPCM demonstrations page contains detailed instructions on how to configure Web Transaction Server, CCF, and Transaction Server to run the WEBPCM demonstrations.

Configure Web Transaction Server Virtual Directories

Decide what the first node should be in the URL and use that node as a virtual directory that maps to the WEBPCM application.

Web Transaction Server treats the first node in a URL that comes after the host name as the virtual directory. For requests that are to be handled by a Transaction Server application using the WEBPCM, that virtual directory must map to the WEBPCM application.

The following URL has /products/ as the virtual directory:

```
http://www.acme.com/products/order?id=widget&quantity=1
```

The node/order might tell the Transaction Server application what specific function is desired in the products category.

Multiple virtual directories in Web Transaction Server can map to the WEBPCM, and the WEBPCM can map to the same Transaction Server application, if desired. The Transaction Server application can determine from the Message Object which virtual directory was used by using the GET_HEADER ($APPLICATION-PATH) function in the WEBAPPSUPPORT library.

You might also want to have another virtual directory that maps to HTML files, graphics, and so on. With that approach, browser users can access a home page, and the Transaction Server application can generate HTML that references the graphics.

***Note**: Although the WEBPCM demonstrations are run with the ATLASADMIN provider, most production work should be done with another Web Transaction Server provider, such as ATLASSUPPORT, so that ATLASADMIN is always available for administration functions.*

Configure CCF (WEBPCM Service)

You need only minimal settings to configure CCF.

Each virtual directory in Web Transaction Server that should map its requests to a Transaction Server application using the WEBPCM must map to a unique service in WEBPCM (ADD SERVICE WEBPCM command). Minimally, the Path and Service attributes must be set, and the Window attribute should also be set.

The path attribute is the same as the virtual directory name in Web Transaction Server.

The service attribute is usually CUCIWEBSERVICE, which is a service defined in CUCIPCM (ADD SERVICE CUCI command) with no attributes. WEBPCM sets the CUCIPCM attributes at dialog establishment time.

The Window attribute is the name of the Transaction Server window to which requests are passed.

Other WEBPCM Service (ADD SERVICE WEBPCM command) attributes that might need to be set are listed in the following table.

Attributes	Description
StringTerminate and Translate	These attributes default to FALSE and TRUE respectively, which are the values most likely used by COBOL applications (pad strings with blanks and convert ASCII to EBCDIC on input, EBCDIC to ASCII on output).
StationControl	The default for station control is for WEBPCM to use Cookies to identify each user, and map their session to a specific station in Transaction Server. Another setting that might be desired is Permanent, so that only one station is used and all requests are mapped to that station.
CheckUserAuth	If you want to restrict access to the Transaction Server applications to only those users who have valid MCP usercodes, set this attribute to TRUE. Note the privileged status of the user is not checked by this setting; the application can check the user's privilege status with the GET_USER_PRIVILEGED procedure in the WEBAPPSUPPORT library.
Usercode	If the Usercode attribute is not specified in the WEBPCM service, the usercode sent to Transaction Server when the dialog opens is the user's usercode if CheckUserAuth is TRUE, or else the Web Transaction Server provider anonymous usercode if CheckUserAuth is FALSE. To ensure that a specific usercode is used, set this attribute in the WEBPCM service.

Configure Transaction Server Window/Program/Agenda

This step is needed only if there is not already an existing Transaction Server definition.

If you are modifying an existing application, you might not need to change the Transaction Server configuration. Instead, ensure that the WEBPCM service maps to the existing Transaction Server window. Otherwise, configure the Transaction Server window, program, and agenda for the application.

***Note**: You might need to increase the Maximum Message Text Size setting in the Global section of the Transaction Server Utility window if the amount of data your application sends back to the station is greater than the current setting.*

Test the Application

Use WEBAPPSUPPORT library tracing to debug the program.

Now you are ready to test your application. The application can request tracing to be done for its calls into WEBAPPSUPPORT by first calling the SET_TRACING procedure in the WEBAPPSUPPORT library. Using this procedure makes debugging an application much easier.

Application Design Considerations

This subsection discusses application design considerations.

Programming Languages Supported by WEBPCM

The following application programming languages are supported by WEBPCM:

- COBOL74
- COBOL85
- ALGOL
- NEWP
- C
- Pascal

Refer to Section 3, “WEBAPPSUPPORT Library Interface” to see which procedures should be used for the programming language chosen.

Remote Files versus Direct-Window Applications

As gateway into Transaction Server, WEBPCM allows the following applications to interface to HTTP (intranet or Internet) users who use the following normal Transaction Server mechanisms:

- Transaction Server Direct Window applications
- Transaction Server Remote File applications

Supported applications use the Direct Window interface or use the Remote File interface and run in a Transaction Server window.

No special requirements apply to using the Transaction Server headers for other than normal Transaction Server mechanisms such as Trancode Routing. The data sent to and from the user uses the standard application message buffers.

Transaction Server Synchronized Recovery

The Transaction Server feature Synchronized Recovery is supported with the WEBPCM with the restriction that an application should not use any of the following WEBAPPSUPPORT procedures if Synchronized Recovery is required:

- GET_HEADER ($PATH-TRANSLATED) (other GET_HEADER calls can be used)
- getHeader ($PATH-TRANSLATED) (other getHeader calls can be used)
- GET_MIME_TYPE
- getMimeType
- GET_REAL_PATH
- getRealPath

These procedures require processing by the Web Transaction Server, and the Web Transaction Server must be waiting on the response to the user's request to process the above calls. Applications that want to use the above functions and need recovery should migrate their databases to use REAPPLYCOMPLETED and INDEPENDENTTRANS.

Requests that exceed 60,000 bytes for an input message object are not supported for Synchronized Recovery.

Delivery Confirmation

The Transaction Server feature Delivery Confirmation is supported with the WEBPCM:

- A positive delivery confirmation is returned to the application if the data was successfully written to the network provider (TCP/IP) for delivery to the client.
- If the data could not be sent, then no delivery confirmation is sent to the application. Failure to send the data can be caused by the client (browser user) terminating the connection, such as by clicking on another Web page link.

Processing Items

Transaction Server processing items process the input to and output from the Transaction Server application for Web messages, just as for other Transaction Server messages.

You do not necessarily need to modify the existing application. Instead you can have a processing item modify the request on input, converting the message so that it looks as though it came from a non-Web user, and modifying the application response, converting the output into appropriate HTML.

The processing items make the calls to WEBAPPSUPPORT instead of, or in addition to, the Transaction Server application.

Character Sets

HTTP and HTML use ASCII-based character sets. The WEBPCM supports converting ASCII strings to EBCDIC on input, and back to ASCII on output, so that applications can use their native character set for processing text strings.

Character set translation is configured with the TRANSLATE option in the WEBPCM service definition. For more information, refer to the WEBPCM ADD SERVICE command. The default is to convert ASCII to EBCDIC.

For generating output in character sets other than ASCII, applications must generate the codes appropriate for the destination. For example, HTML responses that are to contain characters in the Extended Latin1ISO character set should use escaped characters, for example &#232 (hex E8), which is a decimal reference to the character è.

Use the HTML_ESCAPE procedure in WEBAPPSUPPORT to translate escaped characters for Extended ASCII HTML text. As an alternative, applications can generate output in the APPLICATIONCCS service setting, and WEBPCM translates the output to the CLIENTCCS setting by using the MCP MultiLingual System.

String Terminations

Text strings passed to and from the Transaction Server application can either be optionally terminated with a null character or padded to the right with blanks.

COBOL applications typically expect strings to be terminated by blanks.

If terminated by a null character, the rest of the buffer is undefined.

Use the STRINGTERMINATE option in the WEBPCM ADD SERVICE command to control how text strings are terminated.

Serving Both Web and Non-Web Users

An existing application that currently serves non-Web users (such as users using T27-compliant devices) can be modified to also serve Web users and still serve users with the existing interface. The application must first determine the type of device from which the request was sent. By using the WEBPCM, this determination can be accomplished by checking either of the following:

- Trancode field of the Message Object

- StationName of the request, which can be named differently from other station types

The Trancode field of the Message Object always contains fixed text configured with the TRANCODE option in the WEBPCM service definition. Refer to the WEBPCM ADD SERVICE command.

Applications can check the first 17 bytes of the message to determine the message source. Alternately, the Transaction Server Trancode feature can be used to indicate the message source. Refer to Message Object Format (Input/Output Message Format) for information on the format of the Message Object, and see Transaction Server documentation on Trancode functionality.

Use the STATIONNAME option in the ADD SERVICE WEBPCM command to configure StationName format.

Inactivity Timeout

Because of the stateless nature of the Web, users can stop in the middle of a transaction, use other applications, or even turn their systems off. In these cases, the host does not know that the user has gone and might keep application dialogs open that will never be continued.

The WEBPCM supports an optional timeout on inactivity. If no input is received for a station for the elapsed time, the station closes. The default timeout is 12:00:00 (12 hours).

Use the INACTIVITYTIMEOUT option in the WEBPCM ADD SERVICE command to configure inactivity timeout.

Transaction Server Station Closure

If the WEBPCM detects that the station has closed in Transaction Server, it responds to any outstanding request with an error message and closes the dialog.

The error message returned is a response with the HTTP Status Code set to 503 (Service Unavailable), and the following text:

```
The service <X> has become unavailable.
```

In this message, <X> is the PROGRAMID attribute value of the WEBPCM ADD SERVICE command or WEBPCM MODIFY SERVICE command. A Retry-After header is not sent with the response, which means the user must retry the request manually.

For more information, refer to “Using External HTML Versus Internal HTML” later in this section and “Inactivity Timeout” earlier in this section.

Dialogs (stations) can also be closed by an operator with the WEBPCM CLEAR DIALOGS command.

A Transaction Server dialog cannot be closed from the client side.

Learning About HTTP and HTML

Application developers must understand the following two standards to process Web user requests and generate responses:

- HTTP (Hypertext Transfer Protocol) is the protocol used between the Web server (Web Transaction Server) and the client, usually a Web browser. It is defined by the W3C standards organization, and a specification for the protocol is available online at their Web site: http://www.w3c.org/.

  Different browsers support different HTTP levels, and design consideration might be needed for the client level, which can be determined with the $PROTOCOL header name in a call to the GET_HEADER procedure in WEBAPPSUPPORT.
- HTML (Hypertext Markup Language) is the language definition that browsers use to lay out pages. Numerous books on HTML are available. Software tools that generate HTML without the user needing to know the language are also available.

Using External HTML Versus Internal HTML

The Transaction Server application can generate its own HTML responses if it chooses. But generating HTML requires the application to be modified and recompiled with each HTML change, and HTML changes can be frequent. It also might be desirable to edit the HTML file with an HTML editor.

Another option is to reference an external HTML file, one that is maintained outside the program. Three ways to do this are as follows:

- Direct or redirect the user to view a file by returning a 302 (Found) response. Use the SET_REDIRECT procedure in the WEBAPPSUPPORT library for this.
- Read the HTML file into the application and return the file contents to the user.
- Use the MERGE_HTML_FILE_AND_DATA or MERGE_DATA procedure in the WEBAPPSUPPORT library to replace tagged parts of the HTML with fields supplied by the program.

The first two methods work if the HTML data is static. Often, however, fields in the HTML page need to be filled in with data contained in the application or in a database. Use the third method to easily update an HTML page from the application. Refer to MERGE_FILE_AND_DATA or MERGE_DATA procedure and the “Using an External HTML File” example in “Sample Applications.”

User Authorization

WEBPCM

WEBPCM supports these authentication methods:

- HTTP Basic
- NTLM

The authentication method used is controlled by the AUTHENTICATIONTYPE service attribute.

If the WEBPCM service is configured with CHECKUSERAUTH = TRUE, the WEBPCM verifies that the usercode in the request authorization header is a valid MCP usercode. If the usercode is not valid or if no Authorization header is present in the request or for the dialog, the WEBPCM returns to the client a 401 response (Unauthorized), which causes the browser to prompt a user for a usercode and password.

When it validates users, WEBPCM (by way of the Web Transaction Server) uses the SECURITYSUPPORT library of a user, if it is present. For details, refer to "SECURITYSUPPORT Library Support".

Transaction Server Application

The usercode sent to Transaction Server is determined by the following criteria in the WEBPCM SERVICE command:

- If the USERCODE attribute is set in the WEBPCM service, that usercode is sent.
- If the USERCODE attribute is not set and CHECKUSERAUTH = TRUE, the usercode of the client is sent.
- If the USERCODE attribute is not set and CHECKUSERAUTH = FALSE, the anonymous usercode of the Web Transaction Server provider is sent.
- If the USERCODE attribute is not set and STATIONCONTROL = PERMANENT, the usercode sent is the usercode of the first user to use the dialog of the service.

The Transaction Server application can check the MCP privilege status of a user who has passed authorization with the GET_USER_PRIVILEGED procedure of WEBAPPSUPPORT.

In the WEBPCM SERVICE command, the following criteria can affect security:

- If CHECKUSERAUTH = FALSE, the WEBPCM does not do any security checking before sending the request to the application.
- If SHOWPW = TRUE and CHECKUSERAUTH = FALSE, the application can see the password in the Authorization header through the $REMOTE-USER attribute returned from GET_HEADER procedure of WEBAPPSUPPORT. This option enables the application to maintain its own passwords, which should not be the MCP passwords.

Refer to the WEBPCM ADD SERVICE or the WEBPCM MODIFY SERVICE command in the *Custom Connect Facility Administration and Programming Guide* for information about CHECKUSERAUTH, USERCODE, STATIONCONTROL, and SHOWPW attributes.

SECURITYSUPPORT Library Support

When validating users, WEBPCM (through the Web Transaction Server) uses the SECURITYSUPPORT library of a user, if it is present. WEBPCM takes the following action:

- Determines at initialization if SECURITYSUPPORT is present.
- Calls the LOGONCHECK procedure in SECURITYSUPPORT when a user has passed or failed USERDATA authorization. (LOGONCHECK is called for both successful and unsuccessful USERDATA validation.)

The Web Transaction Server calls LOGONCHECK when it validates user authorization for nonanonymous requests.

The default SYMBOL/SECURITYSUPPORT source file has an identification of 10=NXATLAS for MCS TYPE.

If LOGONCHECK returns a result with the field INSTALLATION DETERMINED ERROR CODE set to nonzero, then the Web Transaction Server acts as though USERDATA had rejected the user and returns a 401 (Unauthorized) HTTP response.

For the SECURITY parameter, the following criteria applies:

- A STATION IDENTIFIER is not supplied.
- The USERCODE field contains the usercode being authorized.
- CHARGECODE and ACCESSCODE are not supplied.
- INVALID LOGON ATTEMPT NUMBER is always zero.

If a fault occurs in the SECURITYSUPPORT library, the Web Transaction Server stops calling LOGONCHECK. Web Transaction Server must be terminated and restarted for it to resume calling LOGONCHECK.

Note: *The ATLASSECURITY library works to lock out users who exceed the MCP LOGONATTEMPTS setting for log-in retries.*

Refer also to the *Security Administration Guide* for more information on WEBPCM and security. Refer to the *Custom Connect Facility Administration and Programming Guide* for more information about WEBPCM commands.

Internationalization Considerations

WEBPCM applications can process input and generate responses containing character set encodings other than the defaults of LATIN1EBCDIC or LATIN1ISO. The data supplied by the application can be a different character set encoding than that intended for the response if the CENTRALSUPPORT installed on the MCP supports translation between the two character set encodings.

For example, an application can generate the response in the JapanEBCDICJBIS8 character set encoding, and WEBPCM can translate the output to CODEPAGE932 (also known as Shift_JIS).

The data merging functions also support translations. For example, an HTML file could be coded in the CODEPAGE932 encoding, and data in the JapanEBCDICJBIS8 character set could be merged into the file contents.

HostCCS System Option

The HostCSS system option enables the default host character set to be specified when the default CCSVERSION is ASERIESNATIVE. WEBPCM uses the HostCSS setting in the CENTRALSTATUS() call, if the setting exists, for use as the CCS_HOST_DEFAULT.

Processing Input and Generating Output

WEBPCM determines the character sets used by the application and the client for processing input and output in one of two ways:

- A new configuration setting in the WEBPCM service
- A procedure call that the application makes to the WEBAPPSUPPORT library

If configured in the service, the following service attributes should be set:

- APPLICATIONCCS
- CLIENTCCS

The application can override the settings in the WEBPCM service by making a call to the SET_TRANSLATION procedure in the WEBAPPSUPPORT library. This call determines the character sets to be used for all subsequent processing for that application process.

Restrict non-USASCII characters in HTTP headers to situations where you are sure that all of the users of your site will use only the one character set that your application and CLIENTCCS are designed to handle. This restriction exists because it is not possible to determine the HTTP request that the character set used in the headers.

It is generally good practice to set the Content-Type header in the response to indicate the character set used in the response content. For example, you can set the Content-Type header to the following:

```
text/html; charset=Shift_JIS
```

Merging Data

Before calling the MERGE_DATA or MERGE_FILE_AND_DATA procedures in WEBAPPSUPPORT to merge data into HTML, an application should do the following:

1. Set the CHARSET parameter to the application character set, for example, what the application has coded the DATA_BUFF contents in, and what the application expects the data returned to it in RESULT_BUFF to be coded in.
2. Set the INPUT_CHARSET parameter (MERGE_DATA procedure) to the character set that INPUT_BUFF is coded in.

If an external file contains characters other than LATIN1 characters, such as CODEPAGE932, use the MERGE_I18NFILE_AND_DATA procedure and set the FILE_CHARSET parameter to the character set of the file.

Maintaining Session State Dialogs

HTTP (Web) dialogs are stateless. The client (browser) opens a TCP/IP connection to the server, makes one or a few closely related requests, such as images on an HTML page, waits for the response, and then closes the dialog. This sequence does not match the host application paradigm of a dialog that is kept open through multiple steps of a transaction (or multiple transactions).

The WEBPCM supports two methods for maintaining a session with a user:

- Cookies
- Hidden HTML fields

Use the STATIONCONTROL option in the ADD SERVICE WEBPCM command to configure session options. Refer to the *Custom Connect Facility Administration and Programming Guide* for information about the WEBPCM ADD SERVICE command.

Using Cookies to Maintain Session State

When you configure a service using the WEBPCM ADD SERVICE command or the WEBPCM MODIFY SERVICE command to maintain a session with cookies, an initial request in the session does not have the appropriate cookie header. This situation might result because the "Cookie: header" has expired or has an invalid value from a previous dialog that terminated. In these cases, the WEBPCM creates a cookie for that dialog and uses the "Set-Cookie: header" with a reserved name of WEBPCMTRANSID in the HTTP response. Cookie expiration is not set, so that if the browser is terminated at the client, a new dialog (station) is opened on the next use from that client.

Applications using their own cookies should not use a Cookie called WEBPCMTRANSID.

If cookies are sent to a client that has cookies disabled, successive Transaction Server dialogs (stations) are created for each request from that user, and old ones are left open. Administrators should consider a timeout value for dialogs, either by using the WEBPCM Inactivity Timeout feature, or by implementing a timeout.

Using Hidden HTML Fields to Maintain Session State

If the application developer is concerned that users might have cookie support disabled but still wants sessions to be maintained, the WEBPCM service can be configured to store session information in hidden HTML fields or HTML links using the WEBPCM ADD SERVICE command or WEBPCM MODIFY SERVICE command.

HTML Fields

If the application developer uses HTML fields, the application puts a hidden field with a reserved name of WEBPCMTRANSID into the HTML form. The WEBPCM then routes subsequent requests with that hidden field to the same Transaction Server dialog. The name used for the reserved field is the same as that used for a cookie.

Here is an HTML example:

```
<FORM ACTION="/comsprog1/name" METHOD=POST>
 <INPUT NAME=WEBPCMTRANSID VALUE="0000021,0002307" TYPE="HIDDEN">
 <INPUT NAME=FIRSTNAME VALUE="" TYPE="TEXT" SIZE=30>
 <INPUT NAME="T" VALUE="Transmit" TYPE="SUBMIT">
</FORM>
```

In this example

- The string 0000021,0002307 came from the Message Object through the GET_DIALOG_ID procedure in the WEBAPPSUPPORT library.
- The FORM METHOD used in the HTML can be either GET or PUT.

If multiple forms exist on the HTML page, each form needs to have the hidden field in order to maintain the session.

If the HTML fields are used and the application sends a response without the reserved hidden field, the WEBPCM closes the Transaction Server dialog (station) after the response is sent out. This closure includes sending error responses.

HTML Links

If the HTML links are used to maintain session state, an application can generate code like the following:

```
<A href="/comsdemo2/html/2?WEBPCMTRANSID=0000118,2772669>Next<A/>
```

In this example, the response content must include the cookie name, WEBPCMTRANSID. It must be preceded by NAME=, NAME="<name>", or simply =, as shown here.

The result of this HTML code is that when WEBPCM sends the response to the client, WEBPCM leaves the Transaction Server dialog open.

Maintaining Stateless Dialogs

You can configure the WEBPCM to not retain individual Web users sessions. Two options exist:

- Permanent stations
- Single request stations

Permanent Stations

You can use the WEBPCM ADD SERVICE command or the WEBPCM MODIFY SERVICE command to configure the WEBPCM service so that stations are kept open permanently, which means that the WEBPCM does not close the station after returning a response. Also note that the WEBPCM service station is shared, and that all requests that map to the WEBPCM service go to the same station.

The advantage of permanent stations is that you avoid the processor overhead of opening and closing multiple stations. If the application needs to maintain session state with a specific user, the application must manage that itself, such as with cookies or hidden HTML fields.

Configure permanent stations by setting the STATIONCONTROL option to PERMANENT in the WEBPCM ADD SERVICE command or the WEBPCM MODIFY SERVICE command.

Single Request Stations

You can configure the WEBPCM service so that stations close after the completion of each request.

Configure single request stations by setting the STATIONCONTROL option to NONE in the WEBPCM ADD SERVICE command or the WEBPCM MODIFY SERVICE command.

Performance Considerations

For Permanent Stations

You can reduce the overhead of creating and destroying stations in Transaction Server by declaring the WEBPCM service as having a STATIONCONTROL value of PERMANENT. With PERMANENT, separate sessions with each user are not maintained, and the Transaction Server station is kept open after the first request until the application requests the close.

With permanent stations, the application can either maintain its own session if needed or handle each request based on some attribute of the request, such as the application path or HTML controls used by the end user.

Configure permanent station by setting the STATIONCONTROL option to PERMANENT in the WEBPCM ADD SERVICE command or the WEBPCM MODIFY SERVICE command.

String Termination and Character Sets

Terminating the processing of strings with a null character is more efficient than padding strings with blanks

Avoiding the translation of ASCII to EBCDIC saves processor time if the application can handle ASCII strings directly.

Transaction Flow

User (HTTP) Request

On receiving a request that maps to the WEBPCM, Web Transaction Server signals the AAPI newReqEvt event of the WEBPCM. The WEBPCM uses the application path from the HTTP request to map the request to a configured WEBPCM service (WEBPCM ADD SERVICE command or WEBPCM MODIFY SERVICE command).

If the request does not map to an existing Transaction Server dialog, the WEBPCM then sends an open request to the matching service in CUCIPCM.

If the dialog is successfully opened, an input object is built and sent to the application representing the HTTP request.

Before the transaction ID is sent to the application, the WEBPCM calls two AAPI functions in Web Transaction Server on behalf of the application:

- initRsp (to initialize the response in Web Transaction Server)
- setStatusCode-200 (to create a default of a good status)

Applications do not need to set the status code if 200 (Ok) is the desired status.

Application Response

The Transaction Server application, upon receiving the incoming notification, calls the WEBAPPSUPPORT library to access the request and build the response.

After examining the request and gathering the information requested, the Transaction Server application performs the following actions:

- Builds the response through calls to WEBAPPSUPPORT
- Sends the message back to the user

Usually the application generates HTML that a Web browser processes or causes an existing HTML file to be updated and inserted into the output message.

Server Side Includes (SSIs)

Server Side Includes (SSIs) are directives in HTML pages that are evaluated on the server while the pages are being served. They allow dynamically generated content to be added to a HTML page without having to program a server extension such as a Common Gateway Interface program. Optionally, the Web Transaction Server scans static files and application responses for SSI directives. Valid directives are replaced with the processed text.

For example if the .shtml suffix is configured in the Web Transaction Server for SSI processing and the HTML file with an .shtml suffix contains the following:

```
<p>Today's date is <!--#echo var="DATE_LOCAL" -->.</p>
```

The preceding text might be replaced with the following text:

```
<p>Today's date is Wednesday, July 22, 2008 14:36:22 EDT.</p>
```

The implementation of this feature satisfies the following requirements:

- Implement a subset of Apache HTTP Server SSIs (ϒ) in the Web Transaction Server.
- Include MCP files in documents that are read from MCP disk or responses that are supplied by AAPI or WEBPCM applications.
- Support basic echo functions like time, date, and CGI variables. The time function must be available for formatting

Programming Considerations

This subsection discusses programming considerations for WEBPCM applications.

Application Response

The Transaction Server application, upon receiving the incoming notification, calls the WEBAPPSUPPORT library to access the request and build the response.

After examining the request and gathering the information requested, the Transaction Server application

- Builds the response through calls to WEBAPPSUPPORT
- Sends the message back to the user

Usually the application generates HTML that a Web browser processes or causes an existing HTML file to be updated and inserted into the output message.

Application Creation of Response

The two steps to sending the response back to the Web user are

1. Optionally setting HTTP headers
2. Sending the content data (the actual response)

Setting Status Code, HTTP Headers

Responses go to Web users with an HTTP status code of 200 (OK) by default. If a different status code is needed for the response, the application must call the SET_STATUS_CODE procedure in the WEBAPPSUPPORT library to set the status code.

The application can set and modify HTTP headers sent with the response. For example, the application can set its own cookie headers for its own tracking purposes. HTTP headers are set by calling the SET_HEADER or SET_COOKIE procedures in WEBAPPSUPPORT.

The content type of the message is set by calling the SET_CONTENT_TYPE procedure (if it is different than the default of text/html).

A redirection response, telling the client to go to another resource can be generated with a call to the SET_REDIRECT procedure.

Typically, all application processing to generate the response is done before setting any headers, in case an error response needs to be sent instead.

Adding the Content Data and Returning the Response

To add the content of the response (usually the HTML document), the application calls SET_CONTENT in the WEBAPPSUPPORT library, passing the same message received from the user.

To return the response, the application does a SEND (or WRITE to the remote file) to the station. The WEBPCM handles the calls into Web Transaction Server to send the data.

Multiple calls to SET_CONTENT, each followed by a SEND or WRITE, can be done to send messages larger than the maximum size for each SEND or WRITE. The size of each segment (Message Object) sent must be small enough to fit into one response buffer (that is, less than 392,000 bytes in length for Direct Window applications, 9K bytes for Remote File applications).

To send multiple segments, perform the following steps:

1. Build the response in one or more internal buffers, of any length.
2. Call SET_HEADER and set the Content-Length header to the length of the total content.
3. Call SET_CONTENT with the first segment of data and set the COMPLETE parameter to FALSE.
4. Call GET_MESSAGE_LENGTH and send or write the message.
5. Call SET_CONTENT with DATA_LEN set to zero to clear the stored data in the message object.
6. Repeat steps 3 through 5 for each segment of data, calling SET_CONTENT with COMPLETE = TRUE on the last segment.

If the amount of data to be returned in the response cannot be easily determined at the time the first part of the response is sent, perform the following steps:

1. Build the response in one or more internal buffers, of any length.
2. Call SET_HEADER and set the Connection header to the value ‘close’..
3. Call SET_CONTENT with the first segment of data and set the COMPLETE parameter to FALSE.
4. Call GET_MESSAGE_LENGTH and send or write the message.
5. Call SET_CONTENT with DATA_LEN set to zero to clear the stored data in the message object.
6. Repeat steps 3 through 5 for each segment of data, calling SET_CONTENT with COMPLETE = TRUE on the last segment.

Transaction Server Message Interface

The Transaction Server message interface is applicable to both Direct Window applications and Remote File WEBPCM applications.

Direct Window applications have access to the Transaction Server input and output headers.

Remote File applications do not require use of Transaction Server input and output headers during interactions with HTTP users.

Note: *No special requirements apply to using the Transaction Server input and output headers to access WEBPCM functionality.*

The Transaction Server interface is the interface the targeted applications use to wait for input and to return output. Existing applications can continue to serve their current interfaces (which can, for example, expect T27-type input and output) while extending their support to HTTP users. This approach simplifies the application, eliminating the need to wait on different input sources.

Through the Transaction Server Interface, applications wait for input and return the responses. HTTP requests are passed to applications through this interface.

The following happens on input:

- Applications receive a string of data in the data buffer that uniquely identifies the request, called the Message Object. This data is opaque to the application; that is, the application does not examine it.
- The message is passed to the WEBAPPSUPPORT library to be examined. The message can be thought of as an object, and the WEBAPPSUPPORT library provides the methods that act upon that object.
- The application then makes calls to the WEBAPPSUPPORT library to collect information on the request.

The following happens on output:

- Applications write the message object back to the station from which it was received.
- Changes in the HTTP headers for the response are effected through the WEBAPPSUPPORT library.

Header and Message Formats

Transaction Server Input Header Format

The following fields in the Transaction Server Input Header are important to applications using the WEBPCM.

Field	Description
Function Index	If a trancode has been defined for HTTP messages on this Window, this field contains the trancode index that indicates the message source. Refer to the trancode field of Message Object Format (Input/Output Message Format).
Function Status	If delivery confirmation was requested, this status field contains the result of the delivery.

Transaction Server Output Header Format

The following fields in the Transaction Server Output Header are important to applications using the WEBPCM.

Field	Description
Delivery Confirmation Flag	Positive and negative delivery confirmation for TP-to-TP messages are available. The confirmation for TP-to-TP messages is similar to delivery confirmation for messages sent to stations. This feature is invoked by setting the Delivery Confirmation Flag to 1 and the Delivery Confirmation Key field to a nonzero value in the output header of the TP-to-TP message.
Delivery Confirmation Key	Delivery Confirmation is supported for responses sent through the WEBPCM.

Message Object Format (Input and Output Message Format)

The data message (sent to both Direct Window and Remote File Transaction Server applications on input, and sent out for output) has the following format.

Field	Description
Trancode	Text that is always present in the first 17 bytes of the message. It is the value set by the TRANCODE attribute in the WEBPCM SERVICE definition, right padded with blanks. It can be used for the Transaction Server trancode feature, or an application can examine the field to determine this is a message from the WEBPCM. The character set used for the trancode is EBCDIC. You can place the trancode into the URL, instead of hard-coding it in the service. This practice makes it easier to manage a system in which multiple programs are running in one Transaction Server window. Refer to the TRANCODE service attribute *URL setting in the *Custom Connect Facility Administration and Programming Guide*.
Input and Output store	Variable-length data used by WEBAPPSUPPORT, WEBPCM, or both to process the message. The application should not directly modify any data in this field but should use the functions in the WEBAPPSUPPORT library to view or modify the contents. The length of the entire message can be determined from the WEBAPPSUPPORT procedure GET_MESSAGE_LENGTH.

HTTP Tutorial

The following is a simplified overview of what HTTP messages contain.

Request Line	HTTP Headers	Content

- The Request Line identifies the method used (GET, POST, PUT), the identification of the resource (also known as the URL), an optional query string, and the HTTP protocol level.
- The HTTP Headers are usually name and value pairs that identify such things as client capabilities (for example, language) and restrictions on the request (for example, last modified date).
- Content on requests is used with POST (forms) or PUT (upload file), and is the data for the request. It is optional.

***Note**: Except for the Content part, HTTP messages are in U.S. ASCII text characters. Content format varies with the Content-Type.*

The following text is a sample HTTP request, as sent by a browser. This request is a read access (GET) of the /docs/ directory. The application does not see this raw format. Carriage returns and line feeds are replaced with ~ and ^ respectively. No input data is shown in this example.

```
GET /docs/ HTTP/1.0~^If-Modified-Since: Fri, 21 Aug 1998 13:08:34 GMT;
length=37298~^Referer: http://asn035:2488/~^Connection: Keep-Alive~^User-
Agent: Mozilla/4.5 [en] (Win95; I)~^Host: asn035:2488~^Accept: image/gif,
image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*~^Accept-
Encoding: gzip~^Accept-Language: en~^Accept-Charset: iso-8859-1,*,utf-
8~^Authorization: Basic ZmZmZmZmZmZmZmZmZmZm~^~^
```

HTTP Response (Server to Client)

The following is a simplified overview of what HTTP responses contain.

Status Line	HTTP Headers	Content

- The Status Line contains the protocol level, a status code, and explanatory text.
- The HTTP Headers are usually name-value pairs that identify such things as server identification and content information (for example, length and format).
- The Content is the data for the response and is usually HTML.

The following text is a sample HTTP response, as sent by a server. Carriage returns and line feeds are replaced with ~ and ^ respectively. The Content data (HTML) has been truncated.

```
HTTP/1.0 200 Document follows~^Last-Modified: Fri, 21 Aug 1998 13:08:34
GMT~^Mime-Version: 1.0~^Server: ClearPath NX/Atlas Web Server 5.0~^Date:
Thu, 28 Jan 1999 01
:16:30 GMT~^Content-Length: 37298~^Content-Type: text/html~^Connection:
Keep-Alive
~^Keep-Alive: timeout=10~^~^<HTML><HEAD><META HTTP-EQUIV="Content-Type"
CONTENT="t
ext/html; charset=iso-8859-1"><META NAME="Author" CONTENT="Mitchell
Fisher"><META NAME="GENERATOR" CONTENT="Mozilla/4.03 [en] (Win95; I)
[Netscape]"><TITLE>NX/Atlas
 Web Server Administration Site</TITLE></HEAD><BODY TEXT="#000000"
BGCOLOR="#FFFFFF">
<CENTER><A HREF="docs/"><IMG SRC="atlas.jpg" ALT="ClearPath MCP 8.0
NX/Atlas Web Server Documentation" BORDER=0 HEIGHT=151
WIDTH=426></A></CENTER> <CENTER><TABLE BORDER=0 COLS=1 WIDTH="60%"
><TR><TD><DIR><LI><FONT FACE="Arial,Helvetica"><
```

Related Information

The standards organization, W3C, maintains the HTTP specification, which you can download from http://www.w3.org/ (look for HTTP). This specification defines the HTTP format, status codes, headers, and so forth that make up an HTTP message. Also, books on CGI programming can provide information about using HTTP headers.

Sample Applications

This subsection contains COBOL and ALGOL examples.

COBOL Examples

Basic Example

This example shows a COBOL application that uses the WEBPCM. It is a basic COBOL Direct Window application that generates its own HTML response. It gets one HTTP header from the WEBAPPSUPPORT library to put into the response.

```
This is a COBOL74 sample COMS program that serves Web users
* via NX/Atlas Web Server. It assumes that strings sent and
* received are padded by blanks and are in EBCDIC charset.
* (This compiles with both COBOL74 and COBOL85.)
*
 IDENTIFICATION DIVISION.
 PROGRAM-ID. TEST-WEBAPP.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
  WORKING-STORAGE SECTION.
  01 COMS-MESSAGE-AREA  PIC X(5000).
  01 WEB-MSG  REDEFINES COMS-MESSAGE-AREA.
    03 WEB-TRANCODE  PIC  X(17).
  01 HTML-HEAD    PIC X(54) VALUE IS
     "<HTML><HEAD><TITLE>Web App Sample</TITLE></HEAD><BODY>".
  01 HTML-TAIL    PIC X(14) VALUE IS
     "</BODY></HTML>".
  01 HTML-BUFF    PIC X(5000).
  77 HTML-START    PIC 9(11)  BINARY VALUE IS 1.
  77 HTML-LENGTH   PIC 9(11)  BINARY VALUE IS 0.
  77 MSG-COMPLETE   PIC 9(11)  BINARY VALUE IS 1.
  01 APP-PATH     PIC X(17) VALUE IS "$APPLICATION-PATH".
  01 APP-PATH-VALUE  PIC X(255).
  77 PTR       PIC 9(11)  BINARY VALUE IS 1.
  77 MSG-LENGTH    PIC 9(11)  BINARY VALUE IS 0.
  77 WEB-RESULT    PIC S9(11)  BINARY VALUE IS 0.
    88 WEB-OK     VALUE 1.
    88 WEB-NO-OP    VALUE 0.
    88 WEB-BADID    VALUE -1.
    88 WEB-DENIED   VALUE -2.
    88 WEB-SOFTERR   VALUE -3.
    88 WEB-NOTAVAIL  VALUE -4.
  COMMUNICATION SECTION.
  INPUT HEADER COMS-IN.
  OUTPUT HEADER COMS-OUT.


PROCEDURE DIVISION.
 CONTROLLER.
  CHANGE ATTRIBUTE LIBACCESS OF "WEBAPPSUPPORT" TO BYFUNCTION.
  CHANGE ATTRIBUTE LIBACCESS OF "DCILIBRARY" TO BYINITIATOR.
  ENABLE INPUT COMS-IN KEY "ONLINE".
  PERFORM PROCESS-INPUT THRU PROCESS-INPUT-EXIT
   UNTIL STATUSVALUE OF COMS-IN = 99.
 END-OF-TASK.
  STOP RUN.
```

```
 PROCESS-INPUT.
  RECEIVE COMS-IN MESSAGE INTO COMS-MESSAGE-AREA.
  IF STATUSVALUE OF COMS-IN NOT = 99
   IF NOT FUNCTIONSTATUS OF COMS-IN < 0
*
*    - We have a message to handle -
*
    IF WEB-TRANCODE = "ATLAS-HTTP"
*
*    - Message came from a web user -
*
    CALL "GET_HEADER OF WEBAPPSUPPORT"
     USING  COMS-MESSAGE-AREA, APP-PATH, APP-PATH-VALUE
     GIVING WEB-RESULT
    IF WEB-OK
*
*     - Good result for getting the header, build the HTML -
*
     MOVE 1 TO PTR
     STRING HTML-HEAD,
         "Your Application Path is " DELIMITED BY SIZE,
         APP-PATH-VALUE DELIMITED BY " ",
         HTML-TAIL   DELIMITED BY SIZE
      INTO HTML-BUFF WITH POINTER PTR
     SUBTRACT 1 FROM PTR GIVING HTML-LENGTH
*
*     - Update the output message with the HTML -
*

     CALL "SET_CONTENT OF WEBAPPSUPPORT"
      USING  COMS-MESSAGE-AREA, HTML-BUFF, HTML-LENGTH, HTML-START,
          HTML-LENGTH,    MSG-COMPLETE
      GIVING WEB-RESULT
     IF WEB-OK
*
*      - We need to know how many bytes to send, get the
*       length and send the message back to the station -
*
      CALL "GET_MESSAGE_LENGTH OF WEBAPPSUPPORT"
       USING  COMS-MESSAGE_AREA, MSG-LENGTH
       GIVING WEB-RESULT
      MOVE MSG-LENGTH     TO TEXTLENGTH   OF COMS-OUT
      MOVE 1         TO DESTCOUNT    OF COMS-OUT
      MOVE STATION OF COMS-IN TO DESTINATIONDESG OF COMS-OUT
      SEND COMS-OUT FROM COMS-MESSAGE-AREA.
  PROCESS-INPUT-EXIT.
   EXIT.
```

Using an External HTML File

You might want to use an HTML file that is external to the program, rather than hard code HTML in the application. With this method, the application does not need to be recompiled for each HTML change, which can happen frequently. You might also want to use an HTML editing tool instead of hand coding the HTML.

If the HTML file is static, not needing any changes at the time of the response, the application can direct the browser to read the file with a call to SET_REDIRECT, or read the file and return the contents in the response.

In most cases, some dynamic value might be needed for the HTML, such as data from a database. In that case, you can use the WEBAPPSUPPORT procedure MERGE_FILE_AND_DATA (or MERGE_DATA).

Here are example COBOL code portions that demonstrate its use.

```
01 FILE-NAME  PIC X(255).
  01 REPLACE-DATA-BUFF.
    03 REPLACE-DATA OCCURS 5 TIMES.
       05 RD-NAME  PIC X(20).
       05 RD-VALUE  PIC X(30).
  01 REPLACED-BUFF   PIC X(4000).
  01 RD-BUFF-LENGTH   PIC 9(11) BINARY.
  01 ITEM-COUNT     PIC 9(11) BINARY.
  01 ITEM-NAME-LENGTH  PIC 9(11) BINARY VALUE IS 20.
  01 ITEM-VALUE-LENGTH PIC 9(11) BINARY VALUE IS 30.
  01 TRIM-BLANKS    PIC 9(11) BINARY VALUE IS 1.
   :
*
* load REPLACE-DATA fields:
*
   MOVE SPACES TO REPLACE-DATA-BUFF.
   MOVE "Date"        TO RD-NAME (1).
   MOVE "September 14, 1998" TO RD-VALUE(1).
   MOVE "State"       TO RD-NAME (2).
   MOVE "Alabama"      TO RD-VALUE(2).
   MOVE "Abbrev"       TO RD-NAME (3).
   MOVE "AL"         TO RD-VALUE(3).
   MOVE "State"       TO RD-NAME (4).
   MOVE "Alaska"       TO RD-VALUE(4).
   MOVE "Abbrev"       TO RD-NAME (5).
   MOVE "AK"         TO RD-VALUE(5).
   MOVE 5          TO ITEM-COUNT.
   :
   MOVE "*PUBLIC/WWWROOT/WEBAPP1/""RESPONSE.HTM""" TO FILE-NAME.
   CALL "MERGE_FILE_AND_DATA OF WEBAPPSUPPORT"
    USING CHARSET-EBCDIC,  NO-STRING-TERMINATE,
       FILE-NAME,     REPLACE-DATA-BUFF, ITEM-COUNT,
       ITEM-NAME-LENGTH, ITEM-VALUE-LENGTH, TRIM-BLANKS,
       REPLACED-BUFF,   RD-BUFF-LENGTH
    GIVING WEB-RESULT.
   IF WEB-OK
    MOVE 1 TO PTR
    STRING HTML-HEAD DELIMITED BY SIZE,
       REPLACED-BUFF   FOR     RD-BUFF-LENGTH,
       HTML-TAIL     DELIMITED BY SIZE
      INTO OUT-BUFF WITH POINTER PTR.
*
* (call SET_CONTENT with OUT-BUFF, send response)
*
```

Sample HTML for This Example:

```
<HTML><HEAD><TITLE>States</TITLE></HEAD>
<BODY TEXT="#000000" BGCOLOR="#FFFFFF">
<CENTER><IMG SRC="logo.jpg" ALT="ACME Logo" BORDER=0 HEIGHT=151
WIDTH=426>
<P>States & Their Abbreviations:</P><BR>
<TABLE BORDER=0 COLS=2 WIDTH="70%">$REPLACE-BEGIN
```

```
<TR><TD>$REPLACE=State</TD>     <TD>$REPLACE=Abbrev</TD></TR>$REPLACE-END
</TABLE><BR>$REPLACE=Date</CENTER></BODY></HTML>
```

Resulting HTML from the Previous HTML:

```
<HTML><HEAD><TITLE>States</TITLE></HEAD>
<BODY TEXT="#000000" BGCOLOR="#FFFFFF">
<CENTER><IMG SRC="logo.jpg" ALT="ACME Logo" BORDER=0 HEIGHT=151
WIDTH=426>
<P>States & Their Abbreviations:</P><BR>
<TABLE BORDER=0 COLS=2 WIDTH="70%">
<TR><TD>Alabama</TD><TD>AL</TD></TR>
<TR><TD>Alaska</TD><TD>AK</TD></TR>
</TABLE><BR>September 14, 1998</CENTER></BODY></HTML>
```

In this example, the fields passed are not in the same order as the tag fields in the HTML file. For the most efficient solution, the data fields supplied should be in the same order as the HTML file.

ALGOL Examples

ALGOL Include File

Included with the release is an include file for ALGOL programs that declares the WEBAPPSUPPORT library, its procedures, and some useful DEFINEs.

Basic Example

This example shows a basic ALGOL Remote File application that generates its own HTML response. It gets two HTTP headers from the WEBAPPSUPPORT library to put into the response.

```
BEGIN % Sample ALGOL application that supports Web users.
    % Uses the Remote File interface.
    % Uses ASCII strings, with strings terminated by null byte.
    % The following INCLUDE file contains the WEBAPPSUPPORT library
    % declaration and all of its procedures.
$$INCLUDE "SYSTEM/CCF/WEBPCM/WEBAPPSUPPORT/INCLUDE/ALGOL"
EBCDIC VALUE ARRAY   WebTrancodeText7    (7"ATLAS-HTTP"    47"00"),
                     remoteUserHdrV7    ((7"$REMOTE-USER"  47"00"),
                     applicationPathHdrV7    (7"$APPLICATION-PATH"
47"00");
EBCDIC ARRAY applicationPathValue [0: 255];
BOOLEAN   done;
INTEGER   fs, lenRead;
DEFINE    maxRecSize  = 9168 #;     % remote file max
EBCDIC ARRAY outputArray     [0: maxRecSize];
INTEGER   messageLength;
POINTER   p;
FILE     remoteFile (KIND=REMOTE, FRAMESIZE=8, MYUSE=IO,
            MAXRECSIZE=maxRecSize);
EBCDIC ARRAY remoteUserValue   [0: 100];
EBCDIC ARRAY transId       [0: webTrancodeLen];
INTEGER   webAppResult;
EBCDIC ARRAY webMessage      [0: maxRecSize];
DEFINE
  htmlHead =
   7"<HTML><HEAD><TITLE>ALGOL Web App Sample</TITLE></HEAD><BODY>" #,
  htmlTail = 7"</BODY></HTML>" # ;
%        %---------------
   PROCEDURE processWebInput ;
     %   %---------------
     % We have a Web message, get two headers and build
     % a response.
   BEGIN
     webAppResult := get2Headers
      (webMessage, applicationPathHdr7, applicationPathValue,
            remoteUserHdr7,    remoteUserValue   );
     IF webAppResult = web_Ok
     THEN % got the two headers


BEGIN
        REPLACE p:outputArray [0] BY
         htmlHead, 7"<P>Your application path is ",
         applicationPathValue[0] UNTIL = 47"00", 7"<BR>",
         7"Your user name is ",
         remoteUserValue[0] UNTIL = 47"00", 7".</P>",
```

```
         htmlTail, 47"00";
        % now place the HTML into the output message
        webAppResult := setContent
                 (webMessage, outputArray, 0, OFFSET(p), TRUE);
        IF webAppResult = web_Ok
        THEN % get message length and write message
         BEGIN
           webAppResult := getMessageLength
                       (webMessage, messageLength);
           WRITE (remoteFile, messageLength, webMessage);
         END; % setContent = web_Ok
      END; % get2Headers = web_Ok
   END OF processWebInput;
%------------------------ begin Main Program ---------------------------
  OPEN (remoteFile, OFFER);
  DO
   CASE WAIT (remoteFile.CHANGEEVENT, remoteFile.INPUTEVENT)
   OF BEGIN
    1: BEGIN % CHANGEEVENT
       fs  := remoteFile.FILESTATE;
       done := (  fs = VALUE(CLOSED   )
           OR fs = VALUE(DEACTIVATED) );
     END; % CHANGEEVENT
    2: BEGIN % INPUTEVENT
       REPLACE webMessage [webInTrancodeIx]
        BY " " FOR webTrancodeLen;
       READ (remoteFile, maxRecsize, webMessage);;
       IF webMessage [webInTrancodeIx]
        = webTrancodeText7 FOR webTrancodeLen
       THEN % trancode text tells us this is a Web message
        processWebInput
       ELSE
        ; % non-Web input ...
     END; % INPUTEVENT
   END  % case WAIT
  UNTIL done;
END.
```

Section 3
WEBAPPSUPPORT Library Interface

Overview

The WEBAPPSUPPORT library is a library provided with the Custom Connect Facility (CCF) release. The interface enables applications to call procedures in this library in order to perform the following tasks:

- Process HTTP requests and return HTTP responses as WEBPCM applications
- Parse, create, modify, and transform XML documents
- Make HTTP requests to HTTP servers
- Compress/decompress data using the DEFLATE compression method

How Procedure Name Indicates Application Language

The procedures are described here as they are declared in the WEBAPPSUPPORT library. You can name a procedure based on the language you are using.

- For COBOL and AB Suite applications, use underscores in the name (for example, CREATE_KEY).
- For EAE applications, use dashes in the name (for example, CREATE-KEY).
- For applications written in other languages, especially ALGOL, do not use underscores or dashes in the name (for example, createKey).

WEBAPPSUPPORT Connection Library Interface

The procedures documented in this guide are available to ALGOL/NEWP applications through a Connection Library interface. The interface enables programs that call into WEBAPPSUPPORT to not be forcibly terminated if WEBAPPSUPPORT is terminated, such as by an operator issuing a DS command.

The INTERFACENAME library parameter is “WEBAPPSUPPORTCL”.

An approval procedure is not used.

Note: *Applications that use objects stored in WEBAPPSUPPORT, such as XML document tags or HTTP Client objects must be able to handle the sudden loss of those objects when the WEBAPPSUPPORT library delinks.*

Because the Connection library definition is unique to each application, the WEBAPPSUPPORT include file (*SYSTEM/CCF/WEBPCM/WEBAPPSUPPORT/INCLUDE/ALGOL) is not updated with a Connection Library definition. The following code is sample ALGOL code.

```
TYPE CONNECTION BLOCK WEBAPPSUPPORTCL;
BEGIN
    PROCEDURE CHGPROC (CONN_INDEX, NEW_STATE, REASON, ACTOR, IMDSED);
      VALUE            CONN_INDEX, NEW_STATE, REASON,        IMDSED;
      INTEGER          CONN_INDEX, NEW_STATE, REASON;
      TASK                                                ACTOR;
BOOLEAN                                                    IMDSED;
BEGIN
       (change procedure handling code)
    END; % Procedure CHGPROC


  INTEGER PROCEDURE setTracing  (TRACE_ON);
     VALUE                       TRACE_ON;
     BOOLEAN                     TRACE_ON;
  IMPORTED;

    (more imported WEBAPPSUPPORT procedures)

END WEBAPPSUPPORTCL;

WEBAPPSUPPORTCL SINGLE LIBRARY
              CLWEBAPPSUPPORT (% AUTOLINK      = TRUE,
                   LIBACCESS     = BYFUNCTION,
                   FUNCTIONNAME  = "WEBAPPSUPPORT.",
                   INTERFACENAME = "WEBAPPSUPPORTCL.",
                   CHANGE        = CHGPROC            );

 %--- Begin Client Program ---

          RSLT := LINKLIBRARY (CLWEBAPPSUPPORT, DONTWAITFORFILE);

          IF ISVALID (CLWEBAPPSUPPORT.setTracing)
 THEN
    BEGIN
      DISPLAY ("  setTracing by CL is Valid");
      % Use reference procedures to reference the procedure to use:
      setTracingP := CLWEBAPPSUPPORT.setTracing;
    END
  ELSE  % older WEBAPPSUPPORT, use server library interface
     setTracingP := setTracing;
  setTracingP (TRUE);
```

WEBAPPSUPPORT EAE Interface

The procedures documented in this guide are also available to applications through an EAE interface. The following table describes the notes that appear in the parameter descriptions of these procedures.

Notes	Description
[bin]	The field contains binary data that the application should not examine or display. The application can write LOW-VALUES into the field to set a null value.
[longa]	The field contains alphanumeric data in the application's character set. If the first character is set to a space character, the field is assumed null or empty.

The RESULT field contains the procedure result.

Variable size parameters (usually EBCDIC array parameters) are used with a size field set by the EAE application that precedes the parameter, which specifies the size of the variable size parameter. For example:

```
SD   SOURCE-SIZE         N5   SOURCE size, for example, 10000

SD   SOURCE              An   [longa]
```

The application sets the SOURCE-SIZE field to the value 10000, and then sets the size of SOURCE to 10000 bytes.

For the best performance, use **256** or **2048** for variable size parameters when possible. If a different size is required, try to use that same size most of the time, for example, 10000.

WEBAPPSUPPORT General Parameters File

At initialization, the WEBAPPSUPPORT library refers to an optional parameters file to control general operation if that file exists. This parameters file operates similarly to the parameters file for XML, *SYSTEM/CCF/WEBAPPSUPPORT/PARAMS/XML.

This parameters file is *SYSTEM/CCF/WEBAPPSUPPORT/PARAMS and exists on the SL WEBAPPSUPPORT family. A sample file is released as *SYSTEM/CCF/WEBAPPSUPPORT/PARAMS/EXAMPLE. It contains these directives.

```
% Sample WEBAPPSUPPORT General Parameters
 % TEMPFAMILY   "DISK";
    % TEMPFAMILY is the MCP family used for temporary files to store
    % large amounts of data.  Defaults to DL SORT family.
 % TERMINATENOUSERS  FALSE;
    % If true, WEBAPPSUPPORT terminates when there are no applications
    % linked to it, otherwise WEBAPPSUPPORT continues running.
 % TRACEFAMILY  "DISK";
    % TRACEFAMILY is the MCP family where WEBAPPSUPPORT trace
    % files are created.  Defaults to SL WEBAPPSUPPORT family.
```

The following four directives are supported.

- TEMPFAMILY directive: The syntax for this directive is

  ```
  TEMPFAMILY <family>
  ```

  where <family> is a quoted string for the family where temporary files are created. The default is the DL SORT family.

- TERMINATENOUSERS directive: The syntax for this directive is

  ```
  TERMINATENOUSERS <boolean>
  ```

  where <boolean> can have the following values:

  – FALSE: continue running when there are no applications linked to the WEBAPPSUPPORT library. This is the default value in MCP release 17.0 or later.

  – TRUE: terminate when there are no applications linked to the WEBAPPSUPPORT library. This is the default value in MCP releases prior to 17.0.

- TRACEFAMILY directive: the syntax for this directive is

  ```
  TRACEFAMILY <family>
  ```

  where <family> is a quoted string for the family where trace files are created. The default is the SL WEBAPPSUPPORT family unless the TRACEFILE file attribute FAMILYNAME in the *SYSTEM/CCF/WEBAPPSUPPORT codefile has been changed.

  The TRACEFAMILY directive overrides the codefile modification of the TRACEFILE file attribute FAMILYNAME attribute.

- TRACEERRORS directive: The syntax for this directive is

  ```
  TRACEERRORS <boolean>
  ```

  where <boolean> can have the following values:

  – FALSE: application procedure errors for all applications are not traced.

  – TRUE: application procedure errors for all applications are traced.

WEBAPPSUPPORT Commands

This WEBAPPSUPPORT library offers an operator interface to manage the functions of the library. You can enter commands to WEBAPPSUPPORT through the CCF WEBPCM module or through Accept commands to the WEBAPPSUPPORT library.

For example, from MARC issuing the command through WEBPCM, the operator would enter the following:

```
NA CCF WEBPCM WEBAPPSUPPORT STATUS
```

The response is returned to the MARC operator.

An example of using an Accept command through the WEBAPPSUPPORT library follows. Assume the mix number of the library is 1234:

```
1234AX STATUS
```

The response is displayed in the system messages.

Syntax

```
── WEBAPPSUPPORT ─┬── [STATUS] ──────────────────────────────────────┤
                  ├─ CONFIG[URATION] ────────────────────────────────┤
                  ├─ GC ─────────────────────────────────────────────┤
                  ├─ JPM <jpmnum> ─── LOG ─┬─ CLOSE ─────────────────┤
                  │                        └─ LEVEL <log level>─────┤
                  ├─ QUIT ───────────────────────────────────────────┤
                  ├─ RESTARTXML ─────────────────────────────────────┤
                  ├─ TRACE ──────────────────────────────────────────┤
                  │  ┌─ + ─┬─────── [ALL] ──┬──[ERRORS]──┐
                  │  └─ - ─┘  ┌─<mix >──┐   │            │
                  │           ├─ DEBUG ─────────────────┤
                  │           └─ DUMP ──────────────────┤
                  └──────── CLOSE ──────────────────────┤
```

Where <log level> is OFF, FATAL, ERROR, INFO, WARN, or DEBUG.

Explanation

WEBAPPSUPPORT STATUS

Displays the status and some configuration for each Java Parser Module (JPM).

WEBAPPSUPPORT CONFIG

Displays for the operator the configuration currently in use in the WEBAPPSUPPORT library and identifies the current TEMPFAMILY and TRACEFAMILY settings.

WEBAPPSUPPORT GC

Returns the number of application stacks that have had their memory returned to the available pool. This form of the command also reduces memory used by the WEBAPPSUPPORT library from processing large XML documents that have been released.

This form of the command performs garbage collection on the library to make memory previously held for applications, which have since have terminated, available to new applications. A CU in the library stack might show much reduction in memory held.

WEBAPPSUPPORT JPM

Enables the operator to control the release of JPM logs and the level of JPM logging.

An operator can cause the current JPM log to be closed and a new log opened. The new log has a different timestamp in the log file name from the old log.

The operator can dynamically change the level of JPM logging. If the JPM terminates and restarts, the JPM uses the log level in its configuration file.

WEBAPPSUPPORT QUIT

Directs the WEBAPPSUPPORT library to terminate when there are zero applications linked.

WEBAPPSUPPORT RESTARTXML

Terminates and restarts XML processing.

This command allows the operator to change the XML configuration and have that change take effect without terminating WEBAPPSUPPORT. WEBAPPSUPPORT waits for XML requests that are being processed to complete, and then close the sockets to the JPMs, rereads the XML configuration file, and restarts XML processing

WEBAPPSUPPORT TRACE

Displays the status of tracing for WEBAPPSUPPORT.

WEBAPPSUPPORT TRACE +
WEBAPPSUPPORT TRACE –

Turns on tracing on and off for WEBAPPSUPPORT.

WEBAPPSUPPORT TRACE + ALL
WEBAPPSUPPORT TRACE – ALL

Turns tracing on and off for all WEBAPPSUPPORT application stacks.

WEBAPPSUPPORT TRACE + <mix number>
WEBAPPSUPPORT TRACE – <mix number>

Turns tracing on and off for specified WEBAPPSUPPORT application stacks.

WEBAPPSUPPORT TRACE + DEBUG
WEBAPPSUPPORT TRACE – DEBUG

Turns tracing and internal library debugging on or off.

WEBAPPSUPPORT TRACE + DUMP
WEBAPPSUPPORT TRACE – DUMP

Turns program dumping on or off for the stack of an application when a software fault occurs in the WEBAPPSUPPORT library.

WEBAPPSUPPORT TRACE + ALL ERRORS

WEBAPPSUPPORT TRACE - ALL ERRORS

WEBAPPSUPPORT TRACE + ERRORS

WEBAPPSUPPORT TRACE - ERRORS

Turns on or off error tracing for all applications.

WEBAPPSUPPORT TRACE + <mix> ERRORS

WEBAPPSUPPORT TRACE - <mix> ERRORS

Turns on or off error tracing for a specific WEBAPPSUPPORT application stack.

WEBAPPSUPPORT TRACE CLOSE

Closes tracing for WEBAPPSUPPORT.

Examples

Example 1

This command displays the status of WEBAPPSUPPORT:

```
NA CCF WEBPCM WEBAPPSUPPORT STATUS

  Unisys Corporation WEBAPPSUPPORT
  Version 53.189.8016 Compiled 02/07/2009 @ 11:47
  Connection To WEBPCM:  Linked
  3 Callers Linked
  XML Parser JPM1:
    Host 192.168.16.21, Port 51117
     1 Sockets Open
    Status:    Available
    Standby:  False
    Version:  53.1.189.8016
    Threads:  Current = 10, Min = 10, Max = 40
    Logging:  Level = Warn, File = logs/log.txt
    Documents Parsed/Transformed = 41
    JVM:
      Version:  1.6.0_07
      Free = 96 MB, Total = 127 MB, Max = 511 MB
  XML Parser JPM2:
    Host 192.168.16.31, Port 51117
     1 Sockets Open
    Status:    Available
    Standby:  True
    Version:  53.1.189.8016
    Threads:  Current = 10, Min = 10, Max = 40
    Logging:  Level = Warn, File = logs/log.txt
    Documents Parsed/Transformed = 0
    JVM:
      Version:  1.6.0_07
      Free = 96 MB, Total = 127 MB, Max = 511 MB
```

Example 2

This command displays the status of tracing for WEBAPPSUPPORT:

```
NA CCF WEBPCM WEBAPPSUPPORT TRACE
```

```
  Tracing for All Application Stacks Is Off
  Tracing Is On For Specific Stacks: 4456, 4473
  Internal Debug tracing Is Off
  PDUMPS Will Not Be Taken For Faults
  Tracing To File *TRACE/CCF/WEBAPPSUBPPORT/19990623/"141503.TXT" ON
  521HL
```

Example 3

This command closes the open trace file for WEBAPPSUPPORT, tracing continues in a new trace file:

```
NA CCF WEBPCM WEBAPPSUPPORT TRACE CLOSE

  Tracing File *TRACE/CCF/WEBAPPSUBPPORT/yyyymmdd/"141503.TXT" ON 521HL
  Released
```

Example 4

This command turns on TRACE and DEBUG for WEBAPPSUPPORT:

```
NA CCF WEBPCM WEBAPPSUPPORT TRACE + DEBUG

  Trace (Internal) DEBUG Turned On
```

Example 5

This command turns on tracing for all WEBAPPSUPPORT application stacks:

```
NA CCF WEBPCM WEBAPPSUPPORT TRACE + ALL

  Tracing For All Application Stacks Turned On
```

Example 6

This command turns on tracing for WEBAPPSUPPORT application stack 4457:

```
NA CCF WEBPCM WEBAPPSUPPORT TRACE + 4457

  Tracing Turned On for Stack 4457
```

Example 7

This command executes a garbage collection:

```
NA CCF WEBPCM WEBAPPSUPPORT GC

  Garbage Collect Complete, 12 Stacks Cleared
```

Example 8

This command closes the current log for JPM 1 and starts a new log:

```
NA CCF WEBPCM WEBAPPSUPPORT JPM 1 LOG CLOSE
```

Example 9

This command sets the log level for JPM 1 to debug:

```
NA CCF WEBPCM WEBAPPSUPPORT JPM 1 LEVEL DEBUG
```

Example 10

This command restarts XML processing:

```
NA CCF WEBPCM WEBAPPSUPPORT RESTARTXML

  XML Processing Will Be Restarted
```

Example 11

This command returns the WEBAPPSUPPORT library configuration.

```
NA CCF WEBPCM WEBAPPSUPPORT CONFIG

Current Configuration:
   TEMPFAMILY    DISK
   TRACEFAMILY   DISK

  PARSER 1:
    HOST           192.168.16.21
    PORT           51117
    STANDBY        false
    INITIATEJVM    true
    TARGET         1
    JAVAFAMILY     DISK
    JAVAHOMEDIR    JRE6
    JVMATTRS       -server -Xshare:off -XX:+UseParallelGC
                   -XX:ParallelGCThreads=4 -XX:-UseAdaptiveSizePolicy
                   -Xmn458m -Xms1376M -Xmx1376M
    JPMFAMILY      DISK
    JPMHOMEDIR     XMLJPM
    TASKATTRS      MPID=XMLJPM1; FILE STDOUT=(KIND=DISK, PATHNAME=/-
/DISK/DIR/XMLJPM/JPM1/LOGS/STDOUT-$TIME.TXT, EXTMODE=ASCII,
PROTECTION=PROTECTED, UNIQUETOKEN="$"); FILE STDIN=(KIND=DI SK,
PATHNAME=/-/DISK/DIR/XMLJPM/JPM1/LOGS/STDERR-$TIME.TXT,EX TMODE=ASCII,
PROTECTION=PROTECTED, UNIQUETOKEN="$");
```

Returned Result Values for WEBAPPSUPPORT Procedures

All procedures return the same result values unless noted under each procedure.

Value	Description
1	Successful.
0	No-op: Possibilities include that no data is available to return.
–1	Invalid Transaction ID: Possibilities include that the browser user has terminated the connection.
–2	Response not allowed.
–3	Software Error: Possibilities include a corrupted Message Object, a buffer that is too small, or some other fault on the stack. When the buffer being read from or written into is too small, a trace message is written to the trace file and also displayed at the system ODT indicating the likelihood is that the buffer is too small.
–4	Service Unavailable: WEBAPPSUPPORT cannot link to the WEBPCM, or the WEBPCM is not linked to the Web Transaction Server provider.
–15	Character set not available: The CENTRALSUPPORT and CCSFILE installed on the system do not support the character set.
–16	File character set not available: The EXTMODE of the file used is not supported by the CENTRALSUPPORT and CCSFILE installed on the system.
–17	Translation not available: The mapping between the input and output character sets is not supported by the CENTRALSUPPORT and CCSFILE installed on the system.
–18	Buffer too small: The buffer being read from or written into is too small. A trace message is also written to the trace file and is displayed at the system ODT to indicate the likelihood that the buffer is too small. ***Note:*** *This error is reported only if both WEBAPPSUPPORT and the application program are running at Interface Level 2 or higher; otherwise, the buffer too small condition is reported as a software error.*
–20	Maximum length too small: Either the length of a returned name, including any terminating byte, exceeds the MAX_NAME_LEN parameter, or the length of a returned value, including any terminating bye, exceeds the MAX_VALUE_LEN parameter. ***Note:*** *This error is reported only if both WEBAPPSUPPORT and the application program are running at Interface Level 2 or higher; otherwise, the maximum-length-too-small condition is reported as a software error.*

Procedure Groupings

The WEBAPPSUPPORT library procedures are grouped as follows in this section. An explanation about "Using the Trace File" is included at the end of the General Procedures subsection.

- General Procedures
- WEBPCM Procedures
- XML Procedures
- HTTP Client Procedures
- Regular Expressions Procedures

General Procedures

The procedure topics describe the syntax, parameters, and possible return values. Each topic presents the syntax for

- A COBOL85 entry point, which has uppercase characters and underscores

  An example is CREATE_KEY.
- An ALGOL entry point, which has lower-case and upper-case characters and no underscores

  An example is createKey.
- An EAE entry point, which has upper-case characters and dashes

  An example is CREATE-KEY.

  Note: *For more information on EAE and the notes used in the procedure description text of this guide, refer to "WEBAPPSUPPORT EAE Interface" earlier in this section.*

CLEANUP

Causes the library to clean up its structures used for the application when called by the application.

Syntax

```
PROCEDURE CLEANUP;
```

For example, in COBOL at the program exit, use the following syntax:

```
CALL "CLEANUP OF WEBAPPSUPPORT"

PROCEDURE CLEANUP1 (GLB_PARAM);
   EBCDIC ARRAY     GLB_PARAM [0];
```

Parameters

GLB_PARAM has the format:

```
SG-GLB-PARAM GROUP
  SG-PARAM GROUP
    SD   RESULT       S5
```

CREATE_KEY

Creates a key object in WEBAPPSUPPORT.

Each application stack can have up to 65535 key objects stored at once. Key objects in WEBAPPSUPPORT cannot be shared by application stacks. Key objects can be used for multiple encryptions or decryptions

Syntax

```
INTEGER PROCEDURE CREATE_KEY
               (CONTAINER, ALGORITHM, KEY_SIZE, KEY_VALUE, PERMANENT,
                  GENERATE_KEY, SERVICE_NAME, KEY_TAG);
  EBCDIC ARRAY  CONTAINER,      SERVICE_NAME,   KEY_VALUE [0];
  INTEGER                  ALGORITHM, KEY_SIZE,            PERMANENT,
                  GENERATE_KEY,               KEY_TAG;

INTEGER PROCEDURE createKey
               (CONTAINER, ALGORITHM, KEY_SIZE, KEY_VALUE, PERMANENT,
                  GENERATE_KEY, SERVICE_NAME, KEY_TAG);
  VALUE                    ALGORITHM, KEY_SIZE,            PERMANENT,
                  GENERATE_KEY;
  EBCDIC ARRAY  CONTAINER,      SERVICE_NAME,   KEY_VALUE [*];
  INTEGER                  ALGORITHM, KEY_SIZE,            PERMANENT,
                  GENERATE_KEY,               KEY_TAG;



PROCEDURE CREATE-KEY (GLB_PARAM);

   EBCDIC ARRAY       GLB_PARAM [0];
```

Parameters

CONTAINER is a string in the character set of the application that identifies the key container and can be null if the container is to be temporary. The key container either already exists in MCP Cryptography or is created from this procedure call.

ALGORITHM identifies the encryption algorithm to be used. Values are equivalent to the values for the iAlgorithmID parameter to the McpCryptCryptData procedure in MCAPISUPPORT.

KEY_SIZE is the size in bits of the key to use. If the key container exists, this value should be zero.

KEY_VALUE is binary data that is the unencrypted key value and must be KEY_SIZE bits long.

PERMANENT indicates whether or not the created key container should be permanent (1) or temporary (0). Only temporary is supported in Release 14.0.

GENERATE_KEY indicates whether or not to generate a random symmetric key.

- 0 = do not generate a key. Use the KEY_VALUE parameter for the key.
- 1 = generate the key. The generated key value is returned in the KEY_VALUE parameter..

SERVICE_NAME is a string in the character set of the application. If null, the usercode of the applicaion must match the usercode for the key container. If non-null, the service name of the application is checked for matching to the service name of the key container.

KEY_TAG is the returned tag that references the key object in WEBAPPSUPPORT.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	CONTAINER-SIZE	N5	CONTAINER size, for example, 256
SD	CONTAINER	A*n*	[longa]
SD	ALGORITHM	N12	
SD	KEY-SIZE	N12	
SD	KEY-VALUE-SIZE	N5	KEY-VALUE size, for example, 256
SD	KEY-VALUE	A*n*	[longa]
SD	PERMANENT	N5	
SD	GENERATE-KEY	N5	
SD	SERVICE-NAME-SIZE	N5	SERVICE-NAME size, for example, 256
SD	SERVICE-NAME	A*n*	[longa]
SD	KEY-TAG	A6	[bin]

Possible Result Values

In addition to the standard return results, these possible values can be returned.

Value	Description
–120	The maximum keys are stored.
–121	The XML Encryption License Key is required.
–122	The MCAPI is unavailable.
–124	The container name is invalid.
–125	The algorithm is not supported
–126	The asymmetric key container cannot be created.
–127	The key size is not supported.
–129	The key container does not exist.
–130	The key container cannot be accessed.
–131	Permanent key container is not supported.

CURRENT_UTIME

Returns to a COBOL application the current time in the ALGOL TIME(57) format, which is the current time adjusted for UTC. UTC time is used for all Web-related times, such as time fields in HTTP headers.

Syntax

```
INTEGER PROCEDURE CURRENT_UTIME (TIME57);
  REAL                           TIME57;
```

Parameter

TIME57 is the ALGOL TIME(57) value.

DATE_TO_TIME57

Behaves similarly to the HTTP_DATE_TO_INT procedure except that successful conversion returns a real word containing the equivalent time in the ALGOL TIME(57) format.

Syntax

```
INTEGER PROCEDURE HTTP_DATE_TO_TIME57
                          (CHARSET, STRING_TERMINATE,

                             DATE_STRING, DATE_REAL);
  INTEGER                  CHARSET, STRING_TERMINATE;
  EBCDIC ARRAY                DATE_STRING [0];
  REAL                                   DATE_REAL;

INTEGER PROCEDURE httpDateToTime57
                          (CHARSET, STRING_TERMINATE,

                             DATE_STRING, DATE_REAL);
  VALUE                    CHARSET, STRING_TERMINATE;
  INTEGER                  CHARSET, STRING_TERMINATE;
  EBCDIC ARRAY                DATE_STRING [0];
  REAL                                    DATE_REAL;
```

Parameters

CHARSET is the application character set: 0 = EBCDIC, 1 = ASCII.

STRING_TERMINATE indicates if the application terminates its strings with nulls: 0 = FALSE (blanks are used), 1 = TRUE.

DATE_STRING is the date in rfc1123-date, rfc850-date, or asctime-date format as defined in the HTTP specifications. Examples of the three formats are listed respectively below:

```
Fri, 12 Dec 1997 23:59:59 GMT

Friday, 12-Dec-97 23:59:59 GMT

Fri, Dec 12 23:59:59 1997
```

DATE_REAL is the corresponding TIME(57) format real value.

DECODE_BINARY64

Decodes a string of Binary 64-encoded data into the original form.

Syntax

```
INTEGER PROCEDURE DECODE_BINARY64
                   (CHARSET, SOURCE, SOURCE_START, SOURCE_LEN,
                      DEST, DEST_START, DEST_LEN);
  INTEGER           CHARSET,         SOURCE_START, SOURCE_LEN,
                            DEST_START, DEST_LEN;
  EBCDIC ARRAY               SOURCE,
                      DEST [0];

INTEGER PROCEDURE decodeBinary64
                   (CHARSET, SOURCE, SOURCE_START, SOURCE_LEN,
                      DEST, DEST_START, DEST_LEN);
  VALUE             CHARSET,         SOURCE_START, SOURCE_LEN,
                            DEST_START;
  INTEGER           CHARSET,         SOURCE_START, SOURCE_LEN,
                            DEST_START, DEST_LEN;
  EBCDIC ARRAY               SOURCE,
                      DEST [*];

PROCEDURE DECODE-BINARY64 (GLB_PARAM);
   EBCDIC ARRAY            GLB_PARAM [0];
```

Parameters

CHARSET is the MLS character set in which the data in the SOURCE parameter is encoded.

SOURCE is the array containing the Binary 64-encoded data.

SOURCE_START is the zero-based offset into SOURCE and indicates where the encoded data starts.

SOURCE_LEN is the length of the data in SOURCE.

DEST is the array that receives the unencoded data.

DEST_START is the zero-based offset into DEST and indicates where the unencoded data starts.

DEST_LEN is the length of data returned in the DEST parameter.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	CHARSET	N5	
SD	SOURCE-SIZE	N5	SOURCE size, for example, 2048
SD	SOURCE	An	[longa]
SD	SOURCE-START	N5	
SD	SOURCE-LEN	N5	
SD	DEST-SIZE	N5	DEST size, for example, 256
SD	DEST	An	[longa]
SD	DEST-START	N5	
SD	DEST-LEN	N5	

Possible Result Values

In addition to the standard return results, these possible values can be returned.

Value	Description
–35	The procedure call did not specify a field. The start or length parameters contained an error.
–53	Source is not valid Binary 64.

DECODE_UTF8

Decodes a UTF-8 encoded string of characters into a string of characters in the character set specified in the application.

Syntax

```
INTEGER PROCEDURE DECODE UTF8
                     (CHARSET, UTF_STRING, UTF_LEN,
                       DECODED_STRING, DECODED_LEN);
  INTEGER             CHARSET,             UTF_LEN,
                                        DECODED_LEN;
  EBCDIC ARRAY                  UTF_STRING,  DECODED_STRING [0];

INTEGER PROCEDURE decodeUTF8
                     (CHARSET, UTF_STRING, UTF_LEN,
                       DECODED_STRING, DECODED_LEN);
  VALUE               CHARSET,             UTF_LEN;
  INTEGER             CHARSET,             UTF_LEN,
                                        DECODED_LEN;
  EBCDIC ARRAY                  UTF_STRING,  DECODED_STRING [*];

PROCEDURE DECODE-UTF8 (GLB_PARAM);
   EBCDIC ARRAY        GLB_PARAM [0];
```

Parameters

CHARSET is the character set to which you want to decode: 0 = EBCDIC, 1 = ASCII, or values defined in the *MultiLingual System Guide* as Ccsnumbers that are translatable from UCS2. UCS2NT (84) is also supported.

UTF_STRING is the buffer that contains the UTF-8 encoded characters.

UTF_LEN is the length in bytes of UTF_STRING.

DECODED_STRING is the buffer that is to contain the decoded string.

DECODED_LEN is the length in bytes of DECODED_STRING

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	CHARSET	N5	
SD	UTF-STRING-SIZE	N5	UTF-STRING size, for example, 2048
SD	UTF-STRING	A*n*	[longa]
SD	UTF-LEN	N5	
SD	DECODED-STRING-SIZE	N5	DECODED-STRING size, for example, 2048
SD	DECODED-STRING	A*n*	[longa]
SD	DECODED-LEN	N5	

Possible Result Values

In addition to the standard return results, these possible values can be returned.

Value	Description
–15	Character set not available. The CENTRALSUPPORT and CCSFILE installed on the system does not support the character set.
–17	Translation between CHARSET and UCS2 is not available. The mapping between the two character sets is not supported by the CENTRALSUPPORT and CCSFILE installed on the system.
–32	Invalid UTF-8. UTF_STRING contains invalid UTF-8 value.

DECRYPT_DATA

Decrypts data from an array or from an MCP file into an array or MCP file.

Syntax

```
INTEGER PROCEDURE DECRYPT_DATA
                   (SOURCE_TYPE, SOURCE, SOURCE_START, SOURCE_LEN,
                      DEST_TYPE,   DEST,   DEST_START,   DEST_LEN,
                        IV, REMOVE_PAD, KEY_TAG);

  INTEGER           SOURCE_TYPE,         SOURCE_START, SOURCE_LEN,
                      DEST_TYPE,           DEST_START,   DEST_LEN,
                            REMOVE_PAD, KEY_TAG;
  EBCDIC       ARRAY    IV,      SOURCE, DEST [0];

INTEGER PROCEDURE decryptData
                   (SOURCE_TYPE, SOURCE, SOURCE_START, SOURCE_LEN,
                      DEST_TYPE,   DEST,   DEST_START,   DEST_LEN,
                        IV, REMOVE_PAD, KEY_TAG);
  VALUE             SOURCE_TYPE,         SOURCE_START, SOURCE_LEN,
                      DEST_TYPE,           DEST_START,
                            REMOVE_PAD, KEY_TAG;
  INTEGER           SOURCE_TYPE,         SOURCE_START, SOURCE_LEN,
                      DEST_TYPE,           DEST_START,   DEST_LEN,
                            REMOVE_PAD, KEY_TAG;
  EBCDIC       ARRAY    IV,      SOURCE, DEST [*];

PROCEDURE DECRYPT-DATA (GLB_PARAM);
   EBCDIC ARRAY         GLB_PARAM [0];
```

Parameters

SOURCE_TYPE identifies the type of source of the data to be decrypted.

- 1 = the SOURCE parameter contains the data to be decrypted.
- 2 = the SOURCE parameter contains the MCP file name of the data to be decrypted. See the FILENAME_FORMAT option in the SET_OPTION procedure.

SOURCE is the array containing source information. If SOURCE_TYPE is 2, the file name in SOURCE is coded in the character set of the application.

SOURCE_START is a zero-based offset into the SOURCE array and indicates where the supplied information starts.

SOURCE_LEN is the length in bytes of the data in the SOURCE parameter. If SOURCE_TYPE is 2, then SOURCE_LEN can be zero.

DEST_TYPE identifies the type of destination for data to be decrypted.

- 1 = the DEST parameter contains decrypted data on procedure return.
- 2 = the DEST parameter contains the MCP file name to store the decrypted data. See the FILENAME_FORMAT option in the SET_OPTION procedure.

DEST is the array containing destination information. If DEST_TYPE is 2, DEST is coded in the character set of the application.

DEST_START is a zero-based offset into the DEST array and indicates where the supplied information starts.

DEST_LEN is the length in bytes of the data in the DEST parameter. If DEST_TYPE is 2, then DEST_LEN can be zero.

IV is the initialization vector that was used to encrypt the data. The size of the data in the initialization vector depends on the encryption algorithm used.

REMOVE_PAD indicates whether or not to remove padding bytes from the decrypted data.

- 0 = do not remove any pad bytes after decrypting.
- 1 = remove pad bytes after decrypting. The last pad byte added is the number of pad bytes added to the unencrypted data. For example, if the block size of the encryption method is eight, and the length in bytes of the data before being encrypted was five, the data after decryption might be in hexadecimal: x3132333435000003. Also, the resulting length of the returned data is reduced by three, returning five bytes.

KEY_TAG is the key object used to decrypt the data.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	SOURCE-TYPE	N5	
SD	SOURCE-SIZE	N5	SOURCE size, for example, 2048
SD	SOURCE	A*n*	[longa]
SD	SOURCE-START	N5	
SD	SOURCE-LEN	N5	
SD	DEST-TYPE	N5	
SD	DEST-SIZE	N5	DEST size, for example, 2048
SD	DEST	A*n*	[longa]
SD	DEST-START	N5	
SD	DEST-LEN	N12	
SD	IV-SIZE	N5	IV size, for example, 256
SD	IV	A*n*	[longa]
SD	REMOVE-PAD	N5	
SD	KEY-TAG	A6	[bin]

Possible Result Values

In addition to the standard return results, these possible values can be returned.

Value	Description
–121	The XML Encryption Key is required.
0	No-op for any of the following reasons:\ • The destination length is invalid. • The destination length is not supported. • An attribute error occurred creating the file.
–11	The file is not available.
–13	An error occurred setting the file name.
–25	An error occurred writing the file.
–35	The procedure call did not specify a field.
–47	The source length or start is invalid.
–55	The destination start is invalid.
–122	The MCAPI is unavailable.
–123	The key is invalid.

DEFLATE_DATA

Compresses data using the Deflate method defined in RFC 1951. The XML Parser Java Parser Module (JPM) must be available to use this procedure.

If the source of the uncompressed data is an MCP file, that file is not read through the WEBAPPSUPPORT library file cache.

Only stream files are supported for output.

See the SET_OPTION procedure, options DEFLATE_LEVEL and DEFLATE_STRATEGY.

See also the INFLATE_DATA procedure.

Syntax

```
INTEGER PROCEDURE DEFLATE_DATA
               (SOURCE_TYPE, SOURCE, SOURCE_START, SOURCE_LEN,
                  DEST_TYPE, DEST, DEST_START, DEST_LEN,
                    DEFLATE_FORMAT, CRC_TYPE, CRC);
  INTEGER      SOURCE_TYPE,         SOURCE_START, SOURCE_LEN,
                  DEST_TYPE,       DEST_START, DEST_LEN,
                    DEFLATE_FORMAT, CRC_TYPE, CRC;
  EBCDIC ARRAY              SOURCE,
                            DEST [0];
```

```
INTEGER PROCEDURE deflateData
              (SOURCE_TYPE, SOURCE, SOURCE_START, SOURCE_LEN,
                 DEST_TYPE, DEST, DEST_START, DEST_LEN,
                   DEFLATE_FORMAT, CRC_TYPE, CRC);
  VALUE        SOURCE_TYPE,         SOURCE_START, SOURCE_LEN,
                 DEST_TYPE,       DEST_START,
                   DEFLATE_FORMAT, CRC_TYPE;
  INTEGER      SOURCE_TYPE,         SOURCE_START, SOURCE_LEN,
                 DEST_TYPE,       DEST_START, DEST_LEN,
                   DEFLATE_FORMAT, CRC_TYPE, CRC;
  EBCDIC ARRAY              SOURCE,
                            DEST [*];

PROCEDURE DEFLATE-DATA (GLB_PARAM);
   EBCDIC ARRAY         GLB_PARAM [0];
```

Parameters

SOURCE_TYPE identifies the type of the source for the data to be compressed.

- 1 = SOURCE contains the data to be compressed.
- 2 = SOURCE contains the MCP file name of the data to be compressed. The name is in display format or pathname format. See the FILENAME_FORMAT option in the SET_OPTION procedure.

SOURCE is the array containing the uncompressed data or the name of the file in the application character set that contains the uncompressed data.

SOURCE_START is the zero-based offset into SOURCE array and indicates where the uncompressed data or file name starts.

SOURCE_LEN is the length in bytes of the data in SOURCE.

DEST_TYPE identifies the type of the destination for the compressed data.

- 1 = DEST contains the compressed data.
- 2 = DEST contains the MCP file name of the file to which the compressed data is written. The name is in display format or pathname format. The file is created new, and an existing file of the same name is overwritten. See the FILENAME_FORMAT and FILE_ATTRIBUTES options in the SET_OPTION procedure.
- 3 = DEST contains the MCP file name of the file to which the compressed data is written. The name is in display format or pathname format. The file must already exist, and the compressed data is appended. See the FILENAME_FORMAT and FILE_ATTRIBUTES options in the SET_OPTION procedure.

DEST is the array that receives the compressed data or contains the name of the file in the application character set to which the compressed data is to be written.

DEST_START is the zero-based offset into DEST array and indicates where the compressed data or file name starts.

DEST_LEN is the length in bytes of the compressed data, including the headers.

DEFLATE_FORMAT is the format of the compressed output:

- 1 = zlib format as defined in RFC 1950.
- 2 = gzip format as defined in RFC 1952. A filename is not placed in the header, and the MTIME field is zero.

CRC_TYPE is the type of CRC to calculate.

- 0 = no CRC calculation
- 1 = the Java CRC32
- 2 = the Java Adler32

CRC is the CRC value for the uncompressed data.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	SOURCE-TYPE	N5	
SD	SOURCE-SIZE	N5	SOURCE size, for example, 2048
SD	SOURCE	A*n*	[longa]
SD	SOURCE-START	N5	
SD	SOURCE-LEN	N5	
SD	DEST-TYPE	N5	
SD	DEST-SIZE	N5	DEST size, for example, 2048
SD	DEST	A*n*	[longa]
SD	DEST-START	N5	
SD	DEST-LEN	N12	
SD	DEFLATE-FORMAT	N5	
SD	CRC-TYPE	N5	
SD	CRC	N12	

Possible Result Values

In addition to the standard return results, these possible values can be returned.

Value	Description
–47	An unsupported source value was supplied.
-48	Unable to open a socket to a JPM
-49	Unable to write to the JPM
-50	Unable to read from the JPM
-54	The JPM is not configured.
-55	An unsupported destination value was supplied.
-57	The JPM does not support this function.

ENCODE_BINARY64

Encodes an array of data into Binary 64.

Syntax

```
INTEGER PROCEDURE ENCODE_BINARY64
                   (CHARSET, SOURCE, SOURCE_START, SOURCE_LEN,
                      DEST, DEST_START, DEST_LEN);
  INTEGER           CHARSET,         SOURCE_START, SOURCE_LEN,
                            DEST_START, DEST_LEN;
  EBCDIC ARRAY               SOURCE,
                      DEST [0];

INTEGER PROCEDURE encodeBinary64
                   (CHARSET, SOURCE, SOURCE_START, SOURCE_LEN,
                      DEST, DEST_START, DEST_LEN);
  VALUE             CHARSET,         SOURCE_START, SOURCE_LEN
                            DEST_START;
  INTEGER           CHARSET,         SOURCE_START, SOURCE_LEN,
                            DEST_START, DEST_LEN;
  EBCDIC ARRAY               SOURCE,
                      DEST [*];

PROCEDURE ENCODE-BINARY64 (GLB_PARAM);
   EBCDIC ARRAY            GLB_PARAM [0];
```

Parameters

CHARSET is the MLS character set in which the data in the DEST parameter is encoded to.

SOURCE is the array containing the data to be encoded.

SOURCE_START is the zero-based offset into SOURCE array and indicates where the unencoded data starts.

SOURCE_LEN is the length of the data in the SOURCE parameter.

DEST is the array that receives the encoded data.

DEST_START is the zero-based offset into DEST array and indicates where the encoded data starts.

DEST_LEN is the length of the data returned in the DEST parameter.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	CHARSET	N5	
SD	SOURCE-SIZE	N5	SOURCE size, for example, 2048
SD	SOURCE	A*n*	[longa]
SD	SOURCE-START	N5	
SD	SOURCE-LEN	N5	
SD	DEST-SIZE	N5	DEST size, for example, 2048
SD	DEST	A*n*	[longa]
SD	DEST-START	N5	
SD	DEST-LEN	N5	

Possible Result Values

In addition to the standard return results, these possible values can be returned.

Value	Description
–35	The procedure call did not specify a field. The start or length parameters contained an error.

ENCODE_UTF8

Encodes a string of characters in the character set specified in the application into a UTF-8 encoded string.

Syntax

```
INTEGER PROCEDURE ENCODE UTF8
                    (CHARSET, INPUT_STRING, INPUT_LEN,
                       UTF_STRING, UTF_LEN);
  INTEGER            CHARSET,               INPUT_LEN,
                                   UTF_LEN;
   EBCDIC ARRAY               INPUT_STRING,  UTF_STRING [0];

INTEGER PROCEDURE encodeUTF8
                    (CHARSET, INPUT_STRING, INPUT_LEN,
                        UTF_STRING, UTF_LEN);
  VALUE              CHARSET,             INPUT_LEN;
  INTEGER            CHARSET,             INPUT_LEN,
                                    UTF_LEN;
  EBCDIC ARRAY                INPUT_STRING,  UTF_STRING [*];

PROCEDURE ENCODE-UTF8 (GLB_PARAM);
   EBCDIC ARRAY        GLB_PARAM [0];
```

Parameters

CHARSET is the character set INPUT_STRING. 0 = EBCDIC, 1 = ASCII, or values defined in the *MultiLingual System Guide* as Ccsnumbers that are translatable from UCS2. UCS2NT (84) is also supported.

INPUT_STRING is the buffer that contains the string to be encoded.

INPUT_LEN is the length in bytes of INPUT_STRING.

UTF_STRING is the buffer that is to contain the UTF-8 encoded characters.

UTF_LEN is the length in bytes of UTF_STRING.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	CHARSET	N5	
SD	INPUT-STRING-SIZE	N5	INPUT-STRING size, for example, 2048
SD	INPUT-STRING	A*n*	[longa]
SD	INPUT-LEN	N5	
SD	UTF-STRING-SIZE	N5	UTF-STRING size, for example, 2048
SD	UTF-STRING	A*n*	[longa]
SD	UTF-LEN	N5	

Possible Result Values

In addition to the standard return results, the possible values can be returned.

Value	Description
–15	Character set not available. The CENTRALSUPPORT and CCSFILE installed on the system do not support the character set.
–17	Translation between CHARSET and UCS2 is not available. The mapping between the two character sets is not supported by the CENTRALSUPPORT and CCSFILE installed on the system.

ENCRYPT_DATA

Encrypts data from an array or from an MCP file into an array or MCP file. You can use this procedure with the CREATE_CIPHER_REFERENCE procedure to build an XML document that references the encrypted data at a URL.

See also the ENCRYPT_XML_DOCUMENT procedure.

Syntax

```
INTEGER PROCEDURE ENCRYPT_DATA
                   (SOURCE_TYPE, SOURCE, SOURCE_START, SOURCE_LEN,
                      DEST_TYPE,   DEST,   DEST_START,   DEST_LEN,
                        USE_IV,  IV,  PAD,  KEY_TAG);
  INTEGER           SOURCE_TYPE,         SOURCE_START, SOURCE_LEN,
                      DEST_TYPE,           DEST_START,   DEST_LEN,
                        USE_IV,       PAD,  KEY_TAG;
  EBCDIC ARRAY                   SOURCE, DEST, IV [0];

INTEGER PROCEDURE encryptData
                   (SOURCE_TYPE, SOURCE, SOURCE_START, SOURCE_LEN,
                      DEST_TYPE,   DEST,   DEST_START,   DEST_LEN,
                        USE_IV,  IV,  PAD,  KEY_TAG);
  VALUE             SOURCE_TYPE,         SOURCE_START, SOURCE_LEN,
                      DEST_TYPE,           DEST_START,
                        USE_IV,       PAD,  KEY_TAG;
  INTEGER           SOURCE_TYPE,         SOURCE_START, SOURCE_LEN,
                      DEST_TYPE,           DEST_START,   DEST_LEN,
                        USE_IV,       PAD,  KEY_TAG;
  EBCDIC ARRAY                   SOURCE, DEST, IV [*];

PROCEDURE ENCRYPT-DATA (GLB_PARAM);
   EBCDIC ARRAY         GLB_PARAM [0];
```

Parameters

SOURCE_TYPE identifies the type of source of the data to be encrypted.

- 1 = the SOURCE parameter contains the data to be encrypted.
- 2 = the SOURCE parameter contains the MCP file name of the data to be encrypted. See the FILENAME_FORMAT option in the SET_OPTION procedure.

SOURCE is the array containing source information. If SOURCE_TYPE is 2, the file name in SOURCE is coded in the character set of the application.

SOURCE_START is a zero-based offset into the SOURCE array and indicates where the supplied information starts.

SOURCE_LEN is the length in bytes of the data in the SOURCE parameter. If SOURCE_TYPE is 2, then SOURCE_LEN can be zero.

DEST_TYPE identifies the type of destination for data to be encrypted.

- 1 = the DEST parameter contains encrypted data on procedure return.
- 2 = the DEST parameter contains the MCP file name to store the encrypted data. See the FILENAME_FORMAT option in the SET_OPTION procedure.

DEST is the array containing destination information. If DEST_TYPE is 2, DEST is coded in the character set of the application.

DEST_START is a zero-based offset into the DEST array and indicates where the supplied information starts.

DEST_LEN is the length in bytes of the data in the DEST parameter. If DEST_TYPE is 2, then DEST_LEN can be zero.

USE_IV indicates whether to use the initialization vector supplied by the application or to use an internally generated vector.

- 0 = do not use the IV parameter as the initialization vector. The vector generated is returned in the IV parameter.
- 1 = use the IV parameter as the initialization vector.

IV is the initialization vector. The size of the data in the initialization vector depends on the encryption algorithm used.

PAD indicates whether or not to add padding bytes up to the block size for block encryption algorithms. This parameter is ignored for nonblock encryption algorithms.

- 0 = do not add pad bytes before encrypting.
- 1 = add pad bytes before encrypting to fill out the data to a multiple of the block size. The last pad byte added is the number of pad bytes added to the unencrypted data. For example, if the block size of the encryption method is eight, and the length in bytes of the data to be encrypted is five, the data to be encrypted with padding might be in hexadecimal: x3132333435000003.

KEY_TAG is the key object used to encrypt the data.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	SOURCE-TYPE	N5	
SD	SOURCE-SIZE	N5	SOURCE size, for example, 2048
SD	SOURCE	A*n*	[longa]
SD	SOURCE-START	N5	
SD	SOURCE-LEN	N5	
SD	DEST-TYPE	N5	
SD	DEST-SIZE	N5	DEST size, for example, 2048
SD	DEST	A*n*	[longa]
SD	DEST-START	N5	
SD	DEST-LEN	N12	
SD	USE-IV	N5	
SD	IV-SIZE	N5	
SD	IV	A*n*	IV size, for example, 256
SD	PAD	N5	[longa]
SD	KEY-TAG	A6	[bin]

Possible Result Values

In addition to the standard return results, these possible values can be returned.

Value	Description
–121	The XML Encryption Key is required.
0	No-op for any of the following reasons:\ • The destination length is invalid. • The destination length is not supported. • An attribute error occurred creating the file.
–11	The file is not available.
–13	An error occurred setting the file name.
–25	An error occurred writing the file.
–35	The procedure call did not specify a field.
–47	The source length or start is invalid.
–55	The destination start is invalid.
–122	The MCAPI is unavailable.
–123	The key is invalid.

ESCAPE_TEXT

Encodes the supplied text using different escape functions.

This procedure is useful in protecting the application users from Cross-Site Scripting (XSS) attacks.

Syntax

```
INTEGER PROCEDURE ESCAPE_TEXT
                (ESCAPE_TYPE, CHARSET, ESCAPE_CHARSET,
                   UNESCAPED, UNESCAPED_START, UNESCAPED_LEN,
                     ESCAPED,   ESCAPED_START, ESCAPED_LEN);
    INTEGER     ESCAPE_TYPE, CHARSET, ESCAPE_CHARSET,
                             UNESCAPED_START, UNESCAPED_LEN,
                               ESCAPED_START,   ESCAPED_LEN;
    EBCDIC ARRAY  UNESCAPED, ESCAPED [0];


INTEGER PROCEDURE escapeText
                (ESCAPE_TYPE, CHARSET, ESCAPE_CHARSET,
                   UNESCAPED, UNESCAPED_START, UNESCAPED_LEN,
                     ESCAPED,   ESCAPED_START,   ESCAPED_LEN);
    VALUE        ESCAPE_TYPE, CHARSET, ESCAPE_CHARSET,
                              UNESCAPED_START, UNESCAPED_LEN,
                                ESCAPED_START;
```

```
    INTEGER      ESCAPE_TYPE, CHARSET, ESCAPE_CHARSET,
                              UNESCAPED_START, UNESCAPED_LEN,
                                ESCAPED_START,   ESCAPED_LEN;
    EBCDIC ARRAY   UNESCAPED, ESCAPED [*];


PROCEDURE ESCAPE_TEXT (GLB_PARAM);
   EBCDIC ARRAY        GLB_PARAM [0];
```

Parameters

ESCAPE_TYPE is the type of escape or encoding function to perform:

- 1 = HTML entity encoding. The following character encoding is done:

Character	Encoding
&	&
>	<
>	>
"	"
'	'
/	/

- 2 = Aggressive HTML entity encoding. All non-alphanumeric characters with ASCII values less than 256 are encoded as their ASCII-equivalent in the format &#xHH. This value is useful for encoding data inserted in HTML attributes.
- 3 = JavaScript escape. All non-alphanumeric characters with ASCII values less than 256 are encoded as their ASCII-equivalent in the format \xHH. Use this value only for 8-bit character sets.
- 4 = JavaScript escape to UTF-8. All non-alphanumeric characters are encoded in the format \xHH for each byte in UTF-8 encoding. CHARSET must be translatable to UCS2.
- 5 = JavaScript escape to UTF-16. All non-alphanumeric characters are encoded in the format \uHHHH for each character. CHARSET must be translatable to UCS2.
- 6 = CSS escape. All non-alphanumeric characters are encoded in the format \HH, where HH is the Unicode value up to six hexadecimal digits long, with a space character added after the last H character if the following character is a hexadecimal character and HH is less than six digits long. CHARSET must be translatable to UCS2.
- 7 = URL escape to UTF-8. All non-alphanumeric characters are encoded in the format %HH for each byte in UTF-8 encoding. CHARSET must be translatable to UCS2.

CHARSET is the data character set value of UNESCAPED and ESCAPED as defined in MultiLingual System Administration, Operations, and Programming Guide as Ccsnumbers; for example, 102(CODEPAGE932). The value 2 = UTF-8 is also supported.. See the ESCAPE_TYPE parameter for restrictions on this parameter.

ESCAPE_CHARSET is the data character set value of the data as it will be encoded, as defined in the MultiLingual System Administration, Operations, and Programming Guide as Ccsnumbers, and must be translatable from CHARSET. ESCAPE_CHARSET is ignored for ESCAPE_TYPE values that require translation to UCS2.

For example, if an HTML document is to be encoded in character set iso-8859-1, and the application has the UNESCAPED parameter encoded in Latin1EBCDIC, CHARSET is set to Latin1EBCDIC (12) and ESCAPE_CHARSET is set to Latin1ISO (13). If ESCAPE_TYPE is 2 (aggressive HTML entity encoding), the Latin1EBCDIC character x66 (Latin capital A with tilde) in UNESCAPED is converted to Latin1ISO xC3 and encoded into ESCAPED as &#xC3.

UNESCAPED is the original text.

UNESCAPED_START is a zero-based offset into the UNESCAPED parameter and indicates where the supplied information starts.

UNESCAPED_LEN is the length in bytes of the data in the UNESCAPED parameter. If zero, UNESCAPED contains a string that is terminated by blanks or a null byte.

ESCAPED is the resulting text and should not overwrite UNESCAPED. It is not blank-filled to the right or null byte terminated.

ESCAPED_START is a zero-based offset into the ESCAPED parameter and indicates where the supplied information starts.

ESCAPED_LEN is the length in bytes of the data returned in the ESCAPED parameter.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	ESCAPE-TYPE	N5	
SD	CHARSET	N5	
SD	ESCAPE-CHARSET	N5	
SD	UNESCAPED-SIZE	N5	UNESCAPED size, for example, 2048
SD	UNESCAPED	A*n*	[longa]
SD	UNESCAPED-START	N5	
SD	UNESCAPED-LEN	N5	
SD	ESCAPED-SIZE	N5	ESCAPED size, for example, 2048
SD	ESCAPED	A*n*	[longa]
SD	ESCAPED-START	N5	
SD	ESCAPED-LEN	N5	

Notes:

- *Text containing specially recognized HTML characters that need to be processed by a browser should not be passed to this procedure. For example, <P>If a < b & c > d</P> should not be passed, but If a < b & c > d can be passed.*
- *Blanks are not converted to nonbreaking spaces or plus signs for URLs. Nonbreaking spaces can be coded in the HTML as " " or the hexadecimal character xA0 (NBSP) can be used in HTML. This procedure with ESCAPE_TYPE = 1 then converts xA0 to " ".*
- *The escaped text might be much longer than the unescaped text.*

GENERATE_UUID

Generates a UUID, which is a unique identifier. The following types of UUID are supported:

- A UUID that identifies the MCP system by using its MAC address, varying by the time it was created (version 1).
- A random UUID (version 4).

Syntax

```
INTEGER PROCEDURE GENERATE_UUID
                  (VERSION, FORMAT, UUID, UUID_LENGTH);
  INTEGER          VERSION, FORMAT,       UUID_LENGTH;
  EBCDIC ARRAY                      UUID [0];

INTEGER PROCEDURE generateUUID
                  (VERSION, FORMAT, UUID, UUID_LENGTH);
  VALUE            VERSION, FORMAT;
  INTEGER          VERSION, FORMAT,       UUID_LENGTH;
  EBCDIC ARRAY                      UUID [*];

PROCEDURE GENERATE-UUID (GLB_PARAM);
   EBCDIC ARRAY          GLB_PARAM [0];
```

Parameters

VERSION identifies the type of UUID to be generated.

If the value is 1, UUID is a version 1 UUID as defined in RFC 4122. It is a concatenation of system time and the host MAC address. If UUID_LENGTH is 6 when this procedure is called, the UUID parameter is used as the MAC address; otherwise, the first visible MAC address returned from MCP networking is used.

If the value is 4, the UUID is a version 4 UUID as defined in RFC 4122. It is a random value.

FORMAT identifies the format of the returned UUID parameter.

If the value is 1, the UUID is a 16-byte binary UUID.

If the value is 2, the UUID is a Base 64 encoded string of the 16-byte binary UUID, in the application character set.

If the value is 3, the UUID is a 36-character hexadecimal representation of UUID, including four hyphens, in the application's character set. For example: 13CDBB01-9F77-11E1-8001-08000B00C506.

UUID is the generated UUID value.

UUID_LENGTH is the length in bytes of UUID.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	VERSION	N5	
SD	FORMAT	N5	
SD	UUID-SIZE	N5	UUID size, for example, 256
SD	UUID	A*n*	[longa]
SD	UUID-LENGTH	N5	

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	VERSION or FORMAT parameters are not supported.

HTML_ESCAPE

Parses the supplied string for characters in the extended ASCII character set and for four characters reserved for HTML processing: quotation marks ("), ampersand (&), less than (<), and greater than (>). HTML_ESCAPE returns a string with the special characters replaced by their entity reference sequence for use with HTML text.

For applications at Interface Level 3 or higher, the characters hash mark (#), left parenthesis ((), and right parenthesis ()) are converted to entity references. (See the INTERFACE_VERSION procedure.) This conversion is important for applications because it protects them from cross-site scripting attacks. You should use the HTML_ESCAPE procedure for HTML text that comes from user input and is returned in a response.

The use of the ESCAPE_TEXT procedure is preferred over HTML_ESCAPE.

Syntax

```
INTEGER PROCEDURE HTML_ESCAPE
                (CHARSET, STRING_TERMINATE,
                  UNESCAPED_STRING, ESCAPED_STRING);
  INTEGER       CHARSET, STRING_TERMINATE;
  EBCDIC ARRAY   UNESCAPED_STRING, ESCAPED_STRING [0];

INTEGER PROCEDURE htmlEscape
                (CHARSET, STRING_TERMINATE,
                  UNESCAPED_STRING, ESCAPED_STRING);
  VALUE         CHARSET, STRING_TERMINATE;
  INTEGER       CHARSET, STRING_TERMINATE;
  EBCDIC ARRAY   UNESCAPED_STRING, ESCAPED_STRING [*];
```

Parameters

CHARSET is the application character set: 0 = EBCDIC, 1 = ASCII.

STRING_TERMINATE indicates if the application terminates its strings with nulls:

- 0 =FALSE (blanks are used)
- 1 = TRUE

UNESCAPED_STRING is the original string.

ESCAPED_STRING is the resulting string, and should not overwrite UNESCAPED_STRING.

Notes:

- *Text containing specially recognized HTML characters that need to be processed by a browser should not be passed to this procedure. For example, <P>If a < b & c > d</P> should not be passed, but If a < b & c > d can be passed.*
- *Blanks are not converted to nonbreaking spaces. Nonbreaking spaces can be coded in the HTML as " " or the hexadecimal character xA0 (NBSP) can be used in the HTML. This procedure then converts it to " ".*
- *The escaped string might be up to three times longer than the unescaped string.*
- *The ALGOL strings demonstration shows all the characters handled by the HTML_ESCAPE procedure.*

HTML_UNESCAPE

Parses the supplied string for decimal or entity references and returns a string with the references replaced by their actual ASCII characters.

The following entities are supported:

- Numeric character references in decimal (&#D) and hexadecimal (&#xH) formats.
- Character entity references listed in the following table.

Character Entity Reference	Equivalent ASCII Character
&	&
<	<
>	>
"	"
'	'
Latin1 character references that represent characters in the range xA0 to xFF, such as , ¡	Latin1 characters in the range xA0 to xFF

Syntax

```
INTEGER PROCEDURE HTML_UNESCAPE
               (CHARSET, STRING_TERMINATE,
                 ESCAPED_STRING, UNESCAPED_STRING);
  INTEGER       CHARSET, STRING_TERMINATE;
  EBCDIC ARRAY   ESCAPED_STRING, UNESCAPED_STRING [0];

INTEGER PROCEDURE htmlUnescape
               (CHARSET, STRING_TERMINATE,
                 ESCAPED_STRING, UNESCAPED_STRING);
  INTEGER       CHARSET, STRING_TERMINATE;
  EBCDIC ARRAY   ESCAPED_STRING, UNESCAPED_STRING [*];
```

Parameters

CHARSET is the application character set: 0 = EBCDIC, 1 = ASCII.

STRING_TERMINATE indicates if the application terminates its strings with nulls:

0 = FALSE (blanks are used), 1 = TRUE.

ESCAPED_STRING is the original string.

UNESCAPED_STRING is the resulting string, and should not overwrite ESCAPED_STRING.

HTTP_DATE_TO_INT

Converts an HTTP date or time string to an integer value equal to the number of seconds since year 0 time 0 until the specified date or time. See INT_TO_HTTP_DATE for the reverse function.

Syntax

```
INTEGER PROCEDURE HTTP_DATE_TO_INT
               (CHARSET, STRING_TERMINATE,
                   DATE_STRING, DATE_INT);
  INTEGER       CHARSET, STRING_TERMINATE;
  EBCDIC ARRAY     DATE_STRING [0];
  INTEGER                       DATE_INT;
```

```
INTEGER PROCEDURE httpDateToInt
               (CHARSET, STRING_TERMINATE,
                   DATE_STRING, DATE_INT);
  VALUE         CHARSET, STRING_TERMINATE;
  INTEGER       CHARSET, STRING_TERMINATE;
  EBCDIC ARRAY     DATE_STRING [*];
  INTEGER                       DATE_INT;
```

Parameters

CHARSET is the application character set: 0 = EBCDIC, 1 = ASCII.

STRING_TERMINATE indicates if the application terminates its strings with nulls:

0 = FALSE (blanks are used), 1 = TRUE.

DATE_STRING is the date in rfc1123-date, rfc850-date, or asctime-date format as defined in the HTTP specifications. Examples of the three formats are listed respectively below:

```
Fri, 12 Dec 1997 23:59:59 GMT

Friday, 12-Dec-97 23:59:59 GMT

Fri, Dec 12 23:59:59 1997.
```

DATE_INT is the corresponding integer value.

HTTP_ESCAPE

Parses the supplied string for control or reserved characters and returns a string with the special characters replaced by their ASCII escaped sequence (a percent sign followed by the two-hex digit representation of the character), for use with URIs.

The use of the ESCAPE_TEXT procedure is preferred over HTTP_ESCAPE.

Syntax

```
INTEGER PROCEDURE HTTP_ESCAPE
               (CHARSET, STRING_TERMINATE,
                   UNESCAPED_STRING, ESCAPED_STRING);
  INTEGER       CHARSET, STRING_TERMINATE;
  EBCDIC ARRAY     UNESCAPED_STRING, ESCAPED_STRING [0];

INTEGER PROCEDURE httpEscape
               (CHARSET, STRING_TERMINATE,
                   UNESCAPED_STRING, ESCAPED_STRING);
  VALUE         CHARSET, STRING_TERMINATE;
  INTEGER       CHARSET, STRING_TERMINATE;
  EBCDIC ARRAY     UNESCAPED_STRING, ESCAPED_STRING [*];
```

Parameters

CHARSET is the application character set: 0 = EBCDIC, 1 = ASCII.

STRING_TERMINATE indicates if the application terminates its strings with nulls:

0 = FALSE (blanks are used), 1 = TRUE.

UNESCAPED_STRING is the original string.

ESCAPED_STRING is the resulting string, and should not overwrite UNESCAPED_STRING.

Notes:

- *Blank characters are not converted to plus signs (+) but to their escape sequence instead. So a blank is converted to %20.*
- *The new string might be as long as up to three times the original string.*

HTTP_UNESCAPE

Parses the supplied string for ASCII escape sequences and returns a string with the escape sequences replaced by their actual ASCII characters.

The following entities are supported:

- Numeric character references in decimal (&#D) and hexidecimal (&#xH) formats.
- Character entity references:
  - & to &
  - < to <
  - > to >
  - " to "
  - ' to '
  - Latin1 characters in the range xA0 to xFF (such as , ¡, and ¢)

Syntax

```
INTEGER PROCEDURE HTTP_UNESCAPE
               (CHARSET, STRING_TERMINATE,
                   ESCAPED_STRING, UNESCAPED_STRING);
  INTEGER       CHARSET, STRING_TERMINATE;
  EBCDIC ARRAY     ESCAPED_STRING, UNESCAPED_STRING [0];

INTEGER PROCEDURE httpUnescape
               (CHARSET, STRING_TERMINATE,
                   ESCAPED_STRING, UNESCAPED_STRING);
  VALUE         CHARSET, STRING_TERMINATE;
  INTEGER       CHARSET, STRING_TERMINATE;
  EBCDIC ARRAY     ESCAPED_STRING, UNESCAPED_STRING [*];
```

Parameters

CHARSET is the application character set: 0 = EBCDIC, 1 = ASCII.

STRING_TERMINATE indicates if the application terminates its strings with nulls: 0 = FALSE (blanks are used), 1 = TRUE.

ESCAPED_STRING is the original string.

UNESCAPED_STRING is the resulting string, and should not overwrite ESCAPED_STRING.

INFLATE_DATA

Decompresses data using the Inflate method defined in RFC 1951. The XML Parser JPM must be available to use this procedure.

If the source of the compressed data is an MCP file, that file is not read through the WEBAPPSUPPORT library file cache.

Only stream files are supported for output.

Compressed data that requires a dictionary is not supported.

See the SET_OPTION procedure, INFLATE_METHOD option.

See also the DEFLATE_DATA procedure.

Syntax

```
INTEGER PROCEDURE INFLATE_DATA
               (SOURCE_TYPE, SOURCE, SOURCE_START, SOURCE_LEN,
                  DEST_TYPE, DEST, DEST_START, DEST_LEN,
                    INPUT_FORMAT, CRC_TYPE, CRC);
  EBCDIC ARRAY              SOURCE,
                            DEST [0];
  INTEGER      SOURCE_TYPE,         SOURCE_START, SOURCE_LEN,
                  DEST_TYPE,       DEST_START, DEST_LEN,
                    INPUT_FORMAT, CRC_TYPE, CRC;

INTEGER PROCEDURE inflateData
               (SOURCE_TYPE, SOURCE, SOURCE_START, SOURCE_LEN,
                  DEST_TYPE, DEST, DEST_START, DEST_LEN,
                    INPUT_FORMAT, CRC_TYPE, CRC);
  VALUE        SOURCE_TYPE,         SOURCE_START, SOURCE_LEN,
                  DEST_TYPE,       DEST_START,
                    INPUT_FORMAT, CRC_TYPE;
  EBCDIC ARRAY              SOURCE,
                            DEST [*];
  INTEGER      SOURCE_TYPE,         SOURCE_START, SOURCE_LEN,
                  DEST_TYPE,       DEST_START, DEST_LEN,
                    INPUT_FORMAT, CRC_TYPE, CRC;

PROCEDURE INFLATE-DATA (GLB_PARAM);
   EBCDIC ARRAY         GLB_PARAM [0];
```

Parameters

SOURCE_TYPE identifies the type of the source for the data to be uncompressed.

- 1 = SOURCE contains the data to be uncompressed.
- 2 = SOURCE contains the MCP file name of the file with the data to be uncompressed. The name is in display format or pathname format. See the FILENAME_FORMAT option in the SET_OPTION procedure.

SOURCE is the array containing the compressed data or the name of the file in the application character set that contains the compressed data.

SOURCE_START is the zero-based offset into SOURCE and indicates where the compressed data or file name starts.

SOURCE_LEN is the length in bytes of the data in SOURCE.

DEST_TYPE identifies the type of the destination for the uncompressed data.

- 1 = DEST contains the uncompressed data.
- 2 = DEST contains the MCP file name of the file to which the uncompressed data is to be written. The name is in display format or pathname format. See the FILENAME_FORMAT and FILE_ATTRIBUTES options in the SET_OPTION procedure.

DEST is the array that receives the uncompressed data or contains the name of the file in the application character set to which the uncompressed data will be written.

DEST_START is the zero-based offset into DEST and indicates where the uncompressed data or file name starts.

DEST_LEN is the length in bytes of the uncompressed data.

INPUT_FORMAT is the type of input:

- 1 = zlib
- 2 = gzip

CRC_TYPE is the type of CRC to calculate.

- 0 = no CRC calculation
- 1 = the Java CRC32
- 2 = the Java Adler32

CRC is the CRC value for the uncompressed data.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	SOURCE-TYPE	N5	
SD	SOURCE-SIZE	N5	SOURCE size, for example, 2048
SD	SOURCE	A*n*	[longa]
SD	SOURCE-START	N5	
SD	SOURCE-LEN	N5	
SD	DEST-TYPE	N5	
SD	DEST-SIZE	N5	DEST size, for example, 2048
SD	DEST	A*n*	[longa]
SD	DEST-START	N5	
SD	DEST-LEN	N12	
SD	INPUT-FORMAT	N5	
SD	CRC-TYPE	N5	
SD	CRC	N12	

Possible Result Values

In addition to the standard return results, these possible values can be returned.

Value	Description
–47	An unsupported source value was supplied.
-48	Unable to open a socket to a JPM
-49	Unable to write to the JPM
-50	Unable to read from the JPM
-54	The JPM is not configured.
-55	An unsupported destination value was supplied.
-57	The JPM does not support this function.
-66	A dictionary is required.
-67	The compressed format is invalid.

INT_TO_HTTP_DATE

Converts the number of seconds since year 0 time 0 to the rfc1123-date format.

Syntax

```
INTEGER PROCEDURE INT_TO_HTTP_DATE
               (CHARSET, STRING_TERMINATE,
                  DATE_INT, DATE_STRING);
  INTEGER       CHARSET, STRING_TERMINATE;
  INTEGER         DATE_INT;
  EBCDIC ARRAY              DATE_STRING [0];
```

```
INTEGER PROCEDURE intToHttpDate
               (CHARSET, STRING_TERMINATE,
                  DATE_INT, DATE_STRING);
  VALUE         CHARSET, STRING_TERMINATE,
                  DATE_INT;
  INTEGER       CHARSET, STRING_TERMINATE;
  INTEGER         DATE_INT;
  EBCDIC ARRAY               DATE_STRING [*];
```

Parameters

CHARSET is the application character set: 0 = EBCDIC, 1 = ASCII.

STRING_TERMINATE indicates if the application terminates its strings with nulls: 0 = FALSE (blanks are used), 1 = TRUE.

DATE_INT is the corresponding integer value.

DATE_STRING is the date in rfc1123-date format. For example:

```
Sun, 06 Nov 2005 03:14:24 GMT
```

INT_TO_TIME57

Converts the number of seconds since year 0 time 0 to a time(57)-format word.

Syntax

```
INTEGER PROCEDURE INT_TO_TIME57
               (DATE_INT, DATE_REAL);
  INTEGER       DATE_INT;
  REAL                DATE_REAL;

INTEGER PROCEDURE intToTime57
              (DATE_INT, DATE_REAL);
  VALUE        DATE_INT;
  INTEGER      DATE_INT;
  REAL                   DATE_REAL;
```

Parameters

DATE_INT is the corresponding integer value.

DATE_REAL is the TIME(57) date real value.

INTERFACE_VERSION

Accepts the interface version at which the application program is running as a parameter. The minimum of the interface version at which the application program is running and the interface version supported by the WEBAPPSUPPORT library is returned as the value. The lowest value returned is 1.

Application programs that do not call this procedure are assumed to be running at interface version one (1).

The application program must assume functionality corresponding to the returned interface version of WEBAPPSUPPORT. Each version is inclusive of a lower version. The supported versions are as follows:

- **4**: The application can receive a –33 (Invalid Character) error result instead of –3 (Software Error) when setting a response header that contains an invalid character in the header value.
- **3**: The HTML_ESCAPE and htmlEscape procedures also escape the three characters #,), and (.
- **2**: The application might receive the new error results –18 (Buffer Too Small), –19 (Merge Syntax Error), and –20 (Max len Too Small) instead of –3 (Fault).
- **1**: The original and default level.

Syntax

```
INTEGER PROCEDURE INTERFACE_VERSION (APP_INTERFACE_VERSION);
   INTEGER                           APP_INTERFACE_VERSION;

INTEGER PROCEDURE interfaceVersion (APP_INTERFACE_VERSION);
   VALUE                            APP_INTERFACE_VERSION;

PROCEDURE INTERFACE-VERSION (GLB_PARAM);
   EBCDIC ARRAY              GLB_PARAM [0];
```

Parameters

APP_INTERFACE_VERSION is passed by the application as the highest interface version at which the application is capable of running.

GLB_PARAM has the following format:

```
SG-GLB-PARAM GROUP
  SG-PARAM GROUP
    SD   RESULT              S5
    SD   APP-VERSION         N5
```

Possible Result Values

INTERFACE_VERSION returns a value corresponding to the minimum of APP_INTERFACE_VERSION and the highest version at which WEBAPPSUPPORT is capable of running. INTERFACE_VERSION never returns a value of less than 1. In addition to returning the value to the requesting application, WEBAPPSUPPORT retains a copy of the value and uses it as the requesting application's effective interface version.

Sample ALGOL Application Source

```
    DEFINE
       MyInterfaceVersion = 4 #;
    INTEGER
       EFFInterfaceVersion;
    IF ISVALID(interfaceVersion) THEN
       EFFInterfaceVersion := interfaceVersion(MyInterfaceVersion)
    ELSE
       EFFInterfaceVersion := 1;
```

Sample COBOL Application Source

```
77 MY-INTERFACE-VERSION     PIC  9(11)   BINARY VALUE IS 4.
77 EF-INTERFACE-VERSION     PIC  9(11)   BINARY VALUE IS 1.
PERFORM INTERFACE-VERSION.
INTERFACE-VERSION.
   CALL "INTERFACE_VERSION OF WEBAPPSUPPORT"
     USING   MY-INTERFACE-VERSION
     GIVING  EF-INTERFACE-VERSION.
   INTERFACE-VERSION-EXIT.
EXIT.
```

MERGE_DATA

Uses the passed-in text buffer instead of an external file. The procedure is similar to the MERGE_FILE_AND_DATA procedure.

Syntax

```
INTEGER PROCEDURE MERGE_DATA
              (CHARSET, STRING_TERMINATE, INPUT_CHARSET,
                   INPUT_BUFF, INPUT_LENGTH,  DATA_BUFF,
                  ITEM_COUNT, ITEM_NAME_LEN, ITEM_VALUE_LEN,
                   TRIM_BLANKS, RESULT_BUFF, RESULT_LENGTH);
  INTEGER      CHARSET, STRING_TERMINATE, INPUT_CHARSET,
                              INPUT_LENGTH;
  EBCDIC ARRAY     INPUT_BUFF,                DATA_BUFF [0];
  INTEGER         ITEM_COUNT, ITEM_NAME_LEN, ITEM_VALUE_LEN;
  INTEGER          TRIM_BLANKS;
  EBCDIC ARRAY                  RESULT_BUFF [0];
  INTEGER                                    RESULT_LENGTH;

INTEGER PROCEDURE mergeData
              (CHARSET, STRING_TERMINATE, INPUT_CHARSET,
                   INPUT_BUFF, INPUT_LENGTH,  DATA_BUFF,
                    ITEM_COUNT, ITEM_NAME_LEN, ITEM_VALUE_LEN,
                    TRIM_BLANKS, RESULT_BUFF, RESULT_LENGTH);
  VALUE        CHARSET, STRING_TERMINATE, INPUT_CHARSET,
                                INPUT_LENGTH,
                    ITEM_COUNT, ITEM_NAME_LEN, ITEM_VALUE_LEN,
                    TRIM_BLANKS;
  INTEGER      CHARSET, STRING_TERMINATE, INPUT_CHARSET,
                                INPUT_LENGTH;
  EBCDIC ARRAY     INPUT_BUFF,         DATA_BUFF [*];
  INTEGER           ITEM_COUNT, ITEM_NAME_LEN, ITEM_VALUE_LEN;
  BOOLEAN           TRIM_BLANKS;
  EBCDIC ARRAY                   RESULT_BUFF [*];
  INTEGER                                     RESULT_LENGTH;

PROCEDURE MERGE-DATA (GLB_PARAM);
   EBCDIC ARRAY       GLB_PARAM [0];
```

Parameters

CHARSET is the application character set: 0 = EBCDIC (LATIN1EBCDIC), 1 = ASCII (LATIN1ISO), or values defined in the MLS guide as Ccsnumbers, for example, 102 (CODEPAGE932). Extended characters (for example, accented characters) can appear in the text or the data, and are appropriately translated.

STRING_TERMINATE indicates if the application terminates its strings in DATA_BUFF with nulls: 0 = FALSE (blanks are used), 1 = TRUE.

INPUT_CHARSET is the character set for INPUT_BUFF: 0 = EBCDIC (LATIN1EBCDIC), 1 = ASCII (LATIN1ISO), or values defined in the MLS guide as Ccsnumbers, for example, 102 (CODEPAGE932).

INPUT_BUFF is the buffer containing the raw text. The maximum size supported is 268,435,455 bytes.

INPUT_LENGTH is the length in bytes of the data in INPUT_BUFF.

DATA_BUFF is the buffer containing the fixed field size data entries (strings) to be put into the text. The maximum size supported is 268,435,455bytes.

ITEM_COUNT is the number of items in DATA_BUFF.

ITEM_NAME_LEN is the width in bytes of each name item in DATA_BUFF, including any terminating character.

ITEM_VALUE_LEN is the width in bytes of each value item in DATA_BUFF, including any terminating character.

TRIM_BLANKS indicates whether or not to trim trailing blanks on the value items from DATA_BUFF before inserting into the text: 0 = FALSE, 1 = TRUE. This parameter is ignored if the STRING_TERMINATE parameter is TRUE.

RESULT_BUFF is the buffer into which the updated text is returned. It is returned in the application CHARSET. The maximum size supported is 268,435,455bytes. No terminating character is added.

RESULT_LENGTH is the length of data returned in RESULT_BUFF.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	CHARSET	N5	
SD	STRING-TERMINATE	N5	
SD	INPUT-CHARSET	N5	
SD	INPUT-BUFF-SIZE	N5	INPUT-BUFF size, for example, 2048
SD	INPUT-BUFF	A*n*	[longa]
SD	INPUT-LENGTH	N5	
SD	DATA-BUFF-SIZE	N5	DATA-BUFF size, for example, 8000
SD	DATA-BUFF	A*n*	[longa]
SD	ITEM-COUNT	N5	
SD	ITEM-NAME-LEN	N5	
SD	ITEM-VALUE-LEN	N5	
SD	TRIM-BLANKS	N5	
SD	RESULT-BUFF-SIZE	N5	
SD	RESULT-BUFF	A*n*	RESULT-BUFF size, for example, 10000
SD	RESULT-LENGTH	N5	[longa]

Possible Result Values

In addition to the standard returned results, these possible values can be returned.

Value	Description
–12	INPUT_LENGTH too long to be processed.
–15	Character set not available. The CENTRALSUPPORT and CCSFILE installed on the system do not support the character set.
–17	Translation not available. The mapping between the input and output character sets is not supported by the CENTRALSUPPORT and CCSFILE installed on the system.

Notes:

- *Item-Name fields can contain only LATIN1ISO or LATIN1EBCDIC characters.*
- *The output character set must be defined to the WEBAPPSUPPORT library with a call to SET_OUTPUT_CHARSET if the output character set is other than LATIN1ISO.*

MERGE_FILE_AND_DATA

Takes the items in DATA_BUFF and inserts them into specially marked locations into the text read from a file, usually an HTML file, returning the updated text to the caller. It is similar to the MERGE_DATA procedure.

The input file can be in a different character set from the data being merged into it. For example, the input file can be in ASCII; the data merged in by the application can be EBCDIC; and the data is translated before insertion.

The input file kinds supported are as follows:

- STREAM files, such as those created on PCs using Client Access Services shares,
- MCP text files of type SEQDATA, TEXTDATA, and JOBSYMBOL. The sequence numbers are stripped out,
- CDATA text files.

The maximum input file length supported is 268,435,455 bytes.

The intent is to make it easy for COBOL applications to supply the name and value pair data. For example, in COBOL you might declare the following:

```
 01 DATA-BUFFER.
   03 DATA-PAIR OCCURS 4 TIMES.
     05 DATA-NAME  PIC X(20).
     05 DATA-VALUE PIC X(30).
```

The call to MERGE_FILE_AND_DATA passes DATA-BUFFER, with ITEM_COUNT set to 4 (or less), ITEM_NAME_LEN set to 20, and ITEM_VALUE_LEN set to 30. The first DATA-NAME might contain Customer, the first DATA-VALUE the first customer's name, and so forth, and the file HTML might then contain $REPLACE=CUSTOMER.

An example of using this procedure is shown in "Using an External HTML File" under "Sample COBOL Applications."

Syntax

```
INTEGER PROCEDURE MERGE_FILE_AND_DATA
              (CHARSET, STRING_TERMINATE, FILE_NAME,
                 DATA_BUFF, ITEM_COUNT,
                   ITEM_NAME_LEN, ITEM_VALUE_LEN, TRIM_BLANKS,
                   RESULT_BUFF, RESULT_LENGTH);
  INTEGER      CHARSET, STRING_TERMINATE;
  INTEGER                   ITEM_COUNT;
  EBCDIC ARRAY   DATA_BUFF,               FILE_NAME [0];
  INTEGER          ITEM_NAME_LEN, ITEM_VALUE_LEN, TRIM_BLANKS;
  EBCDIC ARRAY     RESULT_BUFF [0];
  INTEGER                       RESULT_LENGTH;

INTEGER PROCEDURE mergeFileAndData
              (CHARSET, STRING_TERMINATE, FILE_NAME,
                 DATA_BUFF, ITEM_COUNT,
                   ITEM_NAME_LEN, ITEM_VALUE_LEN, TRIM_BLANKS,
                   RESULT_BUFF, RESULT_LENGTH);
  VALUE        CHARSET, STRING_TERMINATE,
                            ITEM_COUNT,
                   ITEM_NAME_LEN, ITEM_VALUE_LEN, TRIM_BLANKS,
  INTEGER      CHARSET, STRING_TERMINATE;
  INTEGER                   ITEM_COUNT;
  EBCDIC ARRAY   DATA_BUFF,         FILE_NAME [*];
  INTEGER          ITEM_NAME_LEN, ITEM_VALUE_LEN;
```

```
  BOOLEAN                                          TRIM_BLANKS;
  EBCDIC ARRAY     RESULT_BUFF [*]
  INTEGER                       RESULT_LENGTH;

PROCEDURE MERGE-FILE-AND-DATA (GLB_PARAM);
   EBCDIC ARRAY                GLB_PARAM [0];
```

Parameters

CHARSET is the application character set: 0 = EBCDIC (LATIN1EBCDIC), 1 = ASCII (LATIN1ISO), or values defined in the *MultiLingual System Guide* as Ccsnumbers, for example, 102 (CODEPAGE932). Extended characters (for example, accented characters) can appear in the HTML or the data, and are appropriately translated.

STRING_TERMINATE indicates if the application terminates its strings with nulls: 0 = FALSE (blanks are used), 1 = TRUE.

FILE_NAME is the name of the input file to read in, expressed in the application CHARSET. If the first character is a forward slash (/), the file is interpreted as PATHNAME format; otherwise, it is interpreted as DISPLAY (TITLE) format. Examples are

```
/-/DISK/USERCODE/AR/CUSTOMERINFO.HTM
(AR)"CUSTOMERINFO.HTM" ON DISK
```

See also the SET_OPTION, FILENAME_FORMAT option later in this section.

DATA_BUFF is the buffer containing the fixed field size data entries (strings) to be put into the text, expressed in the application CHARSET. The maximum size supported is 268,435,455 bytes.

ITEM_COUNT is the number of items (name and value pairs) in DATA_BUFF.

ITEM_NAME_LEN is the width in bytes of each name item in DATA_BUFF, including any terminating character.

ITEM_VALUE_LEN is the width in bytes of each value item in DATA_BUFF, including any terminating character.

TRIM_BLANKS indicates whether or not to trim trailing blanks on the value items copied from DATA_BUFF before inserting into the text: 0 = FALSE, 1 = TRUE. This parameter is ignored if the STRING_TERMINATE parameter is TRUE.

RESULT_BUFF is the buffer into which the updated text is returned. It is returned in the application CHARSET. The maximum size supported is 268,435,455 bytes. No terminating character is added.

RESULT_LENGTH is the length of data returned in RESULT_BUFF.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	CHARSET	N5	
SD	STRING-TERMINATE	N5	
SD	FILE-NAME-SIZE	N5	FILE-NAME size, for example, 256
SD	FILE-NAME	A*n*	[longa]
SD	DATA-BUFF-SIZE	N5	DATA-BUFF size, for example, 8000
SD	DATA-BUFF	A*n*	[longa]
SD	ITEM-COUNT	N5	
SD	ITEM-NAME-LEN	N5	
SD	ITEM-VALUE-LEN	N5	
SD	TRIM-BLANKS	N5	
SD	RESULT-BUFF-SIZE	N5	
SD	RESULT-BUFF	A*n*	RESULT-BUFF size, for example, 10000
SD	RESULT-LENGTH	N5	[longa]

Possible Result Values

In addition to the standard returned results, these possible values can be returned.

Value	Description
–11	Input File not found or not available to caller.
–12	Input File too long to be processed.
–13	Attribute error setting input file name.
–14	I/O error reading input file.
–15	Character set not available. The CENTRALSUPPORT and CCSFILE installed on the system do not support the character set.
–16	File character set not available. The EXTMODE of the file used is not supported by the CENTRALSUPPORT and CCSFILE installed on the system.
–17	Translation not available. The mapping between the input and output character sets is not supported by the CENTRALSUPPORT and CCSFILE installed on the system.

Options Available to Mark Insertion Points in the HTML

$REPLACE = <item name>

The occurrences of the string $REPLACE=<item name> in the source text are replaced with the data items provided. The option <item name>

- Is the name of the value field. This option allows fields to be placed in the form in an order different from that defined in the application.
- Can be enclosed in quotes.
- Can contain alphabetics, numerics, or the underscore and hyphen separators (_ and -). It is case insensitive.

Example HTML Segments

```
<P>Today's date is $REPLACE=date.</P>
<P>The temperature is <B>$REPLACE="temp"&deg;F</B></P>
```

$REPLACE-VALUE=<item name>

Similar to $REPLACE = <item name>, this option precedes the inserted text with VALUE =" and follows it with a ". This string can be used for the value field in a form <INPUT> tag so that if the HTML is viewed in a browser, the $REPLACE does not appear.

Example

<INPUT TYPE=TEXT SIZE=20 $REPLACE-VALUE=text1>might result in the following:

```
<INPUT TYPE=TEXT SIZE=20 VALUE="John Doe">
```

Notes:

- *Data values can be null strings.*
- *If any item names in the input file are not replaced by matching data items, the $REPLACE tag is left in the resulting text.*

$REPLACE-BEGIN and $REPLACE-END
$REPLACE-BEGIN = <loop label> and $REPLACE-END = <loop label>

The input file can contain variable amounts (lists) of data. A repeating string can be defined in the text bounded with $REPLACE-BEGIN = <loop label> (begin) and $REPLACE-END = <loop label> (end) tags, and that text string is repeated with each occurrence of data items that reference the $REPLACE = <item name> tags that appear between the begin and end tags.

The begin and end tags can be enclosed in HTML comments, which might be needed to work properly with some HTML editors.

Example

```
<TABLE>
 <TR><TH>Month</TH><TH>Avg Temp</TH></TR>
 <!-- $REPLACE-BEGIN -->
 <TR><TD>$REPLACE=month</TD><TD>$REPLACE=temp</TD></TR>
 <!-- $REPLACE-END -->
</TABLE>
```

The preceding HTML might result in the following:

```
<TABLE>
 <TR><TH>Month</TH><TH>Avg Temp</TH></TR>
 <TR><TD>Jan 99</TD><TD>79</TD></TR>
 <TR><TD>Feb 99</TD><TD>80</TD></TR>
 <TR><TD>Mar 99</TD><TD>81</TD></TR>a
</TABLE>
```

Lists can be nested. If you nest lists, a loop label is required for all but the outermost occurrences of $REPLACE-BEGIN and $REPLACE-END pair. (For the outermost occurrences, a loop label is recommended but not required.)

The loop label associates a particular instance of $REPLACE-BEGIN with a particular instance of $REPLACE-END. It helps you keep track of matching pairs, and also allows you to dynamically cut off lower levels of nesting for which no data items exist. Every $REPLACE-BEGIN = <loop label> must be matched with a $REPLACE-END = <loop label>; if not, a syntax error is reported and the merging function is terminated.

Nested List Example

```
$REPLACE-BEGIN=releases
  <A NAME="SSR$REPLACE=rel">
  The following IC tapes are available for $REPLACE=softlevel<BR>
  $REPLACE-BEGIN=labels
   <A NAME="$REPLACE=label">
   <H3 ALIGN=LEFT>$REPLACE=tapelabel </H3>
   <TABLE CELLSPACING=1 CELLPADDING=5 BORDER=0>
   <TR>
   $REPLACE-BEGIN=iclevels
     <TD VALIGN=top ALIGN=center>
     <A HREF=http://bxah06/filea>$REPLACE=iclevel</A></TD>
   $REPLACE-END=iclevels
     </TR>
     </TABLE>
     <FONT SIZE=-1><A HREF="#home">Back to top</A></FONT><BR>
  $REPLACE-END=labels
  $REPLACE-END=releases
```

Character Sets

- Item name fields can contain only LATIN1ISO or LATIN1EBCDIC characters.
- The output character set must be defined to the WEBAPPSUPPORT library with a call to SET_OUTPUT_CHARSET if the output character set is other than LATIN1ISO.

MERGE_I18NFILE_AND_DATA

MERGE_I18NFILE_AND_DATA is the same as the MERGE_FILE_AND_DATA procedure except that it allows the character set of the file contents to be specified. The file attribute EXTMODE is ignored. This is useful when the file contents are something other than ASCII (LATIN1ISO) or EBCDIC (LATIN1EBCDIC), such as CODEPAGE932.

Syntax

```
INTEGER PROCEDURE MERGE_I18NFILE_AND_DATA
               (CHARSET, STRING_TERMINATE,  FILE_NAME,
                   FILE_CHARSET, DATA_BUFF, ITEM_COUNT,
                   ITEM_NAME_LEN, ITEM_VALUE_LEN, TRIM_BLANKS,
                   RESULT_BUFF, RESULT_LENGTH);
  INTEGER      CHARSET, STRING_TERMINATE;
  INTEGER         FILE_CHARSET          ITEM_COUNT;
  EBCDIC ARRAY                  DATA_BUFF, FILE_NAME [0];
  INTEGER         ITEM_NAME_LEN, ITEM_VALUE_LEN, TRIM_BLANKS;
  EBCDIC ARRAY    RESULT_BUFF [0];
  INTEGER                       RESULT_LENGTH;
```

```
INTEGER PROCEDURE mergeI18NfileAndData
              (CHARSET, STRING_TERMINATE,  FILE_NAME,
                  FILE_CHARSET, DATA_BUFF, ITEM_COUNT,
                  ITEM_NAME_LEN, ITEM_VALUE_LEN, TRIM_BLANKS,
                  RESULT_BUFF, RESULT_LENGTH);
  VALUE        CHARSET, STRING_TERMINATE,
                  FILE_CHARSET            ITEM_COUNT,
                  ITEM_NAME_LEN, ITEM_VALUE_LEN, TRIM_BLANKS,
  INTEGER      CHARSET, STRING_TERMINATE;
  INTEGER         FILE_CHARSET            ITEM_COUNT;
  EBCDIC ARRAY                  DATA_BUFF, FILE_NAME [0];
  INTEGER         ITEM_NAME_LEN, ITEM_VALUE_LEN; TRIM_BLANKS;
  EBCDIC ARRAY    RESULT_BUFF [0];
  INTEGER                      RESULT_LENGTH;

PROCEDURE MERGE-I18N-FILE-AND-DATA (GLB_PARAM);
   EBCDIC ARRAY                     GLB_PARAM [0];
```

Parameters

FILE_CHARSET is the character set that determines how the file contents are treated. The values supported are 0 = EBCDIC (LATIN1EBCDIC), 1 = ASCII (LATIN1ISO), or values defined in the *MultiLingual System Guide* as Ccsnumbers, for example, 102 (CODEPAGE932).

STRING_TERMINATE indicates if the application terminates its strings with nulls: 0 = FALSE (blanks are used), 1 = TRUE.

FILE_NAME is the name of the input file to read in, expressed in the application CHARSET. If the first character is a forward slash (/), the file is interpreted as PATHNAME format; otherwise, it is interpreted as DISPLAY (TITLE) format. Examples are

```
/-/DISK/USERCODE/AR/CUSTOMERINFO.HTM

(AR)"CUSTOMERINFO.HTM" ON DISK
```

See also the SET_OPTION, FILENAME_FORMAT option later in this section.

DATA_BUFF is the buffer containing the fixed field size data entries (strings) to be put into the text, expressed in the application CHARSET. The maximum size supported is 268,435,455 bytes.

ITEM_COUNT is the number of items (name and value pairs) in DATA_BUFF.

ITEM_NAME_LEN is the width in bytes of each name item in DATA_BUFF, including any terminating character.

ITEM_VALUE_LEN is the width in bytes of each value item in DATA_BUFF, including any terminating character.

TRIM_BLANKS indicates whether or not to trim trailing blanks on the value items copied from DATA_BUFF before inserting into the text: 0 = FALSE, 1 = TRUE. This parameter is ignored if the STRING_TERMINATE parameter is TRUE.

RESULT_BUFF is the buffer into which the updated text is returned. It is returned in the application CHARSET. The maximum size supported is 268,435,455 bytes. No terminating character is added.

RESULT_LENGTH is the length of data returned in RESULT_BUFF.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	CHARSET	N5	
SD	STRING-TERMINATE	N5	
SD	FILE-NAME-SIZE	N5	FILE-NAME size, for example, 256
SD	FILE-NAME	A*n*	[longa]
SD	FILE-CHARSET	N5	
SD	DATA-BUFF-SIZE	N5	DATA-BUFF size, for example, 8000
SD	DATA-BUFF	A*n*	[longa]
SD	ITEM-COUNT	N5	
SD	ITEM-NAME-LEN	N5	
SD	ITEM-VALUE-LEN	N5	
SD	TRIM-BLANKS	N5	
SD	RESULT-BUFF-SIZE	N5	
SD	RESULT-BUFF	A*n*	RESULT-BUFF size, for example, 10000
SD	RESULT-LENGTH	N5	[longa]

Possible Result Values

In addition to the standard returned results, these possible values can be returned.

Value	Description
–15	Character set not available. The CENTRALSUPPORT and CCSFILE installed on the system do not support the character set.
–17	Translation not available. The mapping between the input and output character sets is not supported by the CENTRALSUPPORT and CCSFILE installed on the system.

RELEASE_KEY

Releases a key object in WEBAPPSUPPORT, freeing resources in WEBAPPSUPPORT and MCP Cryptography.

Syntax

```
INTEGER PROCEDURE RELEASE_KEY (KEY_TAG);
  INTEGER                      KEY_TAG;

INTEGER PROCEDURE releaseKey  (KEY_TAG);
  VALUE                        KEY_TAG;
  INTEGER                      KEY_TAG;
```

```
PROCEDURE RELEASE_KEY (GLB_PARAM);
   EBCDIC ARRAY        GLB_PARAM [0];
```

Parameter

KEY_TAG is the tag that references the key object in WEBAPPSUPPORT.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	KEY-TAG	A6	[bin]

Possible Result Values

In addition to the standard returned results, these possible values can be returned.

Value	Description
–123	The key is invalid.

SET_OPTION

Sets options for general use of the WEBAPPSUPPORT library.

Syntax

```
INTEGER PROCEDURE SET_OPTION
                  (OPTION, OPTION_VALUE, OPTION_STRING);
   INTEGER          OPTION;
   REAL                     OPTION_VALUE;
   EBCDIC ARRAY                           OPTION_STRING [0];

INTEGER PROCEDURE setOption
                  (OPTION, OPTION_VALUE, OPTION_STRING);
   VALUE            OPTION, OPTION_VALUE;
   INTEGER          OPTION;
   REAL                     OPTION_VALUE;
   EBCDIC ARRAY                           OPTION_STRING [*];

PROCEDURE SET-OPTION (GLB_PARAM);
   EBCDIC ARRAY       GLB_PARAM [0];
```

Parameters

OPTION is the option being set. The following options are supported.

1 (FILENAME_FORMAT)

OPTION_VALUE the format used for file names that applications pass to WEBAPPSUPPORT. The SET_XML_OPTION procedure, FILENAME_FORMAT (10) option has the same value as this option. See the SET_XML_OPTION procedure, FILENAME_FORMAT (10) option in Section 6.

The default for the MERGE_FILE_AND_DATA and MERGE_I18NFILE_AND_DATA procedures is to not apply SEARCHRULE.

An OPTION_VALUE of 0 represents LTITLE. SEARCHRULE = NATIVE is used for opening files, unless the first character of the file name is a forward slash (/). This value is the default.

An OPTION_VALUE of 1 represents PATHNAME. SEARCHRULE = POSIX is used for opening files.

2 (MAX_CACHE_FILES)

OPTION_VALUE specifies the maximum number of files that can be kept in the WEBAPPSUPPORT's cache. Default = 2. Maximum value = 10. If a value larger than 10 is specified, 10 is used. If zero or less is specified caching for the application is disabled.

3 (MAX_CACHE_FILESIZE)

OPTION_VALUE specifies the maximum size of a file in bytes that can be kept in the WEBAPPSUPPORT cache. Default and maximum is the system maximum array size. If zero or less is specified caching for the application is disabled.

4 (CACHE_TIMEOUT)

OPTION_VALUE specifies the number of seconds to wait before checking the disk for an updated version of the file. The default is 0 (disk is checked on each read).

5 RESERVED

6 (DEFLATE_LEVEL)

OPTION_VALUE specifies the level of compression used by the DEFLATE_DATA procedure. OPTION_VALUE is a range from -1 (default) to 9 (best compression). 0 represents no compression, and 1 represents best speed.

7 (DEFLATE_STRATEGY)

OPTION_VALUE specifies the compression strategy used by the DEFLATE_DATA procedure. OPTION_VALUE is a range from 0 (default strategy, which is the default); 1 = filtered strategy; or 2 = Huffman only strategy.

8 (FILE_ATTRIBUTES)

This option specifies file attributes for files created by the WEBAPPSUPPORT library and is equivalent to using the 11 (FILE_ATTRIBUTES) option in the SET_XML_OPTION procedure. The OPTION_STRING parameter contains a comma-separated list of file attribute settings in the application character set. The default for OPTION_STRING is a null string.

For example, the OPTION_STRING parameter can be

```
SECURITYTYPE=PUBLIC, SECURITYUSE=IN
```

The procedure does not use the OPTION_VALUE parameter for this option. Options cannot be specified that create MCP record files; only stream files are supported.

OPTION_STRING usage is described in the previous option descriptions. If the value of OPTION does not require a value for OPTION_STRING, the application should set OPTION_STRING to a null string.

GLB_PARAM has the following format:

Format	Notes
`SG-GLB-PARAM GROUP`	
`  SG-PARAM GROUP`	
`  SD  RESULT  S5`	
`  SD  OPTION  N5`	
`  SD  OPTION-VALUE  N12`	
`  SD  OPTION-STRING-SIZE  N5`	OPTION-STRING size, for example, 256
`  SD  OPTION-STRING  An`	[longa]

Possible Result Values

In addition to the standard returned results, these possible values can be returned.

Value	Description
0	The application specified an option or value that the procedure does not support.
1	The procedure accepted all settings.

9 (CRUNCH_FILE)

OPTION_VALUE specifies whether or not to crunch created files. If OPTION_VALUE is 0, files are not crunched. If OPTION_VALUE is 1, files are crunched. The default OPTION_VALUE is 1.

SET_STRING_TERMINATE

Sets the default string termination option for the application and overrides the STRINGTERMINATE setting in the WEBPCM service.

Syntax

```
INTEGER PROCEDURE SET_STRING_TERMINATE
                   (STRING_TERMINATE);
  INTEGER           STRING_TERMINATE;

INTEGER PROCEDURE setStringTerminate
                   (STRING_TERMINATE);
  VALUE             STRING_TERMINATE;
  BOOLEAN           STRING_TERMINATE;

PROCEDURE SET-STRING-TERMINATE (GLB_PARAM);
   EBCDIC ARRAY                 GLB_PARAM [0];
```

Parameters

STRING_TERMINATE indicates whether the application uses the null value to terminate a string.

If the value of this parameter is 0 (false), the application uses space characters to terminate a string. The value 0 is the default.

If the value of this parameter is 1 (true), the application does not use space characters to terminate a string.

GLB_PARAM has the following format:

```
SG-GLB-PARAM GROUP
  SG-PARAM GROUP
    SD   RESULT              S5
    SD   STRING-TERMINATE    N5
```

Possible Result Values

In addition to the standard returned results, these possible values can be returned.

Value	Description
1	The procedure accepted the setting.

SET_TRACING

Sets the WEBAPPSUPPORT tracing state for the application. If an operator has turned all (global) tracing on, an application turning its tracing off has its setting ignored; that is, those application calls are still traced. If an application turns its tracing on, and an operator turns global tracing off, the application tracing is still performed. See “Using the WEBAPPSUPPORT Trace File.”

Syntax

```
INTEGER PROCEDURE SET_TRACING (TRACE_TYPE);
  INTEGER                       TRACE_TYPE;

INTEGER PROCEDURE setTracing (TRACE_TYPE);
  VALUE                        TRACE_TYPE;
  BOOLEAN                      TRACE_TYPE;

PROCEDURE SET-TRACING (GLB_PARAM);
   EBCDIC ARRAY        GLB_PARAM [0];
```

Parameters

TRACE_TYPE causes tracing to start:

- 0 = do not trace any procedure calls. This is the default value.
- 1 = trace all library calls.
- 2 = trace only calls that return errors (negative results).

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	TRACE-ON	N5	0=FALSE, 1=TRUE

SET_TRANSLATION

Sets the application character set, which is the character set in which the application supplies and receives data. Calling this procedure overrides the APPLICATIONCCS and CLIENTCCS settings in the WEBPCM service.

The XML Parser parses a document into the application character set, not the document character set.

Syntax

```
INTEGER PROCEDURE SET_TRANSLATION
                                (MLS_APPLICATION_SET, MLS_CLIENT_SET);
     INTEGER                     MLS_APPLICATION_SET, MLS_CLIENT_SET;

INTEGER PROCEDURE setTranslation
                                (MLS_APPLICATION_SET, MLS_CLIENT_SET);
     VALUE                       MLS_APPLICATION_SET, MLS_CLIENT_SET;
     INTEGER                     MLS_APPLICATION_SET, MLS_CLIENT_SET;

PROCEDURE SET-TRANSLATION (GLB_PARAM);
   EBCDIC ARRAY            GLB_PARAM [0];
```

Parameters

MLS_APPLICATION_SET is the character set in which the application sends and receives data. The value of this parameter can be any of the following:

- 0 (ASERIESEBCDIC)
- 1 (ASCII)
- 2 (UTF-8)
- Any value defined in the *MultiLingual System Administration, Operations, and Programming Guide* (8600 0288) as a ccsnumber

  An example of a ccsnumber is 102 (CODEPAGE932).

The default value for applications that are not WEBPCM applications is the ccsnumber 4 (ASERIESEBCDIC).

MLS_CLIENT_SET is not used for XML parsing.

GLB_PARAM has the following format:

```
SG-GLB-PARAM GROUP
  SG-PARAM GROUP
    SD   RESULT              S5
    SD   MLS-APPLICATION-SET N5
    SD   MLS-CLIENT-SET      N5
```

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
–17	The procedure did not set the application character. The CENTRALSUPPORT library and the CCSFILE data file installed on the system do not support mapping between the XML Parser input and output character sets.

TIME57_TO_HTTP_DATE

Converts the TIME(57) formatted word to the rfc1123-date format.

Syntax

```
INTEGER PROCEDURE TIME57_TO_HTTP_DATE
              (CHARSET, STRING_TERMINATE,
                  DATE_REAL, DATE_STRING);
  INTEGER      CHARSET, STRING_TERMINATE;
  REAL            DATE_REAL;
  EBCDIC ARRAY               DATE_STRING [0];

INTEGER PROCEDURE time57ToHttpDate
              (CHARSET, STRING_TERMINATE,
                  DATE_REAL, DATE_STRING);
  VALUE        CHARSET, STRING_TERMINATE,
```

```
                 DATE_REAL;
  INTEGER      CHARSET, STRING_TERMINATE;
  REAL            DATE_REAL;
  EBCDIC ARRAY                DATE_STRING [*];
```

Parameters

CHARSET is the application character set: 0 = EBCDIC, 1 = ASCII.

STRING_TERMINATE indicates whether or not the application terminates its strings with nulls: 0 = FALSE (blanks are used), 1 = TRUE.

DATE_REAL is the corresponding TIME(57) format real value.

DATE_STRING is the date in rfc1123-date format.

TIME57_TO_INT

Converts a TIME(57)-format word to an integer, the value of which is the number of seconds of the specified date and time since day 0 time 0.

Syntax

```
INTEGER PROCEDURE TIME57_TO_INT
              (DATE_REAL, DATE_INT);
  REAL         DATE_REAL;
  INTEGER                 DATE_INT;

INTEGER PROCEDURE time57ToInt
              (DATE_REAL, DATE_INT);
  VALUE        DATE_REAL;
  REAL         DATE_REAL;
  INTEGER                 DATE_INT;
```

Parameters

DATE_REAL is the TIME(57) date real value.

DATE_INT is the corresponding integer value.

TRACE_WEB_MSG

Traces out a text message for the application into the WEBAPPSUPPORT trace file. It is a way for the application programmer to add comments about where the application is in its processing, any special conditions encountered, and so on. See “Using the WEBAPPSUPPORT Trace File.”

Syntax

```
INTEGER PROCEDURE TRACE_WEB_MSG (CHARSET, TRACE_STRING, STRING_LEN);
  EBCDIC ARRAY                             TRACE_STRING [0];
  INTEGER                        CHARSET,               STRING_LEN;

INTEGER PROCEDURE traceWebMsg   (CHARSET, TRACE_STRING, STRING_LEN);
  VALUE                          CHARSET,               STRING_LEN;
```

```
  EBCDIC ARRAY                              TRACE_STRING [*];
  INTEGER                        CHARSET,                STRING_LEN;

PROCEDURE TRACE-WEB-MSG (GLB_PARAM);
   EBCDIC ARRAY          GLB_PARAM [0];
```

Parameters

CHARSET is the application character set: 0 = EBCDIC, 1 = ASCII.

TRACE_STRING is the message to be traced, up to a maximum of 65500 bytes. It is in the application character set.

STRING_LEN is the length of the text in TRACE_STRING to be traced.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	CHARSET	N5	
SD	TRACE-STRING-SIZE	N5	TRACE-STRING size, for example, 256
SD	TRACE-STRING	A*n*	[longa]
SD	STRING-LEN	N5	

Possible Result Values

In addition to the standard returned results, these possible values can be returned.

Result	Description
0	No-op (0) is returned if global tracing is not on, and application-specific tracing has not been requested by the application or an operator.

Using the WEBAPPSUPPORT Trace File

You can use either of the following two WEBAPPSUPPORT procedures from an application to control tracing:

- SET_TRACING to turn tracing on or off. This call only affects tracing for the application stack making the call.
- TRACE_WEB_MSG to trace a diagnostic string.

You can also control tracing for application stacks by the WEBAPPSUPPORT TRACE command and the TRACEERRORS general parameter.

The WEBAPPSUPPORT trace file

```
(TRACE/CCF/WEBAPPSUPPORT/<date>/"<time>.TXT")
```

contains trace information about calls made to the WEBAPPSUPPORT library. It is intended to aid programmers in developing applications for use with the WEBPCM and to help Unisys with problem resolution.

Trace File Name

The WEBAPPSUPPORT trace file is named as follows and is located on the family where WEBAPPSUPPORT is located:

```
*TRACE/CCF/WEBAPPSUPPORT/<date>/"<time>.TXT"
```

In this file name, <date>/<time> is the format yyyymmdd/"hhmmss.txt".

Here is an example:

```
*TRACE/CCF/WEBAPPSUPPORT/19990214/"092712.TXT"
```

Note: *You can set the family where the WEBAPPSUPPORT trace file is located to a family other than where the SL command placed the WEBAPPSUPPORT library. To do this, use the TRACEFAMILY directive in the general parameters file for WEBAPPSUPPORT: *SYSTEM/CCF/WEBAPPSUPPORT/PARAMS. See "WEBAPPSUPPORT General Parameters File" in this section for more information about the TRACEFAMILY directive.*

Trace File Format

The WEBAPPSUPPORT trace file is created with PRIVATE security by default. If PUBLIC files are preferred (so that nonprivileged application developers can view the trace files), perform the following steps:

1. Modify the *SYSTEM/CCF/WEBAPPSUPPORT code file with this statement:

```
WFL MODIFY *SYSTEM/CCF/WEBAPPSUPPORT; FILE TRACEFILE
(SECURITYTYPE=PUBLIC)
```

2. Use the SL (Support Library) system command to reassign the code file to the WEBAPPSUPPORT function.

The previous steps needs to be performed for each Interim Correction (IC) that is installed.

The trace file is formatted with an identification line that includes the time the file was created, followed by a header line for the columns, followed by the trace messages. The columns are listed in the following table.

Column Name	Description
Stack	The mix number of the task making the procedure call.
Time	The local system time the trace record was written.
Procedure	The name of the procedure in WEBAPPSUPPORT that was called, up to 15 characters.

Column Name	Description
CS	The Character Set used by the application, usually derived from the MSG parameter: EB = EBCDIC, AS = ASCII.
T	Whether or not the application has its text strings terminated: Y = yes, N = no.
Res	The result value for the procedure call.
Notes	The parameters of the procedure call, or other text.

Notes:

- *The first time an application has a call traced in a particular trace file, a TraceID record is written to the trace file with the application name.*
- *Only one trailing blank is traced out for a string.*

The trace file is created as a STREAM file and is easily viewable from Client Access Services shares or through a browser, if an appropriate mapping is made in Web Transaction Server. See the WEBPCM Demonstrations Web page, Configuration section, for instruction.

Trace File Creation

A WEBAPPSUPPORT trace file can be started by one of the following methods:

- The operator command NA CCF WEBPCM WEBAPPSUPPORT TRACE +.
- One or more applications that request tracing for itself.

If the application requests the trace file, or if an operator requests tracing for a specific application, only the calls of that application are traced; other applications do not have their calls traced.

Trace File Closure

A WEBAPPSUPPORT trace file is closed when one of the following activities occurs:

- An operator closes the trace file.
- The WEBAPPSUPPORT library terminates when the last application delinks from it.

Turning tracing off for a specific application keeps the trace file open for more tracing.

Sample Trace File

```
WEBAPPSUPPORT Trace File *TRACE/CCF/WEBAPPSUPPORT/19990214/"092712.TXT"
Started 25 August 1998 14:27:12 GMT by Application 1079

Stack  Time  Procedure   CS T Res Notes
-----  ----  ---------   -- - --- -----

 1079 09:27:12  [task id]   (MIS)OBJECT/DBPROC ON PACK
 1079 09:27:12 SET_TRACING      1 TRACE_ON=1
 1079 09:27:13 TRACE_WEB_MSG  EB N 1 "DBPROC received input request"
```

```
 1079 09:27:13 GET_2_HEADERS  EB N 1 NAME01="$REMOTE-USER ",VALUE01=
"JONES ",NAME02="User-Agent ",VALUE02="Mozilla/4.05 [en] (Win95; I) "
```

WEBPCM Procedures

The WEBPCM WEBAPPSUPPORT procedures listed in this section each describe an entry point compatible with COBOL with all uppercase and with underscores and an entry point compatible with ALGOL with mixed upper- and lowercase containing no underscores.

GET_COOKIE

Returns a specifically named cookie if present in the request.

Syntax

```
INTEGER PROCEDURE GET_COOKIE (MSG, COOKIE_NAME, COOKIE_VALUE,
                              COOKIE_VALUE_LEN);
  EBCDIC ARRAY                MSG, COOKIE_NAME, COOKIE_VALUE [0];
  INTEGER                     COOKIE_VALUE_LEN;

INTEGER PROCEDURE getCookie  (MSG, COOKIE_NAME, COOKIE_VALUE,
                              COOKIE_VALUE_LEN);
  EBCDIC ARRAY                MSG, COOKIE_NAME, COOKIE_VALUE [*];
  INTEGER                     COOKIE_VALUE_LEN;
```

Parameters

MSG is the Message Object.

COOKIE_NAME is the name of the requested cookie. It is not case-sensitive. For example: Detail-Preference.

COOKIE_VALUE is the returned cookie value of the first name-value pair that matches COOKIE_NAME.

COOKIE_VALUE_LEN is the length of the data returned in COOKIE_VALUE in bytes.

Possible Result Values

Value	Description
0	The specified cookie is not present in the request.

GET_DIALOG_ID

Returns the dialog ID associated with MSG. It is used by the application with the hidden HTML method of maintaining sessions.

Syntax

```
INTEGER PROCEDURE GET_DIALOG_ID (MSG, STR);
  EBCDIC ARRAY                   MSG, STR [0];
```

```
INTEGER PROCEDURE getDialogID   (MSG, STR);
  EBCDIC ARRAY                   MSG, STR [*];
```

Parameters

MSG is the Message Object.

STR is the dialog ID.

GET_HEADER, GET_n_HEADERS

Syntax

```
INTEGER PROCEDURE GET_HEADER (MSG, HEADER_NAME, HEADER_VALUE);
  EBCDIC ARRAY                MSG [0];
  EBCDIC ARRAY                     HEADER_NAME [0];
  EBCDIC ARRAY                                  HEADER_VALUE [0];

INTEGER PROCEDURE getHeader  (MSG, HEADER_NAME, HEADER_VALUE);
  EBCDIC ARRAY                MSG [*];
  EBCDIC ARRAY                     HEADER_NAME [*];
  EBCDIC ARRAY                                  HEADER_VALUE [*]
```

Parameters

MSG is the Message Object.

HEADER_NAME is the name of the requested header. It is case-sensitive. For example: User-Agent.

HEADER_VALUE is the returned header value, null in length if the value is not present in the request. For example: Mozilla/4.05 [en] (Win95; I).

Possible Result Values

In addition to the standard returned results, these possible values can be returned.

Value	Description
0	A null string and No-op (0) are returned if not present in the request.
–17	Translation not available, and the mapping between the input and output character sets is not supported by the CENTRALSUPPORT and CCSFILE installed on the system.

HEADER_NAME Values

The following HEADER_NAME values are also supported as extensions of the header names defined for HTTP.

Value	Description
$APPLICATION-PATH	Returns the leading part of the request path name that refers to the application (that is, the virtual directory in Web Transaction Server, which equals the PATH attribute in the WEBPCM SERVICE definition) plus the next node in the requested path name. The returned path name is unescaped. For example: /apidemo/apienv.
$AUTH-TYPE	Returns the authentication scheme of the request. Returns a null string and a result of No-op if the authorization header is not present in the request. For example: Basic.
$CONTENT-TYPE	Returns the content type of the request. For example: application/x-www-form-urlencoded.
$METHOD	Returns the request method. For example: GET.
$PATH-INFO	Returns the optional part of the requested path name that follows the application path and immediately precedes the query string. The returned string is unescaped. See also $PATH-TRANSLATED. For example: /extra/path.
$PATH-TRANSLATED	Returns the translated version of the optional part of the requested path name that follows the application path and immediately precedes the query string. The returned string is unescaped. This header name should not be used if Transaction Server Synchronized Recovery is required. See also $PATH-INFO. For example: /-/DISK/PUBLIC/WWWROOT/EXTRA/PATH/.
$PROTOCOL	Returns request protocol string as received from the client. The string returned is in the form of <protocol>/<major version>.<minor version>. For example: HTTP/1.1.
$QUERY-STRING	Returns the query string of the request as that received from the client. Might be escaped. For example: name1=The+first+value&name2=value2.
$REMOTE-ADDRESS	Returns the IP address of the agent that sent the request. For example: 192.63.223.164.
$REMOTE-HOST	Returns the fully qualified host name of the agent that sent the request, if the host name is available. For example: FISHERML.TR.UNISYS.COM.
$REMOTE-USER	Returns the name of the user making the request from the Authorization header of the request. If the Authorization header is not in the request, the value is null (that is, the request is anonymous). If the WEBPCM service attributes SHOWPW is TRUE and CHECKUSERAUTH is FALSE and if the Authorization header is present in the request, then the password from the Authorization header is appended to the user name with a colon separating. Examples: ADMIN and JDOE:ABC123. If NTLM is the authentication method, only the usercode is present.
$REQUEST-LINE	Returns the request line as received from the client. Data from the beginning of the request up to but not including the first request header is returned. It includes a Query String, if present. For example: GET /comsapp/ HTTP/1.0.

Value	Description
$REQUEST-PATH	Returns the requested path name after being unescaped. This might not be exactly the same as the path name received in the request, which might be escaped. The query string, if present, is not included (see $REQUEST-URI and $QUERY-STRING). For example: /apidemo/apienv/extra/path 1.
$REQUEST-URI	Returns the request universal request identifier (URI) as received from the client. Query string, if present, is also included (see $REQUEST-PATH and $QUERY-STRING). For example: /apidemo/apienv/extra/path?name1=The+first+value&name2= value2.
$SERVER-NAME	Returns the TCP/IP host name of the Web server. For example: trprogd.tr.unisys.com. Note that this might not be the same as the Host: header, which is what the user put in the request URL.

Multiple Headers

To make coding easier when multiple headers are needed, the following procedures are also exported, with the parameters matching those for GET_HEADER.

Each NAMEnn and VALUEnn parameter in the following is declared as EBCDIC ARRAY [0]:

```
INTEGER PROCEDURE GET_2_HEADERS (MSG,
           NAME01, VALUE01, NAME02, VALUE02);
INTEGER PROCEDURE GET_3_HEADERS (MSG,
           NAME01, VALUE01, NAME02, VALUE02,
           NAME03, VALUE03);
INTEGER PROCEDURE GET_4_HEADERS (MSG,
           NAME01, VALUE01, NAME02, VALUE02,
           NAME03, VALUE03, NAME04, VALUE04);
INTEGER PROCEDURE GET_5_HEADERS (MSG,
           NAME01, VALUE01, NAME02, VALUE02,
           NAME03, VALUE03, NAME04, VALUE04,
           NAME05, VALUE05);
INTEGER PROCEDURE GET_6_HEADERS (MSG,
           NAME01, VALUE01, NAME02, VALUE02,
           NAME03, VALUE03, NAME04, VALUE04,
           NAME05, VALUE05, NAME06, VALUE06);
INTEGER PROCEDURE GET_7_HEADERS (MSG,
           NAME01, VALUE01, NAME02, VALUE02,
           NAME03, VALUE03, NAME04, VALUE04,
           NAME05, VALUE05, NAME06, VALUE06,
           NAME07, VALUE07);
INTEGER PROCEDURE GET_8_HEADERS (MSG,
           NAME01, VALUE01, NAME02, VALUE02,
           NAME03, VALUE03, NAME04, VALUE04,
           NAME05, VALUE05, NAME06, VALUE06,
           NAME07, VALUE07, NAME08, VALUE08);
```

Each NAMEnn and VALUEnn parameter in the following is declared as EBCDIC ARRAY [*]:

```
INTEGER PROCEDURE get2Headers (MSG,
           NAME01, VALUE01, NAME02, VALUE02);
```

```
INTEGER PROCEDURE get3Headers (MSG,
           NAME01, VALUE01, NAME02, VALUE02,
           NAME03, VALUE03);
INTEGER PROCEDURE get4Headers (MSG,
           NAME01, VALUE01, NAME02, VALUE02,
           NAME03, VALUE03, NAME04, VALUE04);
INTEGER PROCEDURE get5Headers (MSG,
           NAME01, VALUE01, NAME02, VALUE02,
           NAME03, VALUE03, NAME04, VALUE04,
           NAME05, VALUE05);
INTEGER PROCEDURE get6Headers (MSG,
           NAME01, VALUE01, NAME02, VALUE02,
           NAME03, VALUE03, NAME04, VALUE04,
           NAME05, VALUE05, NAME06, VALUE06);
INTEGER PROCEDURE get7Headers (MSG,
           NAME01, VALUE01, NAME02, VALUE02,
           NAME03, VALUE03, NAME04, VALUE04,
           NAME05, VALUE05, NAME06, VALUE06,
           NAME07, VALUE07);
INTEGER PROCEDURE get8Headers (MSG,
           NAME01, VALUE01, NAME02, VALUE02,
           NAME03, VALUE03, NAME04, VALUE04,
           NAME05, VALUE05, NAME06, VALUE06,
           NAME07, VALUE07, NAME08, VALUE08);
```

Possible Result Values

In addition to the standard returned results, these possible values can be returned.

Value	Description
0	No-op. None of the headers are available.
1	Successful. All headers are successfully returned or at least one of the headers is successfully returned. If at least one of the headers is successfully returned, some of the VALUE0n parameters can be null strings.
-1	Invalid Transaction ID. The Message Object is corrupted.
–18	Buffer Too Small. The buffer is too small to receive the header data and the interface level is greater than 1.
–3	Software Error. A software error occurred.

GET_MESSAGE_LENGTH

Returns the actual length in bytes of the entire message, including the Trancode field. This procedure is useful for determining how many bytes to SEND or WRITE when returning a response.

Syntax

```
INTEGER PROCEDURE GET_MESSAGE_LENGTH (MSG, LEN);
  EBCDIC ARRAY                        MSG [0];
  INTEGER                                  LEN;

INTEGER PROCEDURE getMessageLength   (MSG, LEN);
  EBCDIC ARRAY                        MSG [*];
  INTEGER                                 LEN;
```

Parameters

MSG is the Message Object.

LEN is the message length in bytes.

GET_MIME_TYPE

Returns the MIME type associated with the supplied path name. A MIME type is configured to the server for each known file name extension (suffix). If the suffix is unknown, a configured default MIME type is returned.

This procedure should not be used if Transaction Server Synchronized Recovery is required.

Syntax

```
INTEGER PROCEDURE GET_MIME_TYPE (MSG, PATH, MIME_TYPE);
  EBCDIC ARRAY                  MSG [0];
  EBCDIC ARRAY                       PATH [0];
  EBCDIC ARRAY                             MIME_TYPE [0];

INTEGER PROCEDURE getMimeType   (MSG, PATH, MIME_TYPE);
  EBCDIC ARRAY                  MSG [*];
  EBCDIC ARRAY                       PATH [*];
  EBCDIC ARRAY                             MIME_TYPE [*];
```

Parameters

MSG is the Message Object.

PATH is the supplied virtual path name.

MIME_TYPE is the returned real path name. For example, a PATH of /myfile.htm can return a MIME_TYPE of text/html.

GET_POSTED_DATA

Returns the data in the request body (also known as Content Data). The application specifies the maximum data length to be returned, and the server returns the actual length read. One or more calls can be made until the whole request body is read.

Syntax

```
INTEGER PROCEDURE GET_POSTED_DATA
                (MSG, MAX_LEN, POST_DATA, POST_LEN);
  EBCDIC ARRAY   MSG [0];
  INTEGER             MAX_LEN;
  EBCDIC ARRAY                 POST_DATA [0];
  INTEGER                                 POST_LEN;

INTEGER PROCEDURE getPostedData
                (MSG, MAX_LEN, POST_DATA, POST_LEN);
  VALUE               MAX_LEN;
  EBCDIC ARRAY   MSG [*];
```

```
  INTEGER              MAX_LEN;
  EBCDIC ARRAY                 POST_DATA [*];
  INTEGER                                POST_LEN;
```

Parameters

MSG is the Message Object.

MAX_LEN is the maximum length in bytes to be returned, not including a terminator character. It should at least be big enough to hold the longest name and value pair.

POST_DATA is the post data returned, with a terminator character. The value fields are in URL-encoded format. For example: checkbox_1=checked&text1_A%26B. In this example, the user put the value A&B into the text1 field. The HTTP_UNESCAPE procedure can be used to unescape the value fields.

POST_LEN is the amount of data returned, not including the terminator character. If the request is not a POST request, zero is returned.

Note: *If the amount of data to be returned exceeds MAX-LEN, the last name and value pair is not truncated. This means POST-LEN could be returned with a value less than MAX-LEN, but there is still data to be read. The application should loop, calling the GET_POSTED_ DATA procedure until POST_LEN is zero.*

Possible Result Values

In addition to the standard returned results, these possible values can be returned.

Value	Description
0	No-op. All data has been previously read or there is no data.
–17	Translation is not available, and the mapping between the input and output character sets is not supported by the CENTRALSUPPORT and CCSFILE installed on the system.

GET_REAL_PATH

Requests the server to apply the alias rules to the supplied virtual path and returns the corresponding real path.

This procedure should not be used if Synchronized Recovery is required.

Syntax

```
INTEGER PROCEDURE GET_REAL_PATH (MSG, VIRTUAL_PATH, REAL_PATH);
  EBCDIC ARRAY                   MSG [0];
  EBCDIC ARRAY                        VIRTUAL_PATH [0];
  EBCDIC ARRAY                                      REAL_PATH [0];

INTEGER PROCEDURE getRealPath   (MSG, VIRTUAL_PATH, REAL_PATH);
  EBCDIC ARRAY                   MSG [*];
  EBCDIC ARRAY                        VIRTUAL_PATH [*];
  EBCDIC ARRAY                                      REAL_PATH [*];
```

Parameters

MSG is the Message Object.

VIRTUAL_PATH is the supplied virtual path name. For example: /icons/.

REAL_PATH is the returned real path name.
For example: /-/DISK/P UBLIC/WWWROOT/ATLAS/ICONS/.

GET_REQUEST_INFO

Returns general information about the transaction request. It is called by the application to predetermine lengths of request string attributes or data so it can allocate sufficient buffer space to get the attributes values or data.

Syntax

```
INTEGER PROCEDURE GET_REQUEST_INFO
              (MSG, REQUEST_LINE_LENGTH, URI_LENGTH,
                 PATH_LENGTH, QUERY_LENGTH, CONTENT_LENGTH,
                 TOTAL_LENGTH);
  EBCDIC ARRAY MSG [0];
  INTEGER           REQUEST_LINE_LENGTH, URI_LENGTH,
                 PATH_LENGTH, QUERY_LENGTH, CONTENT_LENGTH,
                 TOTAL_LENGTH;

INTEGER PROCEDURE getRequestInfo
              (MSG, REQUEST_LINE_LENGTH, URI_LENGTH,
                 PATH_LENGTH, QUERY_LENGTH, CONTENT_LENGTH,
                 TOTAL_LENGTH);
  EBCDIC ARRAY MSG [*];
  INTEGER           REQUEST_LINE_LENGTH, URI_LENGTH,
                 PATH_LENGTH, QUERY_LENGTH, CONTENT_LENGTH,
                 TOTAL_LENGTH;
```

Parameters

MSG is the Message Object.

REQUEST_LINE_LENGTH is the length of the request line as received from the client. This is the length of the string returned by GET_HEADER ($REQUEST-LINE).

HEADER_LENGTH is the total length of the all request headers (excluding the request line). This is the length of the string returned by utParseHeaders.

URI_LENGTH is the length of request universal request identifier (URI) (including query string) as received from the client. This is the length of the string returned by GET_HEADER ($REQUEST-URI).

PATH_LENGTH is the length of the request path name (excluding query string) after being unescaped. This length might not be the actual length of the requested path name because path name in the request might be escaped. This length is the length of the string returned by GET_HEADER ($REQUEST-PATH).

QUERY_LENGTH is the length of the request query string as received from the client that might have escape characters. This is the length of the string returned by GET_HEADER ($QUERY-STRING). Zero is returned if no query string is present.

CONTENT_LENGTH is the length of the request body. The value is zero if the Content-Length header is not present in the request.

TOTAL_LENGTH is the total length in bytes of the HTTP request, which is everything starting with the method, such as GET ..., to the end of the content data, if any.

GET_SERVER_PORT

Returns the port number on which the request was received.

Syntax

```
INTEGER PROCEDURE GET_SERVER_PORT (MSG, PORTNUM);
  EBCDIC ARRAY                      MSG [0];
  INTEGER                                PORTNUM;

INTEGER PROCEDURE getServerPort   (MSG, PORTNUM);
  EBCDIC ARRAY                      MSG [*];
  INTEGER                                PORTNUM;
```

Parameters

MSG is the Message Object.

PORTNUM is the port number on which the request was received.

GET_USER_AUTHORIZED

Indicates whether or not the user is authorized.

Syntax

```
INTEGER PROCEDURE GET_USER_AUTHORIZED (MSG);
  EBCDIC ARRAY                          MSG [0];

INTEGER PROCEDURE getUserAuthorized   (MSG);
  EBCDIC ARRAY                          MSG [*];
```

Parameters

MSG is the Message Object.

Possible Result Values

In addition to the standard returned results, these possible values can be returned.

Value	Description
0	No-op. The user is not authorized (that is, the user is anonymous).

1	Successful. The user is authorized and has passed validity checking by providing a valid MCP usercode and password with the request.

GET_USER_PRIVILEGE

Indicates whether or not the user has the specified privilege.

Syntax

```
INTEGER PROCEDURE GET_USER_PRIVILEGE (MSG, PRIVILEGE);
  EBCDIC ARRAY                        MSG [0];
  INTEGER                                  PRIVILEGE;

INTEGER PROCEDURE getUserPrivilege   (MSG, PRIVILEGE);
  VALUE                                    PRIVILEGE;
  EBCDIC ARRAY                        MSG [*];
  INTEGER                                  PRIVILEGE;
```

Parameters

MSG is the Message Object.

PRIVILEGE is the privilege requested and is one of the following values:

1: PU

2: SECADMIN

3: SYSADMIN

4: SYSTEMUSER

5: CHANGE

6: CHANGESEC

7: COMSCONTROL

8: CREATEFILE

9: EXECUTE

10: GETSTATUS

11: GSDIRECTORY

12: IDC

13: LOCALCOPY

14: LOGINSTALL

15: LOGOTHERS

16: RESERVED

17: RESERVED

18: READ

19: REMOVE

20: SETSTATUS

21: USERDATA

22: WRITE

Possible Result Values

In addition to the standard returned results, these possible values can be returned.

Value	Description
0	No-op. The user does not have the privilege.
1	Successful. The user has the privilege.
–34	Unsupported Privilege. The privilege value is not supported.

GET_USER_PRIVILEGED

Indicates whether or not the user is privileged.

Syntax

```
INTEGER PROCEDURE GET_USER_PRIVILEGED (MSG);
  EBCDIC ARRAY                          MSG [0];

INTEGER PROCEDURE getUserPrivileged   (MSG);
  EBCDIC ARRAY                          MSG [*];
```

Parameters

MSG is the Message Object.

Possible Result Values

In addition to the standard returned results, these possible values can be returned.

Value	Description
0	No-op. The user is not privileged.
1	Successful. The user is privileged.

PARSE_COOKIES

Parses the cookie headers of the request. The application supplies a buffer into which the server is to return the result of the parsing.

Syntax

```
INTEGER PROCEDURE PARSE_COOKIES
               (MSG, MAX_NAME_LEN, MAX_VALUE_LEN, MAX_PATH_LEN,
                  MAX_DOMAIN_LEN, MAX_PORT_LEN,  VERSION,
                  BUFFER, NUM_COOKIES);
  EBCDIC ARRAY  MSG [0];
  INTEGER            MAX_NAME_LEN, MAX_VALUE_LEN, MAX_PATH_LEN,
```

```
                 MAX_DOMAIN_LEN, MAX_PORT_LEN,  VERSION;
  EBCDIC ARRAY    BUFFER [0];
  INTEGER                 NUM_COOKIES;

INTEGER PROCEDURE parseCookies
                (MSG, MAX_NAME_LEN, MAX_VALUE_LEN, MAX_PATH_LEN,
                   MAX_DOMAIN_LEN, MAX_PORT_LEN,  VERSION,
                   BUFFER, NUM_COOKIES);
  VALUE               MAX_NAME_LEN, MAX_VALUE_LEN, MAX_PATH_LEN,
                   MAX_DOMAIN_LEN, MAX_PORT_LEN;
  EBCDIC ARRAY   MSG [*];
  INTEGER             MAX_NAME_LEN, MAX_VALUE_LEN, MAX_PATH_LEN,
                   MAX_DOMAIN_LEN, MAX_PORT_LEN,  VERSION;
  EBCDIC ARRAY     BUFFER [*];
  INTEGER                  NUM_COOKIES);
```

Parameters

MSG is the Message Object.

MAX_NAME_LEN is the size of the name column.

MAX_VALUE_LEN is the size of the value column.

MAX_PATH_LEN is the size of the path column and is used only if the user agent is using version 1 cookies.

MAX_DOMAIN_LEN is the size of the domain column and is used only if the user agent is using version 1 cookies.

MAX_PORT_LEN is the size of the port column and is used only if the user agent is using version 1 cookies.

VERSION is the version of the cookie: 0 = Netscape format cookie.

BUFFER is the buffer into which the data is returned.

NUM_COOKIES is the number of cookies returned.

Only Version 0 cookies are supported.

Possible Result Values

In addition to the standard returned results, these possible values can be returned.

Value	Description
–17	Translation is not available, and the mapping between the input and output character sets is not supported by the CENTRALSUPPORT and CCSFILE installed on the system.
–3	Software Error. If the length of a returned name, including any terminating byte, exceeds the MAX_NAME_LEN procedure, or if the length of a returned value, including any terminating byte, exceeds the MAX_VALUE_LEN parameter, WEBAPPSUPPORT stops processing the request and returns this value.

Example

In COBOL, you might declare

```
01 COOKIE-BUFFER.
  03 COOKIE-INFO OCCURS 10 TIMES.
    05 COOKIE-NAME  PIC X(20).
    05 COOKIE-VALUE PIC X(100).
    05 COOKIE-PATH  PIC X(100).
    05 COOKIE-DOMAIN PIC X(30).
    05 COOKIE-PORT  PIC  X(5).
```

The call to PARSE_COOKIES passes COOKIE-BUFFER, with MAX_NAME_LEN set to 20, and MAX_VALUE_LEN set to 100, etc.

PARSE_HEADER

Parses the query string of the request and returns all of the HTTP headers of the request. This procedure is similar to the PARSE_QUERY_STRING procedure.

Syntax

```
INTEGER PROCEDURE PARSE_HEADERS
               (MSG, MAX_NAME_LEN, MAX_VALUE_LEN, BUFFER,
                  NUM_PAIRS);
  EBCDIC ARRAY  MSG [0];
  INTEGER           MAX_NAME_LEN, MAX_VALUE_LEN;
  EBCDIC ARRAY                                    BUFFER [0];
  INTEGER         NUM_PAIRS;

INTEGER PROCEDURE parseHeaders
               (MSG, MAX_NAME_LEN, MAX_VALUE_LEN, BUFFER,
                  NUM_PAIRS);
  VALUE              MAX_NAME_LEN, MAX_VALUE_LEN;
  EBCDIC ARRAY  MSG [*];
  INTEGER            MAX_NAME_LEN, MAX_VALUE_LEN;
  EBCDIC ARRAY                                     BUFFER [*];
  INTEGER         NUM_PAIRS;
```

Parameters

MSG is the Message Object.

MAX_NAME_LEN is the size of the name column.

MAX_VALUE_LEN is the size of the value column.

BUFFER is the buffer into which the data is returned, represented as pairs of strings.

NUM_PAIRS is the number of pairs returned.

Possible Result Values

In addition to the standard returned results, these possible values can be returned.

Value	Description
–17	Translation is not available, and the mapping between the input and output character sets is not supported by the CENTRALSUPPORT and CCSFILE installed on the system.

PARSE_POST_DATA

See also GET_HEADER ($QUERY_STRING).

Similar to PARSE_QUERY_STRING except the procedure operates on the request body of a POST request, not on the query string. The Content-type of the request body must be of the form application/x-www-form-urlencoded. See also GET_POSTED_DATA.

You cannot use this procedure for requests with content length greater than 16,777,215 bytes.

Syntax

```
INTEGER PROCEDURE PARSE_POST_DATA
                (MSG, MAX_NAME_LEN, MAX_VALUE_LEN, BUFFER,
                   NUM_PAIRS);
  EBCDIC ARRAY   MSG [0];
  INTEGER            MAX_NAME_LEN, MAX_VALUE_LEN;
  EBCDIC ARRAY                                     BUFFER [0];
  INTEGER          NUM_PAIRS;

INTEGER PROCEDURE parsePostData
                (MSG, MAX_NAME_LEN, MAX_VALUE_LEN, BUFFER,
                   NUM_PAIRS);
  VALUE              MAX_NAME_LEN, MAX_VALUE_LEN;
  EBCDIC ARRAY   MSG [*];
  INTEGER            MAX_NAME_LEN, MAX_VALUE_LEN;
  EBCDIC ARRAY                                     BUFFER [*];
  INTEGER          NUM_PAIRS;
```

Parameters

MSG is the Message Object.

MAX_NAME_LEN is the size of the name column.

MAX_VALUE_LEN is the size of the value column.

BUFFER is the buffer into which the data is returned, represented as pairs of unescaped strings.

NUM_PAIRS is the number of pairs returned.

Possible Result Values

In addition to the standard returned results, these possible values can be returned.

Value	Description
–17	Translation is not available, and the mapping between the input and output character sets is not supported by the CENTRALSUPPORT and CCSFILE installed on the system.

PARSE_QUERY_STRING

Parses the query string of the request. The application supplies a buffer into which the server is to return the result of unescaping the received query string [plus signs (+) are also translated to spaces]. If the data does not contain name and value pairs (meaning the value part is absent), information is returned for the names only.

If the length of a returned name, including any terminating byte, exceeds the MAX_NAME_LEN procedure, or if the length of a returned value, including any terminating byte, exceeds the MAX_VALUE_LEN parameter, WEBAPPSUPPORT stops processing the request and returns a Software Error (–3) result.

The intent is to make it easy for COBOL applications to handle name and value pair data.

Syntax

```
INTEGER PROCEDURE PARSE_QUERY_STRING
               (MSG, MAX_NAME_LEN, MAX_VALUE_LEN, BUFFER,
                  NUM_PAIRS);
  EBCDIC ARRAY  MSG [0];
  INTEGER            MAX_NAME_LEN, MAX_VALUE_LEN;
  EBCDIC ARRAY                                     BUFFER [0];
  INTEGER         NUM_PAIRS;

INTEGER PROCEDURE parseQueryString
               (MSG, MAX_NAME_LEN, MAX_VALUE_LEN, UFFER,
                  NUM_PAIRS);
  VALUE              MAX_NAME_LEN, MAX_VALUE_LEN;
  EBCDIC ARRAY  MSG [*];
  INTEGER            MAX_NAME_LEN, MAX_VALUE_LEN;
```

```
  EBCDIC ARRAY                                    BUFFER [*];
  INTEGER         NUM_PAIRS;
```

Parameters

MSG is the Message Object.

MAX_NAME_LEN is the size of the name column.

MAX_VALUE_LEN is the size of the value column.

BUFFER is the buffer into which the data is returned, represented as pairs of strings.

NUM_PAIRS is the number of pairs returned.

Possible Result Values

In addition to the standard returned results, these possible values can be returned.

Value	Description
–17	Translation is not available, and the mapping between the input and output character sets is not supported by the CENTRALSUPPORT and CCSFILE installed on the system.

Example

In COBOL, you might declare the following:

```
01 NAME-VALUE-BUFFER.
  03 NAME-VALUE-PAIR OCCURS 10 TIMES.
    05 QUERY-NAME PIC X(20).
    05 QUERY-VALUE PIC X(100).
```

The call to PARSE_QUERY_STRING passes NAME-VALUE-BUFFER, with MAX_NAME_LEN set to 20, and MAX_VALUE_LEN set to 100.

SET_CONTENT

Sets the response body. One or more calls to this procedure can be made until the whole content length is set.

Syntax

```
INTEGER PROCEDURE SET_CONTENT
                (MSG, RSP_DATA, DATA_START, DATA_LEN, COMPLETE);
  EBCDIC ARRAY   MSG, RSP_DATA [0];
  INTEGER                      DATA_START, DATA_LEN, COMPLETE;

INTEGER PROCEDURE setContent
                (MSG, RSP_DATA, DATA_START, DATA_LEN, COMPLETE);
  VALUE                        DATA_START, DATA_LEN, COMPLETE;
  EBCDIC ARRAY   MSG, RSP_DATA [*];
  INTEGER                      DATA_START, DATA_LEN;
  BOOLEAN                                            COMPLETE;
```

Parameters

MSG is the Message Object.

RSP_DATA is the data to send, without terminating characters.

DATA_START is the index in RSP_DATA to start copying the data. For calls to SET_CONTENT, this is one-based (a value of 1 indicates that data should be copied from the first byte). For calls to setContent, this is zero-based.

DATA_LEN is the amount of data in RSP_DATA to send. If the value is zero or less, the previously set content in the message object is cleared. The maximum amount of data that can be set depends on the amount of free space in the MSG buffer declared in the application.

COMPLETE indicates this is the last (or only) segment of data: 0 = FALSE, 1 = TRUE.

If multiple segments are to be sent to the user, the HTTP header Content-Length must be set with a call to SET_HEADER prior to the first SEND or WRITE of the message object. If only one segment is being sent, the Content-Length header does not need to be set.

Possible Result Values

In addition to the standard returned results, these possible values can be returned.

Value	Description
–17	Translation is not available, and the mapping between the input and output character sets is not supported by the CENTRALSUPPORT and CCSFILE installed on the system.

SET_CONTENT_TYPE

Sets the Content-type header of the response. By default (that is, this procedure is not called), if the response does not have a body, then the Content-Type header is not sent to the user; otherwise, it is text/html. See also SET_STATUS_CODE.

Syntax

```
INTEGER PROCEDURE SET_CONTENT_TYPE
               (MSG, CONTENT_TYPE);
  EBCDIC ARRAY  MSG [0];
  EBCDIC ARRAY       CONTENT_TYPE [0];

INTEGER PROCEDURE setContentType
               (MSG, CONTENT_TYPE);
  EBCDIC ARRAY  MSG [*];
  EBCDIC ARRAY       CONTENT_TYPE [*];
```

Parameters

MSG is the Message Object.

CONTENT_TYPE is the response content type.

Possible Result Values

In addition to the standard returned results, these possible values can be returned.

Value	Description
–33	An invalid character for a response header is in the text supplied by the application, and the application is Interface Version 4 or higher. If the application is at Interface Version 3 or lower, a –3 (Software Error) is returned instead for an invalid character in a response header.

SET_COOKIE

Sets a Netscape style cookie header. It can be called multiple times, with each call concatenated to the previous.

Syntax

```
INTEGER PROCEDURE SET_COOKIE (MSG, COOKIE_NAME, COOKIE_VALUE,
                      EXPIRES, DOMAIN, PATH, SECURE);
  EBCDIC ARRAY               MSG, COOKIE_NAME, COOKIE_VALUE [0];
  EBCDIC ARRAY         EXPIRES, DOMAIN, PATH [0];
  INTEGER                                     SECURE;

INTEGER PROCEDURE setCookie  (MSG, COOKIE_NAME, COOKIE_VALUE,
                      EXPIRES, DOMAIN, PATH, SECURE);
  VALUE                                       SECURE;
  EBCDIC ARRAY               MSG, COOKIE_NAME, COOKIE_VALUE [*];
  EBCDIC ARRAY         EXPIRES, DOMAIN, PATH [*];
  BOOLEAN                                     SECURE;
```

Parameters

MSG is the Message Object.

COOKIE_NAME is the name of the cookie. This parameter is required to be a nonnull string, for example: CUSTOMER.

COOKIE_VALUE is the value of the cookie. This parameter is required to be a nonnull string. The length of this field should not exceed 4,000 bytes unless it is certain that the client (browser) can handle a longer value. The actual absolute size of this field is 10,000 bytes, for example: WILE_E_COYOTE.

EXPIRES is the date that specifies the valid life of the cookie. Once the expiration date has been reached, the cookie is no longer stored or given out. The parameter must be in RFC1123 format, with the further restriction that the time zone must be GMT, and only dashes can separate the date elements. If null, the expired attribute is absent from the header, for example: Mon, 14-Sep-1998 14:30:00 GMT.

DOMAIN is the Internet domain to which the cookie can be returned. It must contain at least two periods if the top level domain is com, edu, net, org, gov, mil, or int; otherwise, it must contain at least three periods. If the value is null, the domain attribute is absent from the header, and the cookie is sent only to the host that set the cookie. For example: .acme.com sends the cookie to hosts anvil.acme.com and shipping.crate.acme.com.

PATH is the subset of URLs in a domain for which the cookie is valid. If the value is null, the path attribute is absent from the header, and the path is assumed to be the same path as the document being described by the header that contains the cookie. For example: "foot" would match /football and /foot/ball.html.

SECURE indicates the cookie is secure, meaning the cookie is only returned if the communications channel with the host is secure (that is, uses SSL). The values are 0 =FALSE or 1=TRUE.

Possible Result Values

In addition to the standard returned results, these possible values can be returned.

Value	Description
–17	Translation is not available, and the mapping between the input and output character sets is not supported by the CENTRALSUPPORT and CCSFILE installed on the system.
–33	An invalid character for a response header is in the text supplied by the application, and the application is Interface Version 4 or higher. If the application is at Interface Version 3 or lower, a –3 (Software Error) is returned instead for an invalid character in a response header.

SET_HEADER

Sets a string HTTP header of the response.

Syntax

```
INTEGER PROCEDURE SET_HEADER (MSG, HEADER_NAME, HEADER_VALUE);
  EBCDIC ARRAY                MSG [0];
  EBCDIC ARRAY                     HEADER_NAME [0];
  EBCDIC ARRAY                                  HEADER_VALUE [0];

INTEGER PROCEDURE setHeader  (MSG, HEADER_NAME, HEADER_VALUE);
  EBCDIC ARRAY                MSG [*];
  EBCDIC ARRAY                     HEADER_NAME [*];
  EBCDIC ARRAY                                  HEADER_VALUE [*];
```

Parameters

MSG is the Message Object.

HEADER_NAME is the requested header. Client (browser) processing of HTTP headers can be case sensitive. Specify the HEADER_HAME with the same case as that specified by the HTTP specification -for example: Expires.

HEADER_VALUE is the supplied header value -for example: Tue, 14 Jul 1998 17:28:31 GMT.

Possible Result Values

In addition to the standard returned results, these possible values can be returned.

Value	Description
–33	An invalid character for a response header is in the text supplied by the application, and the application is Interface Version 4 or higher. If the application is at Interface Version 3 or lower, a –3 (Software Error) is returned instead for an invalid character in a response header.

SET_REDIRECT

Sets a redirect response with the specified location. Sets the status code to 303 (see Other), and the Location header to the NEW_URL parameter. SET_CONTENT can be called to send response content after SET_REDIRECT is called. Otherwise, the Message Object should be sent to the user after calling this procedure.

Syntax

```
INTEGER PROCEDURE SET_REDIRECT (MSG, NEW_URL);
  EBCDIC ARRAY                  MSG [0];
  EBCDIC ARRAY                       NEW_URL [0];

INTEGER PROCEDURE setRedirect  (MSG, NEW_URL);
  EBCDIC ARRAY                  MSG [*];
  EBCDIC ARRAY                       NEW_URL [*];
```

Parameters

MSG is the Message Object.

NEW_URL is the string value for the Location header.

Possible Result Values

In addition to the standard returned results, these possible values can be returned.

Value	Description
–33	An invalid character for a response header is in the text supplied by the application, and the application is Interface Version 4 or higher. If the application is at Interface Version 3 or lower, a –3 (Software Error) is returned instead for an invalid character in a response header.

SET_SSI

Controls Server Side Include (SSI) processing of content data in the response.

See the *Web Transaction Server Administration and Programming Guide* for a list of supported SSI directives.

Syntax

```
INTEGER PROCEDURE SET_SSI   (MSG, SSI);
  VALUE                           SSI;
  EBCDIC ARRAY                MSG [*];
  BOOLEAN                         SSI;

INTEGER PROCEDURE setSSI    (MSG, SSI);
  VALUE                           SSI;
  EBCDIC ARRAY                MSG [*];
  BOOLEAN                         SSI;
```

Parameters

MSG is the Message Object.

SSI indicates whether or not to process the response content for SSI directives. Values are 0 = FALSE or 1 = TRUE. The default is FALSE.

SET_STATUS_CODE

Sets the status code and optionally sets the reason string in the response. If the reason string is null, the server uses the default reason message for the status code.

If this procedure is not called, the default status code is 200 (OK), with no reason text.

Syntax

```
INTEGER PROCEDURE SET_STATUS_CODE
               (MSG, STATUS_CODE, STATUS_SUBCODE, REASON,
                  REASON_LEN);
  EBCDIC ARRAY  MSG [0];
  INTEGER           STATUS_CODE, STATUS_SUBCODE,
                  REASON_LEN;
  EBCDIC ARRAY                                    REASON [0];

INTEGER PROCEDURE setStatusCode
               (MSG, STATUS_CODE, STATUS_SUBCODE, REASON,
                  REASON_LEN);
  VALUE             STATUS_CODE, STATUS_SUBCODE,
                  REASON_LEN;
  EBCDIC ARRAY  MSG [*];
  INTEGER           STATUS_CODE, STATUS_SUBCODE,
                  REASON_LEN;
  EBCDIC ARRAY                                    REASON [*];
```

Parameters

MSG is the Message Object.

STATUS_CODE is the HTTP status code for the response. Typical values are 400 (Bad Request) and 500 (Internal Software Error).

STATUS_SUBCODE is a subcode for certain status codes, so that unique text responses can be returned depending on the specific reason. This field is ignored if REASON is non null. The list of valid subcodes is defined in the *Web Transaction Server Administration and Programming Guide*.

REASON is the optional reason. The format should match the defined content type for the response, which defaults to text/html.

REASON_LEN is the length of REASON in bytes, up to a maximum of 65000 bytes.

Possible Result Values

In addition to the standard returned results, these possible values can be returned.

Value	Description
–17	Translation is not available, and the mapping between the input and output character sets is not supported by the CENTRALSUPPORT and CCSFILE installed on the system.

VALIDATE_REQUEST

Used to direct the authentication of the requestor. The Web Transaction Server handles the authentication and validates the user to MCP USERDATA.

The application is responsible for returning the response to the user, even if the validation is rejected. This includes returning a challenge response in a multi-step validation, such as for NTLM or Kerberos. The Web Transaction Server sets the WWW-Authenticate: response header.

Also, if present, the SECURITYSUPPORT library is called.

Syntax

```
INTEGER PROCEDURE VALIDATE_REQUEST
                     (MSG, METHOD, ASSUMEUC, REALM,
                        STATUS_CODE, STATUS_SUBCODE,
                          DELAY_RSP_TIME, USER, SUPPLEMENTAL);
 EBCDIC ARRAY       MSG,                     REALM,
                                           USER, SUPPLEMENTAL [0];
   INTEGER                 METHOD, ASSUMEUC,
                       STATUS_CODE, STATUS_SUBCODE,
                         DELAY_RSP_TIME;
```

```
INTEGER PROCEDURE validateRequest
                     (MSG, METHOD, ASSUMEUC, REALM,
                        STATUS_CODE, STATUS_SUBCODE,
                          DELAY_RSP_TIME, USER, SUPPLEMENTAL);
     VALUE                  METHOD, ASSUMEUC;
     EBCDIC ARRAY       MSG,                  REALM,
                                          USER, SUPPLEMENTAL [*];
     INTEGER                METHOD, ASSUMEUC,
                          STATUS_CODE, STATUS_SUBCODE,
                            DELAY_RSP_TIME;
```

Parameters

MSG is the Message Object.

METHOD is the authentication method to use.

- 1 = HTTP Basic
- 2 = NTLM Only
- 3 = Kerberos or NTLM

ASSUMEUC is currently not supported. It should be set to zero by the application.

REALM is the realm to be used when METHOD = 1. It is a string in the application's character set. If null or empty, the WEBPCM service path is used, for example: /comsdemo1/. If METHOD is not 1, then the first byte should be set to either a null byte or a space character.

STATUS_CODE is the HTTP status code that results from the validation process. This value can be used with the SET_STATUS_CODE procedure to return the response. Some examples of returned values are

- 200 = Successful
- 401 = Unauthorized
- 403 = Forbidden

STATUS_SUBCODE is a subcode for certain status codes that result from the validation process. This value can be used with the SET_STATUS_CODE procedure to return a final response. The list of valid subcodes is defined in the *Web Transaction Server Administration and Programming Guide* under the "CustomErrors" directive.

DELAY_RSP_TIME is the time in seconds that the application should wait before returning the response. This value is a non-zero when the Delay Authentication Retry feature in the Web Transaction Server provider is enabled, and the HTTP client has had one or more failed validation attempts.

USER is the authenticated usercode, accesscode, and chargecode in the application's character set. Each value is a string, terminated according to the applications string termination setting. It maps to the COBOL structure:

```
01 USER-BUFFER.
  03 USER-USERCODE    PIC X(18).
  03 USER-ACCESSCODE  PIC X(18).
  03 USER-CHARGECODE  PIC X(61).
  03 USER-EXTERNALID  PIC X(256).
```

SUPPLEMENTAL is reserved for future use. The first byte should be set to either a null byte or a space character.

Possible Result Values

In addition to the standard returned results, these possible values can be returned.

Value	Description
0	Unsupported METHOD parameter, or application must return STATUS_CODE and STATUS_SUBCODE to client.

XML Procedures

Refer to Section 6, "WEBAPPSUPPORT Library Interface for the XML Parser," for information on XML WEBAPPSUPPORT procedures.

HTTP Client Procedures

Refer to Section 9, "HTTP Client Applications," for information on HTTP WEBAPPSUPPORT procedures.

Regular Expressions Procedures

Refer to Section 10, "Using Regular Expressions," for information about the Regular Expressions WEBAPPSUPPORT procedures.

Section 4
XML Parser Administration

Installing the XML Parser

To install the XML Parser, do the following:

1. Ensure that the Custom Connect Facility (CCF) is installed on the MCP system.
2. Install the Java Parser Module (JPM) on any of the following:
   - MCP Java Processor
   - Microsoft Windows system with Sun Java 6.0 or 7.0 JDK

On MCP Java

The WFL named *SYSTEM/CCF/XMLPARSER/WFL/JAVA is supplied with the XML Parser and installs the JPM. Also, if WEBAPPSUPPORT is configured to run the JPM, WEBAPPSUPPORT installs the JPM to MCP Java.

On Microsoft Windows

To install the JPM on a Microsoft Windows system, do the following:

1. Map a drive to the MCP installs share, which is `*SYSTEM/INSTALLS`.
2. Copy the files from the installs share folder \XMLJavaParser on the MCP installs share to a folder on your Windows system.

   For example, you could copy the files to the folder c:\Program Files\Unisys\XMLJAVAPARSER.
3. Set the system environment variable JAVA_HOME to point to the folder for Java JRE.

   This step is required.

   For example, the variable JAVA_HOME could point to c:\Program Files\Java\jre1.6.0_12.
4. Set the system environment variable JPM_HOME to point to the JPM directory.

   This step is required.

   For example, the variable JPM_HOME could point to c:\Program Files\Unisys\XMLJAVAPARSER.

5. Set the system environment variable JPM_OPTS to set the Java options needed to run the JPM.

   These options include memory sizes, garbage collection settings, and other options.

   For example, type

   ```
   -server
   ```

6. In the folder c:\Program Files\Unisys\XMLJAVAPARSER\JPM1\config copy the file JPMConfigSAMPLE.xml as JPMConfig.xml.

   JPMConfig.xml is the file that contains specific configuration for the JPM.

Installed Files

The following XML Parser files are installed to the MCP with the CCF product:

- *SYSTEM/CCF/XMLPARSER/WFL/JAVA

  This file contains a WFL for running the JPM on MCP Java.
- *SYSTEM/CCF/XMLPARSER/SAMPLE/PARSEXML/ALGOL

  This file contains a sample ALGOL application that parses an XML document.
- *SYSTEM/CCF/XMLPARSER/SAMPLE/PARSEXML/COBOL

  This file contains a sample COBOL85 application that parses an XML document.
- *SYSTEM/CCF/XMLPARSER/SAMPLE/CREATEXML/ALGOL

  This file contains a sample ALGOL application that creates an XML document.
- *SYSTEM/CCF/XMLPARSER/SAMPLE/CREATEXML/COBOL

  This file contains a sample COBOL85 application that creates an XML document.
- *SYSTEM/CCF/XMLPARSER/SAMPLE/TRANSFORMXML/ALGOL

  This file contains a sample ALGOL application that transforms an XML document.
- *SYSTEM/CCF/XMLPARSER/SAMPLE/TRANSFORMXML/COBOL

  This file contains a sample COBOL85 application that transforms an XML document.
- *SYSTEM/INSTALLS/XMLJAVAPARSER/BIN/=

  This directory contains binary files and Windows .bat files for the JPM.
- *SYSTEM/INSTALLS/XMLJAVAPARSER/JPM1/CONFIG/"JPMCONFIGSAMPLE.XML"

  This file contains a sample JPM configuration.
- *SYSTEM/INSTALLS/XMLJAVAPARSER/JPM1/CONFIG/"JPMLOGPROPERTIES"

  This file contains sample log4j properties. You probably do not need to modify this file.

- *SYSTEM/INSTALLS/XMLJAVAPARSER/"README.TXT"

  This text file contains instructions for using the JPM on systems other than MCP systems.

- *SYSTEM/CCF/WEBAPPSUPPORT/PARAMS/XML/EXAMPLE

  This file contains sample parameters for XML settings that the WEBAPPSUPPORT library uses.

Installing Updates

After you install the XML Parser, you might need to install an update that Unisys supplied. To install an update, do the following:

1. Update the CCF product.

   Follow the instructions in the CCF Interim Correction (IC) cover letter.

   The IC installation puts a new version of the WEBAPPSUPPORT library on the MCP system. If applications are linked to the old WEBAPPSUPPORT library, the old library does not terminate when you install the IC. The old library terminates only when the applications delink from WEBAPPSUPPORT.

   Two instances of the WEBAPPSUPPORT library can run at the same time, but operator commands entered through WEBPCM only go to the old WEBAPPSUPPORT library.

2. Update the (JPM).

   See "Updating the XML Parser JPM."

Configuring the XML Parser

To configure the XML Parser, do the following:

1. Configure the WEBAPPSUPPORT library for its use of the JPMs.
2. Configure each JPM.

WEBAPPSUPPORT XML Parser Configuration File

The WEBAPPSUPPORT XML Parser Configuration File (*SYSTEM/CCF/WEBAPPSUPPORT/PARAMS/XML) is a text file containing the configuration for the WEBAPPSUPPORT part of XML. This file is changed to support the new PARSER directive. The PARSER directive replaces the use of the PARSERHOST and PARSERPORT directives.

Syntax

```
PARSER <parser number> {
   HOST <domain name|IP address>;
   PORT <port number>;
   STANDBY <true|false>;
   INITIATEJVM <true|false>;
   TARGET <java server number>;
   JAVAFAMILY <quoted string>;
```

```
    JAVAHOMEDIR <quoted string>;
    JVMATTRS <quoted string>;
    JPMFAMILY <quoted string>;
    JPMHOMEDIR <quoted string>;
    TASKATTRS <quoted string>;
}
```

The PARSER attributes shown in the following table are optional unless marked as required.

Attribute	Description
<parser number>	The parser number, starting at 1. Parser numbers must be sequential, for example, 1, 2, 3, and so forth. This attribute is required.
HOST	The domain name or IP address of the JPM. If the JPM is running on MCP Java, this name is recommended to be the EVLAN IP address of the Java Server of the JPM. This attribute is required.
PORT	The port number of the JPM. The default is 51117.
STANDBY	If true, the JPM is used for backup only if an active JPM cannot be used. If false, the JPM is an active JPM. The default is false.
INITIATEJVM	If true, WEBAPPSUPPORT initiates the JPM(s) on MCP Java at the initialization of WEBAPPSUPPORT. If false the JPM is not initiated by WEBAPPSUPPORT. The default is false.
TARGET	The MCP Java server number. The default is 1.
JAVAFAMILY	A <quoted string> that specifies the family where MCP Java is installed. This attribute defaults to the SL family of the JAVASUPPORT function name.
JAVAHOMEDIR	A <quoted string> that specifies the top level directory of the MCP Java installation, for example "JRE6". This attribute is required if INITIATEJVM is true.
JVMATTRS	A <quoted string> of attributes to pass to the JVM. Do not specify a classpath or jar. The default is `"-server -Xshare:off -XX:+UseParallelGC -XX:ParallelGCThreads=4 -XX:-UseAdaptiveSizePolicy -Xmn458m -Xms1376M -Xmx1376M"`
JPMFAMILY	A <quoted string> that specifies the family where the JPM is installed. This attribute defaults to the SL family of the JAVASUPPORT function name.
JPMHOMEDIR	A <quoted string> that specifies the home directory for the JPM. This default is "XMLJPM".
TASKATTRS	A <quoted string> of MCP task attributes to apply when running the JPM. The CURRENTDIRECTORY, FILE JAVAHOMEPATH, and FILE STDIN attributes are set by WEBAPPSUPPORT and should not be specified in this attribute. The default is `"MPID=XMLJPM<parser number>; FILE STDOUT(<stdout parameters>); FILE STDERR(<stderr parameters>);"`

The variables shown in the following table apply to the attributes described in the previous table.

Variable	Description
<quoted string>	A string of characters enclosed by quote (") or apostrophe (') characters. The string can be continued across multiple lines by appending multiple quoted strings.
<java home>	/-/<java family>/DIR/<java home dir>
<jpm home>	/-/<jpm family>/DIR/<jpm home dir>
<stdin parameters>	DISK,TITLE=*DIR/<java home dir>/LICENSE ON <java family>
<stdout parameters>	KIND=DISK, LFILENAME=*DIR/<jpm home dir>/JPM<jpm number>/LOGS/"STDOUT-$DATE-$TIME.TXT", FAMILYNAME=<jpm family>, EXTMODE=ASCII, PROTECTION=PROTECTED, UNIQUETOKEN="$"
<stderr parameters>	KIND=DISK, LFILENAME=*DIR/<jpm home dir>/JPM<jpm number>/LOGS/"STDERR-$DATE-$TIME.TXT", FAMILYNAME=<jpm family>, EXTMODE=ASCII, PROTECTION=PROTECTED, UNIQUETOKEN="$"

A sample minimal configuration file that runs one JPM on Java Server 1 might look as follows:

```
% Configuration To Java Parser Modules on MCP JProcessor
PARSER 1 {
   HOST        192.168.16.21;
   INITIATEJVM true;
   JAVAHOMEDIR "JRE6";
}
```

A sample configuration file with two JPMs each running on a separate JDP, a standby JPM running on Windows, and all attributes specified might look like the following:

```
% Configuration To Java Parser Modules on MCP JProcessor
PARSER 1 {
   HOST        192.168.16.21;
   PORT        51117;
   STANDBY     false;
   INITIATEJVM true;
   TARGET      1;
   JAVAFAMILY  "DISK";
   JAVAHOMEDIR "JRE6";
   JVMATTRS   "-server -Xshare:off -XX:+UseParallelGC
               -XX:ParallelGCThreads=4 -XX:-UseAdaptiveSizePolicy
               -Xmn458m -Xms1376M -Xmx1376M";
   JPMFAMILY   "DISK";
   JPMHOMEDIR  "XMLJPM";
   TASKATTRS   "MPID=XMLJPM1; "

    "FILE STDOUT(KIND=DISK, "
```

```
      'LFILENAME=*DIR/XMLJPM/JPM1/LOGS/"STDOUT-$DATE-$TIME.TXT", '
      "FAMILYNAME=DISK, "
      'EXTMODE=ASCII, PROTECTION=PROTECTED, UNIQUETOKEN="$"); '
    "FILE STDERR(KIND=DISK, "
      'LFILENAME=*DIR/XMLJPM/JPM1/LOGS/"STDERR-$DATE-$TIME.TXT", '
      "FAMILYNAME=DISK, "
      'EXTMODE=ASCII, PROTECTION=PROTECTED, UNIQUETOKEN="$");';
}

PARSER 2 {
   HOST        192.168.16.31;
   PORT        51117;
   STANDBY     false;
   INITIATEJVM true;
   TARGET      2;
   JAVAFAMILY  "DISK";
   JAVAHOMEDIR "JRE6";
      JVMATTRS   "-server -Xshare:off -XX:+UseParallelGC
               -XX:ParallelGCThreads=4 -XX:-UseAdaptiveSizePolicy
               -Xmn458m -Xms1376M -Xmx1376M";
   JPMFAMILY   "DISK";
   JPMHOMEDIR  "XMLJPM";
   TASKATTRS   "MPID=XMLJPM2; "
   "FILE STDOUT(KIND=DISK, "
      'LFILENAME=*DIR/XMLJPM/JPM2/LOGS/"STDOUT-$DATE-$TIME.TXT", '
      "FAMILYNAME=DISK, "
      'EXTMODE=ASCII, PROTECTION=PROTECTED, UNIQUETOKEN="$"); '
    "FILE STDERR(KIND=DISK, "
      'LFILENAME=*DIR/XMLJPM/JPM2/LOGS/"STDERR-$DATE-$TIME.TXT", '
      "FAMILYNAME=DISK, "
      'EXTMODE=ASCII, PROTECTION=PROTECTED, UNIQUETOKEN="$");';
}

PARSER 3 {
   HOST        winserver1.mycompany.com;   % windows server
   PORT        51117;
   STANDBY     true;
   INITIATEJVM false;
}
```

Note: *The PARSERHOST and PARSERPORT directives that were used previously are still supported for defining a single JPM.*

Java Parser Module (JPM

The XML file jpmconfig.xml in the directory

*DIR/XMLJPM/JPM<n>/CONFIG/= configures the JPM.

jpmconfig.xml File with Defaults

The following is the jpmconfig.xml file containing the default values for properties.

```
<?xml version="1.0"?>
<configuration>
  <port>
    <number>51117</number>
    <address>0.0.0.0</address>
  </port>
  <threads>
    <min>10</min>
    <max>100</max>
  </threads>
  <logging>
    <level>warn</level>
    <logfile>log.txt</logfile>
  </logging>
  <httpProxyHost></httpProxyHost>
  <httpProxyPort></httpProxyPort>
</configuration>
```

Properties in the jpmconfig.xml File

The properties in the jpmconfig.xml file are port number, port address, threads min, threads max, logging level, logging logfile, http proxy host, and http proxy port.

port number

This property is the number of the port that the JPM uses to communicate with WEBAPPSUPPORT. The default port number is 51117.

port address

This is the IP address on which the JPM listens to communicate with WEBAPPSUPPORT. If the JPM is on the MCP Java 6.0 or 7.0 Java Processor, Unisys recommends that this address be the EVLAN address of the Java server, using the “evlanjdp” mnemonic. For example:

<address>evlanjdp</address>

If the JPM is on a server that is independent of the MCP, this address is 0.0.0.0 or one of the local IP addresses on the server. The default port address is 0.0.0.0.

threads min

This property is the minimum number of JPM worker threads that can be active have at one time. The default for this property is 10; the minimum value is 1, and the maximum value is the value of the threads max property.

threads max

This property is the maximum number of JPM worker threads that can be active at one time. The default for this property is 100, the minimum value is the value of the threads min property, and the maximum value is the maximum number of the worker threads that the JMP can handle.

logging level

This property is the JVM logging level for an application that the MCP is not tracing. This level can be any of the following case-insensitive values:

- DEBUG
- INFO
- WARN
- ERROR
- FATAL
- OFF

logging logfile

This property is the name of the log file for logging JPM activity and errors. This property is one node. The log file is stored in the directory LOGS in the JPM directory.

http proxy host

This property is the host name or IP address of the HTTP proxy. The default is no value, which indicates that the JPM does not use an HTTP proxy.

http proxy port

This property is the port for the HTTP proxy. The default is no value, which indicates that the JPM does not use an HTTP proxy.

Multiple JPMs

The current configuration of specifying a single PARSERHOST and PARSERPORT is replaced by specifying one or more numbered “parsers”, each with their own set of attributes that define location, whether they are standby or active, and optional configuration for having WEBAPPSUPPORT initiate the JPMs.

JPM Initiation

The WEBAPPSUPPORT library allows configuration of multiple JPMs. The WEBAPPSUPPORT library initiates these JPMs if they run on MCP Java. Each JPM has its own directory for configuration and logging. Parsing, transformation and compression requests can either be load-balanced between multiple active JPMs, or if an attempt to reach a JPM fails the request is automatically attempted on one or more standby JPMs.

On Microsoft Windows, you manually run JPMs with the supplied Windows bat file.

The current recommended method of initiating JPMs on MCP Java with a Unisys supplied WFL is now enhanced with the ability for the WEBAPPSUPPORT library to initiate the JPMs. When WEBAPPSUPPORT initiates and processes its XML parser configuration, any JPMs configured as initiated by WEBAPPSUPPORT are started.

JPM Termination

When no callers are linked to the library, WEBAPPSUPPORT and the JPMs that WEBAPPSUPPORT initiated terminate. If WEBAPPSUPPORT terminates frequently it might be better to initiate the JPMs independently with the WFLs supplied by Unisys.

If JPMs terminate because MCP Java or a Java server is unavailable, the JPMs are restarted when MCP Java or a Java server is available. If JPMs terminate because Networking is unavailable, the JPMs are restarted when Networking is available.

If a JPM initiated by WEBAPPSUPPORT terminates for some other reason, WEBAPPSUPPORT starts a worker that creates a waiting entry, prompting the operator to restart the JPM. The JPM is not restarted until either a RESTARTXML command is entered or an operator directs the restart of the specific JPM.

On a RESTARTXML command, WEBAPPSUPPORT does the following:

- Terminates any JPMs initiated by WEBAPPSUPPORT. Completes requests that are in process by the JPMs first
- Reads the XML configuration file
- Checks the CCF install directory for new JPM files if any JPMs are to be initiated. Prompts the operator to upgrade if new JPM files are found
- Initiates JPMs

The WFL supplied by Unisys for initiating JPMs is still released and is changed to accept a JPM number and JDP target as additional parameters.

JPM Directory Structure

For the new Multiple JPM capability, the released directory structure for JPMs changes from:

```
XMLJPM
 +-- BIN
 +-- CONFIG
 +-- LOGS
```

To the following:

```
XMLJPM
 +-- BIN
 +-- JPM1
      +-- CONFIG
      +-- LOGS
```

To add a second JPM, make a copy of the JPM1 folder and name this copy JPM2.

```
XMLJPM
 +-- BIN
 +-- JPM1
 |    +-- CONFIG
 |    +-- LOGS
 +-- JPM2
      +-- CONFIG
      +-- LOGS
```

Initiate JPMs with a parameter, which is their JPM number (or directory name), so that they easily can find their directory.

Note: *JPM numbers must be sequential and start at 1.*

Request Handling

WEBAPPSUPPORT determines which JPMs handle requests. JPMs are either configured as active or standby. If WEBAPPSUPPORT initiates the JPM, then both active and standby JPMs are WEBAPPSUPPORT library initiations.

When WEBAPPSUPPORT receives a request, it round-robins the requests among the list of active JPMs. If an active server fails to respond to a request (cannot open socket to the JPM, error in sending the request, or error in reading the response), the next active JPM in the list is tried. If no active JPMs can handle the response, the standby JPMs are tried until none can handle the request; in that case, the request fails

If a JPM becomes unavailable, a WEBAPPSUPPORT worker creates a waiting entry. WEBAPPSUPPORT attempts to reach the JPM on the next request if the JPM has been unavailable for at least 30 seconds.

Operator Interface

The WEBAPPSUPPORT STATUS response is modified to show the status of each configured JPM. See “WEBAPPSUPPORT Commands” in Section 3.

Updating the XML Parser JPM

XML Parser software is updated from the installation of an Interim Correction of the CCF (Custom Connect Facility) product.

To determine the method to use to update the XML Parser JPM, consider

- How continuously you want parsing service
- How complex a configuration you want

You can make the parsing service more continuous, that is, reduce the number of interruptions, by

- Installing the JPM on multiple, redundant servers
- Configuring the JPM to be able to use any one of multiple ports at one time based on whichever port is available

However, making the parsing service more continuous requires a configuration that is more complex to install and manage.

Examples of ways you can update the JPM are presented in the following topics.

Updating the JPM When the JPM Runs on One Server and Always Uses the Same Port

To update the JPM, perform the following tasks:

1. If you are running the JPM on Microsoft Windows, copy the XMLJAVAPARSER folder from the MCP installs share, which is the directory for JPM, to your Windows system. Overwrite the current directory.

   If you are running the JPM on MCP Java, either the WEBAPPSUPPORT library or the JPM WFL installs the new JPM files to the running directory.
2. Read the file readme.txt for necessary changes to the JPM configuration.
3. If necessary, edit the file jpmconfig.xml to change to the JPM configuration.
4. Terminate the currently running JPM.

   Terminate the JPM manually, for example by using the <mix>AX QUIT command to terminate the codefile *DIR/JRE7/BIN/JAVA for JPMs running on MCP Java.

   Application requests to parse XML documents can fail while the JPM is terminated. See "Multiple JPMs" in this Section.
5. Initiate the JPM.

   If you are using the MCP Java Processor, the WFL *SYSTEM/CCF/XMLPARSER/WFL/JAVA prompts you to install the new JPM files. Answer the WFL Accept with **Y** to perform the install.

You do not need to change the WEBAPPSUPPORT configuration, and the JPM can use the same port and server.

Updating the JPM When the JPM Uses a Non-Default Port

The procedure that you perform depends on whether the JPM runs on the MCP Java Processor, or on a Windows or Linux system.

When the JPM Runs on the MCP Java Processor

To update the JPM, perform the following tasks:

1. Start the JPM.

   Start the WFL *SYSTEM/CCF/XMLPARSER/WFL/JAVA with the value of one of the following parameters different from the value for the currently running JPM:

   - JPMHOMEDIR WFL parameter, which specifies the directory for JPM
   - JPMFAMILY WFL parameter, which specifies the pack family for JPM

2. If you are updating the JPM for the first time and are prompted to create JPM subdirectories, type **Y** to confirm that you want to create the subdirectories.
3. After the JPM starts, stop the JPM by typing

   ```
   <mix number of *DIR/JRE/BIN/JAVA>DS
   ```

4. Copy the file jpmconfig.xml from the currently active directory for JPM to the new directory for JPM.
5. Edit the new file jpmconfig.xml:

   a. Make the JPM use a different port.

      For example, if the currently running JPM is using port 51117, you can make the new JPM use port 51118.

   b. Read the readme.txt file for any other necessary changes to the JPM configuration.

   c. If necessary, edit the file jpmconfig.xml.

6. Start the new JPM by using the WFL that you used in step 1.

   Two JPMs are now running on the server.

7. On the MCP, edit the file *SYSTEM/CCF/WEBAPPSUPPORT/PARAMS/XML to make the JPM use a different port.

   For example, the edited file with the old 51117 port commented out might look like the following:

```
NEXT+ ....*....1....*....2....*....3....*....4....*....5....*....6
00000100% Configuration To Java Parser Module on Windows
00000200 PARSER 1 {
00000250    HOST winserver1;
00000300  % PORT 51117;
00000400    PORT 51118;
```

8. Type the command **NA CCF WEBPCM WEBAPPSUPPORT RESTARTXML** from MARC or the system ODT.

   This command makes subsequent parsing requests go to the new JPM.

9. Type the command **NA CCF WEBPCM WEBAPPSUPPORT STATUS** to check the status of the new JPM.
10. Type the command **NW TCPIP CONN YOURNAME = <old port number>** to determine when no connections are open to the old JPM.
11. When no connections are open to the old JPM, terminate the old JPM.

    Terminate the JPM manually, for example by using the DS command to terminate the codefile DIR/JRE/BINJAVA.

When you use this procedure, a JPM is always available. Parsing requests from applications do not fail because the JPM is unavailable.

When the JPM Runs on a Windows or Linux System

To update the JPM, perform the following tasks:

1. Copy the contents of the directory *SYSTEM/INSTALLS/XMLJAVAPARSER/=, which is the directory for the JPM, to a new directory on your Windows or Linux system.
2. Copy the file jpmconfig.xml from the currently active directory for JPM to the new directory for JPM.
3. Edit the new file jpmconfig.xml:
   a. Make the JPM use a different port.

      For example, if the currently running JPM is using port 51117, you can make the new JPM use port 51118.

   b. Read the readme.txt file for any other necessary changes to the JPM configuration.

   c. If necessary, edit the file jpmconfig.xml.
4. Start the new JPM.

   Two JPMs are now running on the server.
5. On the MCP, edit the file *SYSTEM/CCF/WEBAPPSUPPORT/PARAMS/XML to make the JPM use a different port.

   For example, the edited file with the old 51117 port commented out might look like the following:

```
NEXT+ ....*....1....*....2....*....3....*....4....*....5....*....6
00000100% Configuration To Java Parser Module on Windows
00000200 PARSER 1 {
00000250    HOST winserver1;
00000300  % PORT 51117;
00000400    PORT 51118;
```

6. Type the command **NA CCF WEBPCM WEBAPPSUPPORT RESTARTXML** from MARC or the system ODT.

   This command makes subsequent parsing requests go to the new JPM.
7. Type the command **NA CCF WEBPCM WEBAPPSUPPORT STATUS** to check the status of the new JPM.

8. Type the command **NW TCPIP CONN YOURNAME = <old port number>** to determine when no connections are open to the old JPM.
9. When no connections are open to the old JPM, terminate the old JPM.

When you use this procedure, a JPM is always available. Parsing requests from applications do not fail because the JPM is unavailable.

Updating the JPM When the JPM Runs on Two Servers

To update the JPM, perform the procedure under "Updating the JPM When the JPM Runs on One Server and Can Use Any One of Multiple Ports" for each server.

Both servers have the same port number for the PARSERPORT property but different domain names or IP addresses in the PARSERHOST property.

When you use this procedure, a JPM is always available. Parsing requests from applications do not fail because the JPM is unavailable.

Preparing to Use the XML Parser

Securing the XML Parser

You need to secure the following for the XML Parser:

- XML Parser configuration file
- XML Parser trace files
- Communication Between the WEBAPPSUPPORT Library and the JPM
- JPM port
- JPM log files
- JPM configuration file
- XML documents on HTTP servers

XML Parser Configuration File

The WEBAPPSUPPORT XML Parser configuration file is *SYSTEM/CCF/WEBAPPSUPPORT/PARAMS/XML and is located on the same family where WEBAPPSUPPORT is located. This file is not usercoded. Set the SECURITYTYPE attribute of this file to PRIVATE to prevent nonprivileged users from viewing or changing the file. You can use a guard file to further protect this file.

XML Parser Trace Files

The trace files that the WEBAPPSUPPORT library creates are not usercoded. WEBAPPSUPPORT sets the SECURITYTYPE attribute of this file to PRIVATE to prevent non-privileged users from viewing or changing these files.

Communication between the WEBAPPSUPPORT Library and the JPM

The WEBAPPSUPPORT library and the JPM communicate with each other over EVLAN if

- The JPM runs on an MCP Java level 5.0 or higher Java Processor.
- The WEBAPPSUPPORT XML Parser configuration file uses the EVLAN IP address.

EVLAN traffic cannot be traced. Unisys recommends that the JPM use EVLAN for better security and performance.

If the JPM runs on an MCP system that does not support EVLAN, the TCPIP Rules file can limit access to the JPM port. Also, if the JPM listens on the local host IP address (127.0.0.1) then the port of the JPM will not be accessible outside of the MCP.

If the JPM runs on a system other than the MCP, protect the TCP connection between the MCP and the JPM as much as possible. The XML information sent over this connection is not encrypted.

JPM Port

If the JPM is running on a server with multiple network interfaces, configure the JPM port address to a specific address, not to the default IPv4 address 0.0.0.0. Configuring this address can limit unauthorized TCP access to the JPM.

JPM Log Files

When the JPM runs on an MCP Java Processor, the JPM creates log files and stores the log files in the directory *DIR/XMLJPM/JPM<n> LOGS/= on the MCP. After the JPM is installed, change the security attributes of the LOGS directory to limit access to these logs. For example, in CANDE type the following:

```
WFL ALTER *DIR/XMLJPM/JPM<n>LOGS (GROUP=ADMIN)
```

Note: *Restricting access to MCP directories that the JPM accesses might require running the JPM under a usercode that can access the directories. Running the JPM under such a usercode might require updating the WFL supplied by Unisys that runs the JPM.*

JPM Configuration File

The Java Parser Module configuration file is *DIR/ XMLJPM/JPM<n>/CONFIG/"JPMCONFIG.XML". If the JPM runs on an MCP Java Processor, protect the CONFIG directory the same way that you protect the JPM log files. See the preceding topic "JPM Log Files."

Securing XML documents on HTTP servers

If an XML document to be processed is on an HTTP server, the JPM must be able to access the documents anonymously. You need to secure the documents because anonymous access can make the documents available to unauthorized users. For example, you can configure the MCP Web Transaction Server to allow HTTP access to the XML documents only from the JPM server IP address.

Improving XML Parser Performance

To improve XML Parser performance, perform the following tasks:

- Allocate enough memory to the JPM Java Virtual Machine (JVM)
- Set the maximum number of JPM threads high enough
- Ensure that the MCP system uses EVLAN to communicate with the JProcessor running the JPM
- Ensure that external files are in locations that the JPM can access quickly
- If HTTP servers serve XML documents or external files, ensure that JPM communication with the HTTP servers is efficient
- Disable processing of external general entity references when an application does not use external entities

Allocating Enough Memory to the JVM

Insufficient memory for the JVM can reduce JPM performance by causing frequent garbage collection and delays in JPM processing.

When the JPM is active and reachable, use the WEBAPPSUPPORT STATUS command to check JVM memory usage statistics. The following is an example of a response to the STATUS command:

```
XML Parser:
  Host 192.168.16.2, Port 51117
   1 Sockets Open
  Version:  12.0.0.12
  Threads:  Current = 10, Min = 10, Max = 20
  Logging:  Level = Debug, File = logs/log.out
  Documents Parsed = 0
  JVM:
    Version:  1.5.0_12
    Free = 11 MB, Total = 15 MB, Max = 63 MB
```

If the amount of JVM free memory is consistently low, the JVM might need more memory.

Setting the Maximum Number of JPM Threads

Set the maximum number of JPM threads to the maximum expected number of application stacks that parse requests. If the number of application stacks parsing requests is greater than the maximum number of JPM threads, connections to the JPM close and re-open more frequently. This closing and re-opening of connections increases MCP processing and lengthens response times.

Configuring EVLAN Communication between the MCP and the JProcessor

Maximize XML Parser performance by ensuring that the MCP system uses the EVLAN path to communicate with the JProcessor running the JPM.

To configure EVLAN communication between the MCP and the JProcessor, do the following:

1. Use the NA JAVA SERVER <n> command to obtain the IP address of the JProcessor.

   For example, on the MCP Operator Display Terminal type

   ```
   NA JAVA SERVER 1
   ```

   The MCP could return

   ```
   Java server: 1

   IP address: 192.168.16.2
   ```

2. Configure the WEBAPPSUPPORT library to use the IP address of the JProcessor.

   In the file *SYSTEM/CCF/WEBAPPSUPPORT/PARAMS/XML, set the HOST property to the IP address of the JProcessor. If the HOST property is localhost, the MCP does not use the EVLAN path.

3. Set the JPM address parameter to configure the JPM to listen on the IP address of the JProcessor.
4. The TARGET property should be set to a nonzero value if WEBAPPSUPPORT is to initiate the JPM, so that the JPM runs on the JDP that matches the IP address.

Locating External DTD and Schema Files for Fast Access

Some files, such as DTD or schema files, are necessary for parsing but are outside XML documents. The JPM might need to open and read external files for any parsing request. Ensure that these files are in locations that the JPM can access quickly.

If the JPM runs on a Windows or Linux server, the JPM might be able to read files from the local server file system or a local HTTP server. Reading files from a local server reduces the number of requests over the network. See “Identifying Files on an HTTP Server” and “Identifying Files on a JPM Server File System” in Section 5 for more information about accessing local files.

Ensuring Efficient Communication between the JPM and HTTP Servers

If HTTP servers serve XML documents or external DTD or schema files, ensure that communication between the JPM and the HTTP servers is efficient.

JPM communication with a MCP Web Transaction Server (WebTS) HTTP server is very efficient. A WebTS can efficiently cache files in memory and does not re-open the cached files.

Communication between an HTTP server and a JPM on a MCP Java Processor that is on a system with EVLAN is very efficient. The JPM can use a URL that uses the EVLAN path. For example, the JPM could use http://evlanmcp/xmlfiles/xmlinvoice.xml.

Disabling Processing of External General Entity References

If an application that is parsing an XML document does not need any external general entities in the document, set the EXTERNAL_GENERAL_ENTITIES option of the SET_XML_OPTION procedure to 0 (zero). Disabling processing of external general entities can improve performance.

Section 5
Developing an XML Parser Application

Using the XML Parser API

An application calls WEBAPPSUPPORT library procedures to use the XML Parser. For descriptions of the API of these procedures, see Section 6.

Examples of Using the API

An application can use the XML Parser to perform any of the following tasks:

- Read specific data in an XML document
- Read data sequentially in an XML document
- Create an XML document
- Modify an XML document
- Release an XML document
- Encrypt an element
- Encrypt data into an XML document
- Encrypt data into a file and generate a cipher reference
- Decrypt an XML element
- Decrypt an XML document containing a cipher reference
- Generate a simple data set as JSON text from an MCP application
- Generate a structured data set as JSON text from an XML source

The following topics are examples of the steps that an application can take to perform the preceding tasks.

Reading Specific Data in an XML Document

To read specific data in an XML document, the application can perform the following steps:

1. Call the SET_XML_OPTION procedure to set options to control the processing of the document

2. Call the PARSE_XML_DOCUMENT procedure to parse the document

   The application receives a document tag that references the parsed document, which is stored in the WEBAPPSUPPORT memory, and contains a reference to the document node.
3. Call the GET_ELEMENTS_BY_TAGNAME procedure, repeatedly if necessary, to request a list of elements under a specific node
4. Call the GET_NODE_NAME procedure to request a specific element name
5. Use one of the following procedures to get the data:
   - Call the GET_NODE_VALUE procedure to get the value of a node
   - Call the GET_ATTRIBUTES procedure to get the list of attribute values for an element
   - Call the GET_ATTRIBUTE_BY_NAME procedure to get the value of a specific attribute for an element

Reading Data in an XML Document Sequentially

To read data in an XML document sequentially, the application can perform the following steps:

1. Call the SET_XML_OPTION procedure to set options to control the processing of the document
2. Call the PARSE_XML_DOCUMENT procedure to parse the document

   The application receives a document tag that references the parsed document, which is stored in the WEBAPPSUPPORT memory, and contains a reference to the document node.
3. Call the GET_NEXT_ITEM procedure to request the first item in the document
4. Complete any or all of the following, if the application needs to read the item:
   - Call the GET_NODE_VALUE procedure to get the value of the node
   - Call the GET_ATTRIBUTES procedure to get the list of attribute values for an element
   - Call the GET_ATTRIBUTE_BY_NAME procedure to get the value of a specific attribute for an element
5. Call the GET_NEXT_ITEM procedure and the procedures in step 4 repeatedly to read the other items in the document

   The application receives the result 0 (zero) for the last GET_NEXT_ITEM procedure. That result indicates that all items are read.

Creating an XML Document

To create an XML document, the application can perform the following steps:

1. Call the CREATE_XML_DOCUMENT procedure, specifying the XML document and character set to use for the document

2. Optionally, call the CREATE_DOCTYPE_NODE procedure to create a DTD and calls the APPEND_CHILD procedure to attach the DTD to the document node
3. Call the CREATE_ELEMENT_NODE procedure to create the high-level element, which is called the document element
4. Call the SET_ATTRIBUTE procedure to add an attribute to the document element, if necessary
5. Call the APPEND_CHILD procedure to attach the element to the document
6. Call procedures to create more nodes and attach these nodes to elements

   For example, the application can call any or all of the following to create a node:

   - CREATE_ELEMENT_NODE for an element
   - CREATE_ATTRIBUTE_NODE for an attribute
   - CREATE_TEXT_NODE for a text node
   - CREATE_COMMENT_NODE for a comment
7. Call the INSERT_CHILD_BEFORE procedure to insert a node or the APPEND_CHILD procedure to append a node
8. Call the GET_XML_DOCUMENT procedure to request the current XML document

   The application receives the XML document in the application array or an MCP file.

Modifying a Node Value

The application can do the following steps to modify a node value in an XML document:

1. Call the PARSE_XML_DOCUMENT procedure to parse the document, if the application did not just create the document
2. Call a procedure such as GET_ELEMENTS_BY_TAGNAME or GET_NEXT_ITEM to get the node to be modified
3. Call the SET_NODE_VALUE procedure to change the node value
4. Call the GET_XML_DOCUMENT procedure to request the updated XML document

   The application receives the XML document in the application array or an MCP file.

Setting or Deleting an Attribute Value

The application can do the following steps to set or delete an attribute value in an XML document:

1. Call the PARSE_XML_DOCUMENT procedure to parse the document, if the application did not just create the document
2. Set or delete the attribute value

   For the steps in setting the value, see “Setting an Attribute Value” following this procedure.

The application can do either of the following to delete the value:

- If an attribute is in an element node, the application can call the SET_ATTRIBUTE procedure and set the value of the attribute to empty.
- If the attribute is in an attribute node, the application can call the REMOVE_NODE procedure to remove the node.

3. Call the GET_XML_DOCUMENT procedure to request the updated XML document

   The application receives the XML document in the application array or an MCP file.

Setting an Attribute Value

The application can do either of the following to set an attribute value:

- If the attribute will contain one text node, the application can call the SET_ATTRIBUTE procedure.
- If the attribute will contain multiple text and reference nodes, the application can do the following.
  1. Create text nodes, entity reference nodes, or both, to contain the value

     The application can use the CREATE_TEXT_NODE procedure, the CREATE_ENTITYREF_NODE procedure, or both.
  2. Attach the text and entity reference nodes to the attribute

     The application can use the APPEND_CHILD procedure or the INSERT_CHILD_BEFORE procedure.

Deleting a Node and the Children of the Node

The application can do the following to delete a node and the children of the node in an XML document:

1. Call a procedure such as GET_ELEMENTS_BY_TAGNAME or GET_NEXT_ITEM to get the element to be deleted
2. Call the REMOVE_NODE procedure to delete the node and its children.

Releasing an XML Document

After the application finishes working with an XML document, the application needs to release the document to free WEBAPPSUPPORT resources.

The application can do any of the following to release a document:

- Call the RELEASE_XML_DOCUMENT procedure, specifying the document tag
- Call the PARSE_XML_DOCUMENT procedure to parse another document or the CREATE_XML_DOCUMENT procedure to create another document, specifying the tag for the current document
- Delink from WEBAPPSUPPORT or call the CLEANUP procedure in WEBAPPSUPPORT

  Delinking releases all XML documents created or parsed by the application.

Encrypting an Element

To encrypt an element in a parsed XML document and then get the XML document to send, an application can perform the following steps.

1. Create a key object with the CREATE_KEY procedure, if one is not already created.
2. Locate the element node to be encrypted in a parsed XML document—perhaps using an XPath expression such as GET_NODE_BY_XPATH.
3. Encrypt the element and its child nodes using the ENCRYPT_XML_DOCUMENT procedure, passing the element node to be encrypted.
4. Call GET_XML_DOCUMENT with the new XML document tag; receive back the XML document in external form.

Note: *After completing the above steps, two XML documents are stored in WEBAPPSUPPORT. The application could make other modifications to the new XML document, such as adding attributes to encrypted elements. The original XML document element could be modified and encrypted again.*

Encrypting Data into an XML Document

To take data that is stored in the application array or in an MCP file and then encrypt that data into an XML document, an application can perform the following steps.

1. Create a key object with the CREATE_KEY procedure, if one is not already created.
2. Locate the element node in a parsed XML document that is to be the parent of the encrypted data—perhaps using an XPath expression such as GET_NODE_BY_XPATH.
3. Call the ENCRYPT_DATA_TO_XML procedure to encrypt the data and insert it into the XML document.

Encrypting Data into a File and Generating a Cipher Reference

To encrypt data that is stored in an application array or in an MCP file into a new MCP file that can be served by MCP Web Transaction Server and then create a cipher reference into an XML document that references the encrypted data, an application can perform the following steps.

1. Create a key object with the CREATE_KEY procedure, if one is not already created.
2. Call the ENCRYPT_DATA procedure to created the encrypted data file.
3. Call the CREATE_CIPHER_REFERENCE procedure to insert a cipher reference into the XML document.

Decrypting an XML Element

To decrypt an encrypted element in a parsed XML document and access the data, an application can perform the following steps. The encrypted data is XML.

1. Create a key object with the CREATE_KEY procedure, if one is not already created.
2. Find the *EncryptedData* element with the GET_NODE_BY_XPATH procedure using an XPath expression.
3. Call the DECRYPT_XML_DOCUMENT procedure to get a new XML document containing the decrypted element.

Decrypting an XML Document Containing a Cipher Reference

To decrypt data that is not stored in an XML document but is instead referenced with a URL in a *CipherReference* element contained within the *EncryptedData* element in the XML document, an application can perform the following steps.

1. Find the *CipherReference* element in the XML document—perhaps using an XPath expression such as GET_NODE_BY_XPATH.
2. Use the WEBAPPSUPPORT HTTP Client feature to access the data at the URL identified by the *URI* attribute in the *CipherReference* element.
3. Check the *CipherReference* element for any contained *Transform* elements that describe transformations required on the data—such as, base64 decoding or an XPath expression to be applied to the retrieved data. Xpath can be used to look for the presence of these elements.
4. Call the DECRYPT_DATA procedure to decrypt the data.

Generating a Simple Data Set as JSON Text from an MCP Application

To generate JSON text from data generated by an application where the data is simple name-value pairs, the application can perform these steps:

1. Perform one of the following actions:
   - Build the data into an array; for example, where <LF> represents the line feed character:

     "a, b, c <LF> 1, 2, 3"
   - Write each row to an MCP record file:

     1 a, b, c
     2 1, 2, 3
2. Call the CONVERT_COMMA_TEXT_TO_JSON procedure to generate the JSON text, which can be returned in an application array or written to a new MCP file.

Generating a Structured Data Set as JSON Text from an XML Source

To generate JSON text from data generated by an application where the data is a structured data set, the application can perform these steps:

1. Store the data in XML format either in an XML file or by creating or parsing an XML document in WEBAPPSUPPORT using either the CREATE_XML_DOCUMENT or PARSE_XML_DOCUMENT procedure.
2. Perform one of the following actions:
   - If the XML is stored in a file or an array of the application, call the CONVERT_XML_DOCUMENT_TO_JSON procedure
   - If the XML document is stored in WEBAPPSUPPORT memory, call the CONVERT_XML_TO_JSON procedure.

   These procedures generate JSON text stored either in an application array or a new MCP file.

Using HTTP Servers

You can use HTTP servers to store the following:

- XML documents
- External DTDs
- XML schema documents
- XSL stylesheets

Note: The XML Parser *must be able to access anonymously a resource that the application specifies as an HTTP URL. An application cannot supply credentials via the XML Parser to access restricted resources on an HTTP server. XML documents on HTTP servers that require credentials can be read by the application using the HTTP Client feature and then passing the XML document to the XML Parser to be parsed or transformed.*

For information on how an application identifies files on an HTTP server, see "Identifying Files on an HTTP Server."

Required File Mappings for the MCP Web Transaction Server

You can use the MCP Web Transaction Server to serve XML documents, stylesheets, XML schema, or DTD files. If you do this, ensure that the following file type mappings are in the file config.cfg for the server, which is usually ATLASSUPPORT.

File Extension	Multipurpose Internet Mail Extensions (Mime) Type
dtd	application/xml-dtd
xml	application/xml
xsl	application/xml

Validating an XML Document by Using a Schema or DTD

The XML Parser can validate an XML document against an XML schema or DTD if the XML Parser can identify the schema or DTD file. See “Identifying Files.”

Specifying a Schema

The XML Parser can use an XML schema to validate an XML document and to define entities. The XML document can use only one schema file.

You can specify an XML schema location in either of the following ways:

- In a schema location statement in an XML document

  The statement can be xsi:schemaLocation or xsi:noNamespaceSchemaLocation.
- Using the SCHEMA_LOCATION option in the SET_XML_OPTION procedure that an application calls

How the schema location is specified depends on how the XML document is accessible to the PARSE_XML_DOCUMENT procedure

- If the XML document and schema file are in the same directory, a relative URL or an absolute URL can specify the schema location.

  For example, if the XML document is http://webserver/xml/statusrequest.xml and contains the schema location statusrequest.xsd in the directory /xml, a relative URL or an absolute URL can specify the schema location.
- If the XML document is in the application array or an MCP file, an absolute HTTP URL must specify the schema location.

Specifying Character Sets

The XML document specifies the character set of the document in an encoding statement, which is also the character set in which the XML Parser `generates a new` document. An XML document that does not specify an encoding uses the UTF-8 encoding. For example, an XML document encoded in Latin 1 characters might start with:

```
<?xml version= "1.0" encoding="iso-8859-1"?>
```

The application specifies the character set of the application, which is the character set in which the application supplies and receives text.

WEBAPPSUPPORT translates the following:

- Application text into the document character set when WEBAPPSUPPORT generates a document
- A document from the document character set into the application character set to enable an application to read a document

Specifying the Application Character Set

The application calls the SET_TRANSLATION procedure to specify the character set in which the application supplies and receives text.

Application Character Sets that the XML Parser Supports

The XML Parser supports the application character sets in the following table. The right column has the coded character set (CCS) number for each character set.

Character Set	CCS Number
ARABIC20EBCDIC	34
ASCII	5
ASERIESEBCDIC/EBCDIC	4
ASKSC	902
ASUTL	82
CANSUPPLEBCDIC	16
EBCDICGB2312	111
EBCDICKSC5601	105
EBCDICUTL	108
EBCDICKSC5601	105
EBCDICUTL	108
GB2312	935
IBMSWEDENEBCDIC	51

JAPANEBCDICJBIS8	100
Character Set	**CCS Number**
JBIS8	80
LATIN1EBCDIC	12
LATIN5EBCDIC	14
LATIN9EBCDIC	47
LATINCYRILLICEBC	29
LATINGREEKEBCDIC	19
LETSJ	104
LETSJISX16	930
LOCALEBCDIC	50
UTF-8	2

All the application character sets except UTF-8 are identified in the MCP Multi-Lingual System (MLS).

UTF-8 is an encoding of the Unicode (UCS2) character set and is the default character set encoding for XML documents that the XML Parser generates.

Specifying the Document Character Set

If the application calls the CREATE_XML_DOCUMENT procedure, the XML_DECLARATION parameter in the procedure might contain an encoding string. The XML Parser uses the encoding string to generate the new XML document.

Encoding Strings that Specify the Character Set

The XML Parser supports the following encoding strings. Each encoding string is associated with a CSS number. The XML Parser must be able to translate from the character set of the application to the CCS number character set to generate a new document.

Encoding String	CCS Number
ascii	5- ASCII
big5	115 – WINBIG5
cp297	39 – IBM297
cp437	36 – CODEPAGE437
cp850	18 – CODEPAGE850
cp851	21 – CODEPAGE851
cp852	28 – CODEPAGE852

cp857	44 – CODEPAGE857
Encoding String	**CCS Number**
cp866	31 – CODEPAGE866
euc-jp	103 – EUCJp
iso-8859-1	13 – Latin1ISO
iso-8859-2	27 – Latin2ISO
iso-8859-5	30 – LatinCyrllicISO
iso-8859-7	20 – LatinGreekISO
iso-8859-9	15 – Latin5ISO
iso-8859-15	48 – Latin9ISO
shift_jis	102 – CODEPAGE932
us-ascii	5 – ASCII
utf-8	(2 – UTF-8)

windows-1250	33 – CODEPAGE1250
windows-1251	32 – CODEPAGE1251
windows-1252	37 – CODEPAGE1252
windows-1253	45 – CODEPAGE1253
windows-1254	43 – CODEPAGE1254

These encoding strings are case-insensitive. The Internet Assigned Numbers Authority (IANA) documents these encoding strings at http://www.iana.org/assignments/character-sets.

UTF-8 is an encoding of the Unicode (UCS2) character set and is the default character set for XML documents that the XML Parser generates.

The XML Parser generates an error if WEBAPPSUPPORT cannot translate directly from the application character set into either of the following:

- The character set for generated XML documents
- A character set that is necessary to generate text into the character set for generated XML documents

  For example, WEBAPPSUPPORT cannot translate from the JBIS8 application character set directly into the UCS2 character set, which is needed to generate text in the UTF-8 document character set.

Examples

Each row of the following table shows an example of the following:

- The application character set
- The document encoding string that the application specifies
- The character set for generated XML documents

Application Character Set	Document Encoding String	Character Set for Generated Documents
ASERIESEBCDIC	utf-8	UTF-8
LATIN1EBCDIC	iso-8859-1	LATIN1ISO

Using Entity References

An XML document that the XML Parser creates or parses can have entity references. An entity reference is a place-holder for text, a document fragment, or other data. The XML Parser can replace an entity reference with data before giving the document to the application.

Using General Entity References

If an entity is parsed, the Document Object Model (DOM) tree contains an entity reference node with the parsed entity as a child of the entity reference node. The nodes in the entity sub-tree cannot be modified or deleted.

Entity References for Simple Strings

If a parsed entity points to a simple string, a text node is the only child of the entity reference node. The following is an example of the entity definition for a simple string:

```
<!ENTITY magazinetitle "Life">
```

In the following XML document code, the entity reference &magazinetitle refers to the preceding entity definition:

```
<TITLE>&magazinetitle; Magazine</TITLE>
```

The preceding code would create the following subtree under the entity reference node:

```
[element:  TITLE]

  +---> [entity reference: magazinetitle]

  |       +---> [text:  Life]

  +---> [text:  Magazine]
```

Entity References for Document Fragments

If the parsed entity points to a document fragment, a multinode subtree under the entity reference node represents the document fragment. For example, the entity definition for the document fragment could be the following:

```
<!ENTITY disclaimer SYSTEM "disclaimer.xml">
```

For this example, the following contents of the document fragment are in the file disclaimer.xml:

```
<DISCLAIMER>

  <STRONG>No warranty implied.</STRONG>

</DISCLAIMER>
```

The preceding code would create the following subtree under the entity reference node:

```
[entity reference:  disclaimer]

  +---> [element:  DISCLAIMER]

          +---> [element:  STRONG]

                  +---> [text:  No warranty implied.]
```

Unparsed Entities

The XML Parser does not support unparsed entities.

Controlling General Entity Processing

The application can control whether the XML Parser replaces general node references with entity values for documents that the GET_XML_DOCUMENT procedure generates. The application calls the SET_XML_OPTION procedure and sets the EXPAND_ENTITY_REFERENCE option.

Using Attribute Node Entity References

The XML Parser always replaces attribute node entity references with data. If the document was parsed, the application always receives the attribute nodes with values, not entity references.

Using Predefined and Character Entity References

The XML Parser always gives the application data that replaces pre-defined entity references and character entity references. For example, the XML Parser always replaces the predefined entity reference **<** with **<**, and replaces the character entity reference **&#25;** with **%**.

When an application creates or modifies text in an XML document, the application must supply the text in the encoded form that is required in the document. For example, the application must supply the character **<** in the encoded form **<**. Before the application calls the procedure to create or modify the text, the application can call the XML_ESCAPE procedure to convert most predefined entity references to the required encoded form.

Using Namespaces

The XML Parser can create and parse XML documents that use namespaces. The XML Parser supports the *Namespaces in XML 1.0* standard.

An element or an attribute can be associated with a namespace.

- The CREATE_ATTRIBUTE_NODE, CREATE_ELEMENT_NODE and SET_ATTRIBUTE procedures support specifying a namespace.
- The GET_ELEMENTS_BY_TAGNAME procedure can be limited to returning only elements that are in the specified namespace.
- The HAS_ATTRIBUTE procedure can be limited to returning only true (successful) if the attribute name including a namespace is present.
- The GET_NEXT_ITEM and GET_NODE_NAME procedures return names in the format that the NAMESPACE_PROCESSING option in the SET_XML_OPTION procedure specifies.

  The NAMESPACE_PROCESSING option indicates whether the XML Parser returns element and attribute names with namespace prefixes.

A namespace URL can be any non-null text, but is usually an HTTP URL. The XML Parser does not validate a namespace URL or access a namespace URL.

Identifying Files

An application must identify the following types of files to the XML Parser:

- XML files to be parsed
- External schema files that are not in an XML document
- XML files to be created on the MCP
- XSL stylesheets that are not specified in the XML document

These files can be located on

- The MCP file system
- An HTTP server, such as an MCP Web Transaction Server or a Microsoft Windows Internet Information Server (IIS)
- The file system of the server on which the XML Parser JPM runs.

XML files do not have to be, but often are, in the same place as the DTD or schema files used to validate the XML files.

Identifying Files on an MCP File System

An application identifies an XML file on the MCP file system by specifying the file name in display format or pathname (POSIX) format. The FILENAME_FORMAT option value in the SET_XML_OPTION procedure implies a SEARCHRULE file attribute value that the XML Parser uses to find the file.

In each of the following MCP file names, the FILENAME_FORMAT value for the name is in parentheses at the end:

```
(MYUSERCODE)"MYXMLFILE.XML" ON MYPACK  (LTITLE)

/-/MYPACK/USERCODE/MYUSERCODE/MYXMLFILE.XML  (PATHNAME)

MYXMLFILE.XML  (LTITLE or PATHNAME)
```

The last file name, MYXMLFILE.XML, does not specify a usercode or family. The following points apply for this file name:

If the FILENAME_FORMAT option is LTITLE, the file is under the application usercode.

If the FILENAME_FORMAT option is PATHNAME, the file is in the current directory of the application.

This file is on the primary or secondary family for the application.

The application stack must be running under a usercode that can access the file that the XML Parser parses or creates.

External DTD and Schema Files

An external DTD file must be identified in absolute pathname format in the XML document.

For example, the XML file could contain

```
xsi:noNamespaceSchemaLocation="/-
/mypack/usercode/myusercode/myxmlfile.xsd">
```

An external schema file must be identified in absolute pathname format in either of the following:

- The XML document
- The SCHEMA_LOCATION option of the SET_XML_OPTION procedure

  For example, the SCHEMA_LOCATION option could be set to

```
/-/MYPACK/USERCODE/MYUSERCODE/MYXMLFILE.XSD
```

The JPM must be running under a usercode that can access the DTD or schema file.

Identifying Files on an HTTP Server

An application specifies an HTTP URL to identify a file on an HTTP server, whether the file is an XML file, an external DTD file, or an external schema file. For example, an application could specify the following URL to identify a file:

http://myserver/xmlfiles/myxmlfile.xml

An application must specify an absolute URL, such as the preceding example, to identify an XML file to be parsed. To identify an external DTD or schema file, the application can specify

- An absolute URL
- A relative URL, if the external file and the XML file are in the same directory on the HTTP server

  For example, an application could specify the following relative URL:

```
<!DOCTYPE Transaction SYSTEM "myxmlfile.dtd">
```

Identifying Files on a JPM Server File System

An application might need to identify an XML file, external DTD file, or external schema file on the same server that the JPM runs on. To do this, the application must specify the file name in the format that the local file system requires.

For example, if the JPM is running on Microsoft Windows, an application could specify the following file name:

```
c:\xmlfiles\myxmlfile.xml
```

Locking an XML Document

An application can use the LOCK_DOCUMENT option in the SET_XML_OPTION procedure to lock each access to an XML document that the application creates or parses.

The application needs to lock a document only if another application might access the document while the first application changes the document. The document stays locked when another application calls a procedure to access the document. When the application that locked the document exits the procedure call that accessed the document, the XML Parser releases the document lock.

If the application requires a more global lock, for example to lock out a sequence of procedure calls to WEBAPPSUPPORT, the application must implement the lock.

Using Sample Source Code

Unisys provides sample COBOL85 code and sample ALGOL code for using an XML document. You can use these samples to write applications. See Section 7 for this source code.

The sample fragments of code in Section 7 show basic calls to the XML Parser API procedures. For more complete working examples released with the XML Parser, see the files in the directory *SYSTEM/CCF/XMLPARSER/SAMPLES/=.

Using WEBAPPSUPPORT Library Trace Files

You can use WEBAPPSUPPORT library trace files when you develop an application. These trace files can record the result of every procedure call that a specific application makes. For details about using these trace files, see the *Custom Connect Facility Administration and Programming Guide* and "Using the WEBAPPSUPPORT Trace File" in Section 3.

Section 6
WEBAPPSUPPORT Library Interface for the XML Parser

XML Mapping Structure

An XML Mapping Structure is a set of data passed in a single parameter that an application uses to direct WEBAPPSUPPORT procedures on how to map application data to or from an XML document. The application data is stored in a format such as a COBOL 01 record structure, with adjacent, fixed-sized fields and some repeated sub-structures.

All mapping structures have the following format:

```
<level><mapping>
```

where <level> is a binary value specifying the mapping format level. The EAE type is an N5. For this release, the value is **1**.

Level 1 Formatting

The <mapping> value for level 1 is formatted as:

```
<num items><items>
```

where:

- <num items> is a binary value specifying the number of items that follow. The EAE type is an N5.
- <items> is a list of items of length <num items>. Each item is defined as:

  ```
  <mapping type><field info>
  ```

  where:

  – <mapping type> is a binary value specifying the mapping type. The EAE type is an N5.

  – <field info> is specific to each mapping type.

The following table describes the supported values for <mapping type> and <field info>.

<mapping type> Value	Description of Value	<field info> Structure
1	Alphanumeric Text Alphanumeric Text maps to a COBOL PIC X() field. Text is encoded the application's character set.	`<element name><text size>` where: <element name> is the name of the element that encloses the text. It is 252 bytes in length. The element must exist within the NODE parameter, and if <element name> is null then the NODE parameter is used to enclose the text. <text size> is a binary value specifying the maximum length of the text. The EAE type is an N5.
2	Integer to BINARY Integer values map to a COBOL PIC S9(11) BINARY field.	`<element name>` where <element name> is the name of the element that encloses the integer. It is 252 bytes in length. The element must exist within the NODE parameter, and if <element name> is null then the NODE parameter is used to enclose the text.
3	Integer to EAE S12 Integer values map to an EAE S12 field.	`<element name>` where <element name> is the name of the element that encloses the integer. It is 252 bytes in length. The element must exist within the NODE parameter, and if <element name> is null then the NODE parameter is used to enclose the text.
4	Integer to COMPUTATIONAL Integer values map to a COBOL PIC S9(11) COMP field, which is stored as packed decimal.	`<element name>` where <element name> is the name of the element that encloses the integer. It is 252 bytes in length. The element must exist within the NODE parameter, and if <element name> is null then the NODE parameter is used to enclose the text.

<mapping type> Value	Description of Value	<field info> Structure
5	Floating Point to REAL Floating Point values map to a COBOL REAL field.	`<element name><decimal character>` where: <element name> is the name of the element that encloses the floating point. It is 252 bytes in length. The element must exist within the NODE parameter, and if <element name> is null then the NODE parameter is used to enclose the text. <decimal character> is a single character in the application's character set that is the decimal character.
6	Floating Point to DISPLAY Floating Point values map to a COBOL PIC 9 field.	`<element name><integer width><decimal width><decimal character>` where: <element name> is the name of the element that encloses the floating point. It is 252 bytes in length. The element must exist within the NODE parameter, and if <element name> is null then the NODE parameter is used to enclose the text. <integer width> is a binary value specifying the number of digits to the left of the decimal point. The EAE type is an N5. <decimal width> is a binary value specifying the number of digits to the right of the decimal point. The EAE type is an N5. <decimal character> is a single character in the application's character set that is the decimal character.
7	Floating Point to COMPUTATIONAL Floating Point values map to a COBOL PIC S9(11) COMP field, which is stored as packed decimal.	`<element name><decimal width><decimal character>` where: <element name> is the name of the element that encloses the floating point. It is 252 bytes in length. The element must exist within the NODE parameter, and if <element name> is null then the NODE parameter is used to enclose the text. <decimal width> is a binary value specifying the number of digits to the right of the decimal point. The EAE type is an N5. <decimal character> is a single character in the application's character set that is the decimal character.

<mapping type> Value	Description of Value	<field info> Structure
8	Floating Point to DOUBLE Floating Point values map to a COBOL DOUBLE field.	`<element name>` where: <element name> is the name of the element that encloses the floating point. It is 252 bytes in length. The element must exist within the NODE parameter and if <element name> is null then the NODE parameter is used to enclose the text.
9	Group Group maps repeated structures containing multiple elements to a COBOL record with an OCCURS phrase.	`<group name><group max><num items><items>` where: <group name> is the enclosing element for the group. It is 252 bytes in length. The element must exist within the NODE parameter, and if <group name> is null then the NODE parameter is used to enclose the text. <group max> is a binary value specifying the maximum number of items in the group. It is equivalent to the COBOL OCCURS value. <num items> is the number of <item>s that follow in the group. <items> cannot include a <mapping type> value of 9 (Group) or 10 (Array).
10	Array Array treats all text values within the specified element as the same type, and returns them as an array of values.	`<array max><mapping type><field info>` where: <array max> is a binary value specifying the maximum number of items in the array. It is equivalent to the COBOL OCCURS value. <mapping type> can be any of the above <mapping type> values except for 9 (Group) or 10 (Array). <field info> is as described above for <mapping type> values. The <element name> in <field info> should be the enclosing element for the array of values.

Examples

Example 1: Simple XML Document with No Repeated Structures

The following is sample code text for an XML document with no repeated structures:

```
<PRODUCT>
  <NAME>Widget</NAME>
  <QUANTITY>100</QUANTITY>
  <PRICE CURRENCY="USD">7.99</PRICE>
</PRODUCT>
```

The following is a matching 01 record to receive the data:

```
01 PRODUCT.
  03 NAME              PIC X(20).
  03 QUANTITY          PIC S9(11) COMP.
  03 PRICE.
    05 PRICE-VALUE     REAL.
```

The application would specify the mapping with the record as follows:

```
01 PRODUCT-XML-MAPPING.
  03 PRODUCT-XML-MAPPING-LEVEL  PIC  9(11) BINARY VALUE IS 1.
  03 NUM-PRODUCT-XML-MAPPINGS   PIC  9(11) BINARY VALUE IS 3.
  03 PROD-XMLMAP-NAME-TYPE      PIC  9(11) BINARY VALUE IS 1.
  03 PROD-XMLMAP-NAME-NAME      PIC X(252) VALUE IS "NAME".
  03 PROD-XMLMAP-NAME-LEN       PIC  9(11) BINARY VALUE IS 20.
  03 PROD-XMLMAP-QUANTITY-TYPE  PIC  9(11) BINARY VALUE IS 4.
  03 PROD-XMLMAP-QUANTITY-NAME  PIC X(252) VALUE IS "QUANTITY".
  03 PROD-XMLMAP-PRICE-TYPE     PIC  9(11) BINARY VALUE IS 5.
  03 PROD-XMLMAP-PRICE-NAME     PIC X(252) VALUE IS "PRICE".
  03 PROD-XMLMAP-PRICE-DECCHAR  PIC   X(1) VALUE IS ".".
```

Example 2: XML Document with Repeated Substructures

The following is sample code text for an XML document with repeated substructures, showing the use of groups:

```
<PRODUCTS>
  <PRODUCT>
    <NAME>Widget1</NAME>
    <QUANTITY>100</QUANTITY>
    <PRICE CURRENCY="USD">7.99</PRICE>
  </PRODUCT>
  <PRODUCT>
    <NAME>Widget2</NAME>
    <QUANTITY>200</QUANTITY>
    <PRICE CURRENCY="USD">14.99</PRICE>
  </PRODUCT>
</PRODUCTS>
```

The following is a matching 01 record to receive the data:

```
01 PRODUCTS.
  03 PRODUCT  OCCURS 100 TIMES.
    05 NAME              PIC X(20).
    05 QUANTITY          PIC S9(11) BINARY.
```

```
    05 PRICE.
      07 PRICE-VALUE     PIC S9(11) COMP.
```

The application would specify the mapping with the record as follows:

```
01 PRODUCTS-XML-MAPPING.
  03 PRODUCT-XML-MAPPING-LEVEL  PIC  9(11) BINARY VALUE IS 1.
  03 NUM-PRODUCT-XML-MAPPINGS   PIC  9(11) BINARY VALUE IS 1.
  03 PRODUCT-XML-GROUP-TYPE     PIC  9(11) BINARY VALUE IS 9.
  03 PRODUCT-XML-GROUP-NAME     PIC X(252) VALUE IS "PRODUCTS".
  03 PRODUCT-XML-GROUP-MAX      PIC  9(11) BINARY VALUE IS 100.
  03 PRODUCT-XML-GROUP-ITEMS    PIC  9(11) BINARY VALUE IS 3.
  03 PROD-XMLMAP-NAME-TYPE      PIC  9(11) BINARY VALUE IS 1.
  03 PROD-XMLMAP-NAME-NAME      PIC X(252) VALUE IS "NAME".
  03 PROD-XMLMAP-NAME-LEN       PIC  9(11) BINARY VALUE IS 20.
  03 PROD-XMLMAP-QUANTITY-TYPE  PIC  9(11) BINARY VALUE IS 4.
  03 PROD-XMLMAP-QUANTITY-NAME  PIC X(252) VALUE IS "QUANTITY".
  03 PROD-XMLMAP-PRICE-TYPE     PIC  9(11) BINARY VALUE IS 7.
  03 PROD-XMLMAP-PRICE-NAME     PIC X(252) VALUE IS "PRICE".
  03 PROD-XMLMAP-PRICE-DECWIDTH PIC  9(11) BINARY VALUE IS 2.
  03 PROD-XMLMAP-PRICE-DECCHAR  PIC   X(1) VALUE IS ".".
```

Example 3: XML Document with an Array of Values

The following is sample code text for an XML document with an array of values:

```
<PRODUCTS>
  <PRODUCT>
    <NAME>Widget1</NAME>
  </PRODUCT>
  <PRODUCT>
    <NAME>Widget2</NAME>
  </PRODUCT>
  <PRODUCT>
    <NAME>Widget3</NAME>
  </PRODUCT>
</PRODUCTS>
```

The following is a matching 01 record to receive the data:

```
01 PRODUCTS.
  03 PRODUCT  OCCURS 50 TIMES.
    05 NAME              PIC X(20).
```

The application would specify the mapping with the record as follows:

```
01 PRODUCTS-XML-MAPPING.
  03 PRODUCT-XML-MAPPING-LEVEL  PIC  9(11) BINARY VALUE IS 1.
  03 NUM-PRODUCT-XML-MAPPINGS   PIC  9(11) BINARY VALUE IS 1.
  03 PRODUCT-XML-ARRAY-TYPE     PIC  9(11) BINARY VALUE IS 10.
  03 PRODUCT-XML-ARRAY-MAX      PIC  9(11) BINARY VALUE IS 50.
  03 PRODUCT-XML-ARRAY-TYPE     PIC  9(11) BINARY VALUE IS 1.
  03 PROD-XMLMAP-NAME-NAME      PIC X(252) VALUE IS "PRODUCTS".
  03 PROD-XMLMAP-NAME-LEN       PIC  9(11) BINARY VALUE IS 20.
```

WEBAPPSUPPORT Library Procedures for the XML Parser

You can use the WEBAPPSUPPORT library procedures described in this subsection in your applications to use the XML Parser.

The procedure subsections describe the syntax, parameters, and possible return values. Each subsection presents the syntax for

- A COBOL85 entry point, which has uppercase characters and underscores

  An example is APPEND_CHILD.
- An ALGOL entry point, which has lower-case and upper-case characters and no underscores

  An example is appendChild.
- An EAE entry point, which has upper-case characters and dashes

  An example is APPEND-CHILD.

  Note: *For more information on EAE and the notes used in the procedure description text of this guide, refer to Section 3, "WEBAPPSUPPORT EAE Interface."*

APPEND_CHILD

Inserts a child node and the tree of which the child is the root into the XML document. This procedure inserts the tree at the end of the list of subtrees of a specific parent node.

If the new child is already attached to another node, this procedure detaches the child from the current parent and then attaches the child to the new parent.

See also the procedure INSERT_CHILD_BEFORE.

Syntax

```
INTEGER PROCEDURE APPEND_CHILD
                                          (DOC_TAG, PARENT, NEW_CHILD);
  INTEGER                                  DOC_TAG, PARENT, NEW_CHILD;


INTEGER PROCEDURE appendChild
                                          (DOC_TAG, PARENT, NEW_CHILD);
  VALUE                                    DOC_TAG, PARENT, NEW_CHILD;
 INTEGER                                  DOC_TAG, PARENT, NEW_CHILD;


PROCEDURE APPEND-CHILD                    (GLB_PARAM);
  EBCDIC ARRAY                             GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

PARENT identifies the parent node.

NEW_CHILD identifies the child node to append to the parent.

The following table shows the types of child nodes that this procedure can attach to each type of parent node.

Parent	Possible Children
document node	one document type node, one element node, comment nodes, processing instruction nodes
element node	element nodes, text nodes, attribute nodes, entity reference nodes, comment nodes, CDATA nodes, processing instruction nodes
attribute node	text nodes, entity reference nodes

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	PARENT	A6	[bin]
SD	NEW-CHILD	A6	[bin]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
–40	The procedure did not find the XML document.
–41	The parent or child is not a valid node.
–42	The parent node cannot be a parent.
–43	The procedure cannot attach this node to the parent.
–44	The document already has an element.
–45	The document already has a DTD.

CONVERT_COMMA_TEXT_TO_JSON

Converts comma-delimited text to JSON format in UTF-8 encoding. The text can come from either an MCP file or application array. The MCP file can be a stream file containing ASCII text or an MCP record file containing EBCDIC text. For MCP record files, each record boundary causes a line feed character to be inserted.

See the SET_XML_OPTION procedure, INDENT option, for control over JSON formatting.

The first row is used as the names. The following text shows an example:

Comma text: a, b, c, <LF> 1, 2, 3

The previous example text becomes the following JSON text:

[{ "b": "2", "c": "3", "a": "1" }]

Syntax

```
INTEGER PROCEDURE CONVERT_COMMA_TEXT_TO_JSON
                  (SOURCE_TYPE, SOURCE, SOURCE_START, SOURCE_LEN,
                     CHARSET, DEST_TYPE, DEST, DEST_START, DEST_LEN);
    INTEGER         SOURCE_TYPE,         SOURCE_START, SOURCE_LEN,
                     CHARSET, DEST_TYPE,       DEST_START, DEST_LEN;
    EBCDIC ARRAY                 SOURCE,  DEST [0];

INTEGER PROCEDURE convertCommaTextToJSON
                  (SOURCE_TYPE, SOURCE, SOURCE_START, SOURCE_LEN,
                     CHARSET, DEST_TYPE, DEST, DEST_START, DEST_LEN);
    VALUE           SOURCE_TYPE,         SOURCE_START, SOURCE_LEN,
                     CHARSET, DEST_TYPE,       DEST_START;
    INTEGER         SOURCE_TYPE,         SOURCE_START, SOURCE_LEN,
                     CHARSET, DEST_TYPE,       DEST_START, DEST_LEN;
    EBCDIC ARRAY                 SOURCE,  DEST [*];

PROCEDURE CONVERT-COMMA-TEXT-TO-JSON (GLB_PARAM);
    EBCDIC ARRAY                      GLB_PARAM [0];
```

Parameters

SOURCE_TYPE identifies the type of source for the comma-delimited text.

- 1 = the SOURCE parameter contains comma-delimited text.
- 2 = the SOURCE parameter contains the MCP file name that contains the comma-delimited text. See the FILENAME_FORMAT option in the SET_OPTION procedure.

SOURCE is the array containing source information. If SOURCE_TYPE is 2, the file name in SOURCE is coded in the character set of the application.

SOURCE_START is a zero-based offset into the SOURCE array and indicates where the supplied information starts.

SOURCE _LEN is the length in bytes of the data in the SOURCE parameter. If SOURCE _TYPE is 2, then SOURCE _LEN can be zero.

CHARSET is the MLS character set in which the data in the SOURCE parameter is encoded when SOURCE_TYPE = 1. A value of 2 represents UTF-8 encoding.

DEST_TYPE identifies the type of destination for the JSON text.

- 1 = the DEST parameter contains JSON text on procedure return.
- 2 = the DEST parameter contains the MCP file name to store the JSON text. See the FILENAME_FORMAT option in the SET_OPTION procedure.

DEST is the array containing destination information. If DEST_TYPE is 2, the file name in DEST is coded in the character set of the application.

DEST_START is a zero-based offset into the DEST array and indicates where the supplied information starts.

DEST_LEN is the length in bytes of the data in the DEST parameter. If DEST_TYPE is 2, then DEST_LEN can be zero. On return, DEST_LEN is set to the length in bytes of the JSON text or can be zero if an error occurred.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	SOURCE-TYPE	N5	
SD	SOURCE-SIZE	N5	SOURCE size, for example, 2048
SD	SOURCE	A*n*	[longa]
SD	SOURCE-START	N5	
SD	SOURCE-LEN	N5	
SD	CHARSET	N5	
SD	DEST-TYPE	N5	
SD	DEST-SIZE	N5	DEST size, for example, 2048
SD	DEST	A*n*	[longa]
SD	DEST-START	N5	
SD	DEST-LEN	N12	

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	The SOURCE_TYPE or DEST_TYPE is not supported; or DEST_LEN is less than zero when DEST_TYPE = 2.
–11	The input file was not found or is not available.
–13	An attribute error occurred while setting the file name.

–14	An I/O error occurred while reading the input file.
–15	The character set is not supported.
–47	The SOURCE_START or SOURCE_LEN was invalid.
–48	The procedure cannot open a socket to the JPM.
–49	The procedure cannot write to the JPM.
–50	The procedure cannot read from the JPM
–55	The DEST_START offset was invalid.
–57	The JPM does not support the procedure.
–111	The comma text format is invalid.

CONVERT_JSON_TO_XML_DOCUMENT

Converts JSON text to an XML document.

See the SET_XML_OPTION procedure, INDENT option, for control over XML formatting.

See the SET_XML_OPTION procedure, CANONICAL_METHOD option, for control over XML serialization.

See also the PARSE_JSON _TO_XML procedure.

Syntax

```
INTEGER PROCEDURE CONVERT_JSON_TO_XML_DOCUMENT
(SOURCE_TYPE, SOURCE, SOURCE_START, SOURCE_LEN,
DEST_TYPE, OUT_FORMAT, DEST, DEST_START,
DEST_LEN);                                INTEGER          SOURCE_TYPE,
SOURCE_START, SOURCE_LEN,                                   DEST_TYPE,
OUT_FORMAT,       DEST_START,
DEST_LEN;     EBCDIC ARRAY                  SOURCE,     DEST [0];

INTEGER PROCEDURE convertJSONtoXMLDocument                      (SOURCE_TYPE,
SOURCE, SOURCE_START, SOURCE_LEN,                       DEST_TYPE,
OUT_FORMAT, DEST, DEST_START,                       DEST_LEN);         VALUE
SOURCE_TYPE,         SOURCE_START, SOURCE_LEN,
DEST_TYPE, OUT_FORMAT,       DEST_START;         INTEGER
SOURCE_TYPE,         SOURCE_START, SOURCE_LEN,
DEST_TYPE, OUT_FORMAT,       DEST_START,
DEST_LEN;     EBCDIC ARRAY                  SOURCE,     DEST [*];

PROCEDURE CONVERT-JSON-TO-XML-DOCUMENT (GLB_PARAM);
   EBCDIC ARRAY                        GLB_PARAM [0];
```

Parameters

SOURCE_TYPE identifies the type of source for the XML document.

- 1 = the SOURCE parameter contains the XML document.

- 2 = the SOURCE parameter contains the MCP file name that contains the XML document. See the FILENAME_FORMAT option in the SET_OPTION procedure.

SOURCE is the array containing source information. If SOURCE_TYPE is 2, the file name in SOURCE is coded in the character set of the application.

SOURCE_START is a zero-based offset into the SOURCE array and indicates where the supplied information starts.

SOURCE _LEN is the length in bytes of the data in the SOURCE parameter. If SOURCE_TYPE is 2, then SOURCE _LEN can be zero.

DEST_TYPE identifies the type of destination for the JSON text.

- 1 = the DEST parameter contains JSON text on procedure return.
- 2 = the DEST parameter contains the MCP file name to store the JSON text. See the FILENAME_FORMAT option in the SET_OPTION procedure.

OUT_FORMAT identifies the output format of the XML document and can be either of the following values:

- 1 = A carriage return and a line feed are at the end of each non-text node. Each line is indented the number of spaces that the INDENT option in the SET_XML_OPTION procedure specifies.
- 2 = No carriage return, line feed, or white space is between nodes.
- DEST is the array containing destination information. If DEST_TYPE is 2, the file name in DEST is coded in the character set of the application.
- DEST_START is a zero-based offset into the DEST array and indicates where the supplied information starts.
- DEST_LEN is the length in bytes of the data in the DEST parameter. If DEST-TYPE is 2, then DEST_LEN can be zero. On return, DEST_LEN is set to the length in bytes of the XML text or might be zero if an error occurred.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	SOURCE-TYPE	N5	
SD	SOURCE-SIZE	N5	SOURCE size, for example, 2048
SD	SOURCE	A*n*	[longa]
SD	SOURCE-START	N5	
SD	SOURCE-LEN	N5	
SD	DEST-TYPE	N5	
SD	OUT-FORMAT	N5	
SD	DEST-SIZE	N5	DEST size, for example, 2048
SD	DEST	A*n*	[longa]
SD	DEST-START	N5	
SD	DEST-LEN	N12	

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	The SOURCE_TYPE or DEST_TYPE is not supported, or DEST_LEN is less than zero when DEST_TYPE = 2.
–11	The input file was not found or is not available.
–13	An attribute error occurred while setting the file name.
–14	An I/O error occurred while reading the input file.
–47	The SOURCE_START or SOURCE_LEN was invalid.
–48	The procedure cannot open a socket to the JPM.
–49	The procedure cannot write to the JPM.
–50	The procedure cannot read from the JPM
–51	One or more parsing errors occurred.
–55	The DEST_START offset was invalid.
–57	The JPM does not support the procedure.

CONVERT_XML_DOCUMENT_TO_JSON

Converts an XML document to JSON format in UTF-8 encoding. The XML document can come from either an MCP file or an application array.

Some information might be lost in this transformation because JSON is a data format and XML is a document format. XML uses elements, attributes, and content text; JSON uses unordered collections of name/value pairs and arrays of values. JSON does not distinguish between elements and attributes, and does not recognize namespaces. Sequences of similar elements are represented as JSON arrays. Content text might be placed in a "content" member. Comments, prologs, DTDs, and <[[ ]]> are ignored.

XML documents using namespaces should not be converted to JSON. If GET_XML_DOCUMENT procedure is used to create the XML document, set the NAMESPACE_PROCESSING option in the SET_XML_OPTION procedure to 3 before calling the procedure.

See the SET_XML_OPTION procedure, INDENT option, for control over JSON formatting.

See also the CONVERT_XML _TO_JSON procedure.

Syntax

```
INTEGER PROCEDURE CONVERT_XML_DOCUMENT_TO_JSON
                 (SOURCE_TYPE, SOURCE, SOURCE_START, SOURCE_LEN,
                    DEST_TYPE, DEST,   DEST_START,   DEST_LEN);
  INTEGER         SOURCE_TYPE,         SOURCE_START, SOURCE_LEN,
                    DEST_TYPE,         DEST_START,   DEST_LEN;
  EBCDIC ARRAY                 SOURCE, DEST [0];

INTEGER PROCEDURE convertXMLDocumentToJSON
                 (SOURCE_TYPE, SOURCE, SOURCE_START, SOURCE_LEN,
                    DEST_TYPE, DEST,   DEST_START,   DEST_LEN);
  VALUE           SOURCE_TYPE,         SOURCE_START, SOURCE_LEN,
                    DEST_TYPE,         DEST_START;
  INTEGER         SOURCE_TYPE,         SOURCE_START, SOURCE_LEN,
                    DEST_TYPE,         DEST_START,   DEST_LEN;
  EBCDIC ARRAY                 SOURCE, DEST [*];

PROCEDURE CONVERT-XML-DOCUMENT-TO-JSON (GLB_PARAM);
   EBCDIC ARRAY                         GLB_PARAM [0];
```

Parameters

SOURCE_TYPE identifies the type of source for the XML document.

- 1 = the SOURCE parameter contains the XML document.
- 2 = the SOURCE parameter contains the MCP file name that contains the XML document. See the FILENAME_FORMAT option in the SET_OPTION procedure.

SOURCE is the array containing source information. If SOURCE_TYPE is 2, the file name in SOURCE is coded in the character set of the application.

SOURCE_START is a zero-based offset into the SOURCE array and indicates where the supplied information starts.

SOURCE _LEN is the length in bytes of the data in the SOURCE parameter. If SOURCE _TYPE is 2, then SOURCE _LEN can be zero.

DEST_TYPE identifies the type of destination for the JSON text.

- 1 = the DEST parameter contains JSON text on procedure return.
- 2 = the DEST parameter contains the MCP file name to store the JSON text. See the FILENAME_FORMAT option in the SET_OPTION procedure.

DEST is the array containing destination information. If DEST_TYPE is 2, the file name in DEST is coded in the character set of the application.

DEST_START is a zero-based offset into the DEST array and indicates where the supplied information starts.

DEST_LEN is the length in bytes of the data in the DEST parameter. If DEST-TYPE is 2, then DEST_LEN can be zero. On return, DEST_LEN is set to the length in bytes of the JSON text or might be zero if an error occurred.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	SOURCE-TYPE	N5	
SD	SOURCE-SIZE	N5	SOURCE size, for example, 2048
SD	SOURCE	A*n*	[longa]
SD	SOURCE-START	N5	
SD	SOURCE-LEN	N5	
SD	DEST-TYPE	N5	
SD	DEST-SIZE	N5	DEST size, for example, 2048
SD	DEST	A*n*	[longa]
SD	DEST-START	N5	
SD	DEST-LEN	N12	

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	The SOURCE_TYPE or DEST_TYPE is not supported, or DEST_LEN is less than zero when DEST_TYPE = 2.
–11	The input file was not found or is not available.
–13	An attribute error occurred while setting the file name.
–14	An I/O error occurred while reading the input file.
–47	The SOURCE_START or SOURCE_LEN was invalid.
–48	The procedure cannot open a socket to the JPM.
–49	The procedure cannot write to the JPM.
–50	The procedure cannot read from the JPM
–51	One or more parsing errors occurred.
–55	The DEST_START offset was invalid.
–57	The JPM does not support the procedure.

CONVERT_XML_TO_JSON

Converts a parsed XML document stored in the WEBAPPSUPPORT library to JSON text in UTF-8 encoding.

Some information might be lost in this transformation because JSON is a data format and XML is a document format. XML uses elements, attributes, and content text; JSON uses unordered collections of name/value pairs and arrays of values. JSON does not distinguish between elements and attributes and does not recognize namespaces.

Sequences of similar elements are represented as JSON arrays. Content text might be placed in a "content" member. Comments, prologs, DTDs, and <[[ ]]> are ignored.

Namespace information in the XML document is removed before converting to JSON.

See the SET_XML_OPTION procedure, INDENT option, for control over JSON formatting.

See also the CONVERT_XML _DOCUMENT_TO_JSON procedure.

Syntax

```
INTEGER PROCEDURE CONVERT_XML_TO_JSON
                 (DOC_TAG, DEST_TYPE, DEST, DEST_START, DEST_LEN);
  INTEGER         DOC_TAG, DEST_TYPE,       DEST_START, DEST_LEN;
  EBCDIC ARRAY                        DEST [0];

INTEGER PROCEDURE convertXMLtoJSON
                 (DOC_TAG, DEST_TYPE, DEST, DEST_START, DEST_LEN);
  VALUE           DOC_TAG, DEST_TYPE,       DEST_START;
  INTEGER         DOC_TAG, DEST_TYPE,       DEST_START, DEST_LEN;
  EBCDIC ARRAY                        DEST [*];

PROCEDURE CONVERT-XML-TO-JSON (GLB_PARAM);
   EBCDIC ARRAY                GLB_PARAM [0];
```

Parameters

DOC_TAG is the XML document.

DEST_TYPE identifies the type of destination for the JSON text.

- 1 = the DEST parameter contains JSON text on procedure return.
- 2 = the DEST parameter contains the MCP file name to store the JSON text. See the FILENAME_FORMAT option in the SET_OPTION procedure.

DEST is the array containing destination information. If DEST_TYPE is 2, the file name in DEST is coded in the character set of the application.

DEST_START is a zero-based offset into the DEST array and indicates where the supplied information starts.

DEST_LEN is the length in bytes of the data in the DEST parameter. If DEST-TYPE is 2, then DEST_LEN can be zero. On return, DEST_LEN is set to the length in bytes of the JSON text or might be zero if an error occurred.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	DEST-TYPE	N5	
SD	DEST-SIZE	N5	DEST size, for example, 2048
SD	DEST	A*n*	[longa]
SD	DEST-START	N5	
SD	DEST-LEN	N12	

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	The DEST_TYPE is not supported, or DEST_LEN is less than zero when DEST_TYPE = 2.
-13	An attribute error occurred while setting the file name.
-47	The DEST_LEN is less than zero.
-48	The procedure cannot open a socket to the JPM.
-49	The procedure cannot write to the JPM.
-50	The procedure cannot read from the JPM
-55	The DEST_START offset was invalid.
-57	The JPM does not support the procedure.

CREATE_ATTRIBUTE_NODE

Creates an attribute node in the XML document.

After an application creates the node, the application needs to attach the node to the element node to which you want the attribute to apply. See the APPEND_CHILD procedure.

For information about setting attribute values, see “Setting or Deleting an Attribute Value” in Section 5.

Syntax

```
INTEGER PROCEDURE CREATE_ATTR_NODE
                                (DOC_TAG, NAMESPACE, QUALIFIED_NAME, NODE);
  INTEGER                        DOC_TAG,                            -NODE;
  EBCDIC ARRAY                            NAMESPACE, QUALIFIED_NAME [0];
```

```
INTEGER PROCEDURE createAttributeNode
                                 (DOC_TAG, NAMESPACE, QUALIFIED_NAME, NODE);
  VALUE                           DOC_TAG;
  INTEGER                         DOC_TAG,                            NODE;
  EBCDIC ARRAY                             NAMESPACE, QUALIFIED_NAME [*];

PROCEDURE CREATE-ATTRIBUTE-NODE   (GLB_PARAM);
   EBCDIC ARRAY                    GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

NAMESPACE is the attribute namespace as a Uniform Resource Identifier (URI) in the application character set. If the NAMESPACE parameter is null, this attribute does not have a namespace. An example of a NAMESPACE value is

```
http://somedomain/mynamespace
```

QUALIFIED_NAME is the attribute name in the application character set and cannot be a null string. If this parameter is specified with prefix text before a colon (:), the prefix is a namespace prefix. The procedure does not validate the prefix against an actual namespace declaration in an element that encloses the node.

NODE is the returned attribute node.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	NAMESPACE-SIZE	N5	NAMESPACE size, for example, 255
SD	NAMESPACE	A*n*	[longa]
SD	QUALIFIED-NAME-SIZE	N5	QUALIFIED-NAME size, for example, 255
SD	QUALIFIED-NAME	A*n*	[longa]
SD	NODE	A6	[bin]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	The procedure did not find the XML document.
-35	The procedure call did not specify a field.
-56	The procedure cannot create another node because the maximum number of nodes already exists.

CREATE_CDATA_NODE

Creates a CDATA node for the XML document.

The application needs to attach the CDATA node to the document node or to an element node. See the APPEND_CHILD and INSERT_CHILD_BEFORE procedures.

Syntax

```
INTEGER PROCEDURE CREATE_CDATA_NODE
                                    (DOC_TAG, CDATA_TEXT, CDATA_NODE);
  INTEGER                            DOC_TAG,             CDATA_NODE;
  EBCDIC ARRAY                                 CDATA_TEXT [0];

INTEGER PROCEDURE createCDATANode
                                    (DOC_TAG, CDATA_TEXT, CDATA_NODE);
  VALUE                              DOC_TAG;
  INTEGER                            DOC_TAG,             CDATA_NODE;
  EBCDIC ARRAY                                  CDATA_TEXT [*];

PROCEDURE CREATE-CDATA-NODE         (GLB_PARAM);
   EBCDIC ARRAY                      GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

CDATA_TEXT is the text for the CDATA node. The application must ensure that it supplies only text data for this parameter. Nontextual data in this parameter can make a document invalid.

The text in this parameter

- Must be in the application character set
- Cannot include the prefix characters **<![CDATA[** or the suffix characters **]]>**
- Cannot be a null string

For example, CDATA_TEXT could be the following:

```
This is unparsed text.
```

The procedure would add the following to the XML document:

```
<![CDATA[ This is unparsed text. ]]>
```

CDATA_NODE is the returned CDATA node.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	CDATA-TEXT-SIZE	N5	CDATA-TEXT size, for example, 255
SD	CDATA-TEXT	A*n*	[longa]
SD	CDATA-NODE	A6	[bin]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	The procedure did not find the XML document.
-35	The procedure call did not specify a field.
-56	The procedure cannot create another node because the maximum number of nodes already exists.

CREATE_CIPHER_REFERENCE

Creates a cipher reference in an existing XML document. A cipher reference uses a URI to reference encrypted keys or data. See also the ENCRYPT_DATA procedure.

The form of the cipher reference created is as follows:

```
<parent>
  <CipherReference URI="urivalue">
    <Transforms/>
  </CipherReference>
```

An example of a cipher reference with a base64 transform created by this procedure follows:

```
<CipherData>
 <CipherReference URI="http://dataserver/reports/sales/january">
   <Transforms>
     <ds:Transform
         Algorithm="http://www.w3.org/2000/09/xmldsig#base64"/>
   </Transforms>
 </CipherReference>
</CipherData>
```

Syntax

```
INTEGER PROCEDURE CREATE_CIPHER_REFERENCE
              (DOC_TAG, PARENT, URI, TRANSFORM_TYPE, TRANSFORMS_NODE);
  INTEGER      DOC_TAG, PARENT,      TRANSFORM_TYPE, TRANSFORMS_NODE;
  EBCDIC ARRAY                  URI [0];

INTEGER PROCEDURE createCipherReference
              (DOC_TAG, PARENT, URI, TRANSFORM_TYPE, TRANSFORMS_NODE);
  VALUE        DOC_TAG, PARENT,      TRANSFORM_TYPE;
  INTEGER      DOC_TAG, PARENT,      TRANSFORM_TYPE, TRANSFORMS_NODE;
  EBCDIC ARRAY                  URI [*];

PROCEDURE CREATE-CIPHER-REFERENCE (GLB_PARAM);
   EBCDIC ARRAY                    GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

PARENT is the node that is to be the parent of the *CipherReference* element, which is added as the last child of the parent node..

URI is a string in the character set of the application that is the value for the URI attribute of the *CipherReference* element.

TRANSFORM_TYPE indicates a predefined *Transform* element that is to be automatically added to the *Transforms* element.

If the value is 0, no *Transform* element is added to the *Transforms* element.

If the value is 1, a base64 transform is added to the *Transforms* element, indicating that the encrypted data accessed by the URI value is encoded in base64. The algorithm attribute for the *Transform* element is "http://www.w3.org/2000/09/xmldsig#base64".

TRANSFORMS_NODE is the generated node that is a child of the *CipherReference* element. The application can add specific transforms to this node, such as XPath expressions.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	PARENT	A6	[bin]
SD	URI-SIZE	N5	URI size, for example, 255
SD	URI	A*n*	[longa]
SD	TRANSFORM-TYPE	N5	
SD	TRANSFORMS-NODE	A6	[bin]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
–35	The procedure call did not specify a field.
–40	The procedure did not find the XML document.
–41	The parent is not a valid node.
–42	The parent node cannot be a parent.
–44	The document already has an element.

CREATE_COMMENT_NODE

Creates a comment node for the XML document.

The application needs to attach the comment node to the document node or to an element node. See the APPEND_CHILD and INSERT_CHILD_BEFORE procedures.

Syntax

```
INTEGER PROCEDURE CREATE_COMMENT_NODE
                                    (DOC_TAG, COMMENT_TEXT, COMMENT_NODE);
  INTEGER                            DOC_TAG,               COMMENT_NODE;
  EBCDIC ARRAY                                COMMENT_TEXT [0];

INTEGER PROCEDURE createCommentNode
                                    (DOC_TAG, COMMENT_TEXT, COMMENT_NODE);
  VALUE                              DOC_TAG;
  INTEGER                            DOC_TAG,               COMMENT_NODE;
  EBCDIC ARRAY                                COMMENT_TEXT [*];

PROCEDURE CREATE-COMMENT-NODE (GLB_PARAM);
   EBCDIC ARRAY                GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

COMMENT_TEXT is the text for the comment node. The text

- Must be in the application character set
- Cannot include the prefix characters **<!--** or the suffix characters **-->**

For example, COMMENT_TEXT could be the following:

```
This is a comment.
```

The procedure would add the following to the XML document:

```
<!-- This is a comment. -->
```

COMMENT_NODE is the returned comment node.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	COMMENT-TEXT-SIZE	N5	COMMENT-TEXT size, for example, 255
SD	COMMENT-TEXT	A*n*	[longa]
SD	COMMENT-NODE	A6	[bin]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	The procedure did not find the XML document.
-35	The procedure call did not specify a field.
-56	The procedure cannot create another node because the maximum number of nodes already exist.

CREATE_DOCTYPE_NODE

Creates a document type node for the XML document. This node contains a DTD.

The application needs to attach the document type node to the document node before the document element node. See the APPEND_CHILD and INSERT_CHILD_BEFORE procedures.

An XML document can have only one DTD. If an application attaches a second document type node to the document node, the XML Parser detaches the first document type node.

Syntax

```
INTEGER PROCEDURE CREATE_DOCTYPE_NODE
                                        (DOC_TAG, DOCTYPE_TEXT,
DOCTYPE_NODE);
  INTEGER                                DOC_TAG,               DOCTYPE_NODE;
  EBCDIC ARRAY                                    DOCTYPE_TEXT [0];

INTEGER PROCEDURE createDoctypeNode
                                       (DOC_TAG, DOCTYPE_TEXT, DOCTYPE_NODE);
  VALUE                                 DOC_TAG;
  INTEGER                               DOC_TAG,               DOCTYPE_NODE;
  EBCDIC ARRAY                                    DOCTYPE_TEXT [*];
```

```
PROCEDURE CREATE-DOCTYPE-NODE          (GLB_PARAM);
   EBCDIC ARRAY                         GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

DOCTYPE_TEXT is the text for the document type node. The text

- Must be in the application character set
- Cannot include the prefix characters **<!DOCTYPE** or end with the suffix character **>**.

For example, DOCTYPE_TEXT could be the following:

```
LABEL SYSTEM "http://xxx/label.dtd"
```

The procedure would add the following to the XML document:

```
<!DOCTYPE LABEL SYSTEM ”http://xxx/label.dtd”>
```

DOCTYPE_NODE is the returned document type node.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	DOCTYPE-TEXT-SIZE	N5	DOCTYPE-TEXT size, for example, 255
SD	DOCTYPE-TEXT	A*n*	[longa]
SD	DOCTYPE-NODE	A6	[bin]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
–35	The procedure call did not specify a field.
–40	The procedure did not find the XML document.
–56	The procedure cannot create another node because the maximum number of nodes already exists.

CREATE_ELEMENT_NODE

Creates an element node for the XML document.

The application needs to attach the element node to the document node or an element node. See the APPEND_CHILD and INSERT_CHILD_BEFORE procedures.

Syntax

```
INTEGER PROCEDURE CREATE_ELEMENT_NODE
                       (DOC_TAG, NAMESPACE, QUALIFIED_NAME, NODE);
 INTEGER                    DOC_TAG,            NODE;
 EBCDIC ARRAY                      NAMESPACE, QUALIFIED_NAME [0];

INTEGER PROCEDURE createElementNode
                       (DOC_TAG, NAMESPACE, QUALIFIED_NAME, NODE);
 VALUE                     DOC_TAG;
 INTEGER                    DOC_TAG,            NODE;
 EBCDIC ARRAY                      NAMESPACE, QUALIFIED_NAME [*];

PROCEDURE CREATE-ELEMENT-NODE      (GLB_PARAM);
   EBCDIC ARRAY                     GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

NAMESPACE is the element namespace, as a URI, in the application character set. If the NAMESPACE parameter is null, this element does not have a namespace. An example of a NAMESPACE value is

```
http://somedomain/mynamespace
```

QUALIFIED_NAME is the element tag name in the application character set. If this parameter is specified with prefix text before a colon (:), the prefix is a namespace prefix. The procedure does not validate the prefix against an actual namespace declaration in an element that encloses the node.

NAME is the element tag name in the application character set.

NODE is the returned element node.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	NAMESPACE-SIZE	N5	NAMESPACE size, for example, 255
SD	NAMESPACE	A*n*	[longa]
SD	QUALIFIED-NAME-SIZE	N5	QUALIFIED-NAME size, for example, 255
SD	QUALIFIED-NAME	A*n*	[longa]
SD	NODE	A6	[bin]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
–35	The procedure call did not specify a field.
–40	The procedure did not find the XML document.
–56	The procedure cannot create another node because the maximum number of nodes already exists.

CREATE_ENTITYREF_NODE

Creates an entity reference node for the XML document.

The application needs to attach the entity reference node to an element or attribute node. See the APPEND_CHILD and INSERT_CHILD_BEFORE procedures.

Syntax

```
INTEGER PROCEDURE CREATE_ENTITYREF_NODE
                                      (DOC_TAG, NAME, NODE);
  INTEGER                              DOC_TAG,       NODE;
  EBCDIC ARRAY                                  NAME [0];

INTEGER PROCEDURE createEntityRefNode
                                      (DOC_TAG, NAME, NODE);
  VALUE                                DOC_TAG;
  INTEGER                              DOC_TAG,       NODE;
  EBCDIC ARRAY                                  NAME [*];

PROCEDURE CREATE-ENTITYREF-NODE       (GLB_PARAM);
   EBCDIC ARRAY                       GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

NAME is the entity reference name in the application character set. Do not put an ampersand (&) or a semi-colon (;) in this name.

NODE is the returned entity reference node.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	NAME-SIZE	N5	NAME size, for example, 255
SD	NAME	A*n*	[longa]
SD	NODE	A6	[bin]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
–35	The procedure call did not specify a field.
–40	The procedure did not find the XML document.
–56	The procedure cannot create another node because the maximum number of nodes already exists.

CREATE_PI_NODE

Creates a processing instruction node for the XML document.

The application needs to attach the processing instruction node to the document node or an element node. See the APPEND_CHILD and INSERT_CHILD_BEFORE procedures.

Syntax

```
INTEGER PROCEDURE CREATE_PI_NODE
                                    (DOC_TAG, PI_TARGET, PI_TEXT, PI_NODE);
  INTEGER                            DOC_TAG,                      PI_NODE;
  EBCDIC ARRAY                                PI_TARGET, PI_TEXT [0];

INTEGER PROCEDURE createPINode
                                    (DOC_TAG, PI_TARGET, PI_TEXT, PI_NODE);
  VALUE                              DOC_TAG;
  INTEGER                            DOC_TAG,                      PI_NODE;
  EBCDIC ARRAY                                PI_TARGET, PI_TEXT [*];

PROCEDURE CREATE-PI-NODE            (GLB_PARAM);
   EBCDIC ARRAY                      GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

PI_TARGET is the text for the target of the processing instruction and must be in the application character set.

PI_TEXT is the text for the processing instruction node. This text must be in the application character set. Do not put the characters **?>** in this parameter.

For example, the PI_TARGET could be the following:

```
xml-stylesheet
```

PI_TEXT could be the following:

```
type="text/xml" href="5-2.xsl"
```

The procedure would add the following to the XML document:

```
<?xml-stylesheet type="text/xml" href="5-2.xsl"?>
```

PI_NODE is the returned processing instruction node.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	PI-TARGET-SIZE	N5	PI-TARGET size, for example, 256
SD	PI-TARGET	A*n*	[longa]
SD	PI-TEXT-SIZE	N5	PI-TEXT size, for example, 2048
SD	PI-TEXT	A*n*	[longa]
SD	NODE	A6	[bin]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
-35	The procedure call did not specify a field.
-40	The procedure did not find the XML document.
-56	The procedure cannot create another node because the maximum number of nodes already exist.

CREATE_TEXT_ELEMENT

Creates an element node with a text node attached for the XML document.

This procedure call combines these functions:

- Create an element.
- Attach the element to its parent node.
- Set zero or more simple text attributes on the element.
- Create a text node.
- Attach the text node to the element

This example shows how the XML might look after this procedure returns (changes in italics)

```
<PARENTNODE>
  <ELEMENTNAME ATTR1="Attr1 Text">Some Text</ELEMENTNAME>
```

Syntax

```
INTEGER PROCEDURE CREATE_TEXT_ELEMENT
            (DOC_TAG, PARENT, NAMESPACE, QUALIFIED_ELEMENT_NAME,
               NUM_ATTRS, MAX_ATTR_NAMESPACE_LEN, MAX_ATTR_NAME_LEN,
                 MAX_ATTR_VALUE_LEN, ATTR_BUFFER, TEXT, TEXT_START,
                   TEXT_LENGTH, ELEMENT_NODE);
  INTEGER    DOC_TAG, PARENT,
               NUM_ATTRS, MAX_ATTR_NAMESPACE_LEN, MAX_ATTR_NAME_LEN,
                 MAX_ATTR_VALUE_LEN,                    TEXT_START,
                   TEXT_LENGTH, ELEMENT_NODE;
  EBCDIC ARRAY                NAMESPACE, QUALIFIED_ELEMENT_NAME,
                                     ATTR_BUFFER, TEXT [0];

INTEGER PROCEDURE createTextElement
            (DOC_TAG, PARENT, NAMESPACE, QUALIFIED_ELEMENT_NAME,
               NUM_ATTRS, MAX_ATTR_NAMESPACE_LEN, MAX_ATTR_NAME_LEN,
                 MAX_ATTR_VALUE_LEN, ATTR_BUFFER, TEXT, TEXT_START,
                   TEXT_LENGTH, ELEMENT_NODE);
  VALUE      DOC_TAG, PARENT,
               NUM_ATTRS, MAX_ATTR_NAMESPACE_LEN, MAX_ATTR_NAME_LEN,
                 MAX_ATTR_VALUE_LEN,                    TEXT_START,
                   TEXT_LENGTH;
  INTEGER    DOC_TAG, PARENT,
               NUM_ATTRS, MAX_ATTR_NAMESPACE_LEN, MAX_ATTR_NAME_LEN,
                 MAX_ATTR_VALUE_LEN,                    TEXT_START,
                   TEXT_LENGTH, ELEMENT_NODE;
  EBCDIC ARRAY                NAMESPACE, QUALIFIED_ELEMENT_NAME,
                                     ATTR_BUFFER, TEXT [*];

PROCEDURE CREATE-TEXT-ELEMENT (GLB_PARAM);
   EBCDIC ARRAY                GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

PARENT identifies the parent node for the new element.

NAMESPACE is the element namespace, as a URI, in the application character set. If the NAMESPACE parameter is null, this element does not have a namespace. An example of a NAMESPACE value is

http://somedomain/mynamespace

QUALIFIED_ELEMENT_NAME is the element tag name in the application character set. If this parameter is specified with prefix text before a colon (:), the prefix is a namespace prefix. The procedure does not validate the prefix against an actual namespace declaration in an element that encloses the node.

NUM_ATTRS is the number of attributes to add to the element.

MAX_ATTR_NAMESPACE_LEN is the length of the namespace field for each attribute in ATTR_BUFFER. The valid range is 0 to 2048.

MAX_ATTR_NAME_LEN is the length in bytes of the attribute name field for each attribute in ATTR_BUFFER. The valid range is 1 to 2048.

MAX_ATTR_VALUE_LEN is the length in bytes of the attribute value field for each attribute in ATTR_BUFFER. The valid range is 1 to 2048.

ATTR_BUFFER is the buffer containing the attributes to be added to the element. This buffer contains three fields for each attribute:

- ATTRIBUTE_NAMESPACE is the attribute namespace, as a URI, in the application character set of up to MAX_ATTR_NAMESPACE_LEN bytes. If the ATTRIBUTE_NAMESPACE parameter is null, this attribute does not have a namespace.

  The following example shows an ATTRIBUTE_NAMESPACE value.

  http://somedomain/mynamespace

- ATTRIBUTE_NAME is the attribute name in the application character set of up to MAX_ATTR_NAME_LEN bytes. If this parameter is specified with prefix text before a colon (:), the prefix is a namespace prefix. The procedure does not validate the prefix against an actual namespace declaration in an element that encloses the node.
- ATTRIBUTE_VALUE is the attribute value in the application character set of up to MAX_ATTR_VALUE_LEN bytes.

TEXT is the text value for the text node in the application character set. The application must ensure that it supplies only text data for this parameter. Nontextual data in this parameter might invalidate an XML document. The text in this parameter cannot be a null string.

TEXT_START is a zero-based offset into TEXT and indicates where the text value starts. A COBOL85 application with arrays that start at 1 must pass 0 (zero).

TEXT_LENGTH is the length of data in TEXT.

ELEMENT_NODE is the created element node.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	PARENT	A6	[bin]
SD	NAMESPACE-SIZE	N5	NAMESPACE size, for example, 256
SD	NAMESPACE	A*n*	[longa]
SD	ELEMENT-NAME-SIZE	N5	ELEMENT-NAME size, for example, 256
SD	ELEMENT-NAME	A*n*	[longa]
SD	NUM-ATTRS	N5	
SD	MAX-ATTR-NS-LEN	N5	
SD	MAX-ATTR-NAME-LEN	N5	
SD	MAX-ATTR-VALUE-LEN	N5	
SD	ATTR-BUFFER-SIZE	N5	
SD	ATTR-BUFFER	A*n*	ATTR-BUFFER size, for example, 2048
SD	TEXT-SIZE	N5	[longa]
SD	TEXT	A*n*	TEXT size, for example, 256
SD	TEXT-START	N5	[longa]
SD	TEXT-LENGTH	N5	
SD	ELEMENT-NODE	A6	
			[bin]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
–35	The procedure call did not specify a required field.
–40	The procedure did not find the XML document.
–41	The parent is not a valid node.
–42	The parent node cannot be a parent
–56	The procedure cannot create another node because the maximum number of nodes already exists.

Example

The following code is an example of ATTR_BUFFER used for this procedure in COBOL

```
01 ATTR-BUFFER.
   03 ATTR-PAIR OCCURS 10 TIMES.
     05 ATTR-NAMESPACE  PIC X(30).
     05 ATTR-NAME       PIC X(10).
     05 ATTR-VALUE      PIC X(20).
```

The call to CREATE_TEXT_ELEMENT passes ATTR-BUFFER, with MAX_ATTR_NAMESPACE_LEN set to 30, MAX_ATTR_NAME_LEN set to 10, and MAX_ATTR_VALUE_LEN set to 20.

CREATE_TEXT_NODE

Creates a text node for the XML document.

The application needs to attach the text node to an element or attribute node. See the APPEND_CHILD and INSERT_CHILD_BEFORE procedures.

Syntax

```
INTEGER PROCEDURE CREATE_TEXT_NODE
                               (DOC_TAG, TEXT, TEXT_START, TEXT_LENGTH,
NODE);
  INTEGER                       DOC_TAG,       TEXT_START, TEXT_LENGTH, NODE;
  EBCDIC ARRAY                            TEXT [0];

INTEGER PROCEDURE createTextNode
                               (DOC_TAG, TEXT, TEXT_START, TEXT_LENGTH,
NODE);
  VALUE                         DOC_TAG        TEXT_START, TEXT_LENGTH;
  INTEGER                       DOC_TAG,       TEXT_START, TEXT_LENGTH, NODE;
  EBCDIC ARRAY                            TEXT [*];

PROCEDURE CREATE-TEXT-NODE     (GLB_PARAM);
   EBCDIC ARRAY                 GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

TEXT is the text value for the node and must be in the application character set. The application must ensure that it supplies only text data for this parameter. Nontextual data in this parameter can make a document invalid. The text in this parameter cannot be a null string.

TEXT_START is the zero-based offset into TEXT and indicates where the text value starts. A COBOL85 application with arrays that start at 1 must pass 0 (zero).

TEXT_LENGTH is the length of data in TEXT. If zero, TEXT contains a string that is terminated by blanks or a null byte.

NODE is the returned text node.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	TEXT-SIZE	N5	TEXT size, for example, 256
SD	TEXT	A*n*	[longa]
SD	TEXT-START	N5	
SD	TEXT-LENGTH	N5	
SD	NODE	A6	[bin]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
–35	The procedure call did not specify a field.
–40	The procedure did not find the XML document.
–56	The procedure cannot create another node because the maximum number of nodes already exist.

CREATE_XML_DOCUMENT

Creates an empty XML document in WEBAPPSUPPORT. The document node is returned.

This procedure identifies the character set in which to create the document. When an application accesses the document, the application must use the character set that was the application character set when the document was created.

The SET_TRANSLATION procedure sets the application character set.

Syntax

```
INTEGER PROCEDURE CREATE_XML_DOCUMENT
                                    (DOC_TAG, XML_DECLARATION, NODE);
  INTEGER                            DOC_TAG,                  NODE;
  EBCDIC ARRAY                                 XML_DECLARATION [0];

INTEGER PROCEDURE createXMLDocument
                                    (DOC_TAG, XML_DECLARATION, NODE);
  INTEGER                            DOC_TAG,                  NODE;
  EBCDIC ARRAY                                 XML_DECLARATION [*];

PROCEDURE CREATE-XML-DOCUMENT       (GLB_PARAM);
   EBCDIC ARRAY                      GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document. If this procedure creates the document, this procedure returns DOC_TAG with a non-zero value.

XML_DECLARATION is the XML document XML declaration in the application character set.

If the application character set is UCS2 (85), then this parameter must be encoded in ASCII.

If this parameter is null, no XML declaration is in the document. The XML Parser does not check the validity of the information in the declaration.

If the XML declaration has an encoding string, the application can use that encoding to determine the character set in the document that the GET_XML_DOCUMENT procedure returns. If the XML declaration does not have an encoding string, the GET_XML_DOCUMENT procedure returns the document encoded in UTF-8 by default.

The <? and ?> prefix and suffix characters must be in the XML_DECLARATION parameter.

Following are two examples of the XML_DECLARATION parameter:

```
<?xml version="1.0"?>

<?xml version="1.0"" encoding="ISO-8859-1"?>
```

NODE is the returned document node. If an error occurs, NODE is 0 (zero).

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	XML-DECLARATION-SIZE	N5	XML-DECLARATION size, for example, 256
SD	XML-DECLARATION	A*n*	[longa]
SD	NODE	A6	[bin]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	No-op. The procedure did not create the document because WEBAPPSUPPORT already has the maximum number of XML documents.
1	The procedure created the document.
-15	The procedure did not create the document because the XML Parser does not support the application character set.

DECRYPT_XML_DOCUMENT

Decrypts an *EncryptedData* element into a new XML document. The data encrypted must be an XML document or fragment. This procedure returns a tag to a new XML document.

Only the *CipherValue* element is supported. The *CipherReference* element is not supported—that is, automatic retrieval of the data from a URI does not occur.

Syntax

```
INTEGER PROCEDURE DECRYPT_XML_DOCUMENT
                     (DOC_TAG, KEY_TAG, NODE, NEW_DOC_TAG;
  INTEGER             DOC_TAG, KEY_TAG, NODE, NEW_DOC_TAG;

INTEGER PROCEDURE decryptXMLdocument
                     (DOC_TAG, KEY_TAG, NODE, NEW_DOC_TAG;
  VALUE               DOC_TAG, KEY_TAG, NODE;
  INTEGER             DOC_TAG, KEY_TAG, NODE, NEW_DOC_TAG;

PROCEDURE DECRYPT-XML-DOCUMENT (GLB_PARAM);
   EBCDIC ARRAY                 GLB_PARAM [0];
```

Parameters

DOC_TAG is the source XML document.

KEY_TAG is the key object used to decrypt the data. This parameter must reference a valid key object that can be used to decrypt the encrypted data.

NODE represents the *EncryptedData* element to be decrypted.

NEW_DOC_TAG is the new XML document containing the decrypted items.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	KEY-TAG	A6	[bin]
SD	NODE	A6	[bin]
SD	NEW-DOC-TAG	A6	[bin]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
–121	The XML Encryption Key is required.
0	No-op for one of the following reasons: • NODE is not an *EncryptedData* element. • WEBAPPSUPPORT already has the maximum number of documents.
–40	The procedure did not find the XML document.
–41	The node is not valid.
–48	The procedure cannot open a socket to the JPM.
–49	The procedure cannot writer to the JPM.
–50	The procedure cannot read from the JPM.
–51	One or more parsing errors occurrec.
–122	MCAPI is unavailable.
–123	The key is invalid
–140	The *EncryptedData* element is not properly formed.

DECRYPT_XML_TO_DATA

Decrypts data in an XML document into an application array or into an MCP file. For example, the data could represent an XML fragment or binary data such as a jpeg file.

For example, the following XML fragment encrypted in an XML document:

```
<CreditCard><Number>1234567890</Number></CreditCard>
```

Could be decrypted from

```
<?xml version='1.0' ?>
<Payment>
  <EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'
                      xmlns='http://www.w3.org/2001/04/xmlenc#'>
    <CipherData>
      <CipherValue>MTIzNDU2Nzg5M==</CipherValue>
    </CipherData>
  </EncryptedData>
</Payment>
```

See also the DECRYPT_XML_DOCUMENT procedure

Syntax

```
INTEGER PROCEDURE DECRYPT_XML_TO_DATA
                    (DOC_TAG, KEY_TAG, NODE,
                       DEST_TYPE, DEST, DEST_START, DEST_LEN);
  INTEGER            DOC_TAG, KEY_TAG, NODE,
                       DEST_TYPE,       DEST_START, DEST_LEN;
  EBCDIC ARRAY                     DEST [0];

INTEGER PROCEDURE decryptXMLtoData
                    (DOC_TAG, KEY_TAG, NODE,
                       DEST_TYPE, DEST, DEST_START, DEST_LEN);
  VALUE              DOC_TAG, KEY_TAG, NODE,
                       DEST_TYPE,       DEST_START;
  INTEGER            DOC_TAG, KEY_TAG, NODE,
                       DEST_TYPE,       DEST_START, DEST_LEN;
  EBCDIC ARRAY                     DEST [*];

PROCEDURE DECRYPT-XML-DATA (GLB_PARAM);
   EBCDIC ARRAY             GLB_PARAM [0];
```

Parameters

DOC_TAG is the XML document containing the encrypted item.

KEY_TAG is the key object used to decrypt the data.

NODE represents the *EncryptedData* element to be decrypted.

DEST_TYPE identifies the type of destination for data to be decrypted.

- 1 = the DEST parameter contains decrypted data on procedure return.
- 2 = the DEST parameter contains the MCP file name to store the decrypted data. See the FILENAME_FORMAT option in the SET_OPTION procedure.

DEST is the array containing destination information. If DEST_TYPE is 2, the file name in DEST is coded in the character set of the application.

DEST_START is a zero-based offset into the DEST array and indicates where the supplied information starts.

DEST_LEN is the length in bytes of the data in the DEST parameter. If DEST-TYPE is 2, then DEST_LEN can be zero.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	KEY-TAG	A6	[bin]
SD	NODE	A6	[bin]
SD	DEST-TYPE	N5	
SD	DEST-SIZE	N5	DEST size, for example, 2048
SD	DEST	A*n*	[longa]
SD	DEST-START	N5	
SD	DEST-LEN	N12	

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
–121	The XML Encryption Key is required.
0	No-op for one of the following reasons: • NODE is not an *EncryptedData* element. • The DEST_TYPE value is not supported. • WEBAPPSUPPORT already ahs the maximum number of documents.
–40	The procedure did not find the XML document.
–41	The NODE parameter is not a valid node.
–55	The DEST_START parameter is invalid.
–122	MCAPI is unavailable.
–123	The key is invalid.

ENCRYPT_DATA_TO_XML

Encrypts data in an application array or in an MCP file into an XML document. For example, the data could represent an XML fragment or binary data such as a jpeg file.

This procedure creates an *EncryptedData* element either appended to the supplied parent node or as the document element if a new XML document is created.

Within the *EncryptedData* element, a *CipherData* element holds the encrypted data.

For example, the following XML fragment encrypted stored in an application array:

```
<CreditCard><Number>1234567890</Number></CreditCard>
```

Could be encrypted and appended to the Payment element as:

```
<?xml version='1.0' ?>
<Payment>
  <EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'
                      xmlns='http://www.w3.org/2001/04/xmlenc#'>
    <CipherData>
      <CipherValue>MTIzNDU2Nzg5M==</CipherValue>
    </CipherData>
  </EncryptedData>
</Payment>
```

See the SET_XML_OPTION procedure, CANONICAL_METHOD option, for control over XML serialization.

See also the ENCRYPT_XML_DOCUMENT procedure.

Syntax

```
INTEGER PROCEDURE ENCRYPT_DATA_TO_XML
              (SOURCE_TYPE, SOURCE, SOURCE_START, SOURCE_LEN, ID,
                 DATA_TYPE, MIME_TYPE, ADD_METHOD,
                   KEY_TAG, DOC_TAG, PARENT, ENCRYPTED_NODE);
  INTEGER      SOURCE_TYPE,         SOURCE_START, SOURCE_LEN,
                                       ADD_METHOD,
                   KEY_TAG, DOC_TAG, PARENT, ENCRYPTED_NODE;
  EBCDIC ARRAY              SOURCE,                          ID,
                 DATA_TYPE, MIME_TYPE [0];

INTEGER PROCEDURE encryptDataToXML
              (SOURCE_TYPE, SOURCE, SOURCE_START, SOURCE_LEN, ID,
                 DATA_TYPE, MIME_TYPE, ADD_METHOD,
                   KEY_TAG, DOC_TAG, PARENT, ENCRYPTED_NODE);
  VALUE        SOURCE_TYPE,         SOURCE_START, SOURCE_LEN,
                                       ADD_METHOD,
                   KEY_TAG,          PARENT, ENCRYPTED_NODE;
  INTEGER      SOURCE_TYPE,         SOURCE_START, SOURCE_LEN,
                                       ADD_METHOD, ADD_KEYSIZE,
                   KEY_TAG, DOC_TAG,  PARENT, ENCRYPTED_NODE;
  EBCDIC ARRAY              SOURCE,                          ID,
                 DATA_TYPE, MIME_TYPE [*];

PROCEDURE ENCRYPT-DATA-TO-XML (GLB_PARAM);
   EBCDIC ARRAY                GLB_PARAM [0];
```

Parameters

SOURCE_TYPE identifies the type of source of the data to be encrypted.

- 1 = the SOURCE parameter contains the data to be encrypted.
- 2 = the SOURCE parameter contains the MCP file name of the data to be encrypted. See the FILENAME_FORMAT option in the SET_OPTION procedure.

SOURCE is the array containing source information. If SOURCE_TYPE is 2, the file name in SOURCE is coded in the character set of the application.

SOURCE_START is a zero-based offset into the SOURCE array and indicates where the supplied information starts.

SOURCE_LEN is the length in bytes of the data in the SOURCE parameter.

ID is a string in the character set of the application that is the value for the *Id* attribute of the *EncryptedData* element. If ID is null, the *Id* attribute is not created.

DATA_TYPE is a string in the character set of the application that is the URL that identifies the type of data being encrypted and is used in the *Type* attribute of the *EncryptedData* element. For example, "http://www.isi.edu/in-notes/iana/assignments/media-types/text/xml" represents the encoding of an XML document. If this string is null, the *Type* attribute is not created.

MIME_TYPE is a string in the character set of the application that identifies the media type of the data that is encrypted and is used in the *MimeType* attribute of the *EncryptedData* element. If this string is null, the *MimeType* attribute is not created.

ADD_METHOD controls whether or not to add the *EncryptionMethod* element.

- 0 = do not add the element.
- 1 = add the element. The *Algorithm* attribute of the *EncryptionMethod* element is generated based on the encryption algorithm used.

KEY_TAG is the key object used to encrypt the data.

DOC_TAG is the XML document containing the encrypted item. If supplied as -1, an XML document only containing the encrypted item is created and the resulting document is returned in this parameter. Otherwise, the encrypted item is added to the document referenced by DOC_TAG.

PARENT is the parent node for the encrypted data in the XML document. If DOC_TAG is supplied as -1, this parameter is ignored. Otherwise, PARENT must represent a valid element or document node, and the *EncryptedData* node is added as the last child of PARENT.

ENCRYPTED_NODE is the *EncryptedData* element node created.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	SOURCE-TYPE	N5	
SD	SOURCE-SIZE	N5	SOURCE size, for example, 2048
SD	SOURCE	A*n*	[longa]
SD	SOURCE-START	N5	
SD	SOURCE-LEN	N5	
SD	ID-SIZE	N5	ID size, for example, 256
SD	ID	A*n*	[longa]
SD	DATA-TYPE-SIZE	N5	DATA-TYPE size, for example, 256
SD	DATA-TYPE	A*n*	[longa]
SD	MIME-TYPE-SIZE	N5	MIME-TYPE size, for example, 256
SD	MIME-TYPE	A*n*	[longa]
SD	ADD-METHOD	N5	
SD	KEY-TAG	A6	[bin]
SD	DOC-TAG	A6	[bin]
SD	PARENT	A6	[bin]
SD	ENCRYPTED-NODE	A6	[bin]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
–121	The XML Encryption Key is required.
0	No-op for one of the following reasons: • The SOURCE_TYPE value is not supported. • WEBAPPSUPPORT already has the maximum number of documents.
–35	The procedure call did not specify a field.
–42	The PARENT parameter is not an element or the document node.
–47	The source length or start is invalid.
–122	MCAPI is unavailable.
–123	The key is invalid

ENCRYPT_XML_DOCUMENT

Encrypts an element (or its contents), text node, or entire XML document, creating a new XML document.

The *Type* attribute is automatically added to the *EncryptedData* element, based on the item encrypted. The *xmlns* attribute is also added to the *EncryptedData* element.

Also, an *EncryptionMethod* element can optionally be added to the *EncryptedData* element.

The formatting applied to the XML text before encryption is controlled by the INDENT option of the SET_XML_OPTION procedure. If INDENT is set to zero, the text is compressed with no whitespace prior to encryption; otherwise, if INDENT is nonzero, whitespace is applied to the text prior to encryption.

For example, an XML document containing an element to be encrypted:

```
<?xml version='1.0' ?>
<Payment>
  <CreditCard>
    <Number>1234567890</Number>
  </CreditCard>
</Payment>
```

Can have the *CreditCard* element and its child nodes encrypted as:

```
<?xml version='1.0' ?>
  <Payment>
    <EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'
                        xmlns='http://www.w3.org/2001/04/xmlenc#'>
      <EncryptedMethod
         Algorithm='http://www.w3.org/2001/04/xmlenc#tripledes-cbc'/>
      <CipherData>
        <CipherValue>MTIzNDU2Nzg5M==</CipherValue>
      </CipherData>
    </EncryptedData>
  </Payment>
```

See the SET_XML_OPTION procedure, CANONICAL_METHOD option, for control over XML serialization.

See also ENCRYPT_DATA_TO_XML procedure.

Syntax

```
INTEGER PROCEDURE ENCRYPT_XML_DOCUMENT
                   (DOC_TAG, KEY_TAG, NODE, CONTENT_ONLY, ID,
                      ADD_METHOD, NEW_DOC_TAG, ENCRYPTED_NODE,
                        METHOD_NODE);
  INTEGER           DOC_TAG, KEY_TAG, NODE, CONTENT_ONLY,
                      ADD_METHOD, NEW_DOC_TAG, ENCRYPTED_NODE,
                        METHOD_NODE;
  EBCDIC ARRAY                                          ID [0];

INTEGER PROCEDURE encryptXMLdocument
                   (DOC_TAG, KEY_TAG, NODE, CONTENT_ONLY, ID,
                      ADD_METHOD, NEW_DOC_TAG, ENCRYPTED_NODE,
                        METHOD_NODE);
  VALUE             DOC_TAG, KEY_TAG, NODE, CONTENT_ONLY,
                      ADD_METHOD;
  INTEGER           DOC_TAG, KEY_TAG, NODE, CONTENT_ONLY,
                      ADD_METHOD, NEW_DOC_TAG, ENCRYPTED_NODE,
                        METHOD_NODE;
  EBCDIC ARRAY                                          ID [*];

PROCEDURE ENCRYPT-XML-DOCUMENT (GLB_PARAM);
   EBCDIC ARRAY                GLB_PARAM [0];
```

Parameters

DOC_TAG is the source XML document.

KEY_TAG is the key object used to encrypt the data.

NODE represents the item to be encrypted. It can be

- The document node, which causes the entire XML document to be encrypted
- An element node, which causes the element and all child nodes to be encrypted
- A text node, which causes the text node to be encrypted

CONTENT_ONLY controls if the element and its content are encrypted or if only the content of the element is encrypted. This parameter is ignored if NODE is not an element.

- 0 = encrypt the element and its content.
- 1 = encrypt the content of the element only.

ID is a string in the character set of the application that is the value for the *Id* attribute of the *EncryptedData* element. If ID is null, the *Id* attribute is not created.

ADD_METHOD controls whether or not to add the *EncryptionMethod* element to the *EncryptedData* element.

- 0 = do not add the *EncryptionMethod* element.
- 1 = add the *EncryptionMethod* element. The *Algorithm* attribute of the *EncryptionMethod* element is generated based on the encryption algorithm used. If the encryption algorithm is not one defined by the XML Encryption standard, this attribute is not added.

NEW_DOC_TAG is the new XML document containing the encrypted item.

ENCRYPTED_NODE is the *EncryptedData* element node that replaced NODE.

METHOD_NODE is the node created if ADD_METHOD is 1. Otherwise, METHOD_NODE is set to -1.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	KEY-TAG	A6	[bin]
SD	NODE	A6	[bin]
SD	CONTENT-ONLY	N5	
SD	ID-SIZE	N5	ID size, for example, 256
SD	ID	A*n*	[longa]
SD	ADD-METHOD	N5	
SD	NEW-DOC-TAG	A6	[bin]
SD	ENCRYPTED-NODE	A6	[bin]
SD	METHOD-NODE	A6	[bin]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
–121	The XML Encryption Key is required.
0	No-op for one of the following reasons: • No data exists to encrypt. • The maximum number of XML documents was exceeded.
–35	The procedure did not specify a field, or the NODE parameter is not an element, text, or document node.
–40	The procedure did not find the XML document.
–41	The NODE parameter is not a valid node.
–56	The maximum number of nodes was exceeded.

–122	MCAPI is unavailable.
–123	The key is invalid.

GET_ATTRIBUTE_BY_NAME

Searches for an attribute by name in an element node.

If the procedure finds the attribute, the procedure returns a successful result and the attribute node. The procedure returns only the first attribute node that has the specified name.

If the procedure does not find the attribute or the node that the application supplies is not an element node, the procedure returns a no-op result.

Syntax

```
INTEGER PROCEDURE GET_ATTRIBUTE_BY_NAME
                                    (DOC_TAG, NODE, ATTR_NAME, ATTR_NODE);
  INTEGER                            DOC_TAG, NODE             ATTR_NODE;
  EBCDIC ARRAY                                        ATTR_NAME [0];

INTEGER PROCEDURE getAttributeByName
                                    (DOC_TAG, NODE, ATTR_NAME, ATTR_NODE);
  VALUE                              DOC_TAG, NODE;
  INTEGER                            DOC_TAG, NODE             ATTR_NODE;
  EBCDIC ARRAY                                        ATTR_NAME [*];

PROCEDURE GET-ATTRIBUTE-BY-NAME     (GLB_PARAM);
   EBCDIC ARRAY                       GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

NODE identifies the element node.

ATTR_NAME is the attribute name in the application character set. If ATTR_NAME is a local name without a namespace prefix, then the procedure returns the first attribute with the name, which might be a qualified name with prefixes. If ATTR_NAME is a qualified name with a namespace prefix, then the procedure returns the first attribute with the qualified name that includes the namespace prefix. Attribute names are case-sensitive.

ATTR_NODE is the attribute node.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	NODE	A6	[bin]
SD	ATTR-NAME-SIZE	N5	ATTR-NAME size, for example, 256
SD	ATTR-NAME	A*n*	[longa]
SD	ATTR-NODE	A6	[bin]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	No-op. The specified node is not an element node.
–35	The procedure call did not specify a field.
40	The procedure did not find the XML document.
41	The specified node is not a valid node.

GET_ATTRIBUTES

Returns a list of attribute nodes for the specified element node. If the element node does not have attributes or is not an element node, the procedure returns a no-op result.

Use the GET_NODE_NAME procedure to get the attribute name.

Use the GET_NODE_VALUE procedure to get the attribute value of an attribute that has one text value.

Use the GET_FIRST_CHILD and GET_NEXT_SIBLING procedures to get the subnodes of each attribute in the list.

Syntax

```
INTEGER PROCEDURE GET_ATTRIBUTES
                                    (DOC_TAG, NODE, ATTR_LIST, LIST_LEN);
  INTEGER                            DOC_TAG, NODE,            LIST_LEN;
  INTEGER ARRAY                                       ATTR_LIST [0];

INTEGER PROCEDURE getAttributes
                                    (DOC_TAG, NODE, ATTR_LIST, LIST_LEN);
  VALUE                              DOC_TAG, NODE;
  INTEGER                            DOC_TAG, NODE,            LIST_LEN;
  INTEGER ARRAY                                       ATTR_LIST [*];
```

```
PROCEDURE GET-ATTRIBUTES              (GLB_PARAM);
   EBCDIC ARRAY                        GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

NODE identifies the element node.

ATTR_LIST is the list of attribute nodes.

LIST_LEN is the number of attributes in ATTR_LIST.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	NODE	A6	[bin]
SD	ATTR-LIST-SIZE	N5	ATTR-LIST size, for example, 300 = 50*6
SD	ATTR-LIST	A*n*	[longa]
SD	LIST-LEN	N5	

ATTR-LIST is an array of A6 [bin]. In the above example, an ATTR-LIST-SIZE of 300 allows up to 50 node IDs to be returned.

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	No-op. The element does not have attributes, or the node is not an element.
–40	The procedure did not find the XML document.
–41	The specified node is not a valid node.

GET_CHILD_NODES

Returns a list of child nodes for the specified parent node. If the specified node does not have children or is a type of node that cannot have children, the procedure returns a no-op result.

Syntax

```
INTEGER PROCEDURE GET_CHILD_NODES
                                    (DOC_TAG, NODE, NODE_LIST, LIST_LEN);
  INTEGER                            DOC_TAG, NODE,            LIST_LEN;
  INTEGER ARRAY                                       NODE_LIST [0];

INTEGER PROCEDURE getChildNodes
                                    (DOC_TAG, NODE, NODE_LIST, LIST_LEN);
  VALUE                              DOC_TAG, NODE;
  INTEGER                            DOC_TAG, NODE,            LIST_LEN;
  INTEGER ARRAY                                       NODE_LIST [*];

PROCEDURE GET-CHILD-NODES           (GLB_PARAM);
   EBCDIC ARRAY                      GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

NODE identifies the parent node.

NODE_LIST is the list of nodes.

LIST_LEN is the number of nodes in the NODE_LIST parameter.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	NODE	A6	[bin]
SD	NODE-LIST-SIZE	N5	NODE-LIST size, for example, 300 = 50*6
SD	NODE-LIST	A*n*	[longa]
SD	LIST-LEN	N5	

NODE-LIST is an array of A6 [bin]. In the above example, a NODE-LIST-SIZE of 300 allows up to 50 node IDs to be returned.

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	No-op. The specified node cannot be a parent node or does not have any children.
–40	The procedure did not find the XML document.
–41	The specified node is not a valid node.

GET_DOCUMENT_ELEMENT

Returns the document element of the XML document. The document element is the top-level element of the document.

Syntax

```
INTEGER PROCEDURE GET_DOCUMENT_ELEMENT
                                    (DOC_TAG, ELEMENT_NODE);
  INTEGER                            DOC_TAG, ELEMENT_NODE;

INTEGER PROCEDURE getDocumentElement
                                    (DOC_TAG, ELEMENT_NODE);
  VALUE                              DOC_TAG;
  INTEGER                            DOC_TAG, ELEMENT_NODE;

PROCEDURE GET-DOCUMENT-ELEMENT      (GLB_PARAM);
   EBCDIC ARRAY                       GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

ELEMENT_NODE is the document top-level element. If an error occurs, null (-1) is returned for ELEMENT_NODE.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	ELEMENT-NODE	A6	[bin]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	No-op. The document does not have an element.
-40	The procedure did not find the XML document.

GET_DOCUMENT_ENCODING

Returns encoding information for the XML document. The encoding information is in text format.

The topic "Specifying the Document Character Set" in Section 5 lists the encoding strings that specify character sets in XML documents.

Syntax

```
INTEGER PROCEDURE GET_DOCUMENT_ENCODING
                                    (DOC_TAG, ENCODING_TEXT);
  INTEGER                            DOC_TAG;
  EBCDIC ARRAY                                ENCODING_TEXT [0];

INTEGER PROCEDURE getDocumentEncoding
                                    (DOC_TAG, ENCODING_TEXT);
  VALUE                              DOC_TAG;
  INTEGER                            DOC_TAG;
  EBCDIC ARRAY                                ENCODING_TEXT [*];

PROCEDURE GET-DOCUMENT-ENCODING     (GLB_PARAM);
   EBCDIC ARRAY                       GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

ENCODING_TEXT is the document encoding that the procedure returns. The text is in the application character set and is the text value for the XML document header.

For example, the XML document header could contain the following:

```
<?xml version="1.0" encoding="KOI8-R"?>
```

The ENCODING_TEXT parameter would contain KOI8-R.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	ENCODING-TEXT-SIZE	N5	ENCODING-TEXT size, for example, 256
SD	ENCODING-TEXT	A*n*	[longa]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	No-op. The document does not specify an encoding.
–40	The procedure did not find the XML document.

GET_DOCUMENT_NODE

Returns the document node of this XML document.

The document node is the root of the tree. The document element, which is the top-level element of the document, is a child of the document node.

Syntax

```
INTEGER PROCEDURE GET_DOCUMENT_NODE
                                          (DOC_TAG, NODE);
  INTEGER                                  DOC_TAG, NODE;

INTEGER PROCEDURE getDocumentNode
                                         (DOC_TAG, NODE);
  VALUE                                   DOC_TAG;
  INTEGER                                 DOC_TAG, NODE;

PROCEDURE GET-DOCUMENT-NODE              (GLB_PARAM);
   EBCDIC ARRAY                           GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

NODE is the returned document node. If an error occurs, the value 0 (zero) is returned.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	DOCUMENT-NODE	A6	[bin]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
–40	The procedure did not find the XML document.

GET_DOCUMENT_VERSION

Returns the XML document version as a string. If the XML document does not declare a version, the procedure returns 1.0.

Syntax

```
INTEGER PROCEDURE GET_DOCUMENT_VERSION
                                    (DOC_TAG, VERSION);
  INTEGER                            DOC_TAG;
  EBCDIC ARRAY                                VERSION [0];

INTEGER PROCEDURE getDocumentVersion
                                    (DOC_TAG, VERSION);
  VALUE                              DOC_TAG;
  INTEGER                            DOC_TAG;
  EBCDIC ARRAY                                 VERSION [*];

PROCEDURE GET-DOCUMENT-VERSION      (GLB_PARAM);
   EBCDIC ARRAY                       GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

VERSION is the document version in the application character set. For example, VERSION can be 1.0.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	VERSION-SIZE	N5	VERSION size, for example, 256
SD	VERSION	A*n*	[longa]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	No-op. The document does not have a version specified.
-40	The procedure did not find the XML document.

GET_ELEMENTS_BY_TAGNAME

Returns a list of element nodes that are under the specified node. The element nodes can be only those that match the name specified or can be all nodes. The specified node can be the document node or an element node.

Searching is case-sensitive.

Syntax

```
INTEGER PROCEDURE GET_ELEMENTS_BY_TAGNAME
                                (DOC_TAG, NODE, NAME, NODE_LIST, LIST_LEN);
  INTEGER                        DOC_TAG, NODE,                 LIST_LEN;
  EBCDIC ARRAY                                   NAME [0];
  INTEGER ARRAY                                        NODE_LIST [0];

INTEGER PROCEDURE getElementsByTagName
                                (DOC_TAG, NODE, NAME, NODE_LIST, LIST_LEN);
  VALUE                          DOC_TAG, NODE;
  INTEGER                        DOC_TAG, NODE,                 LIST_LEN;
  EBCDIC ARRAY                                   NAME [*];
  INTEGER ARRAY                                        NODE_LIST [*];

PROCEDURE GET-ELEMENTS-BY-TAGNAME (GLB_PARAM);
   EBCDIC ARRAY                   GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

NODE identifies the node under which the procedure searches.

NAME is the node name to search for and must be in the application character set. If NAME is a local name without a namespace prefix, then the procedure returns all nodes with the name, which might be a qualified name with prefixes. If NAME is a qualified name with a namespace prefix, then the procedure returns all nodes with the qualified name that includes the namespace prefix.

If NAME is an empty string, all nodes under NODE are returned.

NODE_LIST is the returned list of nodes.

LIST_LEN is the number of nodes in the NODE_LIST parameter.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	NODE	A6	[bin]
SD	NAME-SIZE	N5	NAME size, for example, 256
SD	NAME	A*n*	[longa]
SD	NODE-LIST-SIZE	N5	NODE-LIST size, for example, 300 = 50*6
SD	NODE-LIST	A*n*	[longa]
SD	LIST-LEN	N5	

NODE-LIST is an array of A6 [bin]. In the above example, a NODE-LIST-SIZE of 300 allows up to 50 node IDs to be returned.

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	No-op. The document does not have any elements that match the tag name.
–40	The procedure did not find the XML document.
–41	The specified node is not a valid node.

GET_FIRST_CHILD

Returns the node that is the first child of the specified parent node. If the parent node does not have children, the procedure returns a no-op result.

Syntax

```
INTEGER PROCEDURE GET_FIRST_CHILD
                                    (DOC_TAG, PARENT, FIRST_CHILD);
  INTEGER                            DOC_TAG, PARENT, FIRST_CHILD;

INTEGER PROCEDURE getFirstChild
                                    (DOC_TAG, PARENT, FIRST_CHILD);
  VALUE                              DOC_TAG, PARENT;
  INTEGER                            DOC_TAG, PARENT, FIRST_CHILD;

PROCEDURE GET-FIRST-CHILD           (GLB_PARAM);
   EBCDIC ARRAY                      GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

PARENT identifies the parent node.

FIRST_CHILD is the first child of the parent.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	PARENT	A6	[bin]
SD	FIRST-CHILD	A6	[bin]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	No-op. The specified parent node is not a parent node or does not have any children
–40	The procedure did not find the XML document.
–41	The specified parent node is not a valid node.

GET_LAST_CHILD

Returns the node that is the last child of the specified parent node. If the parent node does not have children, the procedure returns a no-op result.

Syntax

```
INTEGER PROCEDURE GET_LAST_CHILD
                                    (DOC_TAG, PARENT, LAST_CHILD);
  INTEGER                            DOC_TAG, PARENT, LAST_CHILD;

INTEGER PROCEDURE getLastChild
                                   (DOC_TAG, PARENT, LAST_CHILD);
  VALUE                             DOC_TAG, PARENT;
  INTEGER                           DOC_TAG, PARENT, LAST_CHILD;

PROCEDURE GET-LAST-CHILD           (GLB_PARAM);
   EBCDIC ARRAY                     GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

PARENT identifies the parent node.

LAST_CHILD is the last child of the parent.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	PARENT	A6	[bin]
SD	LAST-CHILD	A6	[bin]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	No-op. The specified parent node is not a parent node or does not have any children.
–40	The procedure did not find the XML document.
–41	The specified parent node is not a valid node.

GET_NEXT_ITEM

Returns the following:

- The next node that follows the specified node
- The type of the returned node

Use this procedure in an application to access nodes in the XML document in SAX mode, that is, sequentially.

If the procedure reaches the end of the XML document, the procedure returns a no-op result.

Syntax

```
INTEGER PROCEDURE GET_NEXT_ITEM
                          (DOC_TAG, NODE, NEXT_NODE, NODE_TYPE, NODE_NAME);
  INTEGER                  DOC_TAG, NODE, NEXT_NODE, NODE_TYPE;
  EBCDIC ARRAY                                                NODE_NAME
[0];

INTEGER PROCEDURE getNextItem
                          (DOC_TAG, NODE, NEXT_NODE, NODE_TYPE, NODE_NAME);
  VALUE                    DOC_TAG, NODE;
  INTEGER                  DOC_TAG, NODE, NEXT_NODE, NODE_TYPE;
  EBCDIC ARRAY                                                NODE_NAME
[*];

PROCEDURE GET-NEXT-ITEM   (GLB_PARAM);
   EBCDIC ARRAY            GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

NODE identifies that precedes the returned node.

NEXT_NODE is the next node that follows the node in the NODE parameter.

NODE_TYPE is the type of node that the NEXT_NODE parameter specifies. For the end of an element node, attribute node, or entity reference node with children, the application should pass the *end of Element Node*, *End of Attribute Node*, or *End of Entity Reference Node* value, respectively.

NODE_TYPE can be any value listed in the following table:

Value	Description
0	end of document (for input or output) or processing error (only for output)
1	element node
2	attribute node
3	text node

Value	Description
4	CDATA section node
5	entity reference node
7	processing instruction node
8	comment node
9	document node (only for input)
10	document type node
13	end of element node
14	end of attribute node
15	end of entity reference node

The application needs to pass

- The element node value (13) to indicate that the node is the end of an element node
- The end of attribute node value (14) to indicate that the node is the end of an attribute node
- The end of entity reference node value (15) to indicate that the node is the end of an entity reference node with children

NODE_NAME is the name of the node that the NEXT_NODE parameter indicates and is in the application character set. The NAMESPACE_PROCESSING option in the SET_XML_OPTION procedure controls the format of the returned name.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	NODE	A6	[bin]
SD	NEXT-NODE	A6	[bin]
SD	NODE-TYPE	N5	
SD	NODE-NAME-SIZE	N5	NODE-NAME size, for example, 256
SD	NODE-NAME	A*n*	[longa]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	No-op. The procedure reached the end of the document.
–40	The procedure did not find the XML document.
–41	The specified node is not a valid node.

GET_NEXT_SIBLING

Returns the sibling node that immediately follows the specified node. If no sibling node follows the specified node, the procedure returns a no-op result.

Syntax

```
INTEGER PROCEDURE GET_NEXT_SIBLING
                                    (DOC_TAG, NODE, NEXT);
  INTEGER                            DOC_TAG, NODE, NEXT;

INTEGER PROCEDURE getNextSibling
                                    (DOC_TAG, NODE, NEXT);
  VALUE                              DOC_TAG, NODE;
  INTEGER                            DOC_TAG, NODE, NEXT;

PROCEDURE GET-NEXT-SIBLING          (GLB_PARAM);
   EBCDIC ARRAY                       GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

NODE identifies the node that the application specifies and that precedes the returned node.

NEXT is the next sibling of the node that the NODE parameter identifies.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	NODE	A6	[bin]
SD	NEXT	A6	[bin]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	No-op. No node follows the specified node, or the specified node is the document node.
–40	The procedure did not find the XML document.
–41	The specified node is not a valid node.

GET_NODE_BY_XPATH

Returns the first node in the XML document, relative to the context node supplied, that matches the XPATH expression.

Refer to Section 1, “XML Language (XPath) Support” for any limitations.

See also the GET_NODES_BY_XPATH procedure.

Syntax

```
INTEGER PROCEDURE GET_NODE_BY_XPATH
                           (DOC_TAG, CONTEXT_NODE, XPATH, NODE);
  INTEGER                   DOC_TAG, CONTEXT_NODE,        NODE;
  EBCDIC ARRAY                                      XPATH [0];



INTEGER PROCEDURE getNodeByXPath
                           (DOC_TAG, CONTEXT_NODE, XPATH, NODE);
  VALUE                     DOC_TAG, CONTEXT_NODE;
  INTEGER                   DOC_TAG, CONTEXT_NODE,        NODE;
  EBCDIC ARRAY                                      XPATH [*];

PROCEDURE GET-NODE-BY-XPATH (GLB_PARAM);
   EBCDIC ARRAY              GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

CONTEXT_NODE identifies the context node for starting the XPath expression evaluation.

XPATH is the XPath expression as a string in the application character set. For example (not including quote characters): "//*[@class='city']"

NODE is the returned node.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	CONTEXT-NODE	A6	[bin]
SD	XPATH-SIZE	N5	XPATH size, for example, 256
SD	XPATH	A*n*	[longa]
SD	NODE	A6	[bin]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	The document does not have any nodes that match the expression.
–90	The expression is an invalid/unsupported Xpath expression.

GET_NODE_NAME

Returns the name of the specified node.

The following table specifies the kind of name returned for each node type.

Node Type	Node Name
attribute	<name of attribute>
CDATA section	#cdata-section
comment	#comment
document	#document
document type	#DTD
element	<tag name>
entity reference	<name of entity referenced>
processing instruction	<target>
text	#text

Syntax

```
INTEGER PROCEDURE GET_NODE_NAME
                                (DOC_TAG, NODE, NAMESPACE, NODE_NAME);
  INTEGER                        DOC_TAG, NODE;
  EBCDIC ARRAY                                  NAMESPACE, NODE_NAME [0];
```

```
INTEGER PROCEDURE getNodeName
                                 (DOC_TAG, NODE, NAMESPACE, NODE_NAME);
  VALUE                           DOC_TAG, NODE;
  INTEGER                         DOC_TAG, NODE;
  EBCDIC ARRAY                                   NAMESPACE, NODE_NAME [*];

PROCEDURE GET-NODE-NAME          (GLB_PARAM);
   EBCDIC ARRAY                   GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

NODE identifies the node the name of which the procedure returns.

NAMESPACE is the Uniform Resource Identifier (URI) for the node namespace and is returned in the application character set. If this URI is not declared for an element or attribute node, the default string is returned. The default is

Ǵ

For a node that does not have a name that includes a namespace, the NAMESPACE parameter is returned null.

NODE_NAME is the node name and is returned in the application character set. The NAMESPACE_PROCESSING option of the SET_XML_OPTION procedure controls the format of the NODE_NAME parameter.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	NODE	A6	[bin]
SD	NAMESPACE-SIZE	N5	NAMESPACE size, for example, 256
SD	NAMESPACE	A*n*	[longa]
SD	NODE-NAME-SIZE	N5	NODE-NAME size, for example, 256
SD	NODE-NAME	A*n*	[longa]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
–40	The procedure did not find the XML document.
–41	The specified node is not a valid node.

GET_NODES_BY_XPATH

Returns the set of nodes in the XML document, relative to the context node supplied, that matches the XPATH expression.

Refer to Section 1, "XML Language (XPath) Support" for any limitations.

See also the GET_NODE_BY_XPATH procedure.

Syntax

```
INTEGER PROCEDURE GET_NODES_BY_XPATH
                  (DOC_TAG, CONTEXT_NODE, XPATH, NODE_LIST, LIST_LEN);
  INTEGER          DOC_TAG, CONTEXT_NODE,                   LIST_LEN;
  EBCDIC  ARRAY                            XPATH [0];
  INTEGER ARRAY                                   NODE_LIST [0];

INTEGER PROCEDURE getNodesByXPath
                  (DOC_TAG, CONTEXT_NODE, XPATH, NODE_LIST, LIST_LEN);
  VALUE            DOC_TAG, CONTEXT_NODE;
  INTEGER          DOC_TAG, CONTEXT_NODE,                   LIST_LEN;
  EBCDIC  ARRAY                            XPATH [*];
  INTEGER ARRAY                                   NODE_LIST [*];

PROCEDURE GET-NODES-BY-XPATH (GLB_PARAM);
   EBCDIC ARRAY               GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

CONTEXT_NODE identifies the context node for starting the XPath expression evaluation.

XPATH is the XPath expression as a string in the application character set.

NODE_LIST is the returned list of nodes. Its size might be increased if needed to hold the list of nodes.

LIST_LEN is the number of nodes in the NODE_LIST parameter.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	CONTEXT-NODE	A6	[bin]
SD	XPATH-SIZE	N5	XPATH size, for example, 256
SD	XPATH	A*n*	[longa]
SD	NODE-LIST-SIZE	N5	NODE-LIST size, for example, 300 = 50*6
SD	NODE-LIST	A*n*	[longa]
SD	LIST-LEN	N5	

NODE-LIST is an array of A6 [bin]. In the above example, a NODE-LIST-SIZE of 300 allows up to 50 node IDs to be returned.

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	The document does not have any nodes that match the expression.
–90	The expression is an invalid/unsupported Xpath expression.

GET_NODE_TYPE

Returns the node type of the specified node.

Syntax

```
INTEGER PROCEDURE GET_NODE_TYPE
                                      (DOC_TAG, NODE, NODE_TYPE);
  INTEGER                              DOC_TAG, NODE, NODE_TYPE;

INTEGER PROCEDURE getNodeType
                                     (DOC_TAG, NODE, NODE_TYPE);
  VALUE                               DOC_TAG, NODE;
  INTEGER                             DOC_TAG, NODE, NODE_TYPE;

PROCEDURE GET-NODE-TYPE              (GLB_PARAM);
   EBCDIC ARRAY                       GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

NODE identifies the node for which the procedure returns the node type.

NODE_TYPE is the returned node type and can be any value in the following table.

Value	Node Type
1	element
2	attribute
3	text node
4	CDATA section node
5	entity reference node
7	processing instruction node
8	comment node

9	document
10	document type

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	NODE	A6	[bin]
SD	NODE-TYPE	N5	

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
–40	The procedure did not find the XML document.
–41	The specified node is not a valid node.

GET_NODE_VALUE

Returns the value of a node. The node value of text, comment, cdata section, and processing instruction nodes is their text content.

The node value of an attribute node is returned as follows:

- If the attribute node only contains one text node, the value of that text node is returned as the node value.
- If the attribute node contains multiple text and entity reference nodes, a no-op result is returned. The application should then check for children of the attribute node by using a procedure such as the GET_CHILD_NODES procedure.
- All other nodes have null as their value, which is indicated by returning a no-op result (zero) and a zero for the value length.

Syntax

```
 INTEGER PROCEDURE GET_NODE_VALUE
                      (DOC_TAG, NODE, NODE_VALUE, VALUE_START,
VALUE_LENGTH);
      INTEGER           DOC_TAG, NODE,             VALUE_START, VALUE_LENGTH;
      EBCDIC ARRAY                      NODE_VALUE [0];

INTEGER PROCEDURE getNodeValue
                      (DOC_TAG, NODE, NODE_VALUE, VALUE_START,
VALUE_LENGTH);
      VALUE             DOC_TAG, NODE,             VALUE_START;
```

```
      INTEGER           DOC_TAG, NODE,             VALUE_START, VALUE_LENGTH;
      EBCDIC ARRAY                      NODE_VALUE [*];

PROCEDURE GET-NODE-VALUE (GLB_PARAM);
   EBCDIC ARRAY           GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

NODE identifies the node the value of which the procedure returns.

NODE_VALUE is the returned node value in the application character set.

- The node value of a text, comment, CDATA section, or processing instruction node is the text in the node.
- If an attribute node contains only one text node, the procedure returns the text in the node.
- If an attribute node contains multiple text and entity reference nodes, the procedure returns a no-op result (zero). You can use a procedure such as the GET_CHILD_NODES procedure in an application to check for children of the attribute node.
- The value of all other nodes is null. For these nodes, the procedure returns a no-op result and a zero for the VALUE_LENGTH parameter.

VALUE_START is a zero-based offset into the NODE_VALUE parameter and specifies where the node value is returned.

VALUE_LENGTH is the length of the NODE_VALUE parameter.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	NODE	A6	[bin]
SD	NODE-VALUE-SIZE	N5	NODE-VALUE size, for example, 2048
SD	NODE-VALUE	A*n*	[longa]
SD	VALUE-START	N5	
SD	VALUE-LEN	N5	

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	No-op. The attribute node has more than one child node, or the specified node is a type that does not have a value.
–40	The procedure did not find the XML document.
–41	The specified node is not a valid node.

GET_PARENT_NODE

Returns the node that is the parent of the specified node. If the specified node does not have a parent, the procedure returns a no-op result.

Syntax

```
INTEGER PROCEDURE GET_PARENT_NODE
                              (DOC_TAG, NODE, PARENT);
     INTEGER                   DOC_TAG, NODE, PARENT;

INTEGER PROCEDURE getParentNode
                              (DOC_TAG, NODE, PARENT);
      VALUE                    DOC_TAG, NODE;
      INTEGER                  DOC_TAG, NODE, PARENT;

PROCEDURE GET-PARENT-NODE     (GLB_PARAM);
   EBCDIC ARRAY                GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

NODE identifies the node the parent of which the procedure returns.

PARENT is the parent of the node that the NODE parameter specifies.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	NODE	A6	[bin]
SD	PARENT	A6	[bin]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	No-op. The specified node does not have a parent node.
-40	The procedure did not find the XML document.
-41	The specified node is not a valid node.

GET_PREVIOUS_SIBLING

Returns the sibling node that immediately precedes the specified node. If no sibling node precedes the specified node, the procedure returns a no-op result.

Syntax

```
INTEGER PROCEDURE GET_PREVIOUS_SIBLING
                                 (DOC_TAG, NODE, PREVIOUS);
    INTEGER                       DOC_TAG, NODE, PREVIOUS;

INTEGER PROCEDURE getPreviousSibling
                                 (DOC_TAG, NODE, PREVIOUS);
  VALUE                           DOC_TAG, NODE;
  INTEGER                         DOC_TAG, NODE, PREVIOUS;

PROCEDURE GET-PREVIOUS-SIBLING    (GLB_PARAM);
   EBCDIC ARRAY                   GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

NODE identifies the node that follows the returned node.

PREVIOUS is the previous sibling of the node that the NODE parameter identifies.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	NODE	A6	[bin]
SD	PREVIOUS	A6	[bin]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	No-op. No node precedes the specified node, or the specified node is the document node.
-40	The procedure did not find the XML document.
-41	The specified node is not a valid node.

GET_XML_DOCUMENT

Retrieves the XML document, which the application might have modified, but does not release the document from WEBAPPSUPPORT memory.

Based on the setting of the NAMESPACE_PROCESSING option setting in the SET_XML_OPTION procedure namespace information is put into the generated XML document.

If the XML declaration for the XML document specifies a character encoding string that the procedure can identify, the procedure uses the specified encoding to encode the content of the document. If the XML declaration does not specify an encoding string, the procedure uses UTF-8 encoding for the document. If the XML declaration specifies an encoding string that the procedure cannot identify, the procedure returns an error.

See the topic "Specifying the Document Character Set" for information about specifying the character set for the document.

If the document has element and attribute nodes with names that include namespace prefixes, the generated XML document contains the prefixes.

If the application identifies a file as the destination of the document, the procedure creates a stream file with the following attributes:

```
BLOCKSTRUCTURE   = FIXED
EXTMODE          = ASCII
FILEORGANIZATION = NOTRESTRICTED
FILESTRUCTURE    = STREAM
FILETYPE         = DATA
FRAMESIZE        = 8
MAXRECSIZE       = 1
MINRECSIZE       = 1
SECURITYTYPE     = PRIVATE
SECURITYUSE      = IO
```

You can override the preceding attribute values by setting attributes in the FILE_ATTRIBUTES option of the SET_XML_OPTION procedure.

The application can identify a permanent directory as the document destination, if the directory is part of a directory structure that already exists.

Syntax

```
INTEGER PROCEDURE GET_XML_DOCUMENT
               (DOC_TAG, DEST_TYPE, OUT_FORMAT, DEST, DEST_START, DEST_LEN
);
     INTEGER    DOC_TAG, DEST_TYPE, OUT_FORMAT,       DEST_START, DEST_LEN;
     EBCDIC ARRAY                                DEST [0];

INTEGER PROCEDURE getXMLDocument
               (DOC_TAG, DEST_TYPE, OUT_FORMAT, DEST, DEST_START, DEST_LEN
);
     VALUE      DOC_TAG, DEST_TYPE, OUT_FORMAT,       DEST_START;
     INTEGER    DOC_TAG, DEST_TYPE, OUT_FORMAT,       DEST_START, DEST_LEN;
     EBCDIC ARRAY                                DEST [*];

PROCEDURE GET-XML-DOCUMENT (GLB_PARAM);
   EBCDIC ARRAY             GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

DEST_TYPE identifies the type of destination for the document and can be either of the following values:

- 1= When the procedure returns, the DEST parameter contains the XML document. The maximum size of an XML document returned is 134 MB.
- 2= When the application calls the procedure, the DEST parameter contains the MCP file name to which the procedure writes the XML document. The file name can be in display format or pathname format. See the FILENAME_FORMAT option in the SET_XML_OPTION procedure. If the file exists before the application calls the procedure, the procedure overwrites the file.

OUT_FORMAT identifies the output format of the XML document and can be either of the following values:

- 1= A carriage return and a line feed are at the end of each nontext node. Each line is indented the number of spaces that the INDENT option in the SET_XML_OPTION procedure specifies.
- 2= No carriage return, line feed, or white space is between nodes.
- 3= Canonical format. See the SET_XML_OPTION procedure, CANONICAL_METHOD option.

DEST is the array containing destination information.

If the DEST_TYPE parameter is 1, DEST contains the XML document.

If the DEST_TYPE parameter is 2, DEST identifies the name of the MCP file that contains the document, and this name is in the application character set. The SET_TRANSLATION procedure sets the application character set.

DEST_START is a zero-based offset into DEST and indicates where the procedure returns the XML document.

DEST_LEN, in the procedure return, specifies the length in bytes of the XML document. If the DEST_TYPE parameter is 2, then DEST_LEN, in the procedure call, specifies the length in bytes of the DEST parameter. If DEST_LEN is zero, DEST contains a string that is terminated by blanks or a null byte.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	DEST-TYPE	N5	
SD	OUT-FORMAT	N5	
SD	DEST-SIZE	N5	DEST size, for example, 2048
SD	DEST	A*n*	[longa]
SD	DEST-START	N5	
SD	DEST-LEN	N12	

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	No-op, for any of the following reasons: • The destination length is invalid. • The WEBAPPSUPPORT library does not support the destination type. • The WEBAPPSUPPORT library does not support the output format. • The document does not have children.
-13	An attribute error occurred while setting the file name.
–25	The procedure could not write to the file.
–40	The procedure did not find the XML document.
–46	The WEBAPPSUPPORT library does not support the XML document encoding.
–64	Maximum exceeded.

GET_XML_RECORD

Retrieves data from the XML document into an application's record structure. The application specifies in this call the layout of the record structure and the data to be retrieved from the XML document.

Data is the text node that is the first child of the element. If you map data to a number, and the number is invalid, zero is placed in the record field.

Note: *The mapping of XML data into a record structure may not work for all XML documents. This procedure is designed for simple XML documents with relatively flat structures.*

Syntax

```
INTEGER PROCEDURE GET_XML_RECORD
                     (DOC_TAG, NODE, MAPPING, RECORD);
   INTEGER            DOC_TAG, NODE;
   EBCDIC ARRAY                       MAPPING, RECORD [0];

INTEGER PROCEDURE getXMLRecord
                     (DOC_TAG, NODE, MAPPING, RECORD);
   VALUE              DOC_TAG, NODE;
   INTEGER            DOC_TAG, NODE;
   EBCDIC ARRAY                       MAPPING, RECORD [*];

PROCEDURE GET-XML-RECORD (GLB_PARAM);
   EBCDIC ARRAY           GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

NODE identifies the node under which the procedure searches. It must be the document node or an element.

MAPPING is a structured layout that identifies the XML elements containing data that should be placed into RECORD. See “XML Mapping Structure” for more information.

RECORD is the parameter that receives the mapped data.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	NODE	A6	[bin]
SD	MAPPING-SIZE	N5	MAPPING size, for example, 256
SD	MAPPING	A*n*	[longa]
SD	RECORD-SIZE	N5	RECORD size, for example, 2048
SD	RECORD	A*n*	[longa]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	No-op, no matching data was found, or the NODE parameter is not an element or the document node.
-91	The MAPPING parameter contains unsupported constructs.

HAS_ATTRIBUTE

Determines whether the element node has the specified attribute.

Attributes names are case-sensitive.

Syntax

```
INTEGER PROCEDURE HAS_ATTRIBUTE
                                (DOC_TAG, NODE, ATTR_NAME);
     INTEGER                     DOC_TAG, NODE;
     EBCDIC ARRAY                                ATTR_NAME [0];

INTEGER PROCEDURE hasAttribute
                                (DOC_TAG, NODE, ATTR_NAME);
     VALUE                       DOC_TAG, NODE;
     INTEGER                     DOC_TAG, NODE;
     EBCDIC ARRAY                                ATTR_NAME [*];

PROCEDURE HAS-ATTRIBUTE         (GLB_PARAM);
   EBCDIC ARRAY                  GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

NODE identifies the element node.

ATTR_NAME is the attribute name in the application character set. If ATTR_NAME is a local name without a namespace prefix, then the procedure returns the first attribute with the name, which might be a qualified name with prefixes. If ATTR_NAME is a qualified name with a namespace prefix, then the procedure returns the first attribute with the qualified name that includes the namespace prefix.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	NODE	A6	[bin]
SD	ATTR-NAME-SIZE	N5	ATTR-NAME size, for example, 256
SD	ATTR-NAME	A*n*	[longa]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	No-op. The specified node is not an element node.
-35	The procedure call did not specify a field.
-40	The procedure did not find the XML document.
-41	The specified node is not a valid node.

INSERT_CHILD_BEFORE

Inserts a child node and the tree of which the child is the root under the specified parent node and immediately before another child.

If the application specifies null (-1) for the other child, then the child to be inserted becomes the last subtree of the parent. You can also use the APPEND_CHILD procedure in the application to do that.

If the child to be inserted is already a child of another node, the procedure detaches this child before attaching this child to the new parent.

If the other child is not a child of the parent, the procedure returns a no-op result.

You cannot use this procedure to append attributes to elements.

Syntax

```
INTEGER PROCEDURE INSERT_CHILD_BEFORE
                     (DOC_TAG, PARENT, REF_CHILD, NEW_CHILD);
     INTEGER          DOC_TAG, PARENT, REF_CHILD, NEW_CHILD;

INTEGER PROCEDURE insertChildBefore
                      (DOC_TAG, PARENT, REF_CHILD, NEW_CHILD);
     VALUE             DOC_TAG, PARENT, REF_CHILD, NEW_CHILD;
     INTEGER           DOC_TAG, PARENT, REF_CHILD, NEW_CHILD;

PROCEDURE INSERT-CHILD-BEFORE (GLB_PARAM);
   EBCDIC ARRAY                GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

PARENT identifies the parent node.

REF_CHILD identifies the child node before which the procedure inserts a child.

NEW_CHILD identifies the child node to insert.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	PARENT	A6	[bin]
SD	REF-CHILD	A6	[bin]
SD	NEW-CHILD	A6	[bin]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	No-op. The child before which the procedure is to insert a child is not a child of the parent, or the child to insert is an attribute node.
–40	The procedure did not find the XML document.
–41	A specified node is not a valid node.
–42	The specified parent node is not a parent.
–43	The procedure cannot attach this node to the parent.
–44	The document already has an element.
–45	The document already has a DTD.

PARSE_JSON_TO_XML

Converts JSON text to a parsed XML document stored in the WEBAPPSUPPORT library.

See also the procedure CONVERT_JSON_TO_XML_DOCUMENT.

Syntax

```
INTEGER PROCEDURE PARSE_JSON_TO_XML                    (SOURCE_TYPE, SOURCE,
SOURCE_START, SOURCE_LEN,                                 DOC_TAG, NODE);
INTEGER         SOURCE_TYPE,         SOURCE_START, SOURCE_LEN,
            DOC_TAG, NODE;   EBCDIC ARRAY                 SOURCE [0];

INTEGER PROCEDURE parseJSONtoXML                     (SOURCE_TYPE, SOURCE,
SOURCE_START, SOURCE_LEN,                                 DOC_TAG, NODE);
VALUE           SOURCE_TYPE,         SOURCE_START, SOURCE_LEN;       INTEGER
SOURCE_TYPE,         SOURCE_START, SOURCE_LEN,
  DOC_TAG, NODE;       EBCDIC ARRAY                 SOURCE [*];

PROCEDURE PARSE-JSON-TO-XML (GLB_PARAM);
   EBCDIC ARRAY              GLB_PARAM [0];
```

Parameters

SOURCE_TYPE identifies the type of source for the XML document.

If the value is 1, the SOURCE parameter contains the XML document to be parsed.

If the value is 2, the SOURCE parameter contains the MCP file name of the JSON text to be parsed. The name is in display format or pathname format. See the FILENAME_FORMAT option in the SET_XML_OPTION procedure.

If the value is 3, the SOURCE parameter contains an HTTP URL or JPM server file system identifier that identifies the JSON document to be parsed.

SOURCE is the array containing source information. If the SOURCE_TYPE parameter is 2 or 3, SOURCE is coded in the application character set. If the SOURCE_TYPE parameter is 1, the document is coded in UTF-8.

SOURCE_START is a zero-based offset into the SOURCE parameter and indicates where the supplied information starts.

SOURCE_LEN is the length in bytes of the data in the SOURCE parameter. If zero, SOURCE contains a string that is terminated by blanks or a null byte.

DOC_TAG can have a value in the procedure call or return as follows:

- In the procedure call, the application can set DOC_TAG to either of the following:
  - 0 (zero) to indicate that the application does not currently have a document parsed or to request that a currently parsed document not be released
  - A document tag to release the document before parsing the new document

- In the procedure return, if the procedure can parse the document DOC_TAG identifies the parsed document.

NODE is the returned document node.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	SOURCE-TYPE	N5	
SD	SOURCE-SIZE	N5	SOURCE size, for example, 10000
SD	SOURCE	A*n*	[longa]
SD	SOURCE-START	N5	
SD	SOURCE-LEN	N5	
SD	DOC-TAG	A6	[bin]
SD	NODE	A6	[bin]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	No-op. WEBAPPSUPPORT already has the maximum number of XML documents.
-11	The input file was not found or is not available.
-12	The input file was too long to be processed.
-13	An attribute error occurred while setting the file name.
-14	An I/O error occurred while reading the input file.
-16	A file character set was not available. The CENTRALSUPPORT library and the CCSFILE data file installed on the system do not support the EXTMODE value for the file.
-47	The SOURCE_START offset was invalid, or the SOURCE_TYPE value is not supported.
-48	The procedure cannot open a socket to JPM.
-49	The procedure cannot write to the JPM.
-50	The procedure cannot read from the JPM.
-51	One or more parsing errors occurred.
-52	The URL in the SOURCE parameter is not available.
-54	JPM is not configured.
-56	The procedure cannot create another node because the maximum number of nodes already exists.

PARSE_XML_DOCUMENT

Parses the XML document. After the document is parsed, the application can access or modify the document.

The application character set when the document is created in WEBAPPSUPPORT is the character set that any application must use with the document. The SET_TRANSLATION procedure sets this character set.

See the SET_XML_OPTION procedure, (PRESERVE_WHITESPACE) option, for control over the whitespace when parsing an XML document.

XML Input Files

The PARSE_XML_DOCUMENT procedure can parse XML documents in stream files.

The procedure sets the DEPENDENTINTMODE of the input file to TRUE. Consequently, the procedure does not translate the file contents into the document character set, which must be an ASCII-based character set. The XML document must be encoded in the document character set.

Syntax

```
INTEGER PROCEDURE PARSE_XML_DOCUMENT
                      (SOURCE_TYPE, SOURCE, SOURCE_START, SOURCE_LEN,
                           DOC_TAG, NODE);
     INTEGER           SOURCE_TYPE,         SOURCE_START, SOURCE_LEN,
                           DOC_TAG, NODE;
     EBCDIC ARRAY                   SOURCE [0];

INTEGER PROCEDURE parseXMLDocument
                      (SOURCE_TYPE, SOURCE, SOURCE_START, SOURCE_LEN,
                           DOC_TAG, NODE);
     VALUE             SOURCE_TYPE,         SOURCE_START, SOURCE_LEN;
     INTEGER           SOURCE_TYPE,         SOURCE_START, SOURCE_LEN,
                           DOC_TAG, NODE;
     EBCDIC ARRAY                   SOURCE [*];

PROCEDURE PARSE-XML-DOCUMENT (GLB_PARAM);
   EBCDIC ARRAY               GLB_PARAM [0];
```

Parameters

SOURCE_TYPE identifies the type of source for the XML document.

If the value is 1, the SOURCE parameter contains the XML document to be parsed.

If the value is 2, the SOURCE parameter contains the MCP file name of the XML document to be parsed. The name is in display format or pathname format. See the FILENAME_FORMAT option in the SET_XML_OPTION procedure.

If the value is 3, the SOURCE parameter contains an HTTP URL or JPM server file system identifier that identifies the XML document to be parsed.

SOURCE is the array containing source information. If the SOURCE_TYPE parameter is 2 or 3, SOURCE is coded in the application character set. If the SOURCE_TYPE parameter is 1, the document is coded in the document character set. The SET_TRANSLATION procedure sets the application character set.

The XML document must be encoded in an ASCII-based character set, for example, us-ascii, UTF-8 or iso-8859-1.

SOURCE_START is a zero-based offset into the SOURCE parameter and indicates where the supplied information starts.

SOURCE_LEN is the length in bytes of the data in the SOURCE parameter. If zero, SOURCE contains a string that is terminated by blanks or a null byte.

DOC_TAG can have a value in the procedure call or return:

- In the procedure call, the application can set DOC_TAG to either of the following:
  - 0 (zero), to indicate that the application does not currently have a document parsed or to request that a currently parsed document not be released
  - A document tag to release the document before parsing the new document
- In the procedure return, DOC_TAG identifies the parsed document, if the procedure can parse the document.

NODE is the returned document node.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	SOURCE-TYPE	N5	
SD	SOURCE-SIZE	N5	SOURCE size, for example, 10000
SD	SOURCE	A*n*	[longa]
SD	SOURCE-START	N5	
SD	SOURCE-LEN	N5	
SD	DOC-TAG	A6	[bin]
SD	NODE	A6	[bin]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	No-op. WEBAPPSUPPORT already has the maximum number of XML documents.
-11	The input file was not found or not available.
-12	The input file was too long to be processed.
-13	An attribute error occurred while setting file name.
-14	An I/O error occurred while reading the input file.
-16	A file character set was not available. The CENTRALSUPPORT library and the CCSFILE data file installed on the system do not support the EXTMODE value for the file.
-47	The SOURCE_START offset was invalid, or the SOURCE_TYPE value is not supported.
-48	The procedure cannot open a socket to JPM.
-49	The procedure cannot write to the JPM.
-50	The procedure cannot read from the JPM.
-51	One or more parsing errors occurred.
-52	The URL in the SOURCE parameter is not available.
-54	JPM not configured.
-56	The procedure cannot create another node because the maximum number of nodes already exists.

RELEASE_XML_DOCUMENT

Releases the XML document from WEBAPPSUPPORT memory. After this procedure completes, the application cannot use the document.

Syntax

```
INTEGER PROCEDURE RELEASE_XML_DOCUMENT
                                      (DOC_TAG);
     INTEGER                           DOC_TAG;

INTEGER PROCEDURE releaseXMLDocument
                                      (DOC_TAG);
     INTEGER                           DOC_TAG;

PROCEDURE RELEASE_XML_DOCUMENT        (GLB_PARAM);
   EBCDIC ARRAY                        GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document. If the procedure successfully releases the document, the procedure returns the DOC_TAG parameter with the value 0 (zero).

GLB_PARAM has the following format:

Format		Notes
SG-GLB-PARAM GROUP		
SG-PARAM GROUP		
SD RESULT	S5	
SD DOC-TAG	A6	[bin]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	DOC_TAG does not represent a document, or the document was already released.
1	The procedure released the document.

REMOVE_NODE

Removes the node and the children of the node, if the node has children, from the document. After this procedure completes, applications cannot use the node.

If the node is the document node or not a valid node, the procedure returns a no-op result.

Syntax

```
INTEGER PROCEDURE REMOVE_NODE
                                    (DOC_TAG, NODE);
     INTEGER                         DOC_TAG, NODE;

INTEGER PROCEDURE removeNode
                                    (DOC_TAG, NODE);
     VALUE                           DOC_TAG, NODE;
     INTEGER                         DOC_TAG, NODE;

PROCEDURE REMOVE_NODE               (GLB_PARAM);
   EBCDIC ARRAY                      GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

NODE identifies the node to be removed. The procedure sets this parameter to 0 (zero) if the procedure successfully removes the node.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	NODE	A6	[bin]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	No-op. The node is the document node or not a valid node.
–40	The procedure did not find the XML document.

SET_ATTRIBUTE

Adds or updates an attribute in a specified element node. The attribute value supplied must be a simple text string. If the attribute value is null, the procedure removes the attribute from the node.

If the attribute value is to contain text and entity reference nodes, code the application so that it does the following:

1. Uses the CREATE_ATTRIBUTE_NODE procedure to create the attribute node
2. Uses the CREATE_TEXT_NODE procedure to create the text nodes
3. Uses the CREATE_ENTITYREF_NODE procedure to create the entity reference nodes
4. Uses the APPEND_CHILD or INSERT_CHILD_BEFORE procedure to attach the text and entity reference nodes to the attribute node
5. Uses the APPEND_CHILD procedure to attach the attribute node to the element node

Syntax

```
INTEGER PROCEDURE SET_ATTRIBUTE
                        (DOC_TAG, NODE, NAMESPACE, ATTR_NAME, ATTR_VALUE);
  INTEGER                DOC_TAG, NODE;
  EBCDIC ARRAY                            NAMESPACE, ATTR_NAME, ATTR_VALUE
[0];

INTEGER PROCEDURE setAttribute
                        (DOC_TAG, NODE, NAMESPACE, ATTR_NAME, ATTR_VALUE);
  VALUE                  DOC_TAG, NODE;
  INTEGER                DOC_TAG, NODE;
  EBCDIC ARRAY                            NAMESPACE, ATTR_NAME, ATTR_VALUE
[*];

PROCEDURE SET-ATTRIBUTE  (GLB_PARAM);
   EBCDIC ARRAY           GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

NODE identifies the element node.

NAMESPACE is the attribute namespace, as a Uniform Resource Identifier (URI, in the application character set. An example of a NAMESPACE value is

```
http://somedomain/mynamespace
```

ATTR_NAME is the qualified name of the attribute. The name is in the application character set and case-sensitive. If the name is specified with prefix text before a colon (:), the prefix is a namespace prefix. The procedure does not validate the prefix against an actual namespace declaration in an element that encloses the node.

ATTR_VALUE is the attribute value in the application character set.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	NODE	A6	[bin]
SD	NAMESPACE-SIZE	N5	NAMESPACE size, for example, 256
SD	NAMESPACE	A*n*	[longa]
SD	ATTR-NAME-SIZE	N5	ATTR-NAME size, for example, 256
SD	ATTR-NAME	A*n*	[longa]
SD	ATTR-VALUE-SIZE	N5	ATTR-VALUE size, for example, 256
SD	ATTR-VALUE	A*n*	[longa]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	No-op. The specified node is not an element node, or the procedure did not find the attribute.
–35	The procedure call did not specify a field.
–40	The procedure did not find the XML document.
–41	The element node is not a valid node.

SET_NODE_VALUE

Sets the value of a node. The value can replace the current value.

The format of the value that the application supplies depends on the node type. For a description of the value format for each node type, see the topics for following procedures:

- CREATE_CDATA_NODE
- CREATE_COMMENT_ NODE
- CREATE_DOCTYPE_NODE
- CREATE_PI_NODE
- CREATE_TEXT_NODE

If the node is a type that does not define a value or if the value length is less than 1, the procedure returns a no-op result.

The SET_NODE_VALUE procedure cannot set an attribute value. To set an attribute value of an element node, use one of the following:

- The SET_ATTRIBUTE procedure
- The CREATE_ATTRIBUTE_NODE procedure to create an attribute node
- The CREATE_TEXT_NODE procedure, CREATE_ENTITYREF_NODE procedure, or both procedures to create the attribute value
- The APPEND_CHILD or INSERT_CHILD_BEFORE procedure to attach the new attribute value node or nodes to the attribute
- The APPEND_CHILD or INSERT_CHILD_BEFORE procedure to attach the attribute node to an element node

Syntax

```
INTEGER PROCEDURE SET_NODE_VALUE
                            (DOC_TAG, NODE, NODE_VALUE, VALUE_LENGTH);
     INTEGER                 DOC_TAG, NODE,             VALUE_LENGTH;
     EBCDIC ARRAY                            NODE_VALUE [0];

INTEGER PROCEDURE setNodeValue
                            (DOC_TAG, NODE, NODE_VALUE, VALUE_LENGTH);
     VALUE                   DOC_TAG, NODE;
     INTEGER                 DOC_TAG, NODE,              VALUE_LENGTH;
     EBCDIC ARRAY                             NODE_VALUE [*];

PROCEDURE SET-NODE-VALUE     (GLB_PARAM);
   EBCDIC ARRAY               GLB_PARAM [0];
```

Parameters

DOC_TAG identifies the XML document.

NODE identifies the node.

NODE_VALUE is the node value in the application character set.

VALUE_LENGTH is the length of the NODE_VALUE parameter.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	DOC-TAG	A6	[bin]
SD	NODE	A6	[bin]
SD	NODE-VALUE-SIZE	N5	NODE-VALUE size, for example, 2048
SD	NODE-VALUE	A*n*	[longa]
SD	VALUE-LENGTH	N5	

Possible Results

In addition to the standard results, these possible values can be returned.

Value	Description
0	No-op. The procedure did not supply a value, or the specified node type does not have a value that an application can set.
–40	The procedure did not find the XML document.
–41	The specified node is not a valid node.

SET_XML_OPTION

Sets options for processing of XML documents for the application.

Syntax

```
INTEGER PROCEDURE SET_XML_OPTION
                                (OPTION, OPTION_VALUE, OPTION_STRING);
  INTEGER                        OPTION, OPTION_VALUE;
  EBCDIC ARRAY                                          OPTION_STRING [0];

INTEGER PROCEDURE setXMLOption
                                (OPTION, OPTION_VALUE, OPTION_STRING);
  VALUE                          OPTION, OPTION_VALUE;
  INTEGER                        OPTION, OPTION_VALUE;
  EBCDIC ARRAY                                          OPTION_STRING [*];

PROCEDURE SET-XML-OPTION        (GLB_PARAM);
   EBCDIC ARRAY                  GLB_PARAM [0];
```

Parameters

OPTION is the option being set. The following options are supported.

1 (VALIDATE)

Specifies whether or not the JPM validates the XML document against a DTD or schema when parsing. If the JPM finds an error in the document, the XML Parser returns an error to the application.

If the value is 0, this option does not validate the document. This value is the default.

If the value is 1, this option validates the document. The document must specify a DTD or schema.

2 (NAMESPACE_PROCESSING)

Controls how the XML Parser handles namespaces.

If the value is 1, this option returns namespace prefixes with element and attribute names. This value is the default.

If the value is 2, this option returns namespace URLs as replacements for prefixes in element and attribute names.

If the value is 3, this option removes namespace prefixes from returned element and attribute names.

3 (EXPAND_ENTITY_REFERENCE)

Specifies whether the XML Parser replaces entity references with entities in a parsed document for an application.

If the value is 0, this option does not replace general entities. This value is the default.

If the value is 1, this option replaces general entities.

4 (EXTERNAL_GENERAL_ENTITIES)

Specifies whether the XML Parser places external general entities to be parsed in the entity reference tree.

If the value is 0, this option does not put external general entities in the tree.

If the value is 1, this option puts general entities in the tree. This value is the default.

5 (LOCK_DOCUMENT)

Specifies whether access to documents is locked.

If the value is 0, this option does not lock access to XML documents that are created or parsed after this procedure is executed. This value is the default.

If the value is 1, this option locks access to XML documents that are created or parsed after this procedure is executed. This value is needed if an application might access a document after another application changes the document.

6 (SCHEMA_SUPPORT)

Specifies whether the XML Parser uses schemas to validate XML document and to define entities.

If the value is 0, the XML Parser does not support schemas.

If the value is 1, the XML Parser supports schemas. This value is the default.

7 (SCHEMA_FULL_CHECKING)

Specifies whether the XML Parser supports full schema constraint checking.

If the value is 0, the XML Parser does not support full schema constraint checking. This value is the default.

If the value is 1, the XML Parser supports full schema constraint checking.

8 (SCHEMA_LOCATION)

Specifies the schema location in the application character set. The default for OPTION_STRING is the null string.

If the schema location includes a namespace, the OPTION_STRING parameter must specify the namespace URL and the schema file, separated by whitespace. For example, the value of OPTION_STRING could be

http://library.org/library.xsd

The application does not the use OPTION_VALUE parameter.

9 (SCHEMA__LOCATION_TYPE)

Specifies whether the schema location includes a namespace.

If the value is 0, the schema location does not include a namespace. This value is the default.

If the value is 1, the schema location includes a namespace.

10 (FILENAME_FORMAT)

Specifies the format that the XML Parser uses for file names that applications pass to the XML Parser. The SET_OPTION procedure, FILENAME_FORMAT (1) option has the same value as this option.

If the value is 0 (LTITLE), the XML Parser uses the value NATIVE for the SEARCHRULE file attribute. This value is the default.

If the value is 1 (PATHNAME), the XML Parser uses the value POSIX for the SEARCHRULE file attribute.

11 (FILE_ATTRIBUTES)

Contains a comma-separated list of file attribute settings in the application character set. The default for OPTION_STRING is a null string.

For example, the OPTION_STRING parameter can be

```
SECURITYTYPE=PUBLIC, SECURITYUSE=IN
```

The procedure does not use the OPTION_VALUE parameter.

12 (INDENT)

Specifies the number of space characters to prefix each level in the output of GET_XML_DOCUMENT when the OUT_FORMAT parameter is 1.

Specifies the number of space characters to add for each level of indentation for JSON procedures. Zero means no whitespace is added to the JSON output and produces the most compact representation.

The default value is 2, the minimum value is 0, and the maximum value is 10.

13 (LOGGING)

Controls JPM logging for events generated for this application. Each logging level from 1 to 5 includes the logging for the levels identified by higher numbers. For example, level 1 includes the logging for levels 2 to 5.

If the value is 0, logging uses the JPM logging level setting. This value is the default.

If the value is 1 (Debug), logging provides detailed tracing that you can submit for a JPM problem to Unisys.

If the value is 2 (Info), logging provides basic tracing, not the detailed tracing that debug logging provides.

If the value is 3 (Warn), logging logs warnings that the parser generates.

If the value is 4 (Error), logging logs errors that the parser generates.

If the value is 5 (Fatal), logging logs fatal errors which the parser generates and which prevent document parsing.

If the value is 6 (Off), no logging is in place.

14 (CANONICAL_METHOD)

Controls the method by which XML is serialized, according to the Canonical XML Version 1.0 W3C recommendation. This option affects the procedures GET_XML_DOCUMENT, CONVERT_JSON_TO_XML_DOCUMENT, ENCRYPT_DATA_TO_XML, and ENCRYPT_XML_DOCUMENT.

If the value is 1, XML is serialized using Inclusive Canonicalization, which includes the in scope namespace and xml namespace attribute context from ancestors of the XML being serialized, with comments removed.

If the value is 2, XML is serialized the same way when the value = 1, with comments included.

If the value is 3, XML is serialized using the Exclusive Canonicalization, which includes to the minimum extent practical the namespace prefix binding and xml namespace attribute context inherited from ancestor elements, with comments removed.

If the value is 4, XML is serialized the same way when the value = 3, with comments included. This is the default value.

15 (PRESERVE_WHITESPACE)

Controls whether or not to preserve whitespace when parsing an XML document. This option affects the PARSE_XML_DOCUMENT procedure.

If the value is 0, whitespace in the XML document is not preserved. The GET_XML_DOCUMENT procedure returns an XML document with whitespace that might not match the whitespace in the original XML document. This is the default value.

If the value is 1, whitespace in the XML document is preserved. This value should be used for XML documents that are processed using XML Encryption or if the canonical format is used in the GET_XML_DOCUMENT procedure.

OPTION_STRING

The OPTION parameter descriptions explain how an application uses the OPTION_STRING parameter. If the value of OPTION does not require a value for OPTION_STRING, the application needs to set OPTION_STRING to the null string.

GLB_PARAM

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	OPTION	N5	
SD	OPTION-VALUE	N12	
SD	OPTION-STRING-SIZE	N5	OPTION-STRING size, for example, 256
SD	OPTION-STRING	A*n*	[longa]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	The application specified an option or value that the procedure does not support.
1	The procedure accepted all settings.

TRANSFORM_XML_DOCUMENT

Transforms an XML document into another document.

Syntax

```
INTEGER PROCEDURE TRANSFORM_XML_DOCUMENT
                   (XML_SOURCE_TYPE, XML_SOURCE,
                      XML_SOURCE_START, XML_SOURCE_LEN,
                        XSL_SOURCE_TYPE, XSL_SOURCE,
                          XSL_SOURCE_START, XSL_SOURCE_LEN,
                            DEST_TYPE, DEST, DEST_START, DEST_LEN);
  INTEGER           XML_SOURCE_TYPE,
                      XML_SOURCE_START, XML_SOURCE_LEN,
                        XSL_SOURCE_TYPE,
                          XSL_SOURCE_START, XSL_SOURCE_LEN,
                            DEST_TYPE,       DEST_START, DEST_LEN;
  EBCDIC ARRAY                            XML_SOURCE [0],
                                           XSL_SOURCE [0],
                                        DEST [0];

INTEGER PROCEDURE transformXMLDocument
                   (XML_SOURCE_TYPE, XML_SOURCE,
                      XML_SOURCE_START, XML_SOURCE_LEN,
                        XSL_SOURCE_TYPE, XSL_SOURCE,
                          XSL_SOURCE_START, XSL_SOURCE_LEN,
                            DEST_TYPE, DEST, DEST_START, DEST_LEN);
  VALUE             XML_SOURCE_TYPE,
                      XML_SOURCE_START, XML_SOURCE_LEN,
                        XSL_SOURCE_TYPE,
                          XSL_SOURCE_START, XSL_SOURCE_LEN,
                            DEST_TYPE,       DEST_START;
  INTEGER           XML_SOURCE_TYPE,
                      XML_SOURCE_START, XML_SOURCE_LEN,
                        XSL_SOURCE_TYPE,
                          XSL_SOURCE_START, XSL_SOURCE_LEN,
```

```
                              DEST_TYPE,       DEST_START, DEST_LEN;
  EBCDIC ARRAY                         XML_SOURCE [*],
                                           XSL_SOURCE [*],
                                         DEST [*];

PROCEDURE TRANSFORM-XML-DOCUMENT (GLB_PARAM);
   EBCDIC ARRAY                   GLB_PARAM [0];
```

Parameters

XML_SOURCE_TYPE identifies the type of source for the XML document and can be any of the following values.

If the value is 1, the XML_SOURCE parameter contains the XML document to be transformed.

If the value is 2, the XML_SOURCE parameter contains the MCP file name of the XML document to be transformed. The name is in display format or pathname format. The FILENAME_FORMAT option in the SET_OPTION procedure controls the format.

If the value is 3, the XML_SOURCE parameter contains an HTTP URL or JPM server file system identifier that identifies the XML document to be transformed.

XML_SOURCE is the array containing source information. If the XML_SOURCE_TYPE parameter is 1, XML_SOURCE is coded in the document is character set. If the XML_SOURCE_TYPE parameter is 2 or 3, XML_SOURCE is coded in the application is character set. The SET_TRANSLATION procedure sets the application character set.

The XML document must be encoded in an ASCII-based character set, for example us-ascii, UTF-8, or iso-8859-1.

XML_SOURCE_START is the zero-based offset into the XML_SOURCE parameter and indicates where the supplied information starts.

XML_SOURCE_LEN is the length in bytes of the data in the XML_SOURCE parameter. If zero, XML_SOURCE contains a string that is terminated by blanks or a null byte.

XSL_SOURCE_TYPE identifies the type of source for the XSL stylesheet and can be any of the following values.

If the value is 0, no XSL stylesheet is supplied (the XML document contains a reference to the stylesheet to be used for transforming the document).

If the value is 1, the XSL_SOURCE parameter contains the XSL stylesheet to be applied to the XML document.

If the value is 2, the XSL_SOURCE parameter contains the MCP file name of the XSL stylesheet to be applied to the XML document. The name is in display format or pathname format. The FILENAME_FORMAT option in the SET_OPTION procedure controls the format.

If the value is 3, the XSL_SOURCE parameter contains an HTTP URL or JPM server file system identifier that identifies the XSL stylesheet to be applied to the XML document.

The XSL stylesheet is an XML document and must be encoded in an ASCII-based character set, for example us-ascii, UTF-8, or iso-8859-1.

XSL_SOURCE_START is the zero-based offset in the XSL_SOURCE parameter and indicates where the supplied information starts.

XSL_SOURCE_LEN is the length in bytes of the data in the XSL_SOURCE parameter. This value must be zero if XSL_SOURCE_TYPE is zero. If XSL_SOURCE_TYPE is not zero and XSL_SOURCE_LEN is zero, XSL_SOURCE contains a string that is terminated by blanks or a null byte.

DEST_TYPE identifies the type of destination for the document and can be either of the following values.

If the value is 1, when the procedure returns, the DEST parameter contains the transformed document.

If the value is 2, when the application calls the procedure, the DEST parameter contains the MCP file name to which the procedure writes the transformed document. The file name can be in display format or pathname format. The FILENAME_FORMAT option in the SET_OPTION procedure controls the format. If the file exists before the application calls the procedure, the procedure overwrites the file.

DEST is the array containing destination information. If the DEST_TYPE parameter is 1, DEST contains the transformed document. If the DEST_TYPE parameter is 2, DEST identifies the name of the MCP file that contains the transformed document, and this name is in the application character set. The SET_TRANSLATION procedure sets the application character set.

DEST_START is the zero-based offset in DEST and indicates where the procedure returns the transformed document.

DEST_LEN in the procedure call specifies the length in bytes of the DEST parameter if DEST_TYPE is 2. When the procedure returns, DEST_LEN specifies the length in bytes of the transformed document.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	XML_SOURCE_TYPE	N5	
SD	XML-SOURCE-SIZE	N5	XML-SOURCE size, for example, 2048
SD	XML-SOURCE	A*n*	[longa]
SD	XML-SOURCE_START	N5	
SD	XML-SOURCE_LEN	N5	
SD	XSL-SOURCE_TYPE	N5	
SD	XSL-SOURCE-SIZE	N5	XSL-SOURCE size, for example, 2048
SD	XSL-SOURCE	A*n*	[longa]
SD	XSL-SOURCE_START	N5	
SD	XSL-SOURCE_LEN	N5	
SD	DEST-TYPE	N5	
SD	DEST-SIZE	N5	
SD	DEST	A*n*	DEST size, for example, 2048
SD	DEST-START	N5	[longa]
SD	DEST-LEN	N12	

Possible Result Values

In addition to the standard results, these possible values can be returned.

Values	Description
0	No-op, exceeded maximum number of XML documents in WEBAPPSUPPORT
-11	Input file not found or not available
-12	Input file too long to be processed
-13	Attribute error setting file name.
-14	I/O error reading input file
-16	File character set not available. The EXTMODE of the file used is not supported by the CENTRALSUPPORT and CCSFILE installed on the system.
-47	Source start/offset invalid, or source type not supported
-48	Cannot open socket to Java Parser Module
-49	Cannot write to Java Parser Module
-50	Cannot read from Java Parser Module
-51	Parsing error or errors
-52	URL not available
-54	JPM not configured
-56	Maximum nodes exceeded
-57	JPM too old
-58	Transformation error(s)

XML_ESCAPE

Converts predefined entities in an array of characters into the escaped versions of the entities. This procedure supports the entities listed in the following table:

Unescaped Entity	Escaped Version
& (ampersand)	&
< (less than)	<
> (greater than)	>
' (apostrophe)	'
" (quote)	"

Syntax

```
INTEGER PROCEDURE XML_ESCAPE
                    (CHARSET, SOURCE, SOURCE_START, SOURCE_LEN,
                         DEST, DEST_START, DEST_LEN);
     INTEGER          CHARSET,        SOURCE_START, SOURCE_LEN,
     EBCDIC ARRAY              SOURCE,
                         DEST [0];

INTEGER PROCEDURE xmlEscape
                    (CHARSET, SOURCE, SOURCE_START, SOURCE_LEN,
                         DEST, DEST_START, DEST_LEN);
    VALUE            CHARSET,         SOURCE_START, SOURCE_LEN,
                               DEST_START;
    INTEGER          CHARSET,         SOURCE_START, SOURCE_LEN,
                               DEST_START, DEST_LEN;
    EBCDIC ARRAY              SOURCE,
                         DEST [*];

PROCEDURE XML-ESCAPE (GLB_PARAM);
   EBCDIC ARRAY       GLB_PARAM [0];
```

Parameters

CHARSET is the application character set and can be 0 (EBCDIC) or 1 (ASCII).

SOURCE is the array containing the characters to be escaped.

SOURCE_START is a zero-based offset into the SOURCE array and indicates where the unescaped characters start.

SOURCE_LEN is the length of the data in SOURCE.

DEST is the array that receives the escaped characters.

DEST_START is a zero-based offset into the DEST array and indicates where the escaped characters start.

DEST_LEN is the length of data returned in DEST.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	CHARSET	N5	
SD	SOURCE-SIZE	N5	SOURCE size, for example, 2048
SD	SOURCE	A*n*	[longa]
SD	SOURCE-START	N5	
SD	SOURCE-LEN	N5	
SD	DEST-SIZE	N5	DEST size, for example, 2048
SD	DEST	A*n*	[longa]
SD	DEST-START	N5	
SD	DEST-LEN	N5	

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
-47	The value in the SOURCE_START or SOURCE_LEN parameter is invalid.

Section 7
Using Sample Source Code for Parsing an XML Document

The following sample fragments of code show basic calls to the XML Parser API procedures. For more complete working examples released with the XML Parser, see the files in the directory *SYSTEM/CCF/XMLPARSER/SAMPLE/=.

COBOL85 Code for Parsing an XML Document

```
77 SOURCE-ARRAY     PIC  9(11)   BINARY VALUE IS 1.
01 XMLDOC           PIC  X(1000).
77 START-AT-ZERO    PIC  9(11)   BINARY VALUE IS 0.
77 XML-LENGTH       PIC  9(11)   BINARY.
77 DOC-TAG           PIC  9(11)   BINARY VALUE IS 0.
77 DOC-NODE          PIC  9(11)   BINARY VALUE IS 0.
77 WEB-RESULT        PIC S9(11)   BINARY VALUE IS 0.
    88  WEB-OK               VALUE   1.

CALL "PARSE_XML_DOCUMENT OF WEBAPPSUPPORT"
  USING  SOURCE-ARRAY, XMLDOC, START-AT-ZERO, XML-LENGTH,
         DOC-TAG,      DOC-NODE
  GIVING WEB-RESULT.
```

The preceding procedure call has the following parameters:

- SOURCE-ARRAY indicates that the source for the XML document is the XMLDOC array.
- XMLDOC contains the XML document.
- START-AT-ZERO indicates that the document starts at the beginning of the XMLDOC array.
- XML-LENGTH is the length of the document in the XMLDOC parameter. The application assigned this value to the parameter before calling the procedure.
- DOC-TAG has the returned document tag that the application uses to reference the document in WEBAPPSUPPORT.
- DOC-NODE contains the reference to the document node.
- WEB-RESULT contains the result of the procedure call. The result is 1 if the call is successful.

ALGOL Code for Parsing an XML Document

```
INTEGER       xmlResult;
  DEFINE        sourceTypeArray = 1 #;
  EBCDIC  ARRAY xmlDocument [0:65535];
  INTEGER       xmlDocumentLen;

  INTEGER       DOC_TAG;
  INTEGER       documentNode;

 xmlResult := parseXMLDocument
                (   % indicate xml document is in parameter:
                 sourceTypeArray,
                    % the xml document:
                 xmlDocument,
                    % index into xmlDocument where doc starts:
                 0,


                    % document length in bytes, set by app:
                 xmlDocumentLen,
                    % tag that references the parsed document:
                 DOC_TAG,
                    % document node:
                 documentNode);
```

In the preceding procedure call, the xmlResult parameter receives the result of the procedure call.

Section 8
Monitoring the XML Parser

To monitor the status and usage of the XML Parser, do the following:

- Use the WEBAPPSUPPORT library STATUS command.
- Check the JPM log.

Using the WEBAPPSUPPORT Library STATUS Command

You can enter the WEBAPPSUPPORT library STATUS command as either of the following:

- An NA command, through the WEBPCM Protocol Converter Module (PCM)
- An ACCEPT (AX) command

The following is an example of the STATUS command as an NA command:

```
NA CCF WEBPCM WEBAPPSUPPORT STATUS
```

The preceding command returns a response like the following:

```
Unisys Corporation WEBAPPSUPPORT
Version 54.150.0035 Compiled 11/11/2009 @ 14:51
Connection To WEBPCM:  Linked
3 Callers Linked
XML Parser JPM1:
  Host 192.63.212.61, Port 51117
   0 Sockets Open
  Status:  Available
  Standby:  True
  Version:  54.150.0021
  Threads:  Current = 10, Min = 10, Max = 14
  Logging:  Level = Error, File = JPM1/logs/log.txt
  Documents Parsed/Transformed = 3
  JVM:
   Version:  1.6.0_07
   Free = 1 MB, Total = 5 MB, Max = 63 MB
```

The XML Parser section of the above response shows the following:

- Host

  This listing is the JPM host. This parameter is the HOST parameter in the JPM configuration file (*SYSTEM/CCF/WEBAPPSUPPORT/PARAMS/XML).

- Port

  This listing is the JPM port number. This parameter is the PORT parameter in the JPM configuration file (*SYSTEM/CCF/WEBAPPSUPPORT/PARAMS/XML).

- <number of sockets> Sockets Open

  This construct shows the number of TCP sockets currently open to the JPM.

- Status

  This construct shows the whether or not the JPM is available or unavailable.

- Standby

  If False, the JPM is an active JPM and will receive requests before standby JPMs. If True, the JPM is a standby JPM and will receive requests if no active JPMs are available.

- Version

  This construct shows the JPM software version.

- Threads
  - Current

    The number of currently active JPM threads
  - Min

    The configured minimum number of JPM threads
  - Max

    The configured maximum number of JPM threads

- Logging
  - Level

    The JPM logging level
  - File

    The JPM log file name

- Documents Parsed/Transformed

  This construct lists the number of documents that have been parsed or transformed.

- JVM
  - Version

    The JPM Java Virtual Machine version
  - Free = <number of MB>, Total = <number of MB>, and Max = <number of MB>

    The Free parameter specifies the amount of free memory for the JVM. The Total parameter specifies the amount of memory allocated to the JVM. The Max parameter specifies the maximum amount of memory that can be allocated to the JVM.

Checking the JPM Log

You can check the JPM log to monitor the JPM. The JPM log is a text file. The default file name is log.out, but the file can have a time-stamped name if the JPM is run on MCP Java and the STDOUT file is equated to a file name that has data and time stamps in it.

Section 9
HTTP Client Applications

Developing HTTP Client Applications

The Custom Connect Facility (CCF) WEBAPPSUPPORT library provides an API to COBOL85 and ALGOL applications allowing them to easily make HTTP requests and process the responses.

The SOCKETSUPPORT library provides TCP sockets (including SSL). The AUTHSUPPORT library supports NT LAN Manager (NTLM) processing.

This section provides information about objects, request handling, compression, security, and HTTP Client WEBAPPSUPPORT procedures. Also, some scenarios are included in this section to show how applications might use the HTTP Client capability. These scenarios are provided as examples only and are not considered to be complete.

Sample COBOL85 and ALGOL applications that demonstrate HTTP requests are released with the CCF.

Objects

When an application makes HTTP requests, four types of objects might be created by the application in the WEBAPPSUPPORT library: host objects, client objects, socket objects, and request objects.

The host object contains the following information:

- Hostname or IP address of the server
- A list of IP addresses if the server has multiple addresses
- The TCP port of the server

The client object contains the following information

- The settings for cookie handling
- Cookies that are remembered for hosts
- Credentials

The socket object contains the following information

- Socket attributes, including SSL
- Socket state

The request object contains the following information:

- HTTP method
- Uniform Resource Locator (URL)
- Query string and/or content data
- Request headers
- Response status, headers, and content

Request Handling

The topics in this subsection deal with several aspects of request handling: default request headers; tanking large data; the request header 100-continue; chunked content; synchronous and asynchronous requests; cookie handling; character set handling; compressed content; and security.

Default Request Headers

The following table lists the HTTP request headers that are sent by default. An application can override these headers by setting its own headers.

Header Name	Description
Accept-Encoding	The header value sent is identity
Authorization	If credentials that match the request have been stored in the client object, this header is sent with the request if the server has requested authentication.
Connection	The header value sent is Keep-Alive
Content-Length	If the SET_HTTP_REQUEST_CONTENT procedure has been called, the length in bytes of the data passed is set in this header.
Content-Type	If the request method is POST, by default this header value is set to: application/x-www-form-urlencoded If the request method is not POST, this header is not sent by default.
Cookie	If one or more Netscape-style or RFC2109-style cookies are set in the client object and match the request, this header is sent with the cookies.

Header Name	Description
Cookie2	If one or more RFC2965-style cookies are set in the client object and match the request, this header is sent with the cookies; otherwise, the following value, which tells the server that the application accepts RFC2965-style cookies, is sent: $Version=1
Host	The value set by the application for the HOST parameter of the CREATE_HTTP_HOST procedure is sent as the header value.
TE	The header value sent is chunked, identity, trailers Also, see HTTP RFC 2616.
User-Agent	See the USER_AGENT option of the SET_HTTP_OPTION procedure.

Tanking Large Data

WEBAPPSUPPORT tanks large data to temporary files on disk for the following capabilities:

- Large HTTP Client response content that exceeds 1,000,000 bytes
- Large HTTP Client request content that exceeds 1,000,000 bytes (for example, using the HTTP PUT method)

See "WEBAPPSUPPORT General Parameters File" in Section 3 for information about the TEMPFAMILY directive.

Request Header—Expect: 100-Continue

If the application sets the request header `Expect: 100-continue` on a request that contains content, the EXECUTE_HTTP_REQUEST procedure waits for a 100 (Continue) response from the server before sending the content.

Chunked Content

An application can supply the content in "chunks," which means that multiple calls to SET_HTTP_REQUEST_CONTENT append the content. Chunked content allows the application to send content that is dynamic in size and could come from multiple sources. Sending chunked content is useful when it is difficult to determine the total size of the content.

When using chunked content, the application should not set a Content-Length header. Also, the application should not prefix the content chunks with size indicators.

For a chunked content request, trailing headers are not supported. Normally, the request is sent with one block of content without using HTTP chunked transfer encoding.

The HTTP server can respond with chunked content and WEBAPPSUPPORT then concatenates the chunks into one block of content to give to the application. WEBAPPSUPPORT also adds trailing response headers to the headers received at the start of the response.

Synchronous and Asynchronous Requests

Applications can process requests synchronously, where the application stack does not return from the EXECUTE_HTTP_REQUEST procedure until the response processing is complete.

Applications can process requests asynchronously, where the application stack returns from the EXECUTE_HTTP_REQUEST procedure when the request is sent, and the application periodically checks for response completion.

Use the SYNCHRONOUS option in the SET_HTTP_OPTION procedure to control whether requests are processed synchronously (the default) or asynchronously.

Asynchronous requests free the application stack so it can perform other processing, such as making other HTTP requests. To determine that the request is complete, the application can call the GET_HTTP_RESPONSE_STATUS procedure.

Cookie Handling

Cookies are supported to make it easier for the application to retain state for HTTP servers. This functions similarly to how web browsers remember cookies for the length of their current session.

By default cookies set by a server are stored in the client object, and are re-sent in subsequent requests if the cookie's attributes match that of the request. Also, the cookie's expires or Max-Age setting is honored and the cookie is deleted from the client object when it is expired.

The application may also set or remove cookies in a client object with the SET_CLIENT_ATTR procedure. And if a cookie is set in a request with the SET_HTTP_REQUEST_HEADER procedure it will override the cookie of the same name that comes from the client object.

The stored cookie information is lost if the application frees a client object, de-links from WEBAPPSUPPORT, or the system halt/loads. The application can save cookies externally from WEBAPPSUPPORT (GET_HTTP_CLIENT_COOKIES), such as in a file or database, that need to persist across these events, and re-load them into the client object (SET_CLIENT_ATTR) when processing is restarted.

Although multiple cookie specifications with varying support by HTTP servers and other clients exist, only the following specifications are supported.

- The Netscape "Persistent Client State HTTP Cookies" preliminary specification. (http://curl.haxx.se/rfc/cookie_spec.html). This specification is the original specification for cookies developed by Netscape Corporation.

- RFC 2109, "HTTP State Management Mechanism" (http://www.w3.org/Protocols/rfc2109/rfc2109). This specification is the first cookie specification from the Worldwide Web Consortium (WC3). It defines version 1 cookies using the Cookie and Set-Cookie headers.
- RFC 2965, "HTTP State Management Mechanism" (http://www.ietf.org/rfc/rfc2965.txt). This specification makes RFC 2109 obsolete. It uses Cookie2 and Set-Cookie2 headers.

Character Set Handling

All data supplied by the application for the HTTP request except for the request content is coded in the character set of the application and translated into the client character set before sending the request. The application specifies the request content coding by setting the TRANSLATE parameter in the SET_HTTP_REQUEST_CONTENT procedure call. For example, setting this parameter allows the WEBAPPSUPPORT library to translate EBCDIC data to an ASCII equivalent before sending the request.

All data given to the application from the HTTP response except for the response content is translated from the client character set to the character set of the application before giving the data to the application. The application specifies the response content coding by setting the TRANSLATE parameter in the SET_HTTP_REQUEST_CONTENT procedure call. For example, setting this parameter allows the WEBAPPSUPPORT library to translate ASCII response content to an EBCDIC equivalent before giving it to the application

The character sets used for translation are specified by the setting of the application and client character sets in the SET_TRANSLATION WEBAPPSUPPORT procedure. In this context, the "client" character set represents the data sent to, and received from, the HTTP server.

Compressed Content

HTTP Client applications can send or receive compressed content that is compressed with the Deflate method (RFC 1951). Compression and decompression of data can also be done separately from processing HTTP requests.

Sending Compressed Content

Applications can send compressed content in HTTP requests by following these steps.

- Set the Content-Encoding request header to the value `'deflate'` with the SET_HTTP_REQUEST_HEADER procedure.
- Compress the data with the DEFLATE_DATA procedure.
- Set the request content with the compressed data using the SET_HTTP_REQUEST_CONTENT procedure. This action also sets the Content-Length request header to the compressed content length.
- Set any other request headers and execute the request.

Compressing content with the DEFLATE_DATA procedure requires that the Java Parser Module (JPM) of the XML Parser is available.

Receiving Compressed Content

When servers send responses with compressed content, the application can either receive the content compressed or decompressed, depending on the DECOMPRESS option of the SET_HTTP_OPTION procedure.

If applications receive content compressed by the HTTP server, the INFLATE-DATA procedure can be called to decompress the data.

If applications do not want response content to be compressed by the HTTP server, the Accept-Encoding request header should be set by the application to remove the deflate option that is sent by default. For example, the application can set

```
Accept-Encoding: identity
```

The JPM of the XML Parser can be used to decompress data. See the INFLATE_METHOD option of the SET_OPTION procedure.

Security

Security considerations include encrypted sessions, authentication, and storing credentials.

Encrypted Sessions (https)

Encrypted HTTP using TLS/SSL is supported. This functionality is equivalent to a web browser making an https request.

To have encrypted sessions, the requirements for the MCP system are the following:

- SSL must be enabled in MCP TCPIP.
- The CA certificates of the signers of all HTTP servers must be in a root store of the MCP—either the default root store or a root store specified by the application.
- If a client certificate is to be sent to the HTTP server, the certificate must be in the key container specified by the application.
- The application must configure the socket object to use SSL, minimally setting the SSL_Client_Mode option to Client Mode.

Authentication

The methods of authenticating the client to the server are

- HTTP Basic
- NTLM
- Client Certificates

If the application sets an Authorization header in the request, that Authorization header is sent to the server. If the application does not set an Authorization header in the request, WEBAPPSUPPORT can automatically send an Authentication header if challenged by the server and if the client object has credentials stored that match the challenge by the server.

If credentials sent from the client object are rejected by the server, the rejection response, usually a 401 (Unauthorized) response, is returned to the application.

HTTP Basic

HTTP Basic (RFC 2617) is supported.

If the HTTP server replies with a 401 (Unauthorized) response that requests Basic authentication and the client object has been configured with Basic credentials matching the host and realm of the request, the username and password are automatically sent to the server. See the SET_HTTP_CLIENT_ATTR procedure, SET_CREDENTIAL attribute.

Once Basic credentials have been automatically sent for a particular URI, all subsequent requests that begin with that URI also have the Basic credentials sent.

If the client object is not configured with credentials that match the request, the 401 response is given to the application.

NTLM

NTLM versions 1 and 2 are supported for authentication.

If the HTTP server replies with a 401 (Unauthorized) response that requests NTLM authentication and the client object has been configured with NTLM credentials matching the host of the request, the credentials are automatically sent to the server. See the SET_HTTP_CLIENT_ATTR procedure, SET_CREDENTIAL attribute.

The credentials for NTLM processing can either be a username, password and authentication domain supplied by the application or a username and credentials file created with the MAKECREDENTIALS utility.

If a credentials file is used, the application must be running under the usercode that was used to create the credentials file.

If the client object is not configured with credentials that match the request, the 401 response is given to the application.

Client Certificates

Client certificates can be sent to the HTTP server along with the SSL connection open. Before the socket is opened, the application modifies the socket object to specify a key container that contains the client certificate. See the SET_HTTP_SOCKET_OPTION procedure.

Storing Credentials

Credentials supplied by the application as clear text strings, such as username and password, are stored in the WEBAPPSUPPORT memory in encrypted form and decrypted temporarily for authentication processing.

Scenarios

The scenarios for application use of the HTTP Client function described in this section are basic request, subsequent request and SSL request (https). These scenarios serve as examples only and are not intended to describe all possible ways to complete tasks.

Basic Request Scenario

In this basic request scenario, the application makes a simple HTTP request to a remote server. The steps for making a basic request are as follow:

- The application creates a host object (CREATE_HTTP_HOST), specifying the hostname (for example, "www.serverhost.com") and port (for example, 80) of the server.
- The application creates a client object (CREATE_HTTP_CLIENT), specifying any credentials needed for the request. (SET_HTTP_CLIENT_ATTR)
- The application creates a socket object (CREATE_HTTP_SOCKET), specifying any special socket attributes needed. (SET_HTTP_SOCKET_OPTION)
- The application creates a request object (CREATE_HTTP_REQUEST), specifying the URL and any query string or post data, request headers, and so on. (SET_HTTP_REQUEST_QUERY, SET_HTTP_REQUEST_CONTENT, SET_HTTP_REQUEST_HEADER, and so on.)
- The application executes the request (EXECUTE_HTTP_REQUEST), associating the request with the host, client, and socket objects. The request is sent to the server. On return from the procedure, the application gets the parameters of the response. (GET_HTTP_RESPONSE_CONTENT, GET_HTTP_RESPONSE_HEADER, and so on)

Subsequent Request Scenario

In this scenario, the application is making another request to the same host in the basic request scenario. Assume that the host, client, and socket objects are in the same state as they were after the response in the basic request scenario was received.

- The application creates a new request object (CREATE_HTTP_REQUEST).
- The application executes the request (EXECUTE_HTTP_REQUEST), associating the request with the same host, client, and socket objects. If the HTTP server has not closed the socket, the same socket is reused. The response is handled the same as in the basic request scenario.

SSL Request (https) Scenario

In this scenario, the application is making a request to a secure Web site using SSL to encrypt the messages sent and received.

- The applications completes the steps as given for the basic request except that the port in the host object must be a secure port on the HTTP server, such as port 443.
- After the application creates the socket object, the application modifies the socket with SET_HTTP_SOCKET_OPTION as follows:
  - Set SSL client mode (required).
  - Set any other optional SSL attributes as needed. For example, the application might set a key container to specify a client certificate.

Request Complete

When the request is complete and the application no longer needs the objects that it created for the requests, the application should release the objects. (FREE_HTTP_CLIENT, FREE_HTTP_HOST, FREE_HTTP_SOCKET, and FREE_HTTP_REQUEST)

WEBAPPSUPPORT HTTP Client Procedures

The procedure topics describe the syntax, parameters, and possible return values. Each topic presents the syntax for

- A COBOL85 entry point, which has uppercase characters and underscores

  An example is CREATE_HTTP_CLIENT.
- An ALGOL entry point, which has lower-case and upper-case characters and no underscores

  An example is createHttpClient.
- An EAE entry point, which has upper-case characters and dashes

  An example is CREATE-HTTP-CLIENT.

  Note: *For more information on EAE and the notes used in the procedure description text of this guide, refer to Section 3, "WEBAPPSUPPORT EAE Interface."*

BIND_HTTP_SOCKET

Binds a socket object to a local address.

This procedure must be called after creating the socket object with the CREATE_HTTP_SOCKET procedure and before calling EXECUTE_HTTP_REQUEST. See the SockLib_Bind function in the *MCP Sockets Service Programming Guide* for more information about socket binding.

Syntax

```
INTEGER PROCEDURE BIND_HTTP_SOCKET
                      (SOCKET_TAG, SOCKADDR, ADDRLENGTH, BINDRESULT);
  INTEGER              SOCKET_TAG,           ADDRLENGTH, BINDRESULT;
  REAL ARRAY                        SOCKADDR [0];

INTEGER PROCEDURE bindHttpSocket
                      (SOCKET_TAG, SOCKADDR, ADDRLENGTH, BINDRESULT);
  VALUE                SOCKET_TAG,           ADDRLENGTH;
  INTEGER              SOCKET_TAG,           ADDRLENGTH, BINDRESULT;
  REAL ARRAY                        SOCKADDR [*];

PROCEDURE BIND-HTTP-SOCKET (GLB_PARAM);
   EBCDIC ARRAY             GLB_PARAM [0];
```

Parameters

SOCKET_TAG identifies the socket object.

SOCKADDR and ADDRLENGTH are described in the *MCP Sockets Service Programming Guide*, SockLib_Bind function.

BINDRESULT is the result returned from the Socklib_Bind function.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	SOCKET-TAG	A6	
SD	SOCKADDR-SIZE	N5	SOCKADDR size, for example, 255
SD	SOCKADDR	A*n*	[longa]
SD	BIND-RESULT	S12	

SOCKADDR is the local socket address in display format (EAE only).

Possible Return Values

In addition to the standard results, these possible values can be returned.

Value	Description
-59	A socket bind error occurred.
-60	The IP address is not valid format.
-73	The socket tag is invalid.

CREATE_HTTP_CLIENT

Creates a client object in the WEBAPPSUPPORT library.

Syntax

```
INTEGER PROCEDURE CREATE_HTTP_CLIENT
                            (CLIENT_TAG);
  INTEGER                    CLIENT_TAG;

INTEGER PROCEDURE createHttpClient
                            (CLIENT_TAG);
  INTEGER                    CLIENT_TAG;

PROCEDURE CREATE-HTTP-CLIENT (GLB_PARAM);
   EBCDIC ARRAY               GLB_PARAM [0];
```

Parameters

CLIENT_TAG identifies the created client object.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	CLIENT-TAG	A6	[bin]

Possible Return Values

In addition to the standard results, these possible values can be returned.

Value	Description
-75	Maximum HTTP Objects was exceeded.

CREATE_HTTP_HOST

Creates a host object in the WEBAPPSUPPORT library.

Syntax

```
INTEGER PROCEDURE CREATE_HTTP_HOST
                              (HOST, PORT, HOST_TAG);
  EBCDIC ARRAY                 HOST [0];
  INTEGER                            PORT, HOST_TAG;

INTEGER PROCEDURE createHttpHost
                              (HOST, PORT, HOST_TAG);
  VALUE                              PORT;
  EBCDIC ARRAY                 HOST [*];
  INTEGER                            PORT, HOST_TAG;

PROCEDURE CREATE-HTTP-HOST   (GLB_PARAM);
   EBCDIC ARRAY                GLB_PARAM [0];
```

Parameters

HOST identifies the host name, which can be a domain name or an IP address.

PORT identifies the port number of the host.

HOST_TAG identifies the created host object.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	HOST-SIZE	N5	HOST size, for example, 256
SD	HOST	A*n*	[longa]
SD	PORT	N5	
SD	HOST-TAG	A6	[bin]

Possible Return Values

In addition to the standard results, these possible values can be returned.

Value	Description
-75	Maximum HTTP Objects exceeded
-76	No IP addresses available.
-77	Not a valid host/port number

CREATE_HTTP_OBJECTS

Creates a set of objects for an HTTP Client operation.

Syntax

```
INTEGER PROCEDURE CREATE_HTTP_OBJECTS
                     (CLIENT_TAG, HOST, PORT, HOST_TAG, METHOD, URL,
                        REQUEST_TAG, SOCKET_TAG);
  INTEGER             CLIENT_TAG,       PORT, HOST_TAG,
                        REQUEST_TAG, SOCKET_TAG;
  EBCDIC ARRAY                    HOST,                METHOD, URL [0];

INTEGER PROCEDURE createHTTPobjects

                     (CLIENT_TAG, HOST, PORT, HOST_TAG, METHOD, URL,
                        REQUEST_TAG, SOCKET_TAG);
  VALUE                                PORT;
  INTEGER             CLIENT_TAG,       PORT, HOST_TAG,
                        REQUEST_TAG, SOCKET_TAG;
  EBCDIC ARRAY                    HOST,                METHOD, URL [*];

PROCEDURE CREATE-HTTP-OBJECTS (GLB_PARAM);
   EBCDIC ARRAY                GLB_PARAM [0];
```

Parameters

CLIENT_TAG identifies the created client object.

HOST identifies the host name, which can be a domain name or an IP address in display format.

PORT identifies the port number of the host.

HOST_TAG identifies the created host object.

METHOD identifies the HTTP method used in the request, for example, GET or POST.

URL identifies the request URL, not including a query string.

REQUEST_TAG identifies the created request object.

SOCKET_TAG identifies the created socket object.

Value	Description
-75	Maximum HTTP Objects exceeded
-76	No IP addresses available
-77	Not a valid host/port number
-78	Invalid method or URL

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	CLIENT-TAG	A6	[bin]
SD	HOST-SIZE	N5	HOST size, for example, 256
SD	HOST	A*n*	[longa]
SD	PORT	N5	
SD	HOST-TAG	A6	[bin]
SD	METHOD-SIZE	N5	METHOD size, for example, 256
SD	METHOD	A*n*	[longa]
SD	URL-SIZE	N5	URL size, for example, 256
SD	URL	A*n*	[longa]
SD	REQUEST-TAG	A6	[bin]
SD	SOCKET-TAG	A6	[bin]

CREATE_HTTP_REQUEST

Creates a request object in the WEBAPPSUPPORT library.

Syntax

```
INTEGER PROCEDURE CREATE_HTTP_REQUEST
                             (METHOD, URL, REQUEST_TAG);
  EBCDIC ARRAY                METHOD, URL [0];
  INTEGER                                  REQUEST_TAG;

INTEGER PROCEDURE createHttpRequest
                             (METHOD, URL, REQUEST_TAG);
  EBCDIC ARRAY                METHOD, URL [*];
  INTEGER                                  REQUEST_TAG;

PROCEDURE CREATE-HTTP-REQUEST (GLB_PARAM);
   EBCDIC ARRAY                GLB_PARAM [0];
```

Parameters

METHOD identifies the HTTP method used in the request. Examples are GET, POST, and so on.

URL identifies the request URL, not including a query string.

For example, the URL `/abc` requests the resource identified by /abc from the host identified in the host object. This URL `http://host2/abc` is a proxy request to the host identified in the host object to make a request to host2 for the resource identified by /abc.

REQUEST_TAG identifies the created request object.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	METHOD-SIZE	N5	METHOD size, for example, 256
SD	METHOD	A*n*	[longa]
SD	URL-SIZE	N5	URL size, for example, 256
SD	URL	A*n*	[longa]
SD	REQUEST-TAG	A6	[bin]

Possible Return Values

In addition to the standard results, these possible values can be returned.

Value	Description
–75	Maximum HTTP Objects exceeded
–78	Invalid method or URL

CREATE_HTTP_SOCKET

Creates a socket object in the WEBAPPSUPPORT library.

Syntax

```
INTEGER PROCEDURE CREATE_HTTP_SOCKET
                            (SOCKET_TAG);
  INTEGER                    SOCKET_TAG;

INTEGER PROCEDURE createHttpSocket
                            (SOCKET_TAG);
  INTEGER                    SOCKET_TAG;

PROCEDURE CREATE-HTTP-SOCKET (GLB_PARAM);
   EBCDIC ARRAY               GLB_PARAM [0];
```

Parameters

SOCKET_TAG identifies the created socket object.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	SOCKET-TAG	A6	[bin]

Possible Return Values

In addition to the standard results, these possible values can be returned.

Value	Description
–75	Maximum HTTP Objects exceeded

EXECUTE_HTTP_REQUEST

Causes the HTTP request to be sent to the HTTP server and returns a response from the server. See the REQUEST_TIMEOUT option of the SET_HTTP_OPTION procedure.

Syntax

```
INTEGER PROCEDURE EXECUTE_HTTP_REQUEST
                      (HOST_TAG, CLIENT_TAG, SOCKET_TAG, REQUEST_TAG,
                         STATUS_CODE);
  INTEGER              HOST_TAG, CLIENT_TAG, SOCKET_TAG, REQUEST_TAG,
                         STATUS_CODE;

INTEGER PROCEDURE executeHttpRequest
                      (HOST_TAG, CLIENT_TAG, SOCKET_TAG, REQUEST_TAG,
                         STATUS_CODE);
  VALUE                HOST_TAG, CLIENT_TAG, SOCKET_TAG, REQUEST_TAG;
  INTEGER              HOST_TAG, CLIENT_TAG, SOCKET_TAG, REQUEST_TAG,
                         STATUS_CODE;

PROCEDURE EXECUTE-HTTP-REQUEST (GLB_PARAM);
   EBCDIC ARRAY                GLB_PARAM [0];
```

Parameters

HOST_TAG identifies the host object.

CLIENT_TAG identifies the client object.

SOCKET_TAG identifies the socket object.

REQUEST_TAG identifies the request object. If the request object contains the results of a previous request, the response information from that request is cleared before attempting to execute the request.

STATUS_CODE is the HTTP response code if a server response was received before returning from the procedure, else zero.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	HOST-TAG	A6	[bin]
SD	CLIENT-TAG	A6	[bin]
SD	SOCKET-TAG	A6	[bin]
SD	REQUEST-TAG	A6	[bin]
SD	STATUS-CODE	N5	

Possible Result Values

In addition to the standard return results, these possible values can be returned.

Value	Description
-70	Invalid host tag
-71	Invalid client tag
-72	Invalid request tag
-73	Invalid socket tag
-75	Maximum HTTP Objects exceeded
-79	The procedure cannot open a socket to the HTTP server.
-80	The procedure cannot read from the HTTP server.
-81	The procedure cannot write to the HTTP server.
-82	The HTTP response cannot be parsed.

FREE_HTTP_CLIENT

Frees a client object in the WEBAPPSUPPORT library.

Syntax

```
INTEGER PROCEDURE FREE_HTTP_CLIENT
                              (CLIENT_TAG);
  INTEGER                      CLIENT_TAG;

INTEGER PROCEDURE freeHttpClient
                              (CLIENT_TAG);
  VALUE                        CLIENT_TAG;
  INTEGER                      CLIENT_TAG;

PROCEDURE GET-ATTRIBUTE-BY-NAME (GLB_PARAM);
   EBCDIC ARRAY                  GLB_PARAM [0];
```

Parameters

CLIENT_TAG identifies the client object.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	CLIENT-TAG	A6	[bin]

Possible Return Values

In addition to the standard results, these possible values can be returned.

Value	Description
–71	Invalid client tag

FREE_HTTP_HOST

Frees a host object in the WEBAPPSUPPORT library.

Syntax

```
INTEGER PROCEDURE FREE_HTTP_HOST
                              (HOST_TAG);
  INTEGER                      HOST_TAG;

INTEGER PROCEDURE freeHttpHost
                              (HOST_TAG);
  VALUE                        HOST_TAG;
  INTEGER                      HOST_TAG;

PROCEDURE GET-ATTRIBUTE-BY-NAME (GLB_PARAM);
   EBCDIC ARRAY                   GLB_PARAM [0];
```

Parameters

HOST_TAG identifies the host object.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	HOST-TAG	A6	[bin]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
–70	Invalid host tag

FREE_HTTP_REQUEST

Frees a request object in the WEBAPPSUPPORT library.

Syntax

```
INTEGER PROCEDURE FREE_HTTP_REQUEST
                              (REQUEST_TAG);
  INTEGER                      REQUEST_TAG;
INTEGER PROCEDURE freeHttpRequest
                              (REQUEST_TAG);
  VALUE                        REQUEST_TAG;
  INTEGER                      REQUEST_TAG;

PROCEDURE GET-ATTRIBUTE-BY-NAME (GLB_PARAM);
   EBCDIC ARRAY                  GLB_PARAM [0];
```

Parameters

REQUEST_TAG identifies the request object.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	REQUEST-TAG	A6	[bin]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
–72	Invalid request tag

FREE_HTTP_SOCKET

Frees a socket object in the WEBAPPSUPPORT library.

Syntax

```
INTEGER PROCEDURE FREE_HTTP_SOCKET
                              (SOCKET_TAG);
  INTEGER                      SOCKET_TAG;
INTEGER PROCEDURE freeHttpHost
                              (SOCKET_TAG);
  VALUE                        SOCKET_TAG;
  INTEGER                      SOCKET_TAG;

PROCEDURE GET-ATTRIBUTE-BY-NAME (GLB_PARAM);
   EBCDIC ARRAY                  GLB_PARAM [0];
```

Parameters

SOCKET_TAG identifies the host object.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	SOCKET-TAG	A6	[bin]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
–73	Invalid socket tag

GET_HTTP_COOKIE_STRINGS

Returns to the application the cookies stored in a client object, with their attributes. The cookies are returned as an array of strings, one cookie per array row string. See the SET_HTTP_CLIENT_ATTR procedure, SET_COOKIE attribute, for the format of a cookie string.

Only unexpired cookies are returned. Cookies that are set by the server with no expiration, meaning *Max-Age* or *expires* was not set, are returned with no *expires* setting. If *Max-Age* was set by the server, the expiration time of the cookie is returned as a Netscape-style *expires* attribute, not the original setting of the server.

Syntax

```
INTEGER PROCEDURE GET_HTTP_COOKIE_STRINGS
                    (CLIENT_TAG, MAX_COOKIE_LEN,
                       BUFFER, NUM_COOKIES);
  INTEGER            CLIENT_TAG, MAX_COOKIE_LEN,
                              NUM_COOKIES;
  EBCDIC ARRAY         BUFFER [0];


INTEGER PROCEDURE getHttpCookieStrings
                    (CLIENT_TAG, MAX_COOKIE_LEN,
                       BUFFER, NUM_COOKIES);
  VALUE              CLIENT_TAG, MAX_COOKIE_LEN;
  INTEGER            CLIENT_TAG, MAX_COOKIE_LEN,
                              NUM_COOKIES;
  EBCDIC ARRAY         BUFFER [*];

PROCEDURE GET-HTTP-COOKIE-STRINGS (GLB_PARAM);
   EBCDIC ARRAY                    GLB_PARAM [0];
```

Parameters

CLIENT_TAG identifies the client object.

MAX_COOKIE_LEN is the size of the cookie column. If attributes are present, the cookie value is terminated by a semicolon and the attributes follow.

BUFFER is the buffer in which the data is returned. The buffer is in the character set of the application, represented as one string per cookie.

NUM_COOKIES is the number of cookies returned.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	CLIENT-TAG	A6	[bin]
SD	MAX-COOKIE-LEN	N5	
SD	BUFFER-SIZE	N5	BUFFER size, for example, 1200 = 200*6
SD	BUFFER	A*n*	[longa]
SD	NUM-COOKIES	N5	

BUFFER-SIZE should be a multiple of MAX-COOKIE-LEN. BUFFER contains the cookie strings in portions of BUFFER, each portion MAX-COOKIE-LEN bytes long.

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	No cookies

Value	Description
-20	Maximum length too small
-71	Invalid client tag

Example

In COBOL, you might declare the following:

```
 01 COOKIE-BUFFER.
   03 COOKIE-PAIR OCCURS 10 TIMES.
       05 COOKIE-STRING  PIC   X(1000).
```

The call to GET_HTTP_CLIENT_COOKIES passes COOKIE-BUFFER, with MAX_COOKIE_LEN set to1000. Each occurrence of COOKIE_STRING contains a separate cookie, up to NUM_COOKIES cookies

GET_HTTP_RESPONSE_COOKIES

Returns to the application the cookies received in the response, into a structured buffer.

See also GET_HTTP_COOKIE_STRINGS.

Syntax

```
INTEGER PROCEDURE GET_HTTP_RESPONSE_COOKIES
                    (REQUEST_TAG, MAX_NAME_LEN, MAX_VALUE_LEN,
                       MAX_PATH_LEN, MAX_DOMAIN_LEN,
                         BUFFER, NUM_COOKIES);
  INTEGER            REQUEST_TAG, MAX_NAME_LEN, MAX_VALUE_LEN,
                                  NUM_COOKIES;
  EBCDIC ARRAY           BUFFER [0];


INTEGER PROCEDURE getHttpResponseCookies
                    (REQUEST_TAG, MAX_NAME_LEN, MAX_VALUE_LEN,
                       MAX_PATH_LEN, MAX_DOMAIN_LEN,
                         BUFFER, NUM_COOKIES);
  VALUE              REQUEST_TAG, MAX_NAME_LEN, MAX_VALUE_LEN;
  INTEGER            REQUEST_TAG, MAX_NAME_LEN, MAX_VALUE_LEN,
                       MAX_PATH_LEN, MAX_DOMAIN_LEN,
                                  NUM_COOKIES;
  EBCDIC ARRAY           BUFFER [*];

PROCEDURE GET-HTTP-RESPONSE-COOKIES (GLB_PARAM);
   EBCDIC ARRAY                      GLB_PARAM [0];
```

Parameters

REQUEST_TAG identifies the response object.

MAX_NAME_LEN is the size of the cookie name column and must be greater than zero.

MAX_VALUE_LEN is the size of the cookie value column and must be greater than zero.

MAX_PATH_LEN is the size of the cookie path column and can be zero.

MAX_DOMAIN_LEN is the size of the cookie domain column and can be zero.

BUFFER is the buffer into which the data is returned, in the character set of the application, represented as a repeating set of strings with one set per cookie. The string set is as follows:

- A one character string that represents the cookie type: “1” for Netscape, “2” for RFC2109, or “3” for RFC 2965.
- A one character string that indicates if the cookie is secure: “1” for secure and a space character if not secure.
- A one character string that indicates if the cookie should be discarded: “1” for discard and a space character if not to discard.
- A binary word that stores the expires time as an integer since day 0 time 0 and is compatible with the INT_TO_TIME57 and INT_TO_HTTP_DATA WEBAPPSUPPORT procedures. If the cookie does not specify an expires time, this word is zero.
- The cookie name, up to MAX_NAME_LEN bytes.
- The cookie value, up to MAX_VALUE_LEN bytes.
- The cookie path, up to MAX_PATH_LEN bytes.
- The cookie domain, up to MAX_DOMAIN_LEN bytes.

NUM_COOKIES is the number of pairs of cookies returned.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	REQUEST-TAG	A6	[bin]
SD	MAX-NAME-LEN	N5	
SD	MAX-VALUE-LEN	N5	
SD	MAX-PATH-LEN	N5	
SD	MAX-DOMAIN-LEN	N5	
SD	BUFFER-SIZE	N5	BUFFER size, for example, 7390 = 739*10
SD	BUFFER	A*n*	[longa]
SD	NUM-COOKIES	N5	

The format of the data returned in BUFFER is the same as for COBOL programs. The following EAE example matches the COBOL example:

```
SD; SD-COOKIES  ED A LE 739 INDEXED.BY SD-COOKIE-INX (10)
```

```
SD; SD-COOKIE   GROUP
      SD;  SD-CTYPE      ED A LE   1
      SD;  SD-CSECURE    ED A LE   1
      SD;  SD-CDISCARD   ED A LE   1
      SD;  SD-CEXPIRES   ED A LE   6
      SD;  SD-CNAME      ED A LE  20
      SD;  SD-CPATH      ED A LE 255
      SD;  SD-CDOMAIN    ED A LE 255
    END.GROUP;

MOVE;  (1)         SD-COOKIE-INX
MOVE;  SD-COOKIES  SD-COOKIE
```

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	The response has no cookies.
-20	Maximum length too small
-72	Invalid client tag

Example

In COBOL, you might declare the following:

```
 01 COOKIE-BUFFER.
   03 COOKIE-PAIR OCCURS 10 TIMES.
     05 COOKIE-TYPE     PIC   X(1).
     05 COOKIE-SECURE   PIC   X(1).
     05 COOKIE-DISCARD  PIC   X(1).
     05 COOKIE-EXPIRES  PIC  9(11) BINARY.
     05 COOKIE-NAME     PIC  X(20).
     05 COOKIE-VALUE    PIC X(200).
     05 COOKIE-PATH     PIC X(255).
     05 COOKIE-DOMAIN   PIC X(255).
```

The call to GET_HTTP_RESPONSE_COOKIES passes COOKIE-BUFFER with MAX_NAME_LEN set to 20, MAX_VALUE_LEN set to 200, MAX_PATH_LEN set to 255, and MAX_DOMAIN_LEN set to 255.

GET_HTTP_RESPONSE_CONTENT

Returns the content of the response to the application or writes the content to an MCP file.

If the destination is an application array, you can retrieve the data through multiple calls to this procedure. If all of the data has been retrieved, a zero (No-op) procedure result is returned.

If the destination is a file, the file must be a stream file with the following attributes:

```
BLOCKSTRUCTURE   = FIXED
EXTMODE          = ASCII
FILEORGANIZATION = NOTRESTRICTED
FILESTRUCTURE    = STREAM
FILETYPE         = DATA
FRAMESIZE        = 8
MAXRECSIZE       = 1
MINRECSIZE       = 1
SECURITYTYPE     = PRIVATE
SECURITYUSE      = IO
```

The preceding attributes override setting attributes in the FILE_ATTRIBUTES option of the SET_OPTION procedure.

If the destination is a permanent directory, the directory structures must have been previously created.

Syntax

```
INTEGER PROCEDURE GET_HTTP_RESPONSE_CONTENT
                   (REQUEST_TAG, DEST_TYPE, TRANSLATE,
                      CONTENT, CONTENT_START, CONTENT_LEN);
  INTEGER           REQUEST_TAG, DEST_TYPE, TRANSLATE,
                              CONTENT_START, CONTENT_LEN;
  EBCDIC ARRAY        CONTENT [0];

INTEGER PROCEDURE getHttpResponseContent
                   (REQUEST_TAG, DEST_TYPE, TRANSLATE,
                      CONTENT, CONTENT_START, CONTENT_LEN);
  VALUE             REQUEST_TAG, DEST_TYPE, TRANSLATE,
                              CONTENT_START;
  INTEGER           REQUEST_TAG, DEST_TYPE, TRANSLATE,
                              CONTENT_START, CONTENT_LEN;
  EBCDIC ARRAY        CONTENT [*];

PROCEDURE GET-HTTP-RESPONSE-CONTENT (GLB_PARAM);
   EBCDIC ARRAY                      GLB_PARAM [0];
```

Parameters

REQUEST_TAG identifies the response.

DEST_TYPE is the destination for the response content.

If the DEST_TYPE value is 1, the CONTENT parameter contains the response content.

If the DEST_TYPE value is 2, on input the CONTENT parameter contains the MCP file name to which to write the response content. See the FILE_ATTRIBUTES and FILENAME_FORMAT options in the SET_OPTION procedure.

TRANSLATE indicates whether or not to translate the content before returning.

If the TRANSLATE value is 0, do not translate the content.

If the TRANSLATE value is 1, translate the content from the character set of the client of the application to the character set of the application. See SET_TRANSLATION procedure.

CONTENT contains the response content on output if DEST_TYPE = 1 or if the MCP filename on input is DEST_TYPE = 2.

If DEST_TYPE = 1 on input, CONTENT_LEN is the maximum number of bytes of response content to return. A value of zero for CONTENT_LEN means return all content. If DEST_TYPE = 2 on input, CONTENT_LEN is the length of the MCP file name, and all content from the response is written to the file.

CONTENT_LEN on output is the length in bytes of the content either returned in the CONTENT parameter or written to the file.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	REQUEST-TAG	A6	[bin]
SD	DEST-TYPE	N5	
SD	TRANSLATE	N5	
SD	CONTENT-SIZE	N5	CONTENT size, for example, 2048
SD	CONTENT	A*n*	[longa]
SD	CONTENT-START	N5	
SD	CONTENT-LEN	N12	

Possible Result Values

In addition to the standard return results, these possible values can be returned.

Value	Description
0	Invalid request tag
-72	Response contains no content.
-83	Response is not complete (asynchronous mode).

GET_HTTP_RESPONSE_HEADER

Returns the value of the specified response header to the application. If the same header occurs more than once in the response only the first header is returned. Also, see GET_HTTP_RESPONSE_HEADERS.

Syntax

```
INTEGER PROCEDURE GET_HTTP_RESPONSE_HEADER
                            (REQUEST_TAG, NAME, VALUE);
  INTEGER                    REQUEST_TAG;
```

```
  EBCDIC ARRAY                              NAME, VALUE [0];
INTEGER PROCEDURE getHttpResponseHeader
                            (REQUEST_TAG, NAME, VALUE);
  VALUE                      REQUEST_TAG;
  INTEGER                    REQUEST_TAG;
  EBCDIC ARRAY                              NAME, VALUE [*];

PROCEDURE GET-HTTP-RESPONSE-HEADER (GLB_PARAM);
   EBCDIC ARRAY                      GLB_PARAM [0];
```

Parameters

REQUEST_TAG identifies the response.

NAME identifies the response header, coded in the character set of the application. An example is `Last-Modified`. Matching is case-insensitive.

VALUE identifies the returned header value, coded in the character set of the application. An example is `Mon, 23 Mar 2009 19:44:26 GMT`.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	REQUEST-TAG	A6	[bin]
SD	NAME-SIZE	N5	NAME size, for example, 256
SD	NAME	A*n*	[longa]
SD	VALUE-SIZE	N5	VALUE size, for example, 2048
SD	VALUE	A*n*	[longa]

Possible Result Values

In addition to the standard return results, these possible values can be returned.

Value	Description
0	Invalid request tag
-72	Response contains no content.
-83	Response is not complete (asynchronous mode).

GET_HTTP_RESPONSE_HEADERS

Returns the set of response headers to the application.

Syntax

```
INTEGER PROCEDURE GET_HTTP_RESPONSE_HEADERS
                    (REQUEST_TAG, MAX_NAME_LEN, MAX_VALUE_LEN,
                       BUFFER, NUM_PAIRS);
  INTEGER            REQUEST_TAG, MAX_NAME_LEN, MAX_VALUE_LEN,
```

```
                                NUM_PAIRS;
  EBCDIC ARRAY         BUFFER [0];


INTEGER PROCEDURE getHttpResponseHeaders
                    (REQUEST_TAG, MAX_NAME_LEN, MAX_VALUE_LEN,
                       BUFFER, NUM_PAIRS);
  VALUE              REQUEST_TAG, MAX_NAME_LEN, MAX_VALUE_LEN;
  INTEGER            REQUEST_TAG, MAX_NAME_LEN, MAX_VALUE_LEN,
                                NUM_PAIRS;
  EBCDIC ARRAY         BUFFER [*];

PROCEDURE GET-HTTP-RESPONSE-HEADERS (GLB_PARAM);
   EBCDIC ARRAY                      GLB_PARAM [0];
```

Parameters

REQUEST_TAG identifies the response.

MAX_NAME_LEN is the size of the header name column.

MAX_VALUE_LEN is the size of the header value column.

BUFFER is the buffer into which the data is returned, in the character set of the application, represented as pairs of strings.

NUM_PAIRS is the number of pairs returned.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	REQUEST-TAG	A6	[bin]
SD	MAX-NAME-LEN	N5	
SD	MAX-VALUE-LEN	N5	
SD	BUFFER-SIZE	N5	BUFFER size, for example, 1200 = 120*10
SD	BUFFER	A*n*	[longa]
SD	NUM-PAIRS	N5	

The format of the data returned in BUFFER is the same as for COBOL programs.

Possible Result Values

In addition to the standard return results, these possible values can be returned.

Value	Description
-72	Response contains no content.
-83	Response is not complete (asynchronous mode).

Example

In COBOL, you might declare the following:

```
 01 NAME-VALUE-BUFFER.
   03 NAME-VALUE-PAIR OCCURS 10 TIMES.
     05 HEADER-NAME  PIC  X(20).
     05 HEADER-VALUE PIC X(100).
```

The call to GET_HTTP_RESPONSE_HEADERS passes NAME-VALUE-BUFFER, with MAX_NAME_LEN set to 20 and MAX_VALUE_LEN set to 100.

GET_HTTP_RESPONSE_STATUS

Returns the status of the response to the application with information from the HTTP response status line. The application can use this procedure on asynchronous requests to determine whether or not the response is complete.

Syntax

```
INTEGER PROCEDURE GET_HTTP_RESPONSE_STATUS
                      (REQUEST_TAG, VERSION, STATUS_CODE, REASON);
  INTEGER              REQUEST_TAG,          STATUS_CODE;
  EBCDIC ARRAY                       VERSION,              REASON [0];

INTEGER PROCEDURE getHttpResponseStatus
                      (REQUEST_TAG, VERSION, STATUS_CODE, REASON);
  VALUE                REQUEST_TAG;
  INTEGER              REQUEST_TAG,          STATUS_CODE;
  EBCDIC ARRAY                       VERSION,              REASON [*];

PROCEDURE GET-HTTP-RESPONSE-STATUS (GLB_PARAM);
   EBCDIC ARRAY                     GLB_PARAM [0];
```

Parameters

REQUEST_TAG identifies the response.

VERSION is the HTTP version of the response in the character set of the application. An example is `1.1`.

STATUS_CODE is the HTTP response code if the response is complete; otherwise, the value is zero.

REASON is the HTTP reason phrase in the character set of the application.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	REQUEST-TAG	A6	[bin]
SD	VERSION-SIZE	N5	VERSION size, for example, 256
SD	VERSION	A*n*	[longa]
SD	STATUS-CODE	N12	
SD	REASON-SIZE	N5	REASON size, for example, 256
SD	REASON	A*n*	[longa]

Possible Result Values

In addition to the standard return results, these possible values can be returned.

Value	Description
-72	Response contains no content.
-83	Response is not complete (asynchronous mode) or request not placed.

GET_HTTP_SOCKET_OPTION

Inquiries on a socket object in the WEBAPPSUPPORT library. The socket options supported are those supported by the SOCKETSUPPORT library. See the *MCP Sockets Service Programming Guide*, SockLib_GetSockOpt function for more information.

Two procedures are available for EAE applications to get socket options—one passing integers and one passing a string.

Syntax

```
INTEGER PROCEDURE GET_HTTP_SOCKET_OPTION
                    (SOCKET_TAG, LEVEL, OPTION, OPTVAL, OPTLEN,
OPTRESULT);
  INTEGER            SOCKET_TAG, LEVEL, OPTION,         OPTLEN,
OPTRESULT;
  EBCDIC ARRAY                                   OPTVAL [0];

INTEGER PROCEDURE getHttpSocketOption
                    (SOCKET_TAG, LEVEL, OPTION, OPTVAL, OPTLEN,
OPTRESULT);
  VALUE              SOCKET_TAG, LEVEL, OPTION;
  INTEGER            SOCKET_TAG, LEVEL, OPTION,         OPTLEN,
OPTRESULT;
  EBCDIC ARRAY                                   OPTVAL [*];

PROCEDURE GET-HTTP-SOCKET-INTOPTION (GLB_PARAM);
   EBCDIC ARRAY                      GLB_PARAM [0];

PROCEDURE GET-HTTP-SOCKET-STROPTION (GLB_PARAM);
   EBCDIC ARRAY                      GLB_PARAM [0];
```

Parameters

SOCKET_TAG identifies the socket object.

LEVEL, OPTION, OPTVAL, OPTLEN are described in the *MCP Sockets Service Programming Guide*, SockLib_GetSockOpt function.

OPTRESULT is the result returned from the SockLib_GetSockOpt function.

For the GET-HTTP-SOCKET-INTOPTION procedure, GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	SOCKET-TAG	A6	[bin]
SD	LEVEL	N12	
SD	OPTION	N12	
SD	OPTLEN	N5	6 if only INT1 is needed; otherwise, 12
SD	OPTVAL-INT1	S12	
SD	OPTVAL-INT2	S12	
SD	OPTRESULT	S12	

For the GET-HTTP-SOCKET-STROPTION procedure, GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	SOCKET-TAG	A6	[bin]
SD	LEVEL	N12	
SD	OPTION	N12	
SD	OPTVAL-SIZE	N5	OPTVAL size, for example, 2048
SD	OPTVAL	A*n*	[longa]
SD	OPTLEN	N12	
SD	OPTRESULT	S12	

Possible Result Values

In addition to the standard return results, these possible values can be returned.

Value	Description
–73	Invalid socket tag

INIT_HTTP_REQUEST

Initializes a request object so that it can be reused for another request. Request information set on the request object, such as request headers, is not changed.

Syntax

```
INTEGER PROCEDURE INIT_HTTP_REQUEST
                        (REQUEST_TAG);
  INTEGER                REQUEST_TAG;

INTEGER PROCEDURE initHttpRequest
                        (REQUEST_TAG);
  VALUE                  REQUEST_TAG;
  INTEGER                REQUEST_TAG;

PROCEDURE INIT-HTTP-REQUEST (GLB_PARAM);
   EBCDIC ARRAY              GLB_PARAM [0];
```

Parameters

REQUEST_TAG identifies the request object.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	REQUEST-TAG	A6	[bin]

Possible Result Values

In addition to the standard return results, these possible values can be returned.

Value	Description
–72	The request tag is invalid.

SET_HTTP_CLIENT_ATTR

Sets optional attributes of the HTTP client object.

Syntax

```
INTEGER PROCEDURE SET_HTTP_CLIENT_ATTR
                                (CLIENT_TAG,ATTR, ATTR_VALUE,
ATTR_STRING);
  INTEGER                        CLIENT_TAG,ATTR, ATTR_VALUE;
  EBCDIC ARRAY
ATTR_STRING [0];

INTEGER PROCEDURE setHTTPclientAttr
                                (CLIENT_TAG,ATTR, ATTR_VALUE,
ATTR_STRING);
  VALUE                          CLIENT_TAG,ATTR, ATTR_VALUE;
  INTEGER                        CLIENT_TAG,ATTR, ATTR_VALUE;
  EBCDIC ARRAY
ATTR_STRING [*];

PROCEDURE SET-HTTP-CLIENT-ATTR (GLB_PARAM);
   EBCDIC ARRAY                GLB_PARAM [0];
```

Parameters

CLIENT_TAG identifies the client object.

ATTR is the attribute being set.

1 (SET_COOKIE)

This attribute sets or clears a cookie stored in the client object. See “Cookie Handling” previously in this section for links to specifications with details on cookie attribute values.

If the *domain* attribute is not specified, then the nonstandard host attribute must be specified. Host is the domain name or IP address of the originating server. Cookies are only sent to the originating host if a domain is not specified.

If *expires* is set to a date in the past or *Max-Age* is set to zero, then the cookie is deleted from the client object.

The following points list the values for the ATTR_VALUE parameter for the SET_COOKIE attribute.

- If the value is 1, a Netscape-style cookie is set in the client object. If the cookie already exists (matching the domain or host and the path), its value is overwritten with the new value. The format of the ATTR_STRING parameter must be

  ```
  name=value[; expires=date][; path=path][; domain=domain_name][;
  secure][; host=host]
  ```

  where name is the cookie name; value is the value of the cookie; and host is the domain name or IP address that matches the name of the HOST in the host object. The following is an example.

```
step=step1; domain=.httphost.com
```

- If the value is 2, an RFC 2109-style cookie is set in the client object. If the cookie already exists, its value is overwritten with the new value. The format of the ATTR_STRING parameter must be

  ```
  name=value[; Comment=comment][; Domain=domain][; Max-Age=delta-seconds][; Path=path][; Secure][; Version=digit] [; host=host]
  ```

  where name is the cookie name; value is the value of the cookie; and host is the domain name or IP address that matches the name of the HOST in the host object. The following is an example.

  ```
  step="step1"; Version="1"; Domain=".httphost.com"
  ```

- if the value is 3, an RFC 2965-style cookie is set in the client object. If the cookie already exists, its value is overwritten with the new value. The format of the ATTR_STRING parameter must be.

  ```
  name=value[; Comment=comment][; CommentURL=http-url][; Discard][; Domain=domain][; Max-Age=delta-seconds][; Path=path][; Port=port-list][; Secure][; Version=digit] [; host=host]
  ```

  where name is the cookie name; value is the value of the cookie; and host is the domain name or IP address that matches the name of the HOST in the host object. If Port is specified, a port list must be specified. The following is an example.

  ```
  step="step1"; Version="1"; Domain=".httphost.com"; Port="80"
  ```

2 (SET_CREDENTIAL)

This attribute sets or clears a credential stored in the client object.

The following points list the values for the ATTR_VALUE parameter for the SET_CREDENTIAL attribute.

- If the value is 1, a HTTP Basic credential is set in the client object. The format of the ATTR_STRING parameter must be

  ```
  username:password;host;realm
  ```

  where username and password are the username and password to be sent in the request; host is the domain name or IP address that matches the name of the server in the host object; and realm is the realm of the request.

  If an HTTP Basic credential matching the same host and realm exists, the old credential is replaced by the new one. The following is an example.

  ```
  sjones:pass1;paymentserver.com;/
  ```

- The value of 2 is reserved.
- If the value is 3, an NTLM credential with a username and password is set in the client object. The format of the ATTR_STRING parameter must be

  ```
  username:password;host;domain
  ```

where username and password are the username and password to be sent in the request; host is the domain name or IP address that matches the name of the server in the host object; and domain is the domain in which the user is authenticated. The following is an example.

```
sjones:pass1;paymentserver.com;na
```

- If the value is 4, an NTLM credential using a credentials file is set in the client object. The format of the ATTR_STRING parameter must be

  username;host;server

  where username is the username for the request; host is the domain name or IP address that matches the name of the server in the host object; and server is the host name used for creating the credentials file. The following is an example.

  ```
  sjones:paymentserver.com;paymentserver
  ```

The use of ATTR_STRING is described in the previous points. For any settings that do not define a use for it, the application should set it to a null string.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	CLIENT-TAG	A6	[bin]
SD	ATTR	N5	
SD	ATTR-VALUE	S12	
SD	ATTR-STRING-SIZE	N5	ATTR-STRING size, for example, 256
SD	ATTR-STRING	A*n*	[longa]

Possible Result Values

In addition to the standard return results, these possible values can be returned.

Value	Description
–84	Invalid credential
–85	Invalid cookie

SET_HTTP_OPTION

SET_HTTP_OPTION controls options specific to processing of HTTP client requests.

Syntax

```
INTEGER PROCEDURE SET_HTTP_OPTION
                  (OPTION, OPTION_VALUE, OPTION_STRING);
  INTEGER          OPTION, OPTION_VALUE;
  EBCDIC ARRAY                            OPTION_STRING [0];
```

```
INTEGER PROCEDURE setHTTPOption
                   (OPTION, OPTION_VALUE, OPTION_STRING);
  VALUE             OPTION, OPTION_VALUE;
  INTEGER           OPTION, OPTION_VALUE;
  EBCDIC ARRAY                             OPTION_STRING [*];

PROCEDURE SET-HTTP-OPTION (GLB_PARAM);
   EBCDIC ARRAY            GLB_PARAM [0];
```

Parameters

OPTION is the option being set. Supported options are 1 (DECOMPRESS), 2 (FOLLOW_REDIRECTS), 3 (REQUEST_LEVEL), 4 (STORE_COOKIES), 5 (SYNCHRONOUS), 6 (USER_AGENT), and 7(REQUEST_TIMEOUT).

1 (DECOMPRESS)

This option controls whether or not to automatically decompress compressed content in the response.

If the value of OPTION_VALUE is 0, do not compress. The compressed content is returned to the application in the GET_HTTP_RESPONSE_CONTENT procedure.

If the value of OPTION_VALUE is 1, automatically decompress response content that was compressed by the server. This value is the default.

2 (FOLLOW_REDIRECTS)

This option controls whether or not to automatically follow redirects from the server.

If the OPTION_VALUE is 0, do not follow redirects to the server. The redirect response is returned to the application.

If the OPTION_VALUE is 1, automatically follow redirects from the server. This value is the default.

If the server redirects the request from a non-SSL request (http://) to an SSL request (https://), the only SSL socket option set on behalf of the client is the SSL_Client_Mode option that is set to Client Mode. If other SSL socket settings are needed to process the redirected request, the application should set this option to 0 and handle the redirect itself.

3 (REQUEST_LEVEL)

This option sets the HTTP level to use on requests.

If the OPTION_VALUE is 0, use HTTP/1.0.

If the OPTION_VALUE is 1, use HTTP/1.1. This value is the default.

4 (STORE_COOKIES)

This option controls whether or not to save cookies received from the server and automatically resend them on subsequent requests.

If the OPTION_VALUE is 0, do not store cookies received from the server.

If the OPTION_VALUE is 1, store cookies received from the server in the client object, and resend them on subsequent requests to the same path and domain. This value is the default.

5 (SYNCHRONOUS)

This option controls whether or not requests are made synchronously or asynchronously.

If the OPTION_VALUE is 0, requests are made synchronously. This value is the default.

If the OPTION_VALUE is 1, requests are made asynchronously.

6 (USER_AGENT)

This option controls whether or not the HTTP header User-Agent is sent by default with the request.

If the OPTION_VALUE is 0, do not send a User-Agent header by default.

If the OPTION_VALUE is 1, send the NAME task attribute of the application as the User-Agent header. This value is the default.

7 (REQUEST_TIMEOUT)

This option specifies the number of seconds to wait for a response from the HTTP server.

OPTION_VALUE is the number of seconds. The default is 15 seconds. If OPTION_VALUE is less than 1, then 1 second is used.

The use of OPTION_STRING is described in the previous option descriptions. If the option does not use OPTION_STRING, the application should set OPTION_STRING to a null string.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	OPTION	A6	[bin]
SD	OPTION-VALUE	N12	
SD	OPTION-STRING-SIZE	N5	OPTION-STRING size, for example, 5000
SD	OPTION-STRING	A*n*	[longa]

Possible Result Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	Option or value not supported
1	Setting accepted.

SET_HTTP_REQUEST_CONTENT

Sets content for the request.

The content can come either from an array in the application or from an MCP file. If the content comes from an MCP file, the file must be accessible by the application and can be cached by the WEBAPPSUPPORT library.

This procedure can translate the content before including into the response. An example is translating from ASERIESEBCDIC supplied by the application to ASCII.

Content can either be supplied as one block of data, or with multiple calls to this procedure, it can be supplied in "chunks." Chunked input allows sending content that is dynamic in size. This technique is valuable when supplying the total content in one call to this procedure is difficult.

Syntax

```
INTEGER PROCEDURE SET_HTTP_REQUEST_CONTENT
                   (REQUEST_TAG, SOURCE_TYPE, TRANSLATE, CHUNKED,
                      CONTENT, CONTENT_START, CONTENT_LEN);
  INTEGER           REQUEST_TAG, SOURCE_TYPE, TRANSLATE, CHUNKED,
                                CONTENT_START, CONTENT_LEN;
  EBCDIC ARRAY        CONTENT [0];

INTEGER PROCEDURE setHttpRequestContent
                   (REQUEST_TAG, SOURCE_TYPE, TRANSLATE, CHUNKED,
                      CONTENT, CONTENT_START, CONTENT_LEN);
  VALUE             REQUEST_TAG, SOURCE_TYPE, TRANSLATE, CHUNKED,
                                CONTENT_START, CONTENT_LEN;
  INTEGER           REQUEST_TAG, SOURCE_TYPE, TRANSLATE, CHUNKED,
                                CONTENT_START, CONTENT_LEN;
  EBCDIC ARRAY        CONTENT [*];

PROCEDURE SET-HTTP-REQUEST-CONTENT (GLB_PARAM);
   EBCDIC ARRAY                     GLB_PARAM [0];
```

Parameters

REQUEST_TAG identifies the request object.

SOURCE_TYPE identifies the source of the content.

If SOURCE_TYPE is 1, the CONTENT parameter contains the content to be put into the response.

If SOURCE_TYPE is 2, the CONTENT parameter contains the MCP file name of the content to be put into the response. See the FILENAME_FORMAT option in the SET_OPTION procedure.

TRANSLATE indicates whether or not to translate the content before including it in the response.

If TRANSLATE is 0, do not translate the content.

If TRANSLATE is 1, translate the content before including it in the response and use the character sets of the application and client, respectively, as the source and destination character sets. See the SET_TRANSLATION procedure.

CHUNKED indicates whether or not the content is supplied in one or multiple calls to this procedure.

IF CHUNKED is 0, the content supplied is the only content for the response.

If CHUNKED is 1, the content supplied is one of a set of content chunks.

CONTENT identifies the request content.

CONTENT_START is the zero-based offset into CONTENT and indicates where the procedure finds the start of the content.

CONTENT_LEN is the length of the content in bytes. The maximum supported length is approximately 0.5 TB. (See the TEMPFAMILY information in Section 3, "WEBAPPSUPPORT General Parameters File.") If zero or less is specified, the content stored in the request is cleared.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	REQUEST-TAG	A6	[bin]
SD	SOURCE-TYPE	N5	
SD	TRANSLATE	N5	
SD	CHUNKED	N5	
SD	CONTENT-SIZE	N5	CONTENT size, for example, 2048
SD	CONTENT	A*n*	[longa]
SD	CONTENT-START	N5	
SD	CONTENT-LEN	N5	

Possible Result Values

In addition to the standard return results, these possible values can be returned.

Value	Description
0	The TRANSLATE or CHUNKED value is not supported.
-47	The CONTENT_START offset is invalid.
-72	Invalid request tag

SET_HTTP_REQUEST_HEADER

Enables setting multiple headers with the same name in the request.

Syntax

```
INTEGER PROCEDURE SET_HTTP_REQUEST_HEADER
                           (REQUEST_TAG, NAME, VALUE);
  INTEGER                   REQUEST_TAG;
  EBCDIC ARRAY                            NAME, VALUE [0];

INTEGER PROCEDURE setHttpRequestHeader
                           (REQUEST_TAG, NAME, VALUE);
  VALUE                     REQUEST_TAG;
  INTEGER                   REQUEST_TAG;
  EBCDIC ARRAY                            NAME, VALUE [*];

PROCEDURE SET-HTTP-REQUEST-HEADER (GLB_PARAM);
   EBCDIC ARRAY                    GLB_PARAM [0];
```

Parameters

REQUEST_TAG identifies the request object.

NAME identifies the response header, coded in the character set of the application, and it must not be an empty string. An example is User-Agent.

VALUE identifies the returned header value, coded in the character set of the application, and it can be an empty string. An example is Acme Accounts Receivable.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	REQUEST-TAG	A6	[bin]
SD	NAME-SIZE	N5	NAME size, for example, 256
SD	NAME	A*n*	[longa]
SD	VALUE-SIZE	N5	VALUE size, for example, 256
SD	VALUE	A*n*	[longa]

Possible Result Values

In addition to the standard return results, these possible values can be returned.

Value	Description
–72	Invalid request tag

SET_HTTP_REQUEST_QUERY

Sets a query string for the response. The application can either supply a query string or a set of name-value pairs.

The maximum length of a URL-encoded query string is 2048 bytes.

If name-value pairs are given, the procedure does the following URL encoding of the data:

- Embedded space characters in the string(s) are replaced by plus (+) signs.
- Non-alphanumeric characters are escaped as percent-encoded, a percent sign (%) followed by two hexadecimal characters of the ASCII-equivalent. For example. an ampersand (&) is encoded as %26.

Syntax

```
INTEGER PROCEDURE SET_HTTP_REQUEST_QUERY
                    (REQUEST_TAG, MAX_NAME_LEN, MAX_VALUE_LEN,
                       BUFFER, NUM_PAIRS);
  INTEGER            REQUEST_TAG, MAX_NAME_LEN, MAX_VALUE_LEN,
                                 NUM_PAIRS;
  EBCDIC ARRAY         BUFFER [0];

INTEGER PROCEDURE setHttpRequestQuery
                    (REQUEST_TAG, MAX_NAME_LEN, MAX_VALUE_LEN,
                       BUFFER, NUM_PAIRS);
  VALUE              REQUEST_TAG, MAX_NAME_LEN, MAX_VALUE_LEN;
  INTEGER            REQUEST_TAG, MAX_NAME_LEN, MAX_VALUE_LEN,
                                 NUM_PAIRS;
  EBCDIC ARRAY         BUFFER [*];

PROCEDURE SET-HTTP-REQUEST-QUERY (GLB_PARAM);
   EBCDIC ARRAY                   GLB_PARAM [0];
```

Parameters

REQUEST_TAG identifies the response.

MAX_NAME_LEN is the size of the header name column.

MAX_VALUE_LEN is the size of the header value column.

BUFFER is the array containing the query string or set of name-value pairs in the character set of the application. If BUFFER is a null string, the query string is cleared in the request.

NUM_PAIRS is the number of name-value pairs. If set to zero, BUFFER contains a query string; otherwise, it contains name-value pairs.

GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	REQUEST-TAG	A6	[bin]
SD	MAX-NAME-LEN	N5	
SD	MAX-VALUE-LEN	N5	
SD	BUFFER-SIZE	N5	BUFFER size, for example, 2048
SD	BUFFER	A*n*	[longa]
SD	NUM-PAIRS	N5	

Possible Result Values

In addition to the standard return results, these possible values can be returned.

Value	Description
–72	Invalid request tag

Example

In COBOL, you might declare the following:

```
01 NAME-VALUE-BUFFER.
    03 NAME-VALUE-PAIR OCCURS 10 TIMES.
        05 QUERY-NAME  PIC  X(20).
        05 QUERY -VALUE PIC X(100).
```

The call to SET_HTTP_REQUEST_QUERY passes NAME-VALUE-BUFFER with MAX_NAME_LEN set to 20, MAX_VALUE_LEN set to 100, and NUM_PAIRS set to the number of pairs.

SET_HTTP_SOCKET_OPTION

Modifies a socket object in the WEBAPPSUPPORT library. The socket options supported are those supported by the SOCKETSUPPORT library. See the *MCP Sockets Service Programming Guide*, SockLib_SetSockOpt function for more information.

The options shown in the following table are set by default on each HTTP socket. The application can override these settings by calling the SET_HTTP_SOCKET_OPTION procedure.

Level	Option	Value	Description
SOL_Socket	SO_RcvTimeO	1 second	The application should not change this setting; response timeouts are managed by the WEBAPPSUPPORT library.

Two procedures are available for EAE applications to set socket options—one passing integers and one passing a string.

Syntax

```
INTEGER PROCEDURE SET_HTTP_SOCKET_OPTION
                        (SOCKET_TAG, LEVEL, OPTION, OPTVAL, OPTLEN,
OPTRESULT);
  INTEGER                SOCKET_TAG, LEVEL, OPTION,         OPTLEN,
OPTRESULT;
  EBCDIC ARRAY                                       OPTVAL [0];

INTEGER PROCEDURE setHttpSocketOption
                        (SOCKET_TAG, LEVEL, OPTION, OPTVAL, OPTLEN,
OPTRESULT);
  VALUE                  SOCKET_TAG, LEVEL, OPTION,         OPTLEN;
  INTEGER                SOCKET_TAG, LEVEL, OPTION,         OPTLEN,
OPTRESULT;
  EBCDIC ARRAY                                       OPTVAL [*];

PROCEDURE SET-HTTP-SOCKET-INTOPTION (GLB_PARAM);
   EBCDIC ARRAY                      GLB_PARAM [0];

PROCEDURE SET-HTTP-SOCKET-STROPTION (GLB_PARAM);
   EBCDIC ARRAY                      GLB_PARAM [0];
```

Parameters

SOCKET_TAG identifies the socket object.

LEVEL, OPTION, OPTVAL, OPTLEN are described in the *MCP Sockets Service Programming Guide*, SockLib_SetSockOpt function.

OPTRESULT is the result returned from the SockLib_SetSockOpt function.

For the SET-HTTP-SOCKET-INTOPTION procedure, GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	SOCKET-TAG	A6	[bin]
SD	LEVEL	N12	
SD	OPTION	N12	
SD	OPTLEN	N5	6 if only INT1 is needed; otherwise, 12
SD	OPTVAL-INT1	S12	
SD	OPTVAL-INT2	S12	
SD	OPTRESULT	S12	

For the SET-HTTP-SOCKET-STROPTION procedure, GLB_PARAM has the following format:

Format			Notes
SG-GLB-PARAM GROUP			
SG-PARAM GROUP			
SD	RESULT	S5	
SD	SOCKET-TAG	A6	[bin]
SD	LEVEL	N12	
SD	OPTION	N12	
SD	OPTVAL-SIZE	N5	OPTVAL size, for example, 2048
SD	OPTVAL	A*n*	[longa]
SD	OPTLEN	N12	
SD	OPTRESULT	S12	

Possible Result Values

In addition to the standard return results, these possible values can be returned. Other errors are returned by the SockLib_SetSockOpt function.

Value	Description
–73	Invalid socket tag

Section 10
Using Regular Expressions

The Regular Expressions feature enables applications to apply expressions to data, similar to the way that the Perl compatible Regular Expressions (PCRE) (http://www.pcre.org/) package is used. The PCRE library is a set of functions that implement regular expression pattern matching using the same syntax and semantics as Perl 5.

The product component for Regular Expressions in the CCF is REGEXPRESSION.

The CCF product also provides sample applications that demonstrate using Regular Expressions. The COBOL85 sample application is *SYSTEM/CCF/REGULAREXPRESSION/SAMPLE/COBOL, and the ALGOL sample application is *SYSTEM/CCF/REGULAREXPRESSION/SAMPLE/ALGOL. These sample applications use basic, regular expressions to show multiple dates extracted from a string.

See Section 1, "Regular Expressions" for limitations and character set handling information about the Regular Expressions feature.

PCRE API Mapping to WEBAPPSUPPORT Procedures

Table 10–1 summarizes the mapping of PCRE functions to WEBAPPSUPPORT procedures.

Table 10–1. PCRE Functions Mapped to WEBAPPSUPPORT Procedures

PCRE	WEBAPPSUPPORT
pcre *pcre_compile(const char *pattern, int options, const char **errptr, int *erroffset, const unsigned char *tableptr); Compiles a pattern into an internal form. pcre_compile2 also returns an error code.	The COMPILE_RE_PATTERN procedure performs the equivalent function of pcre_compile2, returning an error code.

Table 10–1. PCRE Functions Mapped to WEBAPPSUPPORT Procedures

PCRE	WEBAPPSUPPORT
int pcre_config(int what, void *where); Allows a pcre client to find out what features have been compiled into the pcre library.	An equivalent procedure is not provided; the defaults for the SET_RE_OPTION procedure identify the compiled settings.
int pcre_copy_named_substring(const pcre *code, const char *subject, int *ovector, int stringcount, const char *stringname, char *buffer, int buffersize); Extracts a captured substring by name.	An equivalent procedure is not provided; the application gets all substrings that were found from an execute call.
int pcre_copy_substring(const char *subject, int *ovector, int stringcount, int stringnumber, char *buffer, int buffersize); Extracts a captured substring by number.	An equivalent procedure is not provided; the application gets all substrings that were found from an execute call.
int pcre_dfa_exec(const pcre *code, const pcre_extra *extra, const char *subject, int length, int startoffset, int options, int *ovector, int ovecsize, int *workspace, int wscount); Matches a subject string against a compiled pattern; returns multiple matches.	The EXECUTE_RE procedure returns multiple matches if the MATCH_ALGORITHM option is set in the SET_RE_OPTION procedure.
int pcre_exec(const pcre *code, const pcre_extra *extra, const char *subject, int length, int startoffset, int options, int *ovector, int ovecsize); Matches a subject string against a compiled pattern; returns the first match.	The EXECUTE_RE procedure returns the first match if the MATCH_ALGORITHM option is reset in the SET_RE_OPTION procedure.
void pcre_free_substring(const char *stringptr); Frees the memory of a call to pcre_get_substring.	An equivalent procedure is not provided; all substrings are given to the application in the EXECUTE_RE call.

Table 10–1. PCRE Functions Mapped to WEBAPPSUPPORT Procedures

PCRE	WEBAPPSUPPORT
void pcre_free_substring_list(const char **stringptr); Frees the memory of a call to pcre_get_substring_list.	An equivalent procedure is not provided; all substrings are given to the application in the EXECUTE_RE call.

int pcre_fullinfo(const pcre *code, const pcre_extra *extra, int what, void *where);
 	
int pcre_info(const pcre *code, int *optptr, int *firstcharptr);
 	
pcre_fullinfo returns information about a compiled pattern; pcre_info returns partial information and is obsolete.
	An equivalent procedure is not provided.

int pcre_get_named_substring(const pcre *code, const char *subject, int *ovector, int stringcount, const char *stringname, const char **stringptr); Extracts a captured substring by name.	An equivalent procedure is not provided; all substrings are given to the application in the EXECUTE_RE call.
int pcre_get_stringnumber(const pcre *code, const char *name); Gets the number of a captured substring by name.	An equivalent procedure is not provided; all substrings are given to the application in the EXECUTE_RE call.
int pcre_get_stringtable_entries(const pcre *code, const char *name, char **first, char **last); Gets full details of all captured substrings for a given name.	An equivalent procedure is not provided; all substrings are given to the application in the EXECUTE_RE call.
int pcre_get_substring(const char *subject, int *ovector, int stringcount, int stringnumber, const char **stringptr); Extracts a captured substring by number.	An equivalent procedure is not provided; all substrings are given to the application in the EXECUTE_RE call.

Table 10–1. PCRE Functions Mapped to WEBAPPSUPPORT Procedures

PCRE	WEBAPPSUPPORT
int pcre_get_substring_list(const char *subject, int *ovector, int stringcount, const char ***listptr); Extracts a captured list of substrings by number.	An equivalent procedure is not provided; all substrings are given to the application in the EXECUTE_RE call.
const unsigned char *pcre_maketables(void); Builds a set of external tables in the current locale for passing to pcre_compile, pcre_exec, or pcre_dfa_exec.	An equivalent procedure is not provided.
int pcre_refcount(pcre *code, int adjust); Maintains a reference count in a data block that contains a compiled pattern.	An equivalent procedure is not provided.
pcre_extra *pcre_study(const pcre *code, int options, const char **errptr); Analyzes a compiled pattern to speed up matching.	Studying patterns is an option of the COMPILE_RE_PATTERN procedure (STUDY option of the SET_RE_OPTION procedure). The studied data is stored in the WEBAPPSUPPORT library along with the compiled pattern.
char *pcre_version(void); Returns a string containing the PCRE version and its date of release.	The GET_RE_VERSION procedure provides similar functionality.
void *(*pcre_malloc)(size_t); void (*pcre_free)(void *); void *(*pcre_stack_malloc)(size_t); void (*pcre_stack_free)(void *); int (*pcre_callout)(pcre_callout_block *);	These procedures are not needed by the application.

WEBAPPSUPPORT Library Procedures for Regular Expressions

The procedures in this section each describe an entry point compatible with COBOL with all uppercase and with underscores, for example EXECUTE_RE, and an entry point compatible with ALGOL with mixed upper- and lowercase containing no underscores, for example executeRE.

COMPILE_RE_PATTERN

Compiles a pattern for use with the EXECUTE_RE procedure.

See the SET_RE_OPTION procedure for options that affect pattern compilation.

Syntax

```
INTEGER PROCEDURE COMPILE_RE_PATTERN
                   (PATTERN, PATTERN_START, PATTERN_LENGTH, PATTERN_TAG,
                      ERROR_CODE, ERROR_TEXT);
  EBCDIC ARRAY      PATTERN,      ERROR_TEXT [0];
  INTEGER                    PATTERN_START, PATTERN_LENGTH, PATTERN_TAG,
                      ERROR_CODE;

INTEGER PROCEDURE compileREpattern
                   (PATTERN, PATTERN_START, PATTERN_LENGTH, PATTERN_TAG,
                      ERROR_CODE, ERROR_TEXT);
  VALUE                      PATTERN_START, PATTERN_LENGTH;
  EBCDIC ARRAY      PATTERN,      ERROR_TEXT [*];
  INTEGER                    PATTERN_START, PATTERN_LENGTH, PATTERN_TAG,
                      ERROR_CODE;
```

Parameters

PATTERN is the pattern string to be compiled in the application character set.

PATTERN_START is the zero-based offset into PATTERN where the pattern string starts.

PATTERN_LENGTH is the length in bytes of the pattern string. If the application character set is translatable to 7-bit ASCII, PATTERN_LENGTH can be zero and PATTERN contains a string terminated by blanks or a null byte.

PATTERN_TAG is the tag that references the compiled pattern.

ERROR_CODE is the error code returned by PCRE when a compilation fails. Zero is returned if the compile succeeds.

ERROR_TEXT is the text for the error code in the application character set. This string is null if the compile succeeds.

Possible Return Values

In addition to the standard results, these possible values can be returned.

Value	Description
-100	The regular expression pattern is invalid.
-101	The maximum number of patterns stored is exceeded.

EXECUTE_RE

Executes a regular expression against a subject string using a pattern compiled with the COMPILE_RE_PATTERN procedure.

See the SET_RE_OPTION procedure for options that affect pattern execution.

Syntax

```
INTEGER PROCEDURE EXECUTE_RE
                   (PATTERN_TAG, SUBJECT, SUBJECT_START, SUBJECT_LEN,
                      NUM_SUBSTRINGS, SUBSTRING_OFFSETS,
                        SUBSTRING_LENS, MAX_SUBSTRING_LEN,
                          SUBSTRING_BUFFER);
  INTEGER           PATTERN_TAG,          SUBJECT_START, SUBJECT_LEN,
                      NUM_SUBSTRINGS;   MAX_SUBSTRING_LEN;
  EBCDIC  ARRAY                  SUBJECT,
                          SUBSTRING_BUFFER [0];
  INTEGER ARRAY                       SUBSTRING_OFFSETS,
                        SUBSTRING_LENS [0];

INTEGER PROCEDURE executeRE
                   (PATTERN_TAG, SUBJECT, SUBJECT_START, SUBJECT_LEN,
                      NUM_SUBSTRINGS, SUBSTRING_OFFSETS,
                        SUBSTRING_LENS, MAX_SUBSTRING_LEN,
                          SUBSTRING_BUFFER);
  VALUE             PATTERN_TAG,          SUBJECT_START, SUBJECT_LEN,
MAX_SUBSTRING_LEN;
  INTEGER           PATTERN_TAG,          SUBJECT_START, SUBJECT_LEN,
                      NUM_SUBSTRINGS;   MAX_SUBSTRING_LEN;
  EBCDIC  ARRAY                  SUBJECT,
                          SUBSTRING_BUFFER [*];
  INTEGER ARRAY                       SUBSTRING_OFFSETS,
                        SUBSTRING_LENS [*];
```

Parameters

PATTERN_TAG is the tag that references the compiled pattern.

SUBJECT is the subject string in the application character set.

SUBJECT_START is the zero-based offset into SUBJECT where the subject string starts.

SUBJECT_LEN is the length in bytes of the subject string. If zero, SUBJECT contains a string terminated by blanks or a null byte.

NUM_SUBSTRINGS is the number of substrings that the expression yielded.

SUBSTRING_OFFSETS is the array of zero-based offsets into SUBJECT where each resulting substring starts.

SUBSTRING_LENS is the array of lengths for each resulting substring.

MAX_SUBSTRING_LEN is the maximum length of a substring returned in SUBSTRING_BUFFER. If less than or equal to zero, no substrings are copied into SUBSTRING_BUFFER.

SUBSTRING_BUFFER is the buffer in the application character set where each substring is stored.

Possible Return Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	No matching strings were found.
1	One or more matching substrings were found.
–102	The pattern tag is invalid.
–103	The UTF-8 sequence is invalid.
–104	The NEWLINE combination is invalid.

Example

Here is an example of SUBSTRING_BUFFER used for this procedure in COBOL:

```
01 SUBSTRING-BUFFER.
  03 SUBSTRING-LIST OCCURS 10 TIMES.
    05 SUBSTRING  PIC X(30).
```

The call to EXECUTE_RE passes SUBSTRING-BUFFER with MAX_SUBSTRING_LEN set to 30.

FREE_RE_PATTERN

Frees a pattern no longer needed, which frees up resources in WEBAPPSUPPORT.

Syntax

```
INTEGER PROCEDURE FREE_RE_PATTERN
                  (PATTERN_TAG);
  INTEGER          PATTERN_TAG;

INTEGER PROCEDURE freeREpattern
                  (PATTERN_TAG);
  VALUE            PATTERN_TAG;
  INTEGER          PATTERN_TAG;
```

Parameters

PATTERN_TAG is the tag that references the compiled pattern.

Possible Return Values

In addition to the standard results, these possible values can be returned.

Value	Description
–102	The pattern tag is invalid.

GET_RE_VERSION

Returns the PCRE version supported.

Syntax

```
INTEGER PROCEDURE GET_RE_VERSION (VERSION);
  EBCDIC ARRAY                    VERSION [0];

INTEGER PROCEDURE getREversion   (VERSION);
  EBCDIC ARRAY                    VERSION [*];
```

Parameters

VERSION is the PCRE version as a string in the application character set. For example: 8.02.

SET_RE_OPTION

Sets an option for Regular Expressions compilation or processing.

Syntax

```
INTEGER PROCEDURE SET_RE_OPTION
                  (OPTION, OPTION_VALUE, OPTION_STRING);
  INTEGER          OPTION, OPTION_VALUE;
  EBCDIC ARRAY                           OPTION_STRING [0];

INTEGER PROCEDURE setREoption
                  (OPTION, OPTION_VALUE, OPTION_STRING);
  VALUE            OPTION, OPTION_VALUE;
  INTEGER          OPTION, OPTION_VALUE;
  EBCDIC ARRAY                           OPTION_STRING [*];
```

Parameters

OPTION is the option being set. The following options are supported.

1 (STUDY)

Controls whether or not to perform an extra study of a pattern when compiling the pattern. Using this option might improve the performance of executing regular expressions

If the value is 0, this option specifies to not study the pattern. This value is the default.

If the value is 1, this option specifies to study the pattern when compiling.

2 (MATCH_ALGORITHM)

Controls the searching for pattern matches in the subject string.

If the value is 0, this option specifies the standard PCRE matching algorithm. The searching stops at the first match to the pattern. The substrings of the match are also returned. This value is the default.

If the value is 1, this option specifies an alternative PCRE matching algorithm. The searching finds all matches to the pattern. The matches are returned in the substring fields. Substrings of the matches are not returned.

3 (ANCHORED)

Controls whether or not to anchor the matching to the first matching point in the string. If the value is set to 1 when the pattern is compiled, resetting this option before an execute call has no effect.

If the value is 0, the pattern is not anchored. This value is the default.

If the value is 1, the pattern is anchored and matches only at the first matching point in the string.

4 (BSR)

Controls the interpretation of the \R escape sequence in a pattern.

If the value is 0, the \R escape sequence is CRLF. This value is the default.

If the value is 1, the \R escape sequence is any combination of LF, CR, or CRLF.

If the value is 2, the \R escape sequence is any Unicode newline sequence. See the NEWLINE option for a description of Unicode newline sequences.

5 (CASELESS)

Controls whether or not the pattern matches both upper- and lowercase characters.

If the value is 0, pattern matching is case sensitive. This value is the default.

If the value is 1, pattern matching is not case sensitive.

6 (DOLLAR_ENDONLY)

Controls whether or not a dollar meta-character in the pattern matches only at the end of the subject string. This option is ignored if the MULTILINE option is set to 1.

If the value is 0, a dollar meta-character matches immediately before the end of each newline. This value is the default.

If the value is 1, a dollar meta-character matches only at the end of the subject string.

7 (DOTALL)

Controls the matching of a dot meta-character in a pattern. A negative class such as [^a] always matches newline characters, independent of the setting of this option.

If the value is 0, a dot meta-character does not match when the current position is at a newline. This value is the default.

If the value is 1, a dot meta-character in the pattern matches all characters, including those that indicate newline.

8 RESERVED

9 (EXTENDED)

Controls the handling of whitespace characters in a pattern. Whitespace does not include the VT character (ASCII code 11).

If the value is 0, the whitespace characters in a pattern are not ignored. This value is the default.

If the value is 1, the whitespace characters in a pattern are ignored unless escaped or inside a character class.

10 (EXTRA)

Controls handling of a backslash character in a pattern that is followed by a character with no special meaning.

If the value is 0, a backslash followed by a character with no special meaning is treated as a literal. This value is the default.

If the value is 1, a backslash followed by a character with no special meaning causes an error.

11 (FIRSTLINE)

Controls whether or not an unanchored pattern is required to match at the first line.

If the value is 0, the pattern is not required to match at the first line. This value is the default.

If the value is 1, the pattern is required to match before or at the first newline in the subject string. The matched text can continue over a newline.

12 (JAVASCRIPT_COMPAT)

Controls whether to be compatible with Perl or JavaScript for the cases that are different between the two.

If the value is 0, this value retains the Perl compatibility. This value is the default.

If the value is 1, this value changes the behavior to be compatible with JavaScript.

13 (MULTILINE)

Controls whether or not the subject string is treated as a single line or as multiple lines. Setting this option has no effect in these three cases: if no newlines exist in a subject string; if no occurrences of ^ exist; or if $ exists in a pattern. See also the DOLLAR_ENDONLY option.

If the value is 0, the subject string is treated as a single line of characters. The "start of line" meta-character (^) matches only at the start of the string, while the "end of line" meta-character ($) matches only at the end of the string, or before a terminating newline (unless PCRE_DOLLAR_ENDONLY is set). This value is the default

If the value is 1, the "start of line" and "end of line" constructs match immediately following or immediately before internal newlines in the subject string, respectively, as well as at the very start and end.

14 (NEWLINE)

Controls the interpretation of newlines.

If the value is 0, the character sequence CRLF is a newline. This value is the default

If the value is 1, the character sequence CR is a newline.

If the value is 2, the character sequence LF is a newline.

If the value is 3, any occurrence of the character sequence CR, LR, or CRLF is a newline.

If the value is 4, any Unicode newline sequence is a newline. The Unicode newline sequences are the settings in OPTION_VALUE = 3, plus the single characters VT (vertical tab, U+000B), FF (form feed, U+000C), NEL (next line, U+0085), LS (line separator, U+2028), and PS (paragraph separator, U+2029).

15 (NO_AUTO_CAPTURE)

Controls whether or not to make use of numbered capturing parenthesis in a pattern.

If the value is 0, the use of numbered capturing parenthesis in a pattern is enabled. This value is the default.

If the value is 1, the use of numbered capturing parenthesis in a pattern is disabled.

16 (UNGREEDY)

Controls the "greediness" of the quantifiers.

If the value is 0, the quantifiers are greedy by default. This value is the default.

If the value is 1, the quantifiers are not greedy by default.

OPTION_STRING use is described in the previous option descriptions. The application should set it to a null string for any settings that do not define a use for it.

Possible Return Values

In addition to the standard results, these possible values can be returned.

Value	Description
0	Option or value is not supported.

These PCRE options are not exported to the application.

Option	Explanation
PCRE_UTF8	This option is set internally if the pattern or subject strings are passed to PCRE as UTF-8 strings.
PCRE_NO_UTF8_CHECK	If WEBAPPSUPPORT performs the translation to UTF-8 as part of compiling the pattern or executing the expression, it sets this option internally. If the application supplies the UTF-8 string this option is reset to PCRE.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

3826 5286–007