



# Rocket UniVerse

## BASIC Extensions

*Version 11.2.3*

April 2014

UNV-1123-BASX-1

# Notices

## Edition

**Publication date:** April 2013

**Book number:** UNV-1123-BASX-1

**Product version:** Rocket UniVerse V11.2.3

## Copyright

© Rocket Software, Inc. or its affiliates 1985-2014. All Rights Reserved.

## Trademarks

Rocket is a registered trademark of Rocket Software, Inc. For a list of Rocket registered trademarks go to: [www.rocketsoftware.com/about/legal](http://www.rocketsoftware.com/about/legal). All other products or services mentioned in this document may be covered by the trademarks, service marks, or product names of their respective owners.

## Examples

This information might contain examples of data and reports. The examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

## License agreement

This software and the associated documentation are proprietary and confidential to Rocket Software, Inc., are furnished under license, and may be used and copied only in accordance with the terms of such license.

**Note:** This product may contain encryption technology. Many countries prohibit or restrict the use, import, or export of encryption technologies, and current use, import, and export regulations should be followed when exporting this product.

## Contact information

Website: [www.rocketsoftware.com](http://www.rocketsoftware.com)

Rocket Software, Inc. Headquarters

77 4<sup>th</sup> Avenue, Suite 100

Waltham, MA 02451-1468

USA

Tel: +1 781 577 4321

Fax: +1 617 630 7100

# Contacting Global Technical Support

If you have current support and maintenance agreements with Rocket Software, you can access the Rocket Customer Portal to report and track a problem, to submit an enhancement request or question, or to find answers in the U2 Knowledgebase. The Rocket Customer Portal is the primary method of obtaining support.

To log in to the Rocket Customer Portal, go to:

[www.rocketsoftware.com/support](http://www.rocketsoftware.com/support)

If you do not already have a Rocket Customer Portal account, you can request one by clicking **Need an account?** on the Rocket Customer Portal login page.

Alternatively, you can contact Global Technical Support by email or by telephone:

**Email:** [u2support@rocketsoftware.com](mailto:u2support@rocketsoftware.com)

**Telephone:**

North America	+1 800 729 3553
United Kingdom/France	+44 (0) 800 773 771 or +44 (0) 20 8867 3691
Europe/Africa	+44 (0) 20 8867 3692
Australia	+1 800 707 703 or +61 (0) 29412 5450
New Zealand	+0800 505 515

# Table of Contents

## Chapter 1

### Chapter 1: Using the Socket Interface

Socket function error return codes . . . . .	1-3
Getting a socket error message . . . . .	1-7
Opening a socket. . . . .	1-8
Opening a secure socket . . . . .	1-9
Closing a socket . . . . .	1-11
Getting information from a socket. . . . .	1-12
Reading from a socket . . . . .	1-14
Writing to a socket . . . . .	1-16
Setting the value for a socket option . . . . .	1-18
Getting the value of a socket option . . . . .	1-20
Initializing a server side socket connection . . . . .	1-23
Initializing a secure server side socket connection . . . . .	1-25
Accepting an incoming connection attempt on the server side . . . . .	1-27
Protocol logging . . . . .	1-29
Socket API example. . . . .	1-31

## Chapter 2

### Chapter 2: Using CallHTTP

Configuring the default HTTP settings . . . . .	2-3
Getting the current HTTP default settings . . . . .	2-7
Creating an HTTP request . . . . .	2-8
Creating a secure HTTP request . . . . .	2-11
Setting additional headers for a request . . . . .	2-14
Adding a parameter to the request . . . . .	2-16
Submitting a request . . . . .	2-18
HTTP digest authentication . . . . .	2-21
How Digest Authentication Works . . . . .	2-21
WWW-authenticate response/proxy-authenticate header parameters	2-23
Authorization request header parameters . . . . .	2-24
Using digest authentication with Microsoft IIS web servers . . . . .	2-25
Secure HTTP authentication through proxy . . . . .	2-26

Multiple same-name headers in response messages . . . . .	2-27
<b>Chapter 3</b>	<b>Chapter 3: Using WebSphere MQ with UniVerse</b>
In this chapter . . . . .	3-2
Preface . . . . .	3-3
Overview of messaging . . . . .	3-4
Overview of IBM WebSphere MQ . . . . .	3-5
U2 WebSphere MQ API . . . . .	3-7
CallMQI setup and configuration . . . . .	3-8
Programming with CallMQI . . . . .	3-11
U2MQI.H INCLUDE file . . . . .	3-12
Error handling in CallMQI . . . . .	3-13
Encoding message IDs and correlation IDs . . . . .	3-13
CallMQI programmatic interfaces . . . . .	3-14
Connecting to a queue manager . . . . .	3-15
Opening a queue . . . . .	3-17
Retrieving a message from a queue . . . . .	3-18
Putting a message on a queue . . . . .	3-21
Putting a single message on a queue . . . . .	3-23
Querying the attributes . . . . .	3-25
Closing a queue. . . . .	3-26
Disconnecting from a queue manager . . . . .	3-28
Returning an error message . . . . .	3-29
CallMQI programming examples . . . . .	3-31
CallMQI dynamic array structures . . . . .	3-36
MQOD dynamic array structure . . . . .	3-36
MQMD dynamic array structure . . . . .	3-39
MQGMO dynamic array structure . . . . .	3-42
MQPMO dynamic array structure . . . . .	3-45
WebSphere MQ API for UniData and UniVerse . . . . .	3-48
WebSphere MQ Application Messaging Interface . . . . .	3-48
Session. . . . .	3-49
Services . . . . .	3-49
Policies. . . . .	3-50
Message objects. . . . .	3-50
Messaging styles . . . . .	3-51
CallAMI setup and configuration . . . . .	3-53
Requirements . . . . .	3-53
Platform availability . . . . .	3-53
Setting up the environment for CallAMI . . . . .	3-53

Configurations . . . . .	3-54
CallAMI programmatic interfaces . . . . .	3-57
Initialize an AMI session . . . . .	3-57
Receiving a message . . . . .	3-58
Receiving a request . . . . .	3-62
Sending a message . . . . .	3-65
Sending a request . . . . .	3-67
Sending a response . . . . .	3-69
Terminating a session . . . . .	3-71
CallAMI programming examples. . . . .	3-73
Additional reading. . . . .	3-83

## **Chapter 4**

## **Chapter 4: Creating XML documents**

XML for U2 UniVerse. . . . .	4-2
Document type definitions . . . . .	4-3
XML schema . . . . .	4-3
The Document Object Model (DOM) . . . . .	4-3
Well-formed and valid XML documents. . . . .	4-4
Creating an XML document from RetrieVe . . . . .	4-5
Create the &XML& file. . . . .	4-5
Mapping modes . . . . .	4-5
XML configuration file. . . . .	4-13
xmlconfig parameters . . . . .	4-15
The Mapping File . . . . .	4-24
Distinguishing elements . . . . .	4-27
Root element attributes . . . . .	4-27
Association elements . . . . .	4-34
Mapping file example . . . . .	4-35
How data is mapped . . . . .	4-39
Mapping example . . . . .	4-41
TCL commands for XML. . . . .	4-43
Session-level TCL commands . . . . .	4-43
XMLSETOPTIONS . . . . .	4-43
XMLGETOPTIONS . . . . .	4-45
XMLGETOPTIONVALUE. . . . .	4-46
Existing TCL command affected by XMLSETOPTIONS command or XMLSetOptions() API . . . . .	4-47
Creating an XML document using RetrieVe . . . . .	4-49
Examples . . . . .	4-50
Creating an XML document with UniVerse SQL . . . . .	4-59

Processing rules for UniVerse SQL SELECT statements . . . . .	4-60
XML limitations in UniVerse SQL . . . . .	4-62
Examples . . . . .	4-62
Creating an XML document through UniVerse BASIC . . . . .	4-71
Using the XMLExecute() function . . . . .	4-72
XMLSetOptions. . . . .	4-75
XMLGetOptions . . . . .	4-77
XMLGetOptionValue. . . . .	4-78
Existing APIs affected by XML options. . . . .	4-93
UniVerse BASIC example . . . . .	4-121
<b>Chapter 5</b>	<b>Chapter 5: Receiving XML documents</b>
Receiving an XML document through UniVerse BASIC . . . . .	5-2
Defining extraction rules . . . . .	5-2
Defining the XPath . . . . .	5-4
Extracting XML data through UniVerse BASIC . . . . .	5-13
Displaying an XML document through Retrieve . . . . .	5-18
Displaying an XML document through UniVerse SQL . . . . .	5-22
<b>Chapter 6</b>	<b>Chapter 6: The Simple Object Access Protocol</b>
SOAP components. . . . .	6-2
The SOAP API for UniVerse BASIC . . . . .	6-4
Sending a SOAP request. . . . .	6-4
SOAP API for UniVerse BASIC programmatic interfaces . . . . .	6-5
SOAPSetDefault . . . . .	6-5
SOAPGetDefault . . . . .	6-7
SOAPCreateRequest . . . . .	6-8
SOAPCreateSecureRequest. . . . .	6-10
SOAPSetParameters . . . . .	6-12
SOAPSetRequestHeader. . . . .	6-14
SOAPSetRequestBody . . . . .	6-15
SOAPSetRequestContent . . . . .	6-16
SOAPRequestWrite . . . . .	6-17
SOAPSubmitRequest. . . . .	6-19
SOAPGetResponseHeader . . . . .	6-20
SOAPGetFault . . . . .	6-21
protocolLogging . . . . .	6-23
SOAP API for UniVerse BASIC example . . . . .	6-25

**Chapter 7****Chapter 7: The Document Object Model**

XPath and the Document Object Model . . . . .	7-3
A sample XML document. . . . .	7-3
Opening and closing a DOM document . . . . .	7-4
Navigating the DOM tree . . . . .	7-4
Building DOM trees from scratch . . . . .	7-5
Transforming XML documents . . . . .	7-7
XML for UniVerse BASIC API programmatic interfaces . . . . .	7-11
XDOMOpen . . . . .	7-11
XDOMCreateNode . . . . .	7-12
XDOMCreateRoot . . . . .	7-13
XDOMWrite . . . . .	7-14
XDOMClose . . . . .	7-15
XDOMValidate . . . . .	7-16
XDOMLocate . . . . .	7-18
XDOMLocateNode . . . . .	7-19
. . . . .	
XDOMRemove . . . . .	7-25
. . . . .	
XDOMAppend . . . . .	7-26
XDOMInsert . . . . .	7-28
XDOMReplace . . . . .	7-29
XDOMAddChild. . . . .	7-31
XDOMClone . . . . .	7-32
XDOMTransform. . . . .	7-33
XDOMGetNodeValue . . . . .	7-35
XDOMGetNodeType . . . . .	7-36
XDOMGetAttribute . . . . .	7-37
XDOMGetOwnerDocument . . . . .	7-38
XDOMGetUserData. . . . .	7-39
XDOMSetNodeValue . . . . .	7-40
XDOMSetUserData . . . . .	7-41
XMLGetError . . . . .	7-42

**Chapter 8****Chapter 8: Data transfer between XML documents and UniVerse files**

Transferring Data from XML to the database . . . . .	8-3
Populating the database . . . . .	8-11
Populating the database from TCL. . . . .	8-11
Populating the database using the UniVerse BASIC XMAP API. . . . .	8-13

<b>Chapter 9</b>	<b>Chapter 9: The XML/DB tool</b>	
	The XMAP API . . . . .	8-14
	XMAPOpen function . . . . .	8-14
	XMAPClose function . . . . .	8-16
	XMAPCreate function . . . . .	8-16
	XMAPReadNext function . . . . .	8-17
	XMAPAppendRec function. . . . .	8-18
	XMAPToXMLDoc function. . . . .	8-19
	Examples . . . . .	8-20
	Transferring data from the database to XML . . . . .	8-23
	Creating an XML document from TCL. . . . .	8-23
<b>Chapter 10</b>	<b>Chapter 10: U2 Dynamic Object</b>	
	U2 Dynamic Object . . . . .	10-2
	XML Support in U2 Dynamic Object . . . . .	10-4
	Converting from XML to JSON . . . . .	10-4

Converting from UDO to XML . . . . .	10-6
U2 Dynamic Object API . . . . .	10-7
UDO definitions . . . . .	10-7
UDO functions . . . . .	10-10
UDORead . . . . .	10-10
UDOWrite . . . . .	10-11
UDOSetOption . . . . .	10-11
UDOGetOption . . . . .	10-12
UDOCClone . . . . .	10-12
UDOFree . . . . .	10-13
UDOGetProperty . . . . .	10-14
UDO SetProperty . . . . .	10-14
UDODeleteProperty . . . . .	10-15
UDOGetPropertyNames . . . . .	10-16
UDOGetNextProperty . . . . .	10-16
UDOCCreate . . . . .	10-18
UDOGetType . . . . .	10-18
UDOIsTypeOf . . . . .	10-19
UDOArrayGetSize . . . . .	10-19
UDOArrayGetItem . . . . .	10-20
UDOArraySetItem . . . . .	10-21
UDOArrayInsertItem . . . . .	10-22
UDOArrayAppendItem . . . . .	10-23
UDOArrayDeleteItem . . . . .	10-24
UDOArrayGetNextItem . . . . .	10-24
UDOGetLastError . . . . .	10-25
XML2UDO options and UDO2XML options . . . . .	10-27
UDOOPTION_XML2UDO_INFERNUMBER . . . . .	10-27
UDOOPTION_XML2UDO_INFERBOOLEAN . . . . .	10-28
UDOOPTION_XML2UDO_INFERNULL . . . . .	10-28
UDOOPTION_XML2UDO_EMPTY2NULL . . . . .	10-29
UDOOPTION_XML2UDO_TRIMWHITESPACE . . . . .	10-30
UDOOPTION_XML2UDO_CASEINSENSITIVE . . . . .	10-31
UDOOPTION_XML2UDO_KEEPROOT . . . . .	10-32
UDOOPTION_UDO2XML_XMLSTYLE . . . . .	10-33
UDOOPTION_UDO2XML_NAMESPACE and UDOOPTION_UDO2XML_NAMESPACEPREFIX . . . . .	10-34
UDOOPTION_UDO2XML_ROOTNAME . . . . .	10-35
UDOOPTION_UDO2XML_NULL2EMPTY . . . . .	10-36



# Chapter 1: Using the Socket Interface

Socket function error return codes . . . . .	1-3
Getting a socket error message . . . . .	1-7
Opening a socket . . . . .	1-8
Opening a secure socket. . . . .	1-9
Closing a socket . . . . .	1-11
Getting information from a socket . . . . .	1-12
Reading from a socket . . . . .	1-14
Writing to a socket. . . . .	1-16
Setting the value for a socket option . . . . .	1-18
Getting the value of a socket option. . . . .	1-20
Initializing a server side socket connection . . . . .	1-23
Initializing a secure server side socket connection . . . . .	1-25
Accepting an incoming connection attempt on the server side . . . . .	1-27
Protocol logging . . . . .	1-29
Socket API example . . . . .	1-31

The UniVerse BASIC Socket API provides the user with the capability of interacting with an application running on another machine via the sockets interface. The Socket API enables you to write distributed UniVerse applications. For example, one application, written in the server side socket interface can function as the server while others can function as clients. The server and the clients can cooperate on tasks through socket communication. This is an efficient way for UniVerse BASIC applications to communicate, and is easy to implement. The Socket functions are not in order of how they would normally be implemented. Refer to "["Socket API example"](#)" for more information on using the Socket API functions.

---

## Socket function error return codes

The following error return codes are used for all socket-related functions described below. Note that only numeric code should be used in UniVerse BASIC programs.

The following table describes each error code and its meaning.

Error Code	Definition
0 -SCK_ENOERROR	No error.
1 - SCK_ENOINITIALISED	On Windows platforms, a successful WSASStartup() call must occur before using this function.
2 - SCK_ENETDOWN	The network subsystem has failed.
3 - SCK_EFAULT	The <i>addrlen</i> parameter is too small or <i>addr</i> is not a valid part of the user address space.
4 - SCK_ENOTCONN	The socket is not connected.
5 - SCK_EINTR	The (blocking) call was cancelled. (NT: through WSACancelBlockign-Call).
6 - SCK_EINPROGRESS	A blocking Windows Sockets 1.1 call is in progress, or the service provider is still processing a callback function.
7 - SCK_EINVAL	This can be caused by several conditions. The listen function was not invoked prior to accept, the socket has not been bound with bind, an unknown flag was specified, or MSG_OOB was specified for a socket with SO_OOBINLINE enabled or (for byte stream sockets only) len was zero or negative.
8 - SCK_EMFILE	The queue is nonempty upon entry to accept and there are no descriptors available.
9 - SCK_ENOBUFS	No buffer space is available.
10 - SCK_ENOTSOCK	The descriptor is not a socket.

---

Socket Function Error Return Codes

Error Code	Definition
11 - SCK_EOPNOTSUPP	The referenced socket is not a type that supports connection-oriented service.
12 - SCK_EWOULDBLOCK	The socket is marked as nonblocking and the requested operation would block.
13 - SCK_ENETRESET	The connection has been broken due to the keep-alive activity detecting a failure while the operation was in progress.
14 - SCK_ESHUTDOWN	The socket has been shut down.
15 - SCK_EMSGSIZE	(For recv()) The message was too large to fit into the specified buffer and was truncated, or (for send()) the socket is message oriented, and the message is larger than the maximum supported by the underlying transport.
16 - SCK_ETIMEDOUT	The virtual circuit was terminated due to a time-out or other failure.
17 - SCK_ECONNABORTED	The connection has been dropped, because of a network failure or because the system on the other end went down without notice.
18 - SCK_ECONNRESET	The virtual circuit was reset by the remote side executing a hard or abortive close. For UDP sockets, the remote host was unable to deliver a previously sent UDP datagram and responded with a "Port Unreachable" ICMP packet. The application should close the socket as it is no longer usable.
19 - SCK_EACCES	The requested address is a broadcast address, but the appropriate flag was not set. Call <b>setSocketOption()</b> with the BROADCAST parameter to allow the use of the broadcast address.
20 - SCK_EHOSTUNREACH	The remote host cannot be reached from this host at this time.
21 - SCK_ENOPROTOOPT	The option is unknown or unsupported for the specified provider or socket.

#### Socket Function Error Return Codes (Continued)

Error Code	Definition
22 - SCK_ESYSNOTREADY	Indicates that the underlying network subsystem is not ready for network communication.
23 -SCK_EVER NOTSUPPORTED	The version of Windows Sockets support requested is not provided by this particular Windows Sockets implementation.
24 - SCK_EPROCLIM	Limit on the number of tasks supported by the Windows Sockets implementation has been reached.
25 - SCK_EAFNOSUPPORT	The specified address family is not supported.
26 - SCK_EPROTONOSUPPORT	The specified protocol is not supported.
27 - SCK_EPROTOTYPE	The specified protocol is the wrong type for this socket.
28 - SCK_ESOCKTNOSUPPORT	The specified socket type is not supported in this address family.
29 - SCK_EBADF	Descriptor socket is not valid.
30 - SCK_EHOST_NOT_FOUND	Authoritative Answer Host not found.
31 - SCK_ETRY AGAIN	Non-authoritative Host not found, or server failure.
32 - SCK_ENO_RECOVERY	A nonrecoverable error occurred.
33 - SCK_ENO_DATA	Valid name, no data record of requested type.
34 - SCK_EACCESS	Attempt to connect datagram socket to broadcast address failed because <b>setSocketOption()</b> BROADCAST is not enabled.
35 - SCK_EADDRINUSE	A process on the machine is already bound to the same fully-qualified address and the socket has not been marked to allow address reuse with REUSEADDR. (See the REUSEADDR socket option under <b>setSocketOption()</b> ).

#### Socket Function Error Return Codes (Continued)

---

<b>Error Code</b>	<b>Definition</b>
36 - SCK_EADDRNOTAVAIL	The specified address is not a valid address for this machine.
37 - SCK_EISCONN	The socket is already connected.
38 - SCK_EALREADY	A nonblocking connect call is in progress on the specified socket.
39 - SCK_ECONNREFUSED	The attempt to connect was forcefully rejected.
40 - SCK_EMALLOC	Memory allocation error.
41 - SCK_ENSLMAP	NLS map not found, or unmapped characters encountered.
42 - SCK_EUNKNOWN	Other unknown errors.
101	Invalid security context handle.
102	SSL/TLS handshake failure (unspecified, peer is not SSL aware).
103	Requires client authentication but does not have a certificate in context.
104	Unable to authenticate server.
105	Client authentication failure.
106	Peer not speaking SSL.
107	Encryption error.
108	Decryption error.

---

**Socket Function Error Return Codes (Continued)**

---

## Getting a socket error message

Use the **getSocketErrorMessage()** function to translate an error code into a text error message.

This function works with all socket functions. The return status of those functions can be passed into this function to get the corresponding error message.

### Syntax

**getSocketErrorMessage(*errCode*, *errMsg*)**

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>errCode</i>	The status return code sent by the socket functions.
<i>errMsg</i>	A string containing corresponding error text.

#### getSocketErrorMessage Parameters

The following table describes the return status of each mode.

Return Code	Description
0	Success.
1	Invalid error code.

#### Return Code Status

## Opening a socket

Use the **openSocket()** function to open a socket connection in a specified mode and return the status.

### Syntax

**openSocket(*name\_or\_IP*, *port*, *mode*, *timeout*, *socket\_handle*)**

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>name_or_IP</i>	DNS name (x.com) or IP address of a server.
<i>port</i>	Port number. If the port number is specified as a value <= 0, CallHTTP defaults to a port number of 40001.
<i>mode</i>	2: non-blocking mode 1: blocking mode
<i>timeout</i>	The timeout value, expressed in milliseconds. If you specify mode as 0, timeout will be ignored.
<i>socket_handle</i>	A handle to the open socket.

#### openSocket Parameters

The following table describes the return status of each mode.

Return Code	Description
0	Success.
99	UniVerse failed to obtain a license for an interactive PHANTOM process.
Non-zero	See Socket Function Error Return Codes.

#### Return Code Status

---

## Opening a secure socket

Use the **openSecureSocket()** function to open a secure socket connection in a specified mode and return the status.

This function behaves exactly the same as the **openSocket()** function, except that it returns the handle to a socket that transfers data in a secured mode (SSL/TLS).

All parameters (with the exception of *context*) have the exact meaning as the **openSocket()** parameters. *Context* must be a valid security context handle.

Once the socket is opened, any change in the associated security context will not affect the established connection.

## Syntax

```
openSecureSocket(name_or_IP, port, mode, timeout, socket_handle,  
context)
```

## Parameters

The following table describes each parameter of the syntax.

---

Parameter	Description
<i>name_or_IP</i>	DNS name (x.com) or IP address of a server.
<i>port</i>	Port number. If the port number is specified as a value <= 0, CallHTTP defaults to a port number of 40001.
<i>mode</i>	0:non-blocking mode 1:blocking mode
<i>timeout</i>	The timeout value, expressed in milliseconds. If you specify mode as 0, timeout will be ignored.
<i>socket_handle</i>	A handle to the open socket.
<i>context</i>	A handle to the security context

---

### openSecureSocket Parameters

The following table describes the return status of each mode.

<b>Return Code</b>	<b>Description</b>
0	Success.
99	UniVerse failed to obtain a license for an interactive PHANTOM process.
1-41	See Socket Function Error Return Codes.
101	Invalid security context handle.
102	SSL/TLS handshake failure (unspecified, peer is not SSL aware).
103	Requires client authentication but does not have a certificate in the security context.
104	Unable to authenticate server.

**Return Code Status**

---

## Closing a socket

Use the **closeSocket()** function to close a socket connection.

### Syntax

**closeSocket(*socket\_handle*)**

Where *socket\_handle* is the handle to the socket you want to close.

The following table describes the status of each return code.

Return Code	Description
0	Success.
Non-zero	See Socket Function Error Return Codes.
Return Code Status	

## Getting information from a socket

Use the `getSocketInformation()` function to obtain information about a socket connection.

### Syntax

```
getSocketInformation(socket_handle, self_or_peer, socket_info)
```

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>socket_handle</i>	The handle to the open socket.
<i>self_or_peer</i>	Get information on the self end or the peer end of the socket. Specify 0 to return information from the peer end and non-zero for information from the self end.
<i>socket_info</i>	Dynamic Array containing information about the socket connection. For information about the elements of this dynamic array, see the following table.

#### getSocketInformation Parameters

The following table describes each element of the socket\_info dynamic array.

Element	Description
1	Open or closed
2	Name or IP
3	Port number
4	Secure or non secure
5	Blocking mode

The following table describes the status of each return code.

Element	Description
1	Open or closed
2	Name or IP
3	Port number
4	Secure or non secure
5	Blocking mode

**Return Code Status**

## Reading from a socket

Use the **readSocket()** function to read data in the socket buffer up to `max_read_size` characters.

### Syntax

```
readSocket(socket_handle, socket_data, max_read_size, time_out, blocking_mode, actual_read_size)
```

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<code>socket_handle</code>	The handle to the open socket.
<code>socket_data</code>	The data to be read from the socket.
<code>max_read_size</code>	The maximum number of characters to return. If this is 0, then the entire buffer should be returned.
<code>time_out</code>	The time (in milliseconds) before a return in blocking mode. This is ignored for non-blocking read.
<code>blocking_mode</code>	0:using current mode 1:blocking 2:non-blocking
<code>actual_read_size</code>	The number of characters actually read. -1 if error.

#### readSocket Parameters

The following table describes the return status of each mode.

Mode	Return Status
Non-Blocking	The function will return immediately if there is no data in the socket. If the <i>max_read_size</i> parameter is greater than the socket buffer then just the socket buffer will be returned.
Blocking	If there is no data in the socket, the function will block until data is put into the socket on the other end. It will return up to the <i>max_read_size</i> character setting.

**Return Mode Status**

The following table describes the status of each return code.

Return Code	Status
0	Success.
1-41	See Socket Function Error Return Codes.
107	Encryption error.
108	Decryption error.

**Return Code Status**

## Writing to a socket

Use the **writeSocket()** function to write data to a socket connection.

### Syntax

```
writeSocket(socket_handle, socket_data, time_out, blocking_mode,  
actual_write_size)
```

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>socket_handle</i>	The handle to the open socket.
<i>socket_data</i>	The data to be written to the socket.
<i>time_out</i>	The allowable time (in milliseconds) for blocking. This is ignored for a nonblocking write.
<i>blocking_mode</i>	0:using current mode 1:blocking 2:nonblocking
<i>actual_write_size</i>	The number of characters actually written.

#### writeSocket Parameters

The following table describes the return status of each mode.

Mode	Return Status
Blocking	The function will return only after all characters in <i>socket_data</i> are written to the socket.
Non-Blocking	The function may return with fewer character written than the actual length (in the case that the socket is full).

#### Return Mode Status

The following table describes the status of each return code.

<b>Return Code</b>	<b>Status</b>
0	Success.
1-41	See Socket Function Error Return Codes.
107	Encryption error.
108	Decryption error.

**Return Code Status**

## Setting the value for a socket option

The **setSocketOptions()** function sets the current value for a socket option associated with a socket of any type.

### Syntax

```
setSocketOptions(socket_handle, options)
```

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>socket_handle</i>	The socket handle from openSocket(), acceptSocket(), or initServer-Socket().
<i>options</i>	Dynamic Array containing information about the socket options and their current settings. The dynamic array is configured as:  optName1<VM>optValue1a[<VM>optValue1b]<FM> optName2<VM>optValue2a[<VM>optValue2b]<FM> optName3...  Where optName is specified by the caller and must be an option name string listed below. The first optValue specifies if the option is ON or OFF and must be one of two possible values: "1" for ON or "2" for OFF. The second optValue is optional and may hold additional data for a specific option. Currently, for the "LINGER" option it contains the delayed time (in milliseconds) before closing the socket. For all other options, it should not be specified as it will be ignored.

#### setSocketOptions Parameters

The following table describes the available options (case-sensitive) for `setSocketOptions`.

<b>Option</b>	<b>Description</b>
DEBUG	Enable/disable recording of debug information.
REUSEADDR	Enable/disable the reuse of a location address (default)
KEEPALIVE	Enable/disable keeping connections alive.
DONTROUTE	Enable/disable routing bypass for outgoing messages.
LINGER	Linger on close if data is present.
BROADCAST	Enable/disable permission to transmit broadcast messages.
OOBINLINE	Enable/disable reception of out-of-band data in band.
SNDBUF	Set buffer size for output (the default value depends on OS type).
RCVBUF	Set buffer size for input (the default value depends on OS type).

#### **setSocketOptions Options**

The following table describes the status of each return code.

<b>Return Code</b>	<b>Status</b>
0	Success.
Non-zero	See Socket Function Error Return Codes.

#### **Return Code Status**

## Getting the value of a socket option

The `getSocketOptions()` function gets the current value for a socket option associated with a socket of any type.

### Syntax

```
getSocketOptions(socket_handle, Options)
```

## Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>socket_handle</i>	The socket handle from openSocket(), acceptSocket(), or initServer-Socket().
<i>options</i>	A dynamic array containing information about the socket options and their current settings. When querying for options, the dynamic array is configured as:  optName1<FM> optName2<FM> optName...  When the options are returned, the dynamic array is configured as:  optName1<VM>optValue1a[<VM>optValue1b]<FM> optName2<VM>optValue2a[<VM>optValue2b]<FM> optName3...  Where optName contains an option name string listed below. The first optValue describes if the option is ON or OFF and must be one of two possible values: "1" for ON or "2" for OFF. The second optValue is optional and may hold additional data for a specific option. Currently, for option "LINGER," it contains the delayed time (in milliseconds) before closing the socket.

### getSocketOptions Parameters

The following table describes the available options (case-sensitive) for `getSocketOptions()`.

<b>Option</b>	<b>Description</b>
DEBUG	Enable/disable recording of debug information.
REUSEADDR	Enable/disable the reuse of a location address (default).
KEEPALIVE	Enable/disable keeping connections alive.
DONTROUTE	Enable/disable routing bypass for outgoing messages.
LINGER	Linger on close if data is present.
BROADCAST	Enable/disable permission to transmit broadcast messages.
OOBINLINE	Enable/disable reception of out-of-band data in band.
SNDBUF	Get buffer size for output (default 4KB).
RCVBUF	Get buffer size for input (default 4KB).
TYPE	Get the type of the socket. <i>Refer to the <code>socket.h</code> file for more information.</i>
ERROR	Get and clear error on the socket.

#### **getSocketOptions Options**

The following table describes the status of each return code.

<b>Return Code</b>	<b>Status</b>
0	Success.
Non-zero	See Socket Function Error Return Codes.
<b>Return Code Status</b>	

---

## Initializing a server side socket connection

Use the **initServerSocket()** function to create a connection-oriented (stream) socket. Associate this socket with an address (*name\_or\_IP*) and port number (*port*), and specify the maximum length the queue of pending connections may grow to.

### Syntax

**initServerSocket(*name\_or\_IP*, *port*, *backlog*, *svr\_socket*)**

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>name_or_IP</i>	DNS name (x.com) or IP address of a server or empty. Empty is equivalent to INADDR_ANY which means the system will choose one for you. Generally, this parameter should be left empty.
<i>port</i>	Port number. If the port number is specified as a value <= 0, CallHTTP defaults to a port number of 40001.
<i>backlog</i>	The maximum length of the queue of pending connections (for example, concurrent client-side connections).
<i>svr_socket</i>	The handle to the server side socket.

---

#### initServerSocket Parameters

The following table describes the status of each return code.

<b>Return Code</b>	<b>Status</b>
0	Success.
99	UniVerse failed to obtain a license for an interactive PHANTOM process.
Nonzero	See Socket Function Error Return Codes.
<b>Return Code Status</b>	

---

## Initializing a secure server side socket connection

Use the **initSecureServerSocket()** function to create a secured connection-oriented stream server socket. It does exactly the same as the **initServerSocket()** function except that the connection will be secure.

Once the server socket is opened, any change in the associated security context will not affect the opened socket.

### Syntax

**initSecureServerSocket(*name\_or\_IP*, *port*, *backlog*, *svr\_socket*, *context*)**

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>name_or_IP</i>	DNS name (x.com) or IP address of a server or empty. Empty is equivalent to INADDR_ANY which means the system will choose one for you. Generally, this parameter should be left empty.
<i>port</i>	Port number. If the port number is specified as a value $\leq 0$ , CallHTTP defaults to a port number of 40001.
<i>backlog</i>	The maximum length of the queue of pending connections (for example, concurrent client-side connections).
<i>svr_socket</i>	The handle to the server side socket.
<i>context</i>	The handle to the security context.

initSecureServerSocket Parameters

The following table describes the status of each return code.

<b>Return Code</b>	<b>Status</b>
0	Success.
1 - 41	See Socket Function Error Return Codes.
99	UniVerse failed to obtain a license for an interactive PHANTOM process.
101	Invalid security context handle.

**Return Code Status**

---

## Accepting an incoming connection attempt on the server side

Use the **acceptConnection()** function to accept an incoming connection attempt on the server side socket.

### Syntax

```
acceptConnection(svr_socket, blocking_mode, timeout, in_addr,  
in_name, socket_handle)
```

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>svr_socket</i>	The handle to the server side socket which is returned by <code>initServerSocket()</code> .
<i>blocking_mode</i>	<i>blocking_mode</i> is one of the following: <ul style="list-style-type: none"><li>■ 0 – default (blocking)</li><li>■ 1 – blocking. If this mode and the current blocking mode of <i>svr_socket</i> is set to blocking, <code>acceptConnection()</code> blocks the caller until a connection request is received or the specified <i>time_out</i> has expired.</li><li>■ 2 – nonblocking. In this mode, if there are no pending connections present in the queue, <code>acceptConnection()</code> returns an error status code. If this mode, <i>time_out</i> is ignored.</li></ul>
<i>time_out</i>	Timeout in milliseconds.

---

#### acceptConnection Parameters

---

Parameter	Description
<i>in_addr</i>	The buffer that receives the address of the incoming connection. If NULL, it will return nothing.
<i>in_name</i>	The variable that receives the name of the incoming connection. If NULL, it will return nothing.
<i>socket_handle</i>	The handle to the newly created socket on which the actual connection will be made. The server will use readSocket(), writeSocket(), and so forth with this handle to communicate with the client.

---

**acceptConnection Parameters (Continued)**

The following table describes the status of each return code.

---

Return Code	Status
0	Success.
1-41	See Socket Function Error Return Codes.
102	SSL Handshake failure.
103	No client certificate.
105	Client authentication failure.
106	Peer not speaking SSL.

---

**Return Code Status**

---

## Protocol logging

This function will start or stop logging and can be used for both the Socket API and CallHTTP API.

### Syntax

**protocolLogging**(*log\_file*, *log\_action*, *log\_level*)

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>log_file</i>	The name of the file the logs will be recorded to. The default log file name is httplog and will be created under the current directory.
<i>log_action</i>	Either ON or OFF. The default is OFF.
<i>log_level</i>	The detail level of logging from 0 - 10. See table below.

**protocolLogging Parameters**

Level	Detail
0	No logging.
1	Socket open/read/write/close action (no real data) HTTP request: hostinfo(URL)
2	Level 1 logging plus socket data statistics (size, and so forth).
3	Level 2 logging plus all data actually transferred.
4-10	More detailed status data to assist debugging.

**protocolLogging Log Levels**

The following table describes the status of each return code.

<b>Return Code</b>	<b>Status</b>
0	Success.
1	Failed to start logging.
<b>Return Status</b>	

---

## Socket API example

The UniVerse BASIC code example below demonstrates how each function can be used and their order in a typical implementation. The following functions are used in this example:

- protocolLogging()
- initServerSocket()
- openSocket()
- getSocketInformation()
- acceptConnection()
- writeSocket()
- readSocket()
- getSocketOptions()
- setSockeOptions()
- closeSocket()

```

***** Declare variables
*
reoptVar="DEBUG,REUSEADDR,KEEPALIVE,DONTROUTE,LINGER,OOBINLINE,SND
BUF,RCVBUF,TYPE,ERROR"
REUSEADDR="1"
CONVERT "," TO @FM IN reoptVar
POP3=@(0,0)
*
*
*
***** Specify Protocol Logging information
*
RESULT=protocolLogging("", "ON", 10)
CRT "Logging started = " :RESULT
*
INADDR=""
INNAME=""
*
***** Specify server name and assign handles to each socket
*
SERVER.IP.ADDRESS="127.0.0.1" ;*myHostName
SOCKET.PORT="8555"
MODE="1"; * 0=non-blocking, 1=blocking
SOCKETHANDLE1="";* Client handle
SOCKETHANDLE2="";* Server handle
SOCKETHANDLE3="";* Acceptor handle
TIMEOUT=10000; * milliseconds
*
BACKLOG="2048"
CRT
SERFLAG="-1";* Self end = Server
CRT "Starting Servers"
*
***** Initialize the Server Socket
*
RESULT=initServerSocket(SERVER.IP.ADDRESS, SOCKET.PORT, BACKLOG, SOCK
ETHANDLE2)
CRT "Init server 1 result = " :RESULT
*
***** Get information from the socket
*
RESULT=getSocketInformation(SOCKETHANDLE2, SERFLAG, SOCKETINFO)
CRT "Server Socket Info"
CRT "-----"
CRT "Status : " :SOCKETINFO<1,1>
CRT "Host : " :SOCKETINFO<1,2>
CRT "Port : " :SOCKETINFO<1,3>
CRT "Secure : " :SOCKETINFO<1,4>
CRT "Mode : " :SOCKETINFO<1,5>
*
PEERFLAG="0"
*
***** Open a Client Socket

```

```

*
CRT "Opening Client"
RESULT=openSocket(SERVER.IP.ADDRESS,SOCKET.PORT,MODE,TIMEOUT,SOCKE
THANDLE1)
CRT "Result of client open = ":RESULT
*
***** Get information from the Client Socket
*
RESULT=getSocketInformation(SOCKETHANDLE1,PEERFLAG,SOCKETINFO)
CRT "Client Socket Info"
CRT "-----"
CRT "Status : ":SOCKETINFO<1,1>
CRT "Host : ":SOCKETINFO<1,2>
CRT "Port : ":SOCKETINFO<1,3>
CRT "Secure : ":SOCKETINFO<1,4>
CRT "Mode : ":SOCKETINFO<1,5>
CRT ""
*
***** Accept Connections on Server
*
CRT "Server Accepting connections"
RESULT=acceptConnection(SOCKETHANDLE2,MODE,TIMEOUT,INADDR,INNAME,S
OCKETHANDLE3)
CRT "Connection ACCEPT Status = ":RESULT
CRT "
*
***** Write to and Read from the Socket
*
SDATLEN=""
CDATLEN=""
SRDATA="Hello Server with this test to see the display and count"
CLDATA=""
ACTSIZ=""
*
RESULT=writeSocket(SOCKETHANDLE1,SRDATA,TIMEOUT,MODE,SDATLEN)
CRT "Wrote status = ":RESULT
RESULT=readSocket(SOCKETHANDLE3,CLDATA,CDATLEN,TIMEOUT,MODE,ACTSIZ
)
CRT "Read status = ":RESULT
*CRT
CRT " Value of inbuf = ":CLDATA
CRT " Actual size of data = ":ACTSIZ
CRT " Value of in_addr = ":INADDR
CRT " Value of in_name = ":INNAME
CRT "
*
***** Set the socket options
*
*wroptVar="SNDBUF:@VM:8192:@FM:RCVBUF:@VM:16384
RESULT=setSocketOptions(SOCKETHANDLE2, wroptVar)
CRT " Set options is : ":RESULT
*
***** Get the socket options
*

```

```
RESULT=getSocketOptions(SOCKETHANDLE1, reoptVar)
PRINT "Result of get socket handle Options is : ":RESULT
PRINT "get socket options list"
LIMIT = DCOUNT(reoptVar,@FM)
FOR I = 1 TO LIMIT
PRINT FMT(reoptVar<I,1>, "L#10"),reoptVar<I,2>,reoptVar<I,3>
NEXT I
CRT
*
***** Close each of the Sockets
*
RESULT=closeSocket(SOCKETHANDLE1)
CRT "result of close client = ":RESULT
RESULT=closeSocket(SOCKETHANDLE2)
CRT "result of close server = ":RESULT
RESULT=closeSocket(SOCKETHANDLE3)
CRT "result of close Acceptor = ":RESULT
CRT
END
```

# Chapter 2: Using CallHTTP

2

Configuring the default HTTP settings. . . . .	2-3
Getting the current HTTP default settings . . . . .	2-7
Creating an HTTP request . . . . .	2-8
Creating a secure HTTP request . . . . .	2-11
Setting additional headers for a request . . . . .	2-14
Adding a parameter to the request . . . . .	2-16
Submitting a request . . . . .	2-18
HTTP digest authentication . . . . .	2-21
How Digest Authentication Works. . . . .	2-21
WWW-authenticate response/proxy-authenticate header parameters	2-23
Authorization request header parameters . . . . .	2-24
Using digest authentication with Microsoft IIS web servers . . . . .	2-25
Secure HTTP authentication through proxy . . . . .	2-26
Multiple same-name headers in response messages . . . . .	2-27

CallHTTP provides users with the capability of interacting with a web server from UniVerse BASIC through the standard HTTP protocol. In order to effectively use the CallHTTP functions, you should have a working knowledge of the HTTP standard.

Internet and web technologies have rapidly changed the way business is conducted by enterprises of all categories. E-commerce is increasingly becoming an essential part of any business. Many companies desire the capability to “call out” to the web from UniVerse BASIC so that their now stand-alone applications can be integrated with other applications through the web.

There are many scenarios where this capability can be beneficial. For example, you may want to integrate a general ledger application with a third-party application that has already been web-enabled. When an account number is given, the general ledger application has to send it to the web application through an HTTP request for validation. The web application then returns a confirmation to the UniVerse BASIC application.

HTTP is a complex standard with a large number of components and methods. The goal for CallHTTP is to provide a basic yet general implementation that enables UniVerse BASIC to act as an HTTP client so that data can be exchanged between a UniVerse BASIC application and a web server. CallHTTP provides the “plumbing” for users to build a specific client, not make UniVerse BASIC a browser of its own.

CallHTTP is implemented with the Socket Interface as its network transport, and this lower level API is also available for direct access by the user.

---

## Configuring the default HTTP settings

The **setHTTPDefault** function configures the default HTTP settings, including proxy server and port, buffer size, authentication credential, HTTP version, and request header values. These settings are used with every HTTP request that follows.

### Syntax

**setHTTPDefault(*option, value*)**

If you require all outgoing network traffic to go through a proxy server, **setHTTPDefault()** should be called with value containing the proxy server name or IP address, as well as the port (if other than the default of 80).

### Parameters

The following table describes each parameter of the syntax.

---

Parameter	Description
<i>option</i>	A string containing an option name. See the table below for the options currently defined.
<i>value</i>	A string containing the appropriate option value.

---

**setHTTPDefault Parameters**

The following table describes the available options for **setHTTPDefault**.

Option	Description
PROXY_NAME	Name or IP address of the proxy server.
PROXY_PORT	The port number to be used on the proxy server. This only needs to be specified if the port is other than the default of 80.
VERSION	The version of HTTP to be used. The default version is 1.1, but it can be set to 1.0 for web servers that understand the older protocol. The string should be "1.0" or "1.1."
<b>setHTTPDefault Options</b>	

---

Option	Description
BUFSIZE	The size of the buffer for HTTP data transfer between UniVerse and the web server. The default is 4096. The buffer size can be increased to improve performance. It should be entered as an integer greater than or equal to 4096.
AUTHENTICATE	The user name, password, and realm to gain access. The string should be:  <code>&lt;userid&gt;:&lt;password&gt;[:&lt;realm&gt;]</code>
	Default Basic authentication can also be set. If a request is denied (HTTP status 401/407), UniVerse BASIC will search for the default credential to automatically resubmit the request. If there is already a same-user/realm credential found on the default authenticate list, UniVerse replaces the old value with the new value. This assures that one current credential is maintained for each user and realm.
HEADERS	The header to be sent with the HTTP request. If <i>default_headers</i> contains an empty string, any current user-specified default header will be cleared. Currently, the only default header UniVerse BASIC sets automatically is "User-Agent UniVerse 9.6." If you do not want to send out this header you should overwrite it with <code>setHTTPDefault()</code> .
	Per RFC 2616, for "net politeness," an HTTP client should always send out this header. UniVerse BASIC will also send a date/time stamp with every HTTP request. According to RFC 2616, the stamp represents time in Universal Time (UT) format. A header should be entered as a dynamic array in the form of <Header-Name>@VM<HeaderValue>@Fm<HeaderName>@VM<HeaderValue>.

---

#### setHTTPDefault Options (Continued)

The following table describes the status of each return code.

Return Code	Status
0	Success
1	Invalid option.
2	Invalid value.

**Return Code Status**



**Note:** All defaults set by `setHTTPDefault()` will be in effect until the end of the current UniVerse session. If you do not want the setting to affect subsequent programs, you will need to clear it before exiting the current program. If the user wishes to set the "Authorization" or "Proxy-Authorization" header as defaults, see the description under `setRequestHeader()`. To clear the default settings, pass an empty string with `PROXY_NAME`, `AUTHENTICATE` and `HEADERS`, and 0 for `PROXY_PORT` and `BUFSIZE`.

---

## Getting the current HTTP default settings

The `getHTTPDefault` function returns the default values of the HTTP settings. See the section under `setHTTPDefault` for additional information.

### Syntax

`getHTTPDefault(option, value)`

### Parameters

The following table describes each parameter of the syntax:

Parameter	Description
<i>option</i>	Currently, the following options are defined:  PROXY_NAME PROXY_PORT VERSION BUFSIZE AUTHENTICATE HEADERS
<i>value</i>	A string containing the appropriate option value.

#### getHTTPDefault Parameters

The following table describes the status of each return code.

Return Code	Status
0	Success.
1	Invalid option.

#### Return Code Status

## Creating an HTTP request

The **createRequest** function creates an HTTP request and returns a handle to the request.

### Syntax

**createRequest(*URL*, *http\_method*, *request\_handle*)**

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>URL</i>	A string containing the URL for a resource on a web server. An accepted URL must follow the specified syntax defined in RFC 1738. The general format is: <code>http://&lt;host&gt;:&lt;port&gt;/&lt;path&gt;?&lt;searchpart&gt;</code> . The host can be either a name string or IP address. The port is the port number to connect to, which usually defaults to 80 and is often omitted, along with the preceding colon. The path tells the web server which file you want, and, if omitted, means “home page” for the system. The searchpart can be used to send additional information to a web server.
<i>http_method</i>	A string which indicates the method to be performed on the resource. See the table below for the available (case-sensitive) methods.
<i>request_handle</i>	A handle to the request object.

#### createRequest Parameters

The following table describes the available methods for *http\_method*.

Method	Description
GET	Retrieves whatever information, in the form of an entity, identified by the Request-URI. If the Request-URI refers to a data-producing process, it is the produced data which shall be returned as the entity in the response and not the source text of the process, unless that text happens to be the output of the process.
POST	[:<MIME-type>] For this method, it can also have an optional MIME-type to indicate the content type of the data the request intends to send. If no MIME-type is given, the default content type will be “application/x-www-form-urlencoded.” Currently, only “multipart/form-data” is internally supported, as described in function addRequestParameter() and submitRequest(), although other “multipart/*” data can also be sent if the user can assemble it on his/her own. (The multipart/form-data format itself is thoroughly described in RFC 2388).
HEAD	The HEAD method is identical to GET except that the server MUST NOT return a message-body in the response. The metainformation contained in the HTTP headers in response to a HEAD request SHOULD be identical to the information sent in response to a GET request. This method can be used for obtaining metainformation about the entity implied by the request without transferring the entity-body itself. This method is often used for testing hypertext links for validity, accessibility, and recent modification.
OPTIONS	The OPTIONS method represents a request for information about the communication options available on the request/response chain identified by the Request-URI. This method allows the client to determine the options and/or requirements associated with a resource, or the capabilities of a server, without implying a resource action or initiating a resource retrieval. HTTP 1.1 and later.
DELETE	The DELETE method requests that the origin server delete the resource identified by the Request-URI. HTTP 1.1 and later.

#### **http\_method Methods**

---

<b>Method</b>	<b>Description</b>
TRACE	The TRACE method is used to invoke a remote, application-layer loop-back of the request message. HTTP 1.1 and later.
PUT	The PUT method is used by an authorized client to upload a resource to a web server. The URI of the PUT request specifies the location of the resource.
CONNECT	The CONNECT specification reserves the method name CONNECT for use with a proxy that can dynamically switch to being a tunnel.

---

#### **http\_method Methods (Continued)**

The following table describes the status of each return code.

---

<b>Return Code</b>	<b>Status</b>
0	Success.
1	Invalid URL (Syntactically).
2	Invalid method (For HTTP 1.0, only GET/POST/HEAD)

---

#### **Return Code Status**



**Note:** If URL does include a searchpart, it must be in its encoded format (space is converted into +, and other non-alphanumeric characters are converted into %HH format. See addRequestParameter() for more details). However, host and path are allowed to have these “unsafe” characters. UniVerse BASIC will encode them before communicating with the web server.

## Creating a secure HTTP request

The `createSecureRequest` function behaves exactly the same as the `createRequest()` function, except for the fourth parameter, a handle to a security context, which is used to associate the security context with the request. If the URL does not start with “https” then the parameter is ignored. If the URL starts with “https” but an invalid context handle or no handle is provided, the function will abort and return with an error status.

### Syntax

```
createSecureRequest(URL, http_method, request_handle,
security_context)
```

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<code>URL</code>	A string containing the URL for a resource on a web server. An accepted URL must follow the specified syntax defined in RFC 2818. The general format is:  <code>https://&lt;host&gt;:&lt;port&gt;/&lt;path&gt;?&lt;searchpart&gt;</code> The host can be either a name string or IP address. The port is the port number to connect to, which usually defaults to 443 and is often omitted, along with the preceding colon. The path tells the web server which file you want, and, if omitted, means “home page” for the system. The searchpart can be used to send additional information to a web server.
<code>http_method</code>	A string which indicates the method to be performed on the resource. See the table below for the available (case-sensitive) methods.
<code>request_handle</code>	A handle to the request object.
<code>securityContext</code>	A handle to the security context.

#### createSecureRequest Parameters

The following table describes the available methods for *http\_method*.

Method	Description
GET	Retrieves whatever information, in the form of an entity, identified by the Request-URI. If the Request-URI refers to a data-producing process, it is the produced data which shall be returned as the entity in the response and not the source text of the process, unless that text happens to be the output of the process.
POST	[:<MIME-type>] For this method, it can also have an optional MIME-type to indicate the content type of the data the request intends to send. If no MIME-type is given, the default content type will be "application/x-www-form-urlencoded". Currently, only "multipart/form-data" is internally supported, as described in function <b>addRequestParameter()</b> and <b>submitRequest()</b> , although other "multipart/*" data can also be sent if the user can assemble it on his/her own. (The multipart/form-data format itself is thoroughly described in RFC 2388).
HEAD	The HEAD method is identical to GET except that the server MUST NOT return a message-body in the response. The meta information contained in the HTTP headers in response to a HEAD request SHOULD be identical to the information sent in response to a GET request. This method can be used for obtaining meta-information about the entity implied by the request without transferring the entity-body itself. This method is often used for testing hypertext links for validity, accessibility, and recent modification.
OPTIONS	The OPTIONS method represents a request for information about the communication options available on the request/response chain identified by the Request-URI. This method allows the client to determine the options and/or requirements associated with a resource, or the capabilities of a server, without implying a resource action or initiating a resource retrieval. HTTP 1.1 and later.
DELETE	The DELETE method requests that the origin server delete the resource identified by the Request-URI. HTTP 1.1 and later.

#### **createRequest Methods**

---

<b>Method</b>	<b>Description</b>
TRACE	The TRACE method is used to invoke a remote, application-layer loop-back of the request message. HTTP 1.1 and later.
PUT	The PUT method is used by an authorized client to upload a resource to a web server. The URI of the PUT request specifies the location of the resource.
CONNECT	/* HTTP/1.1 and later but not supported */

---

#### **createRequest Methods (Continued)**

The following table describes the status of each return code.

---

<b>Return Code</b>	<b>Status</b>
0	Success.
1	Invalid URL (Syntactically).
2	Invalid method (For HTTP 1.0, only GET/POST/HEAD)

---

#### **Return Code Status**



*Note: If URL does include a searchpart, it must be in its encoded format (space is converted into +, and other non-alphanumeric characters are converted into %HH format. See addRequestParameter() for more details). However, host and path are allowed to have these "unsafe" characters. UniVerse BASIC will encode them before communicating with the web server.*

## Setting additional headers for a request

The **setRequestHeader** function enables you to set additional headers for a request.

### Syntax

```
setRequestHeader(request_handle, header_name, header_value)
```

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>request_handle</i>	The handle to the request returned by <code>createRequest()</code> .
<i>header_name</i>	The name of the header.
<i>header_value</i>	The value of the header.

#### setRequestHeader Parameters

The following table describes the status of each return code.

Return Code	Status
0	Success.
1	Invalid request handle.
2	Invalid header (Incompatible with method).
3	Invalid header value.

#### Return Code Status



**Note:** Since a user-defined header or header value can be transferred, it is difficult to check the validity of parameters passed to the function. UniVerse BASIC currently will not perform syntax checking on the parameters, although it will reject setting a response header to a request. Refer to RFC 2616 for valid request headers.

The header set by this function will overwrite settings by `setHTTPDefault()`.

This function supports Base64 encoding for Basic authentication. If `header_name` contains either "Authorization" or "Proxy-Authorization," the `header_value` should then contain ASCII text user credential information in the format of "userid:password" as specified by RFC 2617. This function will then encode the text based on Base64 encoding.

## Adding a parameter to the request

The `addRequestParameter` function adds a parameter to the request.

### Syntax

```
addRequestParameter(request_handle, parameter_name, parameter_value, content_handling)
```

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>request_handle</i>	The handle to the request.
<i>parameter_name</i>	The name of the parameter.
<i>parameter_value</i>	The value of the parameter.
<i>content_handling</i>	The dynamic MIME type for the parameter value.

#### **addRequestParameter Parameters**

The following table describes the status of each return code.

Return Code	Status
0	Success.
1	Invalid request handle.
2	Invalid parameter.
3	Bad content type.

#### **Return Code Status**



*Note:* For a GET request, *content\_handling* is ignored.

For a POST request with default content type, the default for content\_handling is “ContentType:text/plain” if content\_handling is not specified. For a POST request with “Multipart/\*” content-type, content\_handling is a dynamic array containing Content-\* strings separated by field marks (@FM). They will be included in the multipart message before the data contained in parameter\_value is sent. An example of content\_handling:

```
Content-Type: application/XML @FM  
Content-Dispostion: attachment; file="C:\drive\test.dat @FM  
Content-Length: 1923
```

Specifically, for a POST request with content type “multipart/form-data,” a “Content-Disposition:form-data” header will be created (or, in the case of Content-Disposition already in *content\_handling*, “form-data” will be added to it).

For both a GET and a POST request with either no content type specified or specified as “application/x-www-form-urlencoded,” as described in createRequest(), URL encoding is performed on data in *parameter\_value* automatically. Basically, any character other than alphanumeric is considered “unsafe” and will be replaced by %HH, where HH is the ASCII value of the character in question. For example, “#” is replaced by %23, and “/” is replaced by %2F, and so forth. One exception is that by convention, spaces (‘ ’) are converted into “+”.

For a POST method with other MIME-type specified, no encoding is done on data contained in *parameter\_value*.

## Submitting a request

The **submitRequest** function will submit a request and get a response.

The request is formed on the basis of default HTTP settings and previous **setRequestHeader()** and **addRequestParameter()** values. Specifically, for a GET method with parameters added, a parameter string (properly encoded) is created and attached to the URL string after the "?" character.

For a POST request with non-empty post\_data, the data is attached to the request message as is. No encoding is performed, and any parameters added through **addRequestParameter()** will be totally ignored. Otherwise the following processing will be performed.

For a POST request with default content type, the parameter string is assembled, a Content-Length header created, and then the string is attached as the last part of the request message.

For a POST request with multipart/\* content type, a unique boundary string is created and then multiple parts are generated in the sequence they were added through calling **addRequestParameter()**. Each will have a unique boundary, followed by optional Content-\* headers, and data part. The total length is calculated and a Content-Length header is added to the message header.

The request is then sent to the Web server identified by the URL supplied with the request and created through **createRequest()** (maybe via a proxy server). UniVerse Basic then waits for the web server to respond. Once the response message is received, the status contained in the response is analyzed.

If the response status indicates that redirection is needed (status 301, 302, 305 or 307), it will be performed automatically, up to ten consecutive redirections (the limit is set to prevent looping, suggested by RFC 2616).

If the response status is 401 or 407 (access denied), the response headers are examined to see if the server requires (or accepts) Basic authentication. If no Basic authentication request is found, the function returns with an error. Otherwise, default Authentication (set by **setHTTPDefault**) is used to resend the request. If no default authentication is set, and no other cached user authentication is found, the function will return with an error.

If the user provides authentication information through “Authorization” or “Proxy-Authorization” header, the encoded information is cached. If later, a Basic authentication request is raised, no default authentication is found, and only one user/password encoding is cached, it will be used to re-send the request.

The response from the HTTP server is disposed into *response\_header* and *response\_data*. It is the user’s responsibility to parse the headers and data. UniVerse Basic only performs transfer encoding (chunked encoding), and nothing else is done on the data. In other words, content-encoding (gzip, compress, deflate, and so forth) are supposed to be handled by the user, as with all MIME types.

Also, if a response contains header “Content-type: multipart/\*”, all the data (multiple bodies enclosed in “boundary delimiters,” see RFC 2046) is stored in *response\_data*. It is the user’s responsibility to parse it according to “boundary” parameter.

## Syntax

```
submitRequest(request_handle, time_out, post_data, response_headers, response_data, http_status)
```

## Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>request_handle</i>	The handle to the request.
<i>time_out</i>	The time-out value (in milliseconds) before the wait response is abandoned.
<i>post_data</i>	The data sent with the POST request.
<b>submitRequest Parameters</b>	

---

Parameter	Description
<i>response_headers</i>	A dynamic array to store header/value pairs.
<i>response_data</i>	The resultant data (may be in binary format).
<i>http_status</i>	A dynamic array containing the status code and explanatory text.

---

**submitRequest Parameters (Continued)**

The following table describes the status of each return code.

---

Return Code	Status
0	Success.
1	Invalid request handle.
2	Timed out.
3	Network Error.
4	Other Errors.
99	UniVerse failed to obtain a license for an interactive PHANTOM process.

---

**Return Code Status**

---

## HTTP digest authentication

The HTTP 1.1 standard specifies two authentication methods, Basic and Digest. The Basic authentication method passes credentials (user name and password) in clear text format to an HTTP server. These credentials can be intercepted. To improve security, Digest authentication has been added to UniVerse 11.1. With Digest authentication, instead of passing credentials in clear text, a digest is passed. The digest uses a time stamp and random data in the authentication message header, and the real password is not exposed. Digest Authentication is based on RFC2617.

## How Digest Authentication Works

When a Web resource is protected, a client that wants to access the resource through an HTTP request message has to authenticate itself to the server. According to the RFC 2617 specification, if the initial client request does not contain valid authentication data, the HTTP server sends back a response message with HTTP status code 401. For Digest authentication, the response message looks similar to the following example:

```
HTTP/1.0 401 UnauthorisedServer: HTTPD/0.9Date: Sun, 10 Apr 2009 20:26:47
GMTWWW-Authenticate: Digest realm="testrealm@host.com qop="auth,auth-int",
nonce="dc98b7102dd2f0e8b11df600bfb0c093",
opaque="5ccc069c403ebaf9f0171e9517f40e41"Content-Type: text/htmlContent-
Length: 311
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/1999/REC-html401-19991224/loose.dtd">
<HTML>
<HEAD>
<TITLE>Error</TITLE>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-1">
</HEAD>
<BODY><H1>401 Unauthorised.</H1></BODY>
</HTML>
```

When receiving this response, the HTTP client has to obtain credentials, calculate the digest, and append an “Authorization” header in a new request, similar to the following example:

```
GET /dir/index.html HTTP/1.0
Host: localhost
Authorization: Digest username="Mufasa",
realm="testrealm@host.com",
nonce="dc98b7102dd2f0e8b1d0f600fb0c093",
uri="/dir/index.html",
qop=auth,
nc=00000001,
cnonce="0a4f113b",
response="6629fae49393a05397450978507c4ef1",
opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

According to RFC 2617, this is how the digest is calculated:

$$HA1 = MD5(A1) = MD5(\text{username}:\text{realm}:\text{password})$$

If the Quality of Protection (qop) value in the WWW-Authenticate header is “auth” or unspecified, the digest is calculated as:

$$HA2 = MD5(A2) = MD5(\text{method}:\text{digestURI})$$

If the qop value is “auth-int” this is how the digest is calculated:

$$HA2 = MD5(A2) = MD5(\text{method}:\text{digestURI}:\text{H(entity-body)})$$

The response is calculated as:

$$\text{Response} = MD5(HA1:\text{nonce}:\text{nonceCount}:\text{clientNonce}:\text{qop}:HA2)$$

The difference between a the “auth” qop value and the “auth-int” qop value is that “auth-int” include the request’s body, not the headers, into the calculation of HA2, which can detect any modification of the request content during transfer. This ensures data integrity. These calculations are performed by UniVerse BASIC.

---

## WWW-authenticate response/proxy-authenticate header parameters

The following table describes the parameters for the WWW-Authenticate Response and Proxy-Authenticate headers:

Parameter	Description
Digest	Indicates that the server is asking for Digest authentication.
Realm	Specifies the authentication realm. The client understands the realm and gives the client an idea of the credentials being requested.
Nonce	A server-generated value used in the calculation of the digest, returned to the server unchanged. This proves that the client has a valid response.
Opaque	A value the client must return to the server unchanged.
Stale	This will normally be in response to the client's request containing an authorization header. If the value is "true," the nonce used in the digest calculation is aged, signaling a possible replay attack.
Algorithm	Specifies the hash algorithm to use when computing the digest. The default is MD5, but can be specified as MD5-sess, which means the client calculates the digest only once for a session.
Qop	Quality of protection. Valid options are authentication only or authentication plus integrity protection.

---

WWW-Authenticate Response/Proxy-Authenticate Header Parameters

## Authorization request header parameters

The following table describes the parameters in the Authorization request header.

Parameter	Description
Digest	Indicates that the client is responding to a Digest access authentication challenge.
Realm	Specifies the authentication realm. This realm should be understood by the client, and gives the client an idea of which credentials are being requested. It must match the value in the WWW-Authenticate response header.
Nonce	A server-generated value to use in the calculation of a digest, which should return to the server unchanged. This is used to prove that the client has a valid response.
Opaque	A value the client must return to the server unchanged. Opaque does not participate in a digest.
Algorithm	Specifies a hash algorithm to use when computing a digest. The default is MD5. This must be the same as the algorithm in the WWW-Authenticate header on the server.
Qop	Quality of protection, either authentication only or authentication plus integrity protection.
Uri	Specifies the requested URI on the server. The URI is repeated here to guarantee it will work in a multi-intermediate proxy environment.
Nc	Client nonce count. Indicates the number of times the client has used this nonce for calculating digests.
Cnonce	Client nonce.
Response	Digest calculated based algorithm.

Authorization Request Header Parameters

---

## **Using digest authentication with Microsoft IIS web servers**

The following conditions must be met if using HTTP Digest Authentication against Microsoft IIS Web servers:

- The IIS server must be running Windows Server 2000 or above.
- The domain to which the IIS server is a member must include a domain controller that is running Windows 2000 or Windows Server 2003.
- The IIS server and a user logging on to the IIS server must belong to the same domain. They can, however, be joined through trusts.
- Each user that needs to be authenticated must have a legitimate account in the Active Directory on the particular domain controller.
- User passwords must be stored in a reversibly encrypted format in the Active Directory. You can use the Active Directory Users and the computer console to access the Account tab of the Properties dialog box to enable reversible encryption.
- Recent Windows 2003 and IIS versions support “Advanced Digest Authentication” which does not require that the user password be stored in reversibly encrypted format. The password is stored in predigested format. To enable this feature, the IIS server and the domain controller must both be running on Windows 2003 with the latest service pack.

## **Secure HTTP authentication through proxy**

In secure HTTP mode, if a request goes through an intermediary, such as a proxy server, the request is using “tunnel mode” through the proxy, which means that the encrypted message is not decrypted by the proxy, just passed through to the destination server. Before this release, UniVerse could not handle a situation where the proxy itself requested authentication. At this release, Proxy authentication is supported in both Basic and Digest formats.

In previous releases of UniVerse, a secure HTTP request could be sent through a proxy server, and UniVerse could perform authentication to the destination web server through the proxy with a secure connection. However, if the proxy server required UniVerse to authenticate, indicated by a response with a 407 status, UniVerse could not establish a secure connection.,

In UniVerse 11.1, UniVerse performs proxy authentication automatically when a secure HTTP connection is requested. You must set up credentials, including username, realm and password, prior to invoking a secure HTTP request.

---

## Multiple same-name headers in response messages

An HTTP response message can contain multiple headers with the same name. For example, you can receive a response message with "Set-Cookie" headers, or multiple "WWW-Authenticate" headers.

The `getResponseHeader()` function currently always returns the first header with a matching name. At this release, the `getResponseHeader()` and `SOAPGetResponseHeader()` functions return all values with the specified header to a dynamic array.

Consider the following response message:

```
HTTP/1.1 401 Unauthorized
Server: Microsoft-IIS/5.1
Date: Sun, 22 Nov 2009 20:23:34 GMT
WWW-Authenticate: Negotiate
WWW-Authenticate: NTLM
WWW-Authenticate: Basic realm="defaultRealm"
Connection: close
Content-Type: text/html
Content-Length: 4046
```

After calling `getResponseHeader(httphandle,"WWW-Authenticate",headers)`, the output parameter will contain the following attribute mark-separated value:

```
Negotiate<FM>NTLM<FM>Basic realm="defaultRealm"
```

# Chapter 3: Using WebSphere MQ with UniVerse

# 3

In this chapter . . . . .	3-2
Preface. . . . .	3-3
Overview of messaging . . . . .	3-4
Overview of IBM WebSphere MQ . . . . .	3-5
U2 WebSphere MQ API . . . . .	3-7
CallMQI setup and configuration . . . . .	3-8
Programming with CallMQI . . . . .	3-11
U2MQI.H INCLUDE file . . . . .	3-12
Error handling in CallMQI . . . . .	3-13
Encoding message IDs and correlation IDs . . . . .	3-13
CallMQI programmatic interfaces . . . . .	3-14
Connecting to a queue manager . . . . .	3-15
Opening a queue . . . . .	3-17
Retrieving a message from a queue . . . . .	3-18
Putting a message on a queue . . . . .	3-21
Putting a single message on a queue . . . . .	3-23
Querying the attributes . . . . .	3-25
Closing a queue . . . . .	3-26
Disconnecting from a queue manager . . . . .	3-28
Returning an error message . . . . .	3-29
CallMQI programming examples . . . . .	3-31
CallMQI dynamic array structures . . . . .	3-36
MQOD dynamic array structure . . . . .	3-36
MQMD dynamic array structure . . . . .	3-39
MQGMO dynamic array structure . . . . .	3-42
MQPMO dynamic array structure . . . . .	3-45
WebSphere MQ API for UniData and UniVerse . . . . .	3-48

WebSphere MQ Application Messaging Interface . . . . .	3-48
Session . . . . .	3-49
Services . . . . .	3-49
Policies . . . . .	3-50
Message objects . . . . .	3-50
Messaging styles . . . . .	3-51
CallAMI setup and configuration . . . . .	3-53
Requirements . . . . .	3-53
Platform availability . . . . .	3-53
Setting up the environment for CallAMI . . . . .	3-53
Configurations . . . . .	3-54
CallAMI programmatic interfaces . . . . .	3-57
Initialize an AMI session . . . . .	3-57
Receiving a message . . . . .	3-58
Receiving a request . . . . .	3-62
Sending a message . . . . .	3-65
Sending a request . . . . .	3-67
Sending a response . . . . .	3-69
Terminating a session . . . . .	3-71
CallAMI programming examples . . . . .	3-73
Additional reading . . . . .	3-83

---

## In this chapter

This chapter describes how to set up and configure the U2 WebSphere MQ API (CallMQI) and WebSphere MQ API for UniData and UniVerse (CallAMI).

This chapter consists of the following sections:

- “ Preface”
- “ Overview of messaging”
- “ Overview of IBM WebSphere MQ”
  
- “ U2 WebSphere MQ API”
- “ CallMQI setup and configuration”
- “ Programming with CallMQI”
- “ CallMQI programmatic interfaces”
- “ CallMQI programming examples”
- “ CallMQI dynamic array structures”
  
- “ WebSphere MQ API for UniData and UniVerse”
- “ CallAMI setup and configuration”
- “ CallAMI programmatic interfaces”
- “ CallAMI programming examples”
  
- “ Additional reading”

## Preface

UniVerse supports two interfaces for interacting with WebSphere MQ: The U2 WebSphere MQ API (CallMQI) and WebSphere MQ API for UniData and UniVerse (CallAMI).

CallMQI makes use of the WebSphere MQ Message Queue Interface API (MQI), while CallAMI utilizes the WebSphere MQ Application Messaging Interface (AMI).

CallMQI is a low-level interface that exposes most of the functionality available in WebSphere MQ via its core MQI API. CallAMI is a higher level API that offers conveniences such as built-in support of request-response style messaging, and repository-based application settings. CallAMI is available on a limited subset of UniVerse platforms.

The AMI is available for download as a WebSphere MQ SupportPac from [www.ibm.com](http://www.ibm.com). You need to download and install this SupportPac before you can use CallAMI. Detailed information on the AMI can be found in the *WebSphere MQ Application Messaging Interface* manual, which you can download from the same location.

This manual assumes you have a general understanding of WebSphere MQ. Numerous documents for WebSphere MQ are available for download from the IBM Publications Center at:

<http://www.ibm.com/shop/publications/order>

## Overview of messaging

Distributed applications are a common occurrence in companies today. They may come about from business acquisitions that bring together disparate systems which must then interact with each other, or from the purchase of new software systems that must interact with existing facilities. Other distributed systems are intentionally designed as such in order to improve scalability and overall reliability.

Regardless of the circumstances of how these heterogeneous systems come about, they require a means for internal communication. A common solution to this challenge is messaging. Messaging middleware products provide the concept of message queues, which applications can use to communicate with each other through the exchange of messages. When one application has information to deliver, it places a message in a queue. Another application can then retrieve the message and act upon it. This is a flexible paradigm, allowing many different types of communication. Examples include the Datagram ("send-and-forget") messaging style, where an application simply delivers a message and then disconnects, and the Request/Response messaging style, which follows a client-server style of communication.

A core feature of messaging middleware products is guaranteed, once-only message delivery. This frees applications from having to take on this burden themselves with low-level communication details. An MQI application making use of the services of a messaging middleware product is guaranteed that its message will be delivered to the destination queue, and that the message will be delivered one time only.

Another feature that message-based applications take advantage of is the benefits of being loosely-coupled. One application can go offline without affecting other applications in the system. When the application comes back online, it can then retrieve any messages that have been waiting for it from the message queue. Allowing components to be loosely-coupled can improve overall system reliability.

## Overview of IBM WebSphere MQ

IBM WebSphere MQ is IBM's messaging middleware product. Through its services, applications can communicate with each other as messaging clients.

WebSphere MQ makes use of message queues and queue managers in providing its services to message-based applications. A queue manager is a service that allows access to and administration of message queues. It handles the details of message delivery, such as guaranteeing once-only delivery, and forwarding messages across a network to other queue managers when required. Message queues act as the destinations for message delivery, holding the incoming messages until retrieved by another application.

With WebSphere MQ, when an application needs to deliver a message, it connects to a WebSphere MQ queue manager, opens a message queue, and places the message on the queue. If required, the queue manager then forwards the message to a queue running under a different queue manager on another machine. A receiving application can then connect to that second queue manager, open the destination queue, and retrieve the message.

The following excerpt from *The WebSphere MQ Application Programming Guide* provides a description of WebSphere MQ queue managers and message queues:

### **What is a message queue?**

*A message queue, known simply as a queue, is a named destination to which messages can be sent. Messages accumulate on queues until they are retrieved by programs that service those queues. Queues reside in, and are managed by, a queue manager (see "What is a queue manager?" on page 5). The physical nature of a queue depends on the operating system on which the queue manager is running. A queue can either be a volatile buffer area in the memory of a computer, or a data set on a permanent storage device (such as a disk). The physical management of queues is the responsibility of the queue manager and is not made apparent to the participating application programs.*

## **What is a queue manager?**

*A queue manager is a system program that provides queuing services to applications. It provides an application programming interface so that programs can put messages on, and get messages from, queues. A queue manager provides additional functions so that administrators can create new queues, alter the properties of existing queues, and control the operation of the queue manager. Many different applications can make use of the queue manager's services at the same time and these applications can be entirely unrelated. For a program to use the services of a queue manager, it must establish a connection to that queue manager. For applications to be able to send messages to applications that are connected to other queue managers, the queue managers must be able to communicate among themselves. WebSphere MQ implements a store-and-forward protocol to ensure the safe delivery of messages between such applications.*

## **U2 WebSphere MQ API**

The U2 WebSphere MQ API (CallMQI) is a BASIC API that allows BASIC programs to interact with WebSphere MQ. The CallMQI interface is based on WebSphere MQ's core API, the Message Queue Interface (MQI).

---

## CallMQI setup and configuration

To connect to WebSphere MQ, you must install the WebSphere MQ Client on the machine running UniData or UniVerse. With the WebSphere MQ Client installed, the UniData or UniVerse database can connect to a queue manager via the WebSphere MQ “client libraries.”

Complete the following steps to connect to a queue manager:

1. Install the WebSphere MQ Client on the database server.
2. Verify your installation of WebSphere MQ.
3. Test the sample program.

### *Installing the WebSphere MQ client*

CallMQI requires the WebSphere MQ Client. Ensure that the WebSphere MQ Client is installed on the same machine as UniData or UniVerse. You can find information on installing the WebSphere MQ Client in the *WebSphere MQ Quick Beginnings* manual for your platform.

### *Verifying the WebSphere MQ Client installation*

You should verify the WebSphere MQ Client installation before you use the CallMQI API. Complete details on how to install the WebSphere MQ Client can be found in the *WebSphere MQ Quick Beginnings* manual for your platform. Make note of the names of the queue manager and queue used in the verification procedure, for use in the next step.

### *Testing with a sample CallMQI program*

Use the following sample CallMQI program to verify that CallMQI is working properly. This sample program relies on the setup steps found in the *WebSphere MQ Quick Beginnings* manual for verifying a WebSphere MQ installation.

The sample program connects to a queue manager. It then opens a queue and places a test message on the queue. Once the message is on the queue, it retrieves the test message and prints it to the screen. It then closes the queue, and disconnects from the queue manager.

Compile and run the program to verify that CallMQI is functioning correctly. Note that you may need to replace the names of both the queue manager and the queue with the respective names used in the *WebSphere MQ Quick Beginnings Guide*. Also, ensure that you set up a client channel appropriately, as described in the *WebSphere MQ Quick Beginnings Guide* before running the sample program from your terminal session.

```
$INCLUDE UNIVERSE.INCLUDE U2MQI.H
*
QUEUE.MANAGER = "saturn.queue.manager"
QUEUE = "QUEUE1"
*
HCONN = 0
RET = MQCONN(QUEUE.MANAGER, HCONN)
IF RET # MQCC_OK THEN
    PRINT "RET = " : RET
    MQERR.RET = MQGETERROR(REASONCODE, REASONMSG)
    PRINT "MQCONN: " : REASONCODE " : " : REASONMSG
END
*
OBJ.DESC = MQOD.DFLT
OPTIONS = MQOO_OUTPUT + MQOO_INPUT_SHARED
HOBJ = 0
OBJ.DESC<MQOD.ObjectName> = QUEUE
RET = MQOPEN(HCONN, OBJ.DESC, OPTIONS, HOBJ)
IF RET # MQCC_OK THEN
    MQERR.RET = MQGETERROR(REASONCODE, REASONMSG)
    PRINT "MQOPEN: " : REASONCODE " : " : REASONMSG
END
*
MSG.DESC = MQMD.DFLT
PUT.MSG.OPTS = MQPMO.DFLT
MSG = "The time is now " : TIMEDATE()
PRINT MSG : ">>"
RET = MQPUT(HCONN, HOBJ, MSG.DESC, PUT.MSG.OPTS, MSG)
IF RET # MQCC_OK THEN
```

```

MQERR.RET = MQGETERROR(REASONCODE, REASONMSG)
PRINT "MQPUT: " : REASONCODE ":" : REASONMSG
END
*
MSG.DESC = MQMD.DFLT
GET.MSG.OPTS = MQGMO.DFLT
MSG.BUFFER.LEN = 128
MSG = ""
MSG.DATA.LEN = 0
RET = MQGET(HCONN, HOBJ, MSG.DESC, GET.MSG.OPTS, MSG.BUFFER.LEN,
MSG, MSG.DATA.LEN)
IF RET # MQCC_OK THEN
    MQERR.RET = MQGETERROR(REASONCODE, REASONMSG)
    PRINT "MQGET: " : REASONCODE ":" : REASONMSG

    END
PRINT "<< " : MSG
*
OPTIONS = MQCO_NONE
RET = MQCLOSE(HCONN, HOBJ, OPTIONS)
IF RET # MQCC_OK THEN
    MQERR.RET = MQGETERROR(REASONCODE, REASONMSG)
    PRINT "MQCLOSE: " : REASONCODE ":" : REASONMSG
    END
*
RET = MQDISC(HCONN)
IF RET # MQCC_OK THEN
    MQERR.RET = MQGETERROR(REASONCODE, REASONMSG)
    PRINT "MQDISC: " : REASONCODE ":" : REASONMSG
    END

```

## Programming with CallMQI

The MQI is the core programming interface into WebSphere MQ. Its programming model consists of two primary facets:

- A set of functions for calling messaging services.
- A set of data structures passed as parameters to those functions.

The number of functions in the MQI is relatively small. They define, in a coarse-grained way, the types of messaging actions available through the API. The richness of the API is conveyed through the data structures. These data structures, supplied as parameters to the MQI functions, allow one to fine tune how the messaging actions behave.

The MQI itself is ported to many different programming languages, and the nature of these data structures depends on the language in question. In UniVerse BASIC and UniBasic, the data structures are represented as dynamic arrays.

The BASIC dynamic arrays that comprise the MQI data structures are of fixed format. Information is stored in predetermined attribute positions.

As an example, consider the MQI's Object Descriptor data structure (MQOD). An MQOD data structure is used in conjunction with the MQOPEN function. MQOPEN can open various types of WebSphere MQ objects, including queues, channels, and queue managers. To specify what type of object to open, as well as the name of the object, you pass in an MQOD data structure to the MQOPEN call. The MQOD data structure holds information about the object to be opened, including the type of object and the name of the object. In BASIC, the MQOD is represented as a fixed-format dynamic array. Information, such as the object type and object name, are stored in specific attribute positions of the array.

The object type, shown in the following example, is stored in attribute position **three**, and the object name is stored in attribute position **four**:

```
MY.MQOD<3> = MQOT_Q ;* MQOT_Q denotes an object of type queue.
```

```
MY.MQOD<4> = "MYQUEUE" ;* The name of the queue to open.
```

The INCLUDE file U2MQI.H holds a set of EQUATE statements that can be used to give programmer-friendly names to the attribute positions of the MQI dynamic arrays. For example, the previous example could be written as:

```
MY.MQOD<MQOD.ObjectType> = MQOT_Q
```

```
MY.MQOD<MQOD.ObjectName> = "MYQUEUE"
```

The names generated by the EQUATE statements correspond to the data structure field names found in the core MQI. These names are prefixed with the name of the data structure, for instance, "MQOD", to avoid name collisions. A complete list of the EQUATE statements related to MQI dynamic arrays can be found on [page 36, CallMQI dynamic array structures](#).

The INCLUDE file U2MQI.H holds a pre-initialized "default" EQUATE statement for each MQI dynamic array structure. You can use these EQUATE statements to initialize your own dynamic array instances with all attributes filled in with typical default values. For example,

```
MY.MQOD = MQOD.DFLT
```

```
MY.MQOD<MQOD.ObjectType> = MQOT_Q
```

```
MY.MQOD<MQOD.ObjectName> = "MYQUEUE"
```

Using the pre-initialized defaults allows you to avoid having to explicitly fill every attribute of your MQI dynamic array structure.

## **U2MQI.H INCLUDE file**

The U2MQI.H INCLUDE file contains a set of EQUATE statements that are used for working with CallMQI. In addition to the dynamic array structure definitions and defaults described above, the INCLUDE file also contains a set of named constants for use with the API. These constants correspond to constants in the core MQI API, and can be used in your BASIC programs.

## Error handling in CallMQI

The return value of all CallMQI functions is a status code indicating success (MQCC\_OK), a warning (MQCC\_WARNING) or a failure (MQCC\_FAILED). In the case of a warning or a failure, additional information can be found by calling the MQGETERROR function. When called, MQGETERROR returns the last occurring reason code and corresponding reason description in its two output parameters.

## Encoding message IDs and correlation IDs

The CallMQI interface uses BASIC dynamic arrays to hold the equivalent of MQI structures, and these dynamic arrays are delimited with value marks and sub-value marks, so it is important that the data stored in these arrays not contain embedded mark characters. Correlation IDs and Message IDs that are auto-generated by WebSphere MQ can potentially contain these characters. To protect against these mark characters corrupting the structure of the dynamic arrays, the CallMQI interface uses Base64-encoding when working with these fields, according to the following methodology:

- When a Correlation ID or a Message ID is pulled from the MQI layer into one of these arrays, it is converted to a Base-64 encoded string.
- Just before a Message ID or Correlation ID is presented to the MQI layer from UniVerse, it is converted back to its regular representation.

If you generate your own Correlation IDs or Message IDs from within a BASIC program, be sure to use the ENCODE function to Base-64 encode the ID before placing it in the dynamic array. Similarly, if you need to access the actual value of the ID from within your BASIC program, use the ENCODE function (with action set to "2" - decode).

## CallMQI programmatic interfaces

This section provides information on CallMQI functions and parameters.



*Note: A set of named constants are available in the INCLUDE file INCLUDE/U2MQI.H.*

The MQI is the core programming interface into WebSphere MQ. Its programming model consists of two primary facets:

- A set of functions for calling messaging services
- A set of data structures passed as parameters to those functions

The number of functions in the MQI is relatively small. They define the types of messaging actions available through the API. The richness of the API is conveyed through the data structures. These data structures, supplied as parameters to the MQI functions, allow one to fine-tune how the messaging actions behave.

The MQI itself is ported to many different programming languages, and the nature of these data structures depends on the language in question. In UniVerse BASIC and UniBasic, the data structures are represented as dynamic arrays.

The BASIC dynamic arrays that comprise the MQI's data structures are of fixed format. Information is stored in predetermined attribute positions.

As an example, consider the MQI's Object Descriptor data structure (MQOD). An MQOD data structure is used in conjunction with the MQOPEN function. MQOPEN can open various types of WebSphere MQ objects, including queues, channels, and queue managers. To specify what type of object to open, as well as the name of the object, you must pass in an MQOD to the MQOPEN call. The MQOD data structure holds information about the object to be opened, including the type of object and the name of the object. In BASIC, the MQOD is represented as a fixed-format dynamic array. Information, such as the object type and object name, are stored in specific attribute positions of the array. The object type, for example, is stored in attribute position three, and the object name is stored in attribute position four, as shown below:

MY.MQOD<3> = MQOT\_Q ;\* MQOT\_Q denotes an object of type queue.

MY.MQOD<4> = "MYQUEUE" ;\* The name of the queue to open

As a programming aid, the INCLUDE file U2MQI.H holds a set of EQUATE statements that give programmer-friendly names to the attribute positions of the MQI dynamic arrays. For example, the above could be written instead as

MY.MQOD<MQOD.ObjectType> = MQOT\_Q

MY.MQOD<MQOD.ObjectName> = "MYQUEUE"

These names correspond to the data structure field names found in the core MQI, prefixed with the name of the data structure, for instance, "MQOD.", to avoid name-collisions. For a complete listing of the EQUATE statements related to MQI dynamic arrays, see the section in this manual for the MQI dynamic array structure in question.

As an additional programming aid, the INCLUDE file U2MQI.H holds an EQUATE statements of a pre-initialized "default" for each MQI dynamic array structure. You can use these EQUATE statements to initialize your own dynamic array instances with all attributes filled in with typical default values. For example,

MY.MQOD = MQOD.DFLT

MY.MQOD<MQOD.ObjectType> = MQOT\_Q

MY.MQOD<MQOD.ObjectName> = "MYQUEUE"

Using the pre-initialized defaults allows you to avoid having to explicitly fill every attribute of your MQI dynamic array structure.

## Connecting to a queue manager

The **MQCONN()** function connects an application to a WebSphere MQ queue manager.

### Syntax

status =**MQCONN**(*qManager*, *hConn*)

## Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>qManager</i>	The name of the queue manager to which you are connecting. [IN]
<i>hConn</i>	The handle denoting the connection to the Websphere MQ queue manager. Use this handle in subsequent calls to other CallMQI functions. [IN]

### MQCONN Parameters

The following table describes the meaning of each return code.

Return Code	Description
0 - MQCC_OK	Function call completed successfully.
1 - MQCC_WARNING	The function call succeeded, but a warning was returned. The MQGETERROR function can be called for further details about the warning.
2 - MQCC_FAILED	The function call failed. The MQGETERROR function can be called for further details about the failure.

### Return Code Status

## Usage notes

MQGETERROR ()- If the return code status is MQCC\_WARNING or MQCC\_FAILED, the MQGETERROR function can be called to get detailed information about a warning or error.

A complete list of the reason codes can be found in Appendix C.

Refer to the *WebSphere MQ Application Programming Reference* manual for additional information about the MQCONN function.

Refer to the *WebSphere MQ Messages* manual for more information about the WebSphere MQ reason codes.

## Opening a queue

The **MQOPEN()** function opens a queue or other WebSphere MQ object.

### Syntax

```
status = MQOPEN (hConn, objDesc, options, hObj,)
```

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>hConn</i>	The handle representing the WebSphere MQ connection to the queue manager. [IN]
<i>objDesc</i>	The object descriptor. <i>objDesc</i> is an MQOD Dynamic Array Structure that identifies the object to be opened. For convenience, the U2MQI interface provides a default MQOD dynamic array, MQOD.DFLT.  In most scenarios, this parameter is solely an input parameter. However, if the object to be opened is a model queue, then a dynamic local queue is created based on the attributes of the model queue; in this case, the name of the newly created dynamic queue is returned in <i>objDesc</i> <MQOD.ObjectName>. [IN/OUT]
<i>options</i>	One or more option codes (MQOO_*) that specify how the WebSphere MQ queue or object is to be opened. If required, multiple option codes can be supplied by adding them together. [IN]
<i>hObj</i>	The object handle representing the opened object upon call completion. [IN/OUT]

### MQOPEN Parameters

The following table describes the meaning of each return code.

Return Code	Description
0 - MQCC_OK	Function call completed successfully.
1 - MQCC_WARNING	The function call succeeded, but a warning was returned. The MQGETERROR function can be called for further details about the warning.
2 - MQCC_FAILED	The function call failed. The MQGETERROR function can be called for further details about the failure.
<b>Return Code Status</b>	

### ***Usage notes***

MQGETERROR ()- If the return code status is MQCC\_WARNING or MQCC\_FAILED, the MQGETERROR function can be called to get detailed information about a warning or error.

A complete list of the reason codes can be found in Appendix C.

Refer to the *WebSphere MQ Application Programming Reference* manual for additional information about the MQOPEN function.

Refer to the *WebSphere MQ Messages* manual for more information about the WebSphere MQ reason codes.

## **Retrieving a message from a queue**

The **MQGET** function retrieves a message from the queue manager using the MQOPEN call.

### **Syntax**

```
status =MQGET(hConn, hObj, msgDesc, getMsgOpts, msgBufferLen,  
msg, msgDataLen)
```

## **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>hConn</i>	The handle to a WebSphere MQ connection.[IN]
<i>hObj</i>	The handle to a WebSphere MQ queue or object [IN]
<i>msgDesc</i>	An MQMD dynamic array describing the attributes of the message being retrieved from the queue. Upon return of MQGET, the dynamic array fields contain information on the message. Please see the section MQMD Dynamic Array Structure for details on the structure and fields of this dynamic array.  For convenience, the CallMQI interface provides a default MQMD dynamic array, MQMD.DFLT, for initializing MQMD dynamic arrays. [IN/OUT]
<i>getMsgOpts</i>	An MQGMO dynamic array describing options on how the message is to be retrieved from the queue. Upon return of MQGET, the dynamic array fields contain information on the status of the get. Please see the section MQGMO Dynamic Array Structure for details on the structure and fields of this dynamic array.  For convenience, the CallMQI interface provides a default MQGMO dynamic array, MQGMO.DFLT, for initializing MQGMO dynamic arrays. [IN/OUT]

### **MQGET Parameters**

---

Parameter	Description
<i>msgBufferLen</i>	The size of the buffer needed to retrieve the message. Specify a size of zero if the message is to be removed from the queue and discarded (also specify the MGMO_ACCEPT_TRUNCATED_MSG in getMsgOpts<MQGMO.Options> in this case). [IN]
<i>msg</i>	The message retrieved from the queue. [OUT]
<i>msgDataLen</i>	The actual length of the message retrieved from the queue. This may be different than the length of the result stored in the <i>msg</i> parameter if truncated messages are allowed. The queue manager sets this output parameter to the actual size of the message regardless of whether truncated messages are accepted. This allows the program to reissue the request with a larger <i>msgBufferLen</i> if needed. [OUT]

---

#### MQGET Parameters (Continued)

The following table describes the meaning of each return code.

---

Return Code	Description
0 - MQCC_OK	Function call completed successfully.
1 - MQCC_WARNING	The function call succeeded, but a warning was returned. The MQGETERROR function can be called for further details about the warning.
2 - MQCC_FAILED	The function call failed. The MQGETERROR function can be called for further details about the failure.

---

#### Return Code Status

### ***Usage notes***

MQGETERROR ()- If the return code status is MQCC\_WARNING or MQCC\_FAILED, the MQGETERROR function can be called to get detailed information about a warning or error.

A complete list of the reason codes can be found in Appendix C.

Refer to the *WebSphere MQ Application Programming Reference* manual for additional information about the MQGET function.

Refer to the *WebSphere MQ Messages* manual for more information about the WebSphere MQ reason codes.

## Putting a message on a queue

The **MQPUT()** function puts a message on the WebSphere MQ queue using the MQOPEN call.

### Syntax

```
status = MQPUT (hConn, hObj, msgDesc, putMsgOpts, msg)
```

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>hConn</i>	The handle representing the WebSphere MQ connection to the queue manager. [IN]
<i>hObj</i>	The handle to a WebSphere MQ queue or object. [IN]

**MQPUT Parameters**

---

Parameter	Description
<i>msgDesc</i>	An MQMD dynamic array describing the attributes of the message being retrieved from the queue. Upon return of MQGET, the dynamic array fields contain information on the message. Please see the section MQMD Dynamic Array Structure for details on the structure and fields of this dynamic array.
<i>putMsgOpts</i>	For convenience, the CallMQI interface provides a default MQMD dynamic array, MQMD.DFLT. [IN/OUT]
<i>msg</i>	An MQPMO dynamic array describing options on how the message is to be placed on the queue. Upon return of MQPUT, the dynamic array fields contain information on the status of the put. Please see the section MQPMO Dynamic Array Structure for details on the structure and fields of this dynamic array.
	For convenience, the CallMQI interface provides a default MQPMO dynamic array, MQPMO.DFLT. [IN/OUT]
	The message to be put on the queue. [IN]

---

#### MQPUT Parameters (Continued)

The following table describes the meaning of each return code.

---

Return Code	Description
0 - MQCC_OK	Function call completed successfully.
1 - MQCC_WARNING	The function call succeeded, but a warning was returned. The MQGETERROR function can be called for further details about the warning.
2 - MQCC_FAILED	The function call failed. The MQGETERROR function can be called for further details about the failure.

---

#### Return Code Status

### **Usage notes**

MQGETERROR ()- If the return code status is MQCC\_WARNING or MQCC\_FAILED, the MQGETERROR function can be called to get detailed information about a warning or error.

A complete list of the reason codes can be found in Appendix C.

Refer to the *WebSphere MQ Application Programming Reference* manual for additional information about the MQPUT function.

## **Putting a single message on a queue**

The **MQPUT1()** function opens a queue or other WebSphere MQ object.

### **Syntax**

```
status = MQPUT1(hConn, objDesc, msgDesc, putMsgOpts, msg)
```

### **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>hConn</i>	The handle representing the WebSphere MQ connection to the queue manager. [IN]
<i>hObj</i>	The handle to a WebSphere MQ queue or object. [IN]

**MQPUT1 Parameters**

---

<b>Parameter</b>	<b>Description</b>
<i>msgDesc</i>	An MQMD dynamic array describing the attributes of the message being retrieved from the queue. Upon return of MQGET, the dynamic array fields contain information on the message. Please see the section MQMD Dynamic Array Structure for details on the structure and fields of this dynamic array.
<i>putMsgOpts</i>	For convenience, the CallMQI interface provides a default MQMD dynamic array, MQMD.DFLT, for initializing new MQMD dynamic arrays. [IN/OUT]
<i>msg</i>	An MQPMO dynamic array describing options on how the message is to be placed on the queue. Upon return of MQPUT, the dynamic array fields contain information on the status of the put. Please see the section MQPMO Dynamic Array Structure for details on the structure and fields of this dynamic array.
	For convenience, the CallMQI interface provides a default MQPMO dynamic array, MQPMO.DFLT, for initializing new MQPMO dynamic arrays. [IN/OUT]
	The message to be put on from the queue. [IN]

---

#### **MQPUT1 Parameters (Continued)**

The following table describes the meaning of each return code.

---

<b>Return Code</b>	<b>Description</b>
0 - MQCC_OK	Function call completed successfully.
1 - MQCC_WARNING	The function call succeeded, but a warning was returned. The MQGETERROR function can be called for further details about the warning.
2 - MQCC_FAILED	The function call failed. The MQGETERROR function can be called for further details about the failure.

---

#### **Return Code Status**

## **Usage notes**

MQGETERROR ()- If the return code status is MQCC\_WARNING or MQCC\_FAILED, the MQGETERROR function can be called to get detailed information about a warning or error.

A complete list of the reason codes can be found in Appendix C.

Refer to the *WebSphere MQ Application Programming Reference* manual for additional information about the MQPUT1 function.

Refer to the *WebSphere MQ Messages* manual for more information about the WebSphere MQ reason codes.

## **Querying the attributes**

The **MQINQ()** function queries the attributes of a queue or other object.

### **Syntax**

status =**MQINQ**(*hConn*, *hObj*, *selectors*, *attrs*)

### **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>hConn</i>	The handle representing the queue manager connection. [IN]
<i>hObj</i>	The handle to a WebSphere MQ queue or object. [IN]
<i>selectors</i>	An @AM-delimited list of selectors to inquire upon. See the <i>WebSphere MQ Application Programming Reference</i> manual for more information on selectors. [IN]
<i>attrs</i>	An @AM-delimited list of inquired upon values that correspond to the list of selectors passed in. [OUT]

**MQINQ Parameters**

The following table describes the meaning of each return code.

Return Code	Description
0 - MQCC_OK	Function call completed successfully.
1 - MQCC_WARNING	The function call succeeded, but a warning was returned. The MQGETERROR function can be called for further details about the warning.
2 - MQCC_FAILED	The function call failed. The MQGETERROR function can be called for further details about the failure.
<b>Return Code Status</b>	

### ***Usage notes***

MQGETERROR ()- If the return code status is MQCC\_WARNING or MQCC\_FAILED, the MQGETERROR function can be called to get detailed information about a warning or error.

A complete list of the reason codes can be found in Appendix C.

Refer to the *WebSphere MQ Application Programming Reference* manual for additional information about the MQINQ function.

Refer to the *WebSphere MQ Messages* manual for more information about the WebSphere MQ reason codes.

## **Closing a queue**

The **MQCLOSE()** function closes access to a queue or other object. When you close the queue, the queue and all uncommitted messages on the queue are deleted.

### **Syntax**

status =MQCLOSE(*hConn*, *hObj*, *options*)

## Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>hConn</i>	A handle denoting the connection to the queue manager. [IN]
<i>hObj</i>	The handle to the WebSphere MQ queue, or object being closed. Upon successful completion of MQCLOSE, <i>hObj</i> is set to MQHO_UNUSABLE_HOBJ. [IN/OUT]
<i>options</i>	One or more option codes (MQCO_*) that specify how the WebSphere MQ queue or object is to be closed. If required, multiple option codes can be supplied by adding them together.  For a complete description of the option codes available to MQCLOSE, please see the WebSphere MQ Application Programming Reference manual. [IN]

### MQCLOSE Parameters

The following table describes the meaning of each return code.

Return Code	Description
0 - MQCC_OK	Function call completed successfully.
1 - MQCC_WARNING	The function call succeeded, but a warning was returned. The MQGETERROR function can be called for further details about the warning.
2 - MQCC_FAILED	The function call failed. The MQGETERROR function can be called for further details about the failure.

### Return Code Status

## Usage notes

MQGETERROR ()- If the return code status is MQCC\_WARNING or MQCC\_FAILED, the MQGETERROR function can be called to get detailed information about a warning or error.

A complete list of the reason codes can be found in Appendix C.

Refer to the *WebSphere MQ Application Programming Reference* manual for additional information about the MQCLOSE function.

Refer to the *WebSphere MQ Messages* manual for more information about the WebSphere MQ reason codes.

## Disconnecting from a queue manager

The **MQDISC** function terminates connections to the queue manager that were created using the MQCONN function. The input for this function is the hConn connection handle returned by the MQCONN function.

### Syntax

```
status = MQDISC(hConn)
```

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>hConn</i>	The handle denoting the connection to the Websphere MQ queue manager. Upon successful completion, the MQDISC function sets this to MQHC_UNUSABLE_CONNECTION. [IN/OUT]

#### MQDISC Parameters

The following table describes the meaning of each return code.

Return Code	Description
0 - MQCC_OK	Function call completed successfully.
1 - MQCC_WARNING	The function call succeeded, but a warning was returned. The MQGETERROR function can be called for further details about the warning.
2 - MQCC_FAILED	The function call failed. The MQGETERROR function can be called for further details about the failure.

#### Return Code Status

## **Usage notes**

**MQGETERROR ()-** If the return code status is MQCC\_WARNING or MQCC\_FAILED, the MQGETERROR function can be called to get detailed information about a warning or error.

A complete list of the reason codes can be found in Appendix C.

Refer to the *WebSphere MQ Application Programming Reference* manual for additional information about the MQDISC function.

Refer to the *WebSphere MQ Messages* manual for more information about the WebSphere MQ reason codes.

## **Returning an error message**

The **MQGETERROR** function returns detailed error or warning information about the last error or warning that occurred in a CallMQI call.

### **Syntax**

status =**MQGETERROR**(reasonCode, reasonMessage)

### **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>reasonCode</i>	The reason code for the last error or warning that occurred during the call.
<i>reasonMessage</i>	A detailed message that describes the reason for the last error message or warning.

**MQGETERROR Parameters**

The following table describes the status of each error code.

Return Code	Status
0 - MQCC_OK	Function call completed successfully.
2 - MQCC_FAILED	The function call failed.
<b>Return Code Status</b>	

### ***Usage notes***

A complete list of the reason codes can be found in Appendix D.

Refer to the *WebSphere MQ Messages* manual for more information about the WebSphere MQ reason codes.

## CallIMQI programming examples

### ***MQCLOSE***

The following example demonstrates how to close the queue referred to by HOBJ:

```
RES = MQCLOSE (HCONN, HOBJ, MQCO_NONE)
IF RES # MQCC_OK THEN
    * Call MQGETERROR for details...
END
```

Close the permanent dynamic queue referred to by HOBJ, and delete it upon closing

```
RES = MQCLOSE (HCONN, HOBJ, MQCO_DELETE)
```

### ***MQCONN***

The following example demonstrates how to open a connection to the queue manager:

```
RES = MQCONN ( "my.queue.manager" , HCONN)
IF RES # MQCC_OK THEN
    * Call MQGETERROR for details
    RES = MQGETERROR (REASONCODE, MSG)
    * ...
END
```

### ***MQDISC***

The following example demonstrates how to close a connection to the queue manager:

```
RES = MQDISC (HCONN)
IF RES # MQCC_OK THEN
    * Call MQGETERROR for details
    RES = MQGETERROR (REASONCODE, MSG)
    * ...
END
```

## **MQGET**

The following example demonstrates how to get a message that is on the message queue:

```
MSGDESC      = MQMD.DFLT
GETMSGOPTS   = MQGMO.DFLT
MSGBUFFERLEN = 128
MSG          = ""
MSGDATALEN   = 0
RES = MQGET(HCONN, HOBJ, MSGDESC, GETMSGOPTS, MSGBUFFERLEN, MSG,
MSGDATALEN)
IF RES # MQCC_OK THEN
    * Call MQGETERROR for details...
END
```

## **MQGETERROR**

The following example demonstrates how to retrieve the error or warning details after a function call:

```
RES = MQCONN("my.queue.manager", HCONN)
IF RES = MQCC_FAILED OR RES = MQCC_WARNING THEN
    * Call MQGETERROR for details
    RES2 = MQGETERROR(REASONCODE, MSG)
    IF RES = MQCC_FAILED THEN
        PRINT "MQCONN failed: " : REASONCODE : ":" : MSG
    END ELSE IF RES = MQCC_WARNING THEN
        PRINT "MQCONN returned with warning: " : REASONCODE : ":" :
MSG
    END
END
```

## **MQINQ**

The following example demonstrates how to inquire about some attribute of a message queue:

```
SELECTORS = ""
SELECTORS<1> = MQIA_CURRENT_Q_DEPTH
SELECTORS<2> = MQIA_OPEN_INPUT_COUNT
SELECTORS<3> = MQCA_CREATION_DATE
SELECTORS<4> = MQCA_CREATION_TIME
RES = MQINQ(HCONN, HOBJ, SELECTORS, ATTRS)
IF RES = MQCC_OK THEN
    PRINT "Current Q Depth = " : SELECTORS<1>
    PRINT "Open Input Count = " : SELECTORS<2>
    PRINT "Creation Date   = " : SELECTORS<3>
    PRINT "Creation Time   = " : SELECTORS<4>
END
```

## **MQOPEN**

### *Opening a Queue for Input and Output*

The following example demonstrates how to open a queue using the default options and parameters:

```
OBJDESC = MQOD.DFLT
OBJDESC<MQOD.ObjectName> = "QUEUE1"
RES = MQOPEN(HCONN, OBJDESC, MQOO_INPUT_SHARED + MQOO_OUTPUT,
HOBJ)
IF RES # MQCC_OK THEN
    * Check MQGETERROR for details...
END
```

### *Opening a Queue for Browsing*

The following example demonstrates how to open a queue for browsing:

```
OBJDESC = MQOD.DFLT
OBJDESC<MQOD.ObjectName> = "QUEUE1"
RES = MQOPEN (HCONN, OBJDESC, MQOO_BROWSE, HOBJ)
IF RES # MQCC_OK THEN
    * Check MQGETERROR for details...
END
```

## *Opening a Queue with a Specific User ID*

The following example demonstrates opening a queue using a specific user ID:

```
OBJDESC = MQOD.DFLT
OBJDESC<MQOD.ObjectName> = "QUEUE1"
OBJDESC<MQOD.AlternateUserID> = "someuser"
RES = MQOPEN (HCONN, OBJDESC, MQOO_ALTERNATE_USER_AUTHORITY +
MQOO_INPUT_SHARED + MQOO_OUTPUT, HOBJ)
IF RES # MQCC_OK THEN
    * Check MQGETERROR for details...
    RES = MQGETERROR(REASONCODE, MSG)
    *
END
```

## *Opening a WebSphere MQ Process Definition*

The following example demonstrates opening a WebSphere MQ process definition:

```
OBJDESC = MQOD.DFLT
OBJDESC<MQOD.ObjectName> = "MYPROCDEFN"
OBJDESC<MQOD.ObjectType> = MQOT_PROCESS
RES = MQOPEN (HCONN, OBJDESC, MQOO_INPUT_SHARED, HOBJ)
IF RES # MQCC_OK THEN
    * Check MQGETERROR for details...
    RES = MQGETERROR(REASONCODE, MSG)
    *
END
```

## ***MQPUT***

### *Putting a Message on the Queue*

The following example demonstrates how to put a message on a queue:

```
MSGDESC      = MQMD.DFLT
PUTMSGOPTS  = MQPMO.DFLT
MSG = "Test Message"
RES = MQPUT(HCONN, HOBJ, MSGDESC, PUTMSGOPTS, MSG)
IF RES # MQCC_OK THEN
    * Call MQGETERROR for details
    RES = MQGETERROR(REASONCODE, MSG)
    *
END
```

## **MQPUT1**

### *Putting a Message on the Queue*

The following example demonstrates how to put a message onto a queue named “QUEUE1”:

```
OBJDESC = MQOD.DFLT
OBJDESC<MQOD.ObjectName> = "QUEUE1"
MSGDESC = MQMD.DFLT
PUTMSGOPTS = MQPMO.DFLT
MSG = "Test Message"
RES = MQPUT1(HCONN, OBJDESC, MSGDESC, PUTMSGOPTS, MSG)
IF RES # MQCC_OK THEN
    * Call MQGETERROR for details
    RES = MQGETERROR(REASONCODE, MSG)
    *
    ...
ENDIF
```

### *Putting a Single Message on the Queue Using a User ID*

The following example demonstrates how to put a single message on the queue using a specific user ID:

```
OBJDESC = MQOD.DFLT
OBJDESC<MQOD.ObjectName> = "QUEUE1"
OBJDESC<MQOD.AlternateUserId> = "someuser"
MSGDESC = MQMD.DFLT
PUTMSGOPTS = MQPMO.DFLT
MSG = "Test Message"
RES = MQPUT1(HCONN, OBJDESC, MSGDESC, PUTMSGOPTS, MSG)
IF RES # MQCC_OK THEN
    * Call MQGETERROR for details
    RES = MQGETERROR(REASONCODE, MSG)
    *
    ...
ENDIF
```

## CallMQI dynamic array structures

CallMQI makes use of predefined dynamic arrays in place of the C-structures found in the C-based MQI API. Default values for these predefined dynamic arrays are found in the BASIC header file U2MQI.H. When a function requires one of these MQI dynamic arrays, the default definition can be used to initialize the local variable being used as the function parameter. Once initialized in this way, specific field values for the variable can be set appropriately. As a convenience, the U2MQI.H header file also defines names for the various fields of the MQI dynamic arrays.

## MQOD dynamic array structure

The dynamic array representation of a WebSphere MQ Object Descriptor (MQOD) structure is described in the following table. The attribute column defines the attribute position of the element in the dynamic array. The identifier column holds an EQUATE constant that can be used to reference the attribute position, for convenience.

This table describes the MQOD dynamic array structure.

Attribute	Identifier	Description
1	MQOD.StrucId	Structure identifier.
2	MQOD.Version	Structure version number.
3	MQOD.ObjectType	Object type.
4	MQOD.ObjectName	Object name.
5	MQOD.ObjectQMgrName	Object queue manager name.
6	MQOD.DynamicQName	Dynamic queue name.
7	MQOD.AlternateUserId	Alternate user identifier.
8	MQOD.RecsPresent	Number of object records present.

MQOD Dynamic Array Structure

Attribute	Identifier	Description
9	MQOD.KnownDestCount	Number of local queues opened successfully.
10	MQOD.UnknownDestCount	Number of remote queues opened successfully.
11	MQOD.InvalidDestCount	Number of queues that failed to open.
12	MQOD.ObjectRecOffset	Offset of first object record from start of MQOD.
13	MQOD.ResponseRecOffset	Offset of first response record from start of MQOD.
14	MQOD.ObjectRecPtr	Address of first object record.
15	MQOD.ResponseRecPtr	Address of first response record.
16	MQOD.AlternateSecurityId	Alternate security identifier.
17	MQOD.ResolvedQName	Resolved queue name.
18	MQOD.ResolvedQMgrName	Resolved queue manager name.

#### **MQOD Dynamic Array Structure (Continued)**

### ***MQOD.DFLT***

MQOD.DFLT is a constant that defines default fields for an MQOD dynamic array. You can use it as a convenience when initializing your own MQOD dynamic arrays. The table below describes these default values.

Attribute	Identifier	Description
1	MQOD.DFLT<MQOD.StrucId>	MQOD_STRUC_ID
2	MQOD.DFLT<MQOD.Version>	MQOD_VERSION_1
3	MQOD.DFLT<MQOD.ObjectType>	MQOT_Q

#### **MQOD.DFLT Dynamic Array Structure**

---

Attribute	Identifier	Description
4	MQOD.DFLT<MQOD.ObjectName>	""
5	MQOD.DFLT<MQOD.ObjectQMgrName>	""
6	MQOD.DFLT<MQOD.DynamicQName>	"AMQ.*"
7	MQOD.DFLT<MQOD.AlternateUserId>	""
8	MQOD.DFLT<MQOD.RecsPresent>	0
9	MQOD.DFLT<MQOD.KnownDest-Count>	0
10	MQOD.DFLT<MQOD.UnknownDest-Count>	0
11	MQOD.DFLT<MQOD.InvalidDest-Count>	0
12	MQOD.DFLT<MQOD.ObjectRecOffset>	0
13	MQOD.DFLT<MQOD.Respon-seRecOffset>	0
14	MQOD.DFLT<MQOD.ObjectRecPtr>	MQPTR
15	MQOD.DFLT<MQOD.ResponseRecPtr>	MQPTR
16	MQOD.DFLT<MQOD.AlternateSecu-rityId>	MQBYTE40
17	MQOD.DFLT<MQOD.ResolvedQName>	""
18	MQOD.DFLT<MQOD.Resolved-QMgrName>	""

---

**MQOD.DFLT Dynamic Array Structure (Continued)**

## MQMD dynamic array structure

The dynamic array representation of a WebSphere MQ Object Descriptor (MQMD) structure is described in the following table. The attribute column defines the attribute position of the element in the dynamic array. The identifier column holds an EQUATE constant that can be used to reference the attribute position, for convenience.

This table describes the MQMD dynamic array structure.

Attribute	Identifier	Description
1	MQMD.StrucId	Structure identifier.
2	MQMD.Version	Structure version number.
3	MQMD.Report	Options for report messages.
4	MQMD.MsgType	Message type.
5	MQMD.Expiry	Message lifetime.
6	MQMD.Feedback	Feedback or reason code.
7	MQMD.Encoding	Numeric encoding of message data.
8	MQMD.CodedCharSetId	Character set identifier of message data.
9	MQMD.Format	Format name of message data.
10	MQMD.Priority	Message priority.
11	MQMD.Persistence	Message persistence.
12	MQMD.MsgId	Message identifier.
13	MQMD.CorrelId	Correlation identifier.
14	MQMD.BackoutCount	Backout counter.
15	MQMD.ReplyToQ	Name of reply queue.
16	MQMD.ReplyToQMgr	Name of reply queue manager.
17	MQMD.UserId	User identifier.

MQMD Dynamic Array Structure

---

Attribute	Identifier	Description
18	MQMD.AccountingToken	Accounting token.
19	MQMD.ApplIdentityData	Application data relating to identity.
20	MQMD.PutApplType	Type of application that put the message.
21	MQMD.PutApplName	Name of application that put the message.
22	MQMD.PutDate	Date when message was put.
23	MQMD.PutTime	Time when message was put.
24	MQMD.ApplOriginData	Application data relating to origin.
25	MQMD.GroupId	Group identifier.
26	MQMD.MsgSeqNumber	Sequence number of logical message within group.
27	MQMD.Offset	Offset of data in physical message from start of logical message.
28	MQMD.MsgFlags	Message flags.
29	MQMD.OriginalLength	Length of original message.

---

**MQMD Dynamic Array Structure (Continued)**

## **MQMD.DFLT**

MQMD.DFLT is a constant that defines default fields for an MQMD dynamic array. You can use it as a convenience when initializing your own MQMD dynamic arrays. The table below describes these default values.

Attribute	Identifier	Description
1	MQMD.DFLT<MQMD.StrucId>	MQMD_STRUC_ID
2	MQMD.DFLT<MQMD.Version>	MQMD_VERSION_1
3	MQMD.DFLT<MQMD.Report>	MQRO_NONE
4	MQMD.DFLT<MQMD.MsgType>	MQMT_DATAGRAM
5	MQMD.DFLT<MQMD.Expiry>	MQEI_UNLIMITED
6	MQMD.DFLT<MQMD.Feedback>	MQFB_NONE
7	MQMD.DFLT<MQMD.Encoding>	MQENC_NATIVE
8	MQMD.DFLT<MQMD.CodedCharSetId>	MQCCSI_Q_MGR
9	MQMD.DFLT<MQMD.Format>	MQFMT_NONE
10	MQMD.DFLT<MQMD.Priority>	MQPRI_PRIORITY_AS_Q_DEF
11	MQMD.DFLT<MQMD.Persistence>	MQPER_PERSISTENCE_AS_Q_DEF
12	MQMD.DFLT<MQMD.MsgId>	MQBYTE24
13	MQMD.DFLT<MQMD.CorrelId>	MQBYTE24
14	MQMD.DFLT<MQMD.BackoutCount>	0
15	MQMD.DFLT<MQMD.ReplyToQ>	""
16	MQMD.DFLT<MQMD.ReplyToQMgr>	""
17	MQMD.DFLT<MQMD.Identifier>	""
18	MQMD.DFLT<MQMD.AccountingToken>	MQBYTE32

### **MQMD.DFLT Dynamic Array Structure**

Attribute	Identifier	Description
19	MQMD.DFLT<MQMD.ApplIdentifyData>	""
20	MQMD.DFLT<MQMD.PutApplType>	MQAT_NO_CONTEXT
21	MQMD.DFLT<MQMD.PutApplName>	""
22	MQMD.DFLT<MQMD.PutDate>	""
23	MQMD.DFLT<MQMD.PutTime>	""
24	MQMD.DFLT<MQMD.ApplOriginData>	""
25	MQMD.DFLT<MQMD.GroupId>	MQBYTE24
26	MQMD.DFLT<MQMD.MsgSeqNumber>	1
27	MQMD.DFLT<MQMD.Offset>	0
28	MQMD.DFLT<MQMD.MsgFlags>	MQMF_NONE
29	MQMD.DFLT<MQMD.OriginalLength>	MQOL_UNDEFINED

#### **MQMD.DFLT Dynamic Array Structure (Continued)**

## **MQGMO dynamic array structure**

The dynamic array representation of a WebSphere MQ Get-Message Option (MQGMO) structure is described in the following table. The attribute column defines the attribute position of the element in the dynamic array. The identifier column holds an EQUATE constant that can be used to reference the attribute position, for convenience.

This table describes the MQGMO dynamic array structure.

Attribute	Identifier	Description
1	MQOD.StrucId	Structure identifier.
2	MQOD.Version	Structure version number.
3	MQGMO.Options	Options that control the action of MQGET.

#### **MQGMO Dynamic Array Structure**

Attribute	Identifier	Description
4	MQGMO.WaitInterval	Wait interval.
5	MQGMO.Signal1	Signal.
6	MQGMO.Signal2	Signal identifier
7	MQGMO.ResolvedQName	Resolved name of destination queue
8	MQGMO.MatchOptions	Options controlling selection criteria used for MQGET.
9	MQGMO.GroupStatus	Flag indicating whether message retrieved is in a group.
10	MQGMO.SegmentStatus	Flag indicating whether message retrieved is a segment of a logical message.
11	MQGMO.Segmentation	Flag indicating whether further segmentation is allowed for the message retrieved.
12	MQGMO.Reserved1	Reserved.
13	MQGMO.MsgToken	Message token.
14	MQGMO.ReturnedLength	Length of message data returned (bytes).

**MQGMO Dynamic Array Structure (Continued)**

## ***MQGMO.DFLT***

MQGMO.DFLT is a constant that defines default fields for an MQGMO dynamic array. You can use it as a convenience when initializing your own MQGMO dynamic arrays.

Attribute	Identifier	Description
1	MQOD.DFLT<MQOD.StrucId>	MQOD_STRUC_ID
2	MQOD.DFLT<MQOD.Version>	MQOD_VERSION_1
3	MQGMO.DFLT<MQGMO.Options>	MQGMO_NO_WAIT
4	MQGMO.DFLT<MQGMO.WaitInterval>	0
5	MQGMO.DFLT<MQGMO.Signal1>	0
6	MQGMO.DFLT<MQGMO.Signal2>	0
7	MQGMO.DFLT<MQGMO.Resolved-QName>	""
8	MQGMO.DFLT<MQGMO.MatchOptions>	MQMO_MATCH_MSG_ID + MQMO_MATCH_COR- REL_ID
9	MQGMO.DFLT<MQGMO.GroupStatus>	MQGS_NOT_IN_GROUP
10	MQGMO.DFLT<MQGMO.Segment-Status>	MQSS_NOT_A_SEGMENT
11	MQGMO.DFLT<MQGMO.Segmen-tation>	MQSEG_INHIBITED
12	MQGMO.DFLT<MQGMO.Reserved1>	""
13	MQGMO.DFLT<MQGMO.MsgToken>	MQBYTE16
14	MQGMO.DFLT<MQGMO.Return-edLength>	MQRL_UNDEFINED

**MQGMO.DFLT Dynamic Array Structure**

## MQPMO dynamic array structure

The dynamic array representation of a WebSphere MQ Put-Message Option (MQPMO) structure is described in the table below. The attribute column defines the attribute position of the element in the dynamic array. The identifier column holds an EQUATE constant that can be used to reference the attribute position, for convenience.

This table describes the MQPMO dynamic array structure.

Attribute	Identifier	Description
1	MQOD.StrucId	Structure identifier.
2	MQOD.Version	Structure version number.
3	MQPMO.Options	Options that control the action of MQPUT and MQPUT1.
4	MQPMO.Timeout	Reserved.
5	MQPMO.Context	Object handle of input queue.
6	MQPMO.KnownDestCount	Number of messages sent successfully to local queues.
7	MQPMO.UnknownDestCount	Number of messages sent successfully to remote queues.
8	MQPMO.InvalidDestCount	Number of messages that could not be sent.
9	MQPMO.ResolvedQName	Resolved name of destination queue.
10	MQPMO.ResolvedQMgrName	Resolved name of destination queue manager.
11	MQPMO.RecsPresent	Number of put message records or response records present.

MQPMO Dynamic Array Structure

Attribute	Identifier	Description
12	MQPMO.PutMsgRecFields	Flags indicating which MQPMR fields are present.
13	MQPMO.PutMsgRecOffset	Offset of first put message record from start of MQPMO.
14	MQPMO.ResponseRecOffset	Offset of first response record from start of MQPMO.
15	MQPMO.PutMsgRecPtr	Address of first put message record.
16	MQPMO.ResponseRecPtr	Address of first response record.

#### **MQPMO Dynamic Array Structure (Continued)**

### **MQGPO.DFLT**

MQPMO.DFLT is a constant that defines default fields for an MQPMO dynamic array. You can use it as a convenience when initializing your own MQPMO dynamic arrays.

Attribute	Identifier	Description
1	MQPMO.DFLT<MQPMO.StrucId>	MQPMO_STRUC_ID
2	MQPMO.DFLT<MQPMO.Version>	MQPMO_VERSION_1
3	MQPMO.DFLT<MQPMO.Options>	MQPMO_NONE
4	MQPMO.DFLT<MQPMO.Timeout>	-1
5	MQPMO.DFLT<MQPMO.Context>	0
6	MQPMO.DFLT<MQPMO.KnownDest-Count>	0
7	MQPMO.DFLT<MQPMO.Unknown-DestCount>	0

#### **MQGPO.DFLT Dynamic Array Structure**

Attribute	Identifier	Description
8	MQPMO.DFLT<MQPMO.InvalidDest-Count>	0
9	MQPMO.DFLT<MQPMO.Resolved-QName>	""
10	MQPMO.DFLT<MQPMO.Resolved-QMgrName>	""
11	MQPMO.DFLT<MQPMO.RecsPresent>	0
12	MQPMO.DFLT<MQPMO.PutMsgRec-Fields>	MQPMRF_NONE
13	MQPMO.DFLT<MQPMO.PutMs-gRecOffset>	0
14	MQPMO.DFLT<MQPMO.Respon-seRecOffset>	0
15	MQPMO.DFLT<MQPMO.PutMs-gRecPtr>	MQPTR
16	MQPMO.DFLT<MQPMO.Respon-seRecPtr>	MQPTR

**MQGPO.DFLT Dynamic Array Structure (Continued)**

---

## WebSphere MQ API for UniData and UniVerse

### WebSphere MQ Application Messaging Interface

The WebSphere MQ API for UniData and UniVerse (CallAMI) makes use of the WebSphere MQ Application Messaging Interface (AMI), available as a WebSphere MQ SupportPac (SupportPac ma0f) at <http://www.ibm.com>.

The WebSphere MQ AMI utilizes a repository of definitions that describe how and where messages are to be delivered. Service definitions determine where messages are to be delivered, and policy definitions govern how those messages are delivered. Typically, a WebSphere MQ administrator creates and administers the definitions. AMI-based programs can then reference the definitions by name. This simplifies software development and maintenance by moving messaging complexity away from the program and into the repository. The repository itself exists as an XML file on the machine running the application (although LDAP-based repositories are supported as well).

CallAMI utilizes service and policy definitions by accepting them as parameters to the various API calls. For example, when placing a message on a queue with `amSendMsg`, the properties defined by the service name and policy name passed in to the function determine where and how the message is sent.

For Windows platforms, WebSphere MQ AMI provides the AMI Administration Tool to manage the repository. It allows administration of the AMI repository through a GUI interface, rather than through direct editing of the XML file. For more information about the WebSphere MQ AMI repository, and about the AMI Administration Tool, see the *WebSphere MQ Application Messaging Interface* manual.

## Session

In the CallAMI, the sending and receiving of messages takes place within a session. You start sessions via the API call **amInitialize**, and end them with **amTerminate**. Upon successful completion, **amInitialize** returns a valid session handle in the *hSession* output variable. You then use this session handle in all subsequent WebSphere API for UniData and UniVerse calls, until you close the session with **amTerminate**. Only one session may be active at a time from within a given instance of a running BASIC program.

## Services

The CallAMI makes use of AMI *services* in its API calls. An AMI service represents a destination from where a message is to be sent to or retrieved. In other words, a service represents a particular message queue running under a particular queue manager.

Interacting with queues can be complex, owing to the many variables associated with them. Services hide this complexity by encapsulating queue parameters within the service definitions themselves. Thus, rather than explicitly setting all of the message queue parameters directly in the program, you simply reference a particular service definition by name in the CallAMI function call.

A WebSphere MQ administrator creates service definitions, and they are stored in the AMI repository. Default services are also available, and can be referenced by using an empty string ("") in place of an actual service name in the CallAMI API calls. See the *WebSphere MQ Application Messaging Interface* manual, Chapter 19: Defining Services, Policies, and Policy Handlers, under the section "Service Definitions," for default values of service properties.

For more information about AMI *services*, and about the AMI Repository, see the *WebSphere MQ Application Messaging Interface* manual.

## Policies

CallAMI makes use of AMI *policies* in its API calls. A policy represents how a message is to be sent or retrieved. For example, a policy can dictate whether truncated messages are allowed, or how many times a failed message delivery should be automatically retried. By referencing policies directly by name in the various CallAMI function calls, you avoid having to explicitly code message delivery details in your programs. Instead, this complexity is embedded in the policies themselves.

A WebSphere MQ administrator creates policy definitions, and they are stored in the AMI repository. Default policies are also available, and you can reference them by using an empty string ("") in place of an actual policy name in CallAMI calls. See the *WebSphere MQ Application Messaging Interface* manual, Chapter 19: Defining Services, Policies, and Policy Handlers under the section “Policy Definitions,” for default policy values.

For more information about AMI *policies*, and about the AMI repository, see the *WebSphere MQ Application Messaging Interface* manual.

## Message objects

When a message is sent or received via CallAMI calls, an AMI message object is created implicitly, behind the scenes. The message object encapsulates details of the message, such as its WebSphere MQ message headers, including the Message ID and Correlation ID, along with the message data itself. You can name, or *tag*, these message objects by providing names for them in the API calls. For example, in the **amSendRequest** function, you can use the *sndMsgName* parameter to specify a name for the underlying message object that gets created during the call. By tagging a message object in this manner, you can later reference that same message object in subsequent API calls. This is particularly important when correlating requests and responses in the Request/Response messaging style.



*Note:* It is not required to name the underlying message objects. You can use an empty string (""), especially in cases where the message object will not need to be referenced later, as is often the case for the Datagram messaging style.

## Messaging styles

CallAMI supports two messaging styles, Datagram and Request/Response.

### Datagram messaging style

The Datagram messaging style, also known as “Send-and-Forget,” is the simplest type of messaging. It involves an application placing a message on a queue without requiring a response back. A receiving application picks up the message and performs work based on the message contents, but does not respond back to the sending application.

Datagram messaging involves the use of two functions:

- amSendMsg
- amReceiveMsg

### Request/response messaging style

Request/Response messaging follows the client/server paradigm. One application, acting as the client, sends a *request message* to a queue. A server application picks up and processes the message, and then sends a *response message* to another queue, which the client monitors for a reply. The client then picks up the response message and acts on it appropriately.

Request/Response messaging relies on four functions, separated according to client or server functionality.

#### *Client Request/Response Functions*

- amSendRequest
- amReceiveMsg

#### *Server Request/Response Functions*

- amReceiveRequest
- amSendResponse

Due to the nature of messaging, where communication is accomplished through an intermediary—for example, a message queue—the notion of correlating requests and responses is important for the Request/Response messaging style. Without a means for this correlation, a requesting application would not be able to differentiate the response message from any other message appearing on the queue. WebSphere MQ accomplishes this correlation through the use of Correlation ID's, which are stored in the message header. In Request/Response messaging, the responding application copies the request message's Message ID into the Correlation ID field of the response message. The requesting application can then select the correct response message from the queue through the use of this Correlation ID.

CallAMI handles this correlation process transparently through the use of AMI *message objects*. Thus, you need not be concerned with the details of copying Message ID's to Correlation ID fields, or filtering messages based on their Correlation ID.

In the client application, this is accomplished by using the message object created in **amSendRequest** as the message selection criteria in **amReceiveMsg**. So, in the call to **amSendRequest**, you tag the underlying request message object by giving it a name via the *sndMsgName* parameter. This name is then used as the *selMsgName* parameter in the subsequent call to **amReceiveMsg**. Behind the scenes, the Message ID of the request message sent out via **amSendRequest**, is used by **amReceiveMsg** to select the correct response message based on its Correlation ID field.

In the server application, a similar process takes place between the calls **amReceiveRequest** and **amSendResponse**. When receiving a request message through **amReceiveRequest**, you tag the underlying request message object by giving it a name via the *rcvMsgName* parameter. This named message object is then used in the subsequent call to **amSendResponse**, through that function's *rcvMsgName* parameter. Behind the scenes, the Message ID from the request message is copied over to the Correlation ID field of the response message, which is then sent out through **amSendResponse**.

## **CallAMI setup and configuration**

### **Requirements**

- UniData 6.0 or later
- UniVerse 10.1 or later
- WebSphere MQ version 6.0
- WebSphere MQ Application Messaging Interface SupportPac  
SupportPac ma0f - available for download from  
<http://www.ibm.com/support>.

### **Platform availability**

CallAMI is available on the following platforms.

- AIX
- SUN Solaris
- HP-UX - V11.0 PA2
- Windows NT/2000/XP/2003

### **Setting up the environment for CallAMI**

You must complete the following steps in order to set up your environment to utilize CallAMI. First you must install either WebSphere MQ, or WebSphere MQ Client on the UniData or UniVerse machine. For information about installing WebSphere MQ, see the *WebSphere MQ Quick Beginnings* manual that ships with your copy of WebSphere MQ. For information about installing the WebSphere MQ Client, see the *WebSphere MQ* manual *WebSphere MQ Clients*. These manuals are also available for download from <http://www.ibm.com/shop/publications/order>.

Once WebSphere MQ or the WebSphere MQ Client is installed on the UniData or UniVerse machine, you should install the WebSphere MQ AMI SupportPac. The AMI is available for download from:

<http://www.ibm.com/support>.

For information about installing the WebSphere MQ AMI SupportPac, see the *WebSphere MQ Application Messaging Interface* manual, available for download from the same location, and from:

<http://www.ibm.com/shop/publications/order>.

More information on using WebSphere MQ Client from within UniData or UniVerse can be found in "["WebSphere MQ and UniData or UniVerse on separate machines."](#)"

## Configurations

Although in most cases a WebSphere MQ queue manager runs on the same machine as the UniData or UniVerse database, this is not strictly necessary. Instead, UniData or UniVerse can communicate directly with a queue manager running on a different machine by acting as a WebSphere MQ Client. In this configuration, CallAMI function calls make use of WebSphere MQ client libraries that marshall the calls across the network to the remote queue manager.

Both scenarios have their particular benefits, and the choice of configuration depends on the particular application. For more information about running an application as a WebSphere MQ Client, see the *WebSphere MQ Clients* manual, which ships with WebSphere MQ, and is also available for download from <http://www.ibm.com/shop/publications/order>.

### ***WebSphere MQ and UniData or UniVerse on the same machine***

To connect to a WebSphere MQ queue manager running on the same machine as UniData or UniVerse, no special steps are required. This is often referred to in various WebSphere MQ documentation as connecting via the WebSphere MQ "server libraries." By default, AMI policy settings are such that the WebSphere MQ server libraries are used if they are installed on the given machine (the machine running UniData or UniVerse). The server libraries are installed when the full version of WebSphere MQ is installed. The server libraries allow applications to connect only to queue managers running on the local machine. When messages must be delivered across a network, the queue managers handle this automatically (after being configured appropriately by the WebSphere MQ Administrator).

## ***WebSphere MQ and UniData or UniVerse on separate machines***

It is also possible to run WebSphere MQ on a separate machine from the UniData or UniVerse database. In this scenario, you must install the WebSphere MQ Client on the machine running UniData or UniVerse. In addition to the WebSphere MQ Client, you must also install the WebSphere MQ AMI SupportPac on the UniData or UniVerse machine. With the WebSphere MQ Client installed, the UniData or UniVerse database can connect to a remote queue manager via the WebSphere MQ “client libraries.” To connect to a queue manager via the WebSphere MQ client libraries, the AMI policy governing the connection must be configured appropriately.

Complete the following steps to connect to a remote queue manager:

1. Install the WebSphere MQ Client on the database server.
2. Install the WebSphere MQ AMI SupportPac on the database server.
3. Configure your AMI policy to connect as a WebSphere MQ Client.
4. Set up a listener for the queue manager on the remote machine.

### *Install the WebSphere MQ client on the UniData/UniVerse machine*

For WebSphere MQ Client installation instructions, see the *WebSphere MQ Client* manual, available with the WebSphere MQ installation, and also for download from <http://www.ibm.com/shop/publications/order>.

### *Install the WebSphere MQ AMI SupportPac on the UniData/UniVerse machine*

For WebSphere MQ AMI installation instructions, see the *WebSphere MQ Application Messaging Interface* manual, available from the WebSphere MQ AMI SupportPac page (<http://www.ibm.com>).

### *Configure your AMI policy to connect as an WebSphere MQ Client*

The following AMI Policy attributes are used when connecting through the WebSphere MQ client libraries:

#### *Connection type*

If Connection Type is set to “Auto” (the default), the application automatically detects if it should connect directly (via the server libraries), or as a client. If Connection Type is “Client,” the application connects as a client. If Connection Type is “Server,” the application connects directly to the local queue manager.

#### *Client channel name*

For a WebSphere MQ client connection, the name of the server-connection channel. See the *WebSphere MQ Client* manual for more information about WebSphere MQ Channels. You can use this attribute, in combination with the Client TCP Server Address attribute, instead of the MQSERVER environment variable on the WebSphere MQ client. For more information about the MQSERVER environment variable, see the *WebSphere MQ Clients* manual.

#### *Client TCP server address*

For an WebSphere MQ client connection, the TCP/IP host name or IP address, and optional port, of the WebSphere MQ server, in the form *hostname(portnumber)*. You can use this attribute, in combination with the Client Channel Name attribute, instead of the MQSERVER environment variable on the WebSphere MQ client. For more information about the MQSERVER environment variable, see the *WebSphere MQ Clients* manual.

#### *Setup a listener for the queue manager on the remote machine*

The queue manager running on the remote machine must have a listener configured to accept the requests from the WebSphere MQ client application. See the *WebSphere MQ Clients* manual, available with the WebSphere MQ installation, and also for download from <http://www.ibm.com/shop/publications/order>, for information on setting up an WebSphere MQ listener.

# CallAMI programmatic interfaces

This section provides information on the WebSphere MQ functions and parameters for UniData and UniVerse.



**Note:** A set of named constants are available in the include file INCLUDE/U2AMI.H.

## Initialize an AMI session

The **amInitialize()** function creates and opens an AMI session. The output parameter *hSession* is a session handle which is valid until the session is terminated. The function returns a status code indicating success, warning or failure. You can also use The BASIC STATUS() function to obtain this value.

### Syntax

```
ret = amInitialize(hSession, appName, policyName, reasonCode)
```

The following table describes each parameter of the syntax.

Parameter	Description
<i>hSession</i>	Upon successful return, holds a handle to a session. You can then use this handle in other UniData and UniVerse WebSphere MQ API calls. [OUT]
<i>appName</i>	An optional name you can use to identify the session. [IN]
<i>policyName</i>	The name of a policy. If you specify as "" (null), amInitialize uses the system default policy name. [IN]
<i>reasonCode</i>	Holds an AMI Reason Code when the function returns a status indicating an AMI warning or an AMI error occurred. You can use the AMI Reason Code to obtain more information about the cause of the warning or error. See Appendix A, “ <a href="#">Appendix A: MQSeries API for UniData and UniVerse reason codes</a> ” for a list of AMI Reason Codes and their descriptions. [OUT]

### amInitialize Parameters

The following table describes the status of each return code.

Return Code	Status
0 - AMCC_SUCCESS	Function completed successfully.
1 - AMCC_WARNING	A warning was returned from AMI. The <i>reasonCode</i> output parameter contains an AMI reason code with further details about the warning.
2 - AMCC_FAILED	An error was returned from AMI. The <i>reasonCode</i> output parameter contains an AMI reason code with further details about the error.
99	UniVerse failed to obtain a license for an interactive PHANTOM process.
115 - U2AMI_ERR_SESSION_IN_USE	An active session already exists (under a different <i>hSession</i> variable than the one being passed in – see Usage Notes for more details).
Other	A non-AMI error occurred.

#### Return Code Status

### **Usage notes**

Only one session may be active at one time. If you call `amInitialize` while another session is active, an error code of `U2AMI_ERR_SESSION_IN_USE` is returned. The one exception to this case is if the subsequent call to `amInitialize` uses the same *hSession* variable from the first call. In this case, the session is automatically terminated using the same AMI policy with which it was initialized, and a new session is started in its place.

### **Receiving a message**

The `amReceiveMsg()` function receives a message sent by the `amSendMsg()` function.

## Syntax

```
ret = amReceiveMsg(hSession, receiverName, policyName,  
selMsgName, maxMsgLen, dataLen, data, rcvMsgName, reasonCode)
```

## Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>hSession</i>	The session handle returned by <b>amInitialize</b> . [IN]
<i>receiverName</i>	The name of a receiver service. If you specify “” (null), amReceiveMsg uses the system default receiver name. [IN]
<i>policyName</i>	The name of a policy. If you specify “” (null), amReceiveMsg uses the system default policy name. [IN]
<i>selMsgName</i>	Optional parameter specifying the name of a message object containing information (such as a Correl ID) to use to retrieve the required message from the queue. See the Usage Notes for additional information about the use of this parameter.[IN]
<i>maxMsgLen</i>	The maximum message length the application will accept. Specify as -1 to accept messages of any length, or use the optional parameter U2AMI_RESIZEBUFFER. See the Usage Notes for additional information about the use of this parameter. [IN]
<i>dataLen</i>	The length of the received message data, in bytes. Specify as “” (null) if you do not require this parameter. [OUT]
<i>data</i>	The received message data. [OUT]
<i>rcvMsgName</i>	The name of a message object for the retrieved message. If you specify “” (null) for this parameter, amReceiveMsg uses the system default name (constant AMSD_RCV_MSG). See the Usage Notes for additional information about the use of this parameter. [IN]
<i>reasonCode</i>	Holds an AMI Reason Code when the function returns a status indicating an AMI warning or an AMI error occurred. You can use the AMI Reason Code to obtain more information about the cause of the warning or error. See Appendix A, <a href="#">“Appendix A: MQSeries API for UniData and UniVerse reason codes,”</a> for a list of AMI Reason Codes and their descriptions. [OUT]

### amReceiveMsg parameters

---

<b>Parameter</b>	<b>Description</b>
<i>optional parameter</i>	If you specify U2AMI_LEAVEMSG for this parameter, and Accept Truncated Messages is not set in the policy receive attributes, UniVerse returns the message length in the <i>dataLen</i> parameter, but the message itself remains on the queue.
U2AMI_DISCARDMSG	If you specify U2AMI_DISCARDMSG for this parameter and Accept Truncated Messages is set in the policy receive attributes, UniVerse discards the message at the WebSphere MQ level with an AMRC_MSG_TRUNCATED warning. This behavior is preferable to discarding the message at the UniVerse level.
U2AMI_RESIZEBUFFER	If you specify U2AMI_RESIZEBUFFER for this parameter, UniVerse handles the details of the buffer size used to retrieve the message. If you do not specify this parameter, you must specify the buffer size. See Usage Notes for more information about this option.

---

**amReceiveMsg parameters (Continued)**

The following table describes the status of each return code.

---

<b>Return Code</b>	<b>Status</b>
0 - AMCC_SUCCESS	Function completed successfully.
<b>Return Code Status</b>	

---

---

Return Code	Status
1 - AMCC_WARNING	A warning was returned from AMI. The <i>reasonCode</i> output parameter contains an AMI reason code with further details about the warning.
2 - AMCC_FAILED	An error was returned from AMI. The <i>reasonCode</i> output parameter contains an AMI reason code with further details about the error.
Other	A non-AMI error occurred.

---

**Return Code Status (Continued)**

### **Usage notes**

*selMsgName* – You can use this parameter in Request/Reply messaging to tell **amReceiveMsg** to retrieve from the queue only those messages that correlate with a message previously placed on the queue with *amSendRequest*. When used in this manner, use the *sndMsgName* parameter of the *amSendRequest* call as the value for *selMsgName* in *amReceiveMsg*. Message correlation occurs here due to the following:

1. The underlying message object that was created when the request message was sent, and referenced by the name *sndMsgName*, holds information about the sent message, such as its Correlation Id and Message Id.
2. When you use this message object (*sndMsgName*) as the *selMsgName* parameter to **amReceiveMsg**, the information held in this message object will be used to ensure that only correlating response messages are retrieved from the queue.

*maxMsgLen* – You can use this parameter to define the maximum length message that **amReceiveMsg** retrieves from the queue. If the value of *maxMsgLen* is less than the length of the message to be retrieved, behavior depends on whether you set the *Accept Truncated Message* parameter in the policy receive attributes to true. If *Accept Truncated Message* is set to true, the data is truncated and there is an *AMRC\_MSG\_TRUNCATED* warning in the *reasonCode* parameter. If you set *Accept Truncated Message* to false (the default), **amReceiveMsg** fails with return status *AMCC\_FAILED* (2), and *reasonCode* *AMRC\_RECEIVE\_BUFF\_LEN\_ERR*.



**Note:** If `amReceiveMsg` returns `AMRC_RECEIVE_BUFF_LEN_ERR` as the `reasonCode`, the message length value is contained in `dataLen` parameter, even though the call failed with return value `MQCC_FAILED`.

If you do not specify the `U2AMI_RESIZE BUFFER` optional parameter and the buffer size you specify with the `maxMsgLen` parameter is too small, the function fails with the `AMRC_RECEIVE_BUFF_LEN_ERR`. If this happens, UniVerse returns the necessary buffer size in the `dataLen` parameter so you can reissue the request with the correct size.

If you specify the `U2AMI_RESIZEBUFFER` parameter, UniVerse uses a default buffer size of 8K. If this buffer size is too small, UniVerse automatically reissues the request with the correct buffer size. While convenient, this behavior can result in performance degradation for the following reasons:

- If the default buffer size is larger than necessary for the received message, UniVerse incurs an unnecessary overhead.
- If the default buffer size is too small for the received message, UniVerse must issue two requests to the queue before successfully retrieving the message.

For performance reasons, we recommend you set the `maxMsgLen` parameter to the expected size of the message whenever possible.

`rcvMsgName` – This parameter allows the application to attach a name to the underlying message object used to retrieve the message. Though supported, this parameter is mainly intended for use in conjunction with future additions to the WebSphere MQ for UniData and UniVerse API.

## Receiving a request

The `amReceiveRequest()` function receives a request message.

### Syntax

```
ret = amReceiveRequest(hSession, receiverName, policyName,  
maxMsgLen, dataLen, data, rcvMsgName, senderName, reasonCode  
[U2AMI_LEAVEMSG | U2AMI_DISCARDMSG |  
U2AMI_RESIZEBUFFER])
```

## Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>hSession</i>	The session handle returned by <b>amInitialize</b> . [IN]
<i>receiverName</i>	The name of a receiver service. If you specify "" (null), amReceiveRequest uses the system default receiver name. [IN]
<i>policyName</i>	The name of a policy. If you specify "" (null), amReceiveRequest uses the system default policy name. [IN]
<i>maxMsgLen</i>	The maximum message length the application will accept. Specify as -1 to accept messages of any length, or use the optional parameter U2AMI_RESIZEBUFFER. See Usage Notes for additional information about the use of this parameter. [IN]
<i>dataLen</i>	The length of the received message data, in bytes. Specify "" (null) if you do not require this parameter. [OUT]
<i>data</i>	The received message data. [OUT]
<i>rcvMsgName</i>	The name of the message object for the received message. If you specify "" (null), amReceiveRequest uses the system default receiver service. The value for <i>rcvMsgName</i> will be used in the subsequent call to <b>amSendResponse</b> . [IN]
<i>senderName</i>	The name of a special type of sender service known as a response sender, to which the response message will be sent. If you do not specify a name, amReceiveRequest uses the system default response sender service. [IN]  <b>Note:</b> The sender name you specify must exist in your AMI repository.
<i>reasonCode</i>	Holds an AMI Reason Code when the function returns a status indicating an AMI warning or an AMI error occurred. You can use the AMI Reason Code to obtain more information about the cause of the warning or error. See Appendix A, " <a href="#">Appendix A: MQSeries API for UniData and UniVerse reason codes</a> ," for a list of AMI Reason Codes and their descriptions. [OUT]

amReceiveRequest parameters

---

<b>Parameter</b>	<b>Description</b>
<i>optional parameter</i>	U2AMI_LEAVEMSG If you specify U2AMI_LEAVEMSG for this parameter, and Accept Truncated Messages is not set in the policy receive attributes, UniVerse returns the message length in the <i>dataLen</i> parameter, but the message itself remains on the queue.
	U2AMI_DISCARDMSG If you specify U2_DISCARDMSG for this parameter and Accept Truncated Messages is set in the policy receive attributes, UniVerse discards the message at the WebSphere MQ level with an AMRC_MSG_TRUNCATED warning. This behavior is preferable to discarding the message at the UniVerse level.
	U2AMI_RESIZEBUFFER If you specify U2AMI_RESIZEBUFFER for this parameter, UniVerse handles the details of the buffer size used to retrieve the message. If you do not specify this parameter, you must specify the buffer size. See Usage Notes for more information about this option.

---

#### **amReceiveRequest parameters (Continued)**

The following table describes the status of each return code.

---

<b>Return Code</b>	<b>Status</b>
0 - AMCC_SUCCESS	Function completed successfully.
1 - AMCC_WARNING	A warning was returned from AMI. The reasonCode output parameter contains an AMI reason code with further details about the warning.
2 - AMCC_FAILED	An error was returned from AMI. The reasonCode output parameter contains an AMI reason code with further details about the error.
other	A non-AMI error occurred.

---

#### **Return Code Status**

## Usage notes

*maxMsgLen* – You can use this parameter to define the maximum length message that **amReceiveRequest** retrieves from the queue. If the value of *maxMsgLen* is less than the length of the message to retrieve, behavior depends on whether you set *Accept Truncated Message* in the policy receive attributes to true. If *Accept Truncated Message* is set to true, the data is truncated and there is an *AMRC\_MSG\_TRUNCATED* warning in the *reasonCode* parameter. If *Accept Truncated Message* is false (the default), **amReceiveRequest** fails with return status *AMCC\_FAILED* (2), and *reasonCode* *AMRC\_RECEIVE\_BUFF\_LEN\_ERR*.



**Note:** If *AMRC\_RECEIVE\_BUFF\_LEN\_ERR* is returned as the *reasonCode*, the message length value is contained in *dataLen* parameter, even though the call failed with return value *MQCC\_FAILED*.

If you do not specify the *U2AMI\_RESIZEBUFFER* optional parameter and the buffer size you specify with the *maxMsgLen* parameter is too small, the function fails with the *AMRC\_RECEIVE\_BUFF\_LEN\_ERR*. If this happens, UniVerse returns the necessary buffer size in the *dataLen* parameter so you can reissue the request with the correct size.

If you specify the *U2AMI\_RESIZEBUFFER* parameter, UniVerse uses a default buffer size of 8K. If this buffer size is too small, UniVerse automatically reissues the request with the correct buffer size. While convenient, this behavior can result in performance degradation for the following reasons:

- If the default buffer size is larger than necessary for the received message, UniVerse incurs an unnecessary overhead.
- If the default buffer size is too small for the received message, UniVerse must issue two requests to the queue before successfully retrieving the message.

For performance reasons, we recommend you set the *maxMsgLen* parameter to the expected size of the message whenever possible.

## Sending a message

The **amSendMsg()** function sends a datagram (send and forget) message.

## Syntax

```
ret = amSendMsg(hSession, senderName, policyName, data,  
sndMsgName, reasonCode)
```

## Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>hSession</i>	The session handle returned by <b>amInitialize</b> . [IN]
<i>senderName</i>	The name of a sender service. If you specify "" (null), amSendMsg uses the system default receiver name. [IN]
<i>policyName</i>	The name of a policy. If you specify "" (null), amSendMsg uses the system default policy name. [IN]
<i>data</i>	The message data to be sent. [IN]
<i>sndMsgName</i>	The name of a message object for the message being sent. If you specify "" (null), amSendMsg uses the system default policy name. [IN]
<i>reasonCode</i>	Holds an AMI Reason Code when the function returns a status indicating an AMI warning or an AMI error occurred. You can use the AMI Reason Code to obtain more information on the cause of the warning or error. See Appendix A, " <a href="#">Appendix A: MQSeries API for UniData and UniVerse reason codes</a> ," for a list of AMI Reason Codes and their descriptions. [OUT]

### amSendMsg Parameters

The following table describes the status of each return code.

Return Code	Status
0 - AMCC_SUCCESS	Function completed successfully.
1 - AMCC_WARNING	A warning was returned from AMI. The <i>reasonCode</i> output parameter contains an AMI reason code with further details about the warning.
2 - AMCC_FAILED	An error was returned from AMI. The <i>reasonCode</i> output parameter contains an AMI reason code with further details about the error.
Other	A non-AMI error occurred.

#### Return Code Status

You can also use the BASIC STATUS() to obtain the return status from the function.

## Sending a request

The **amSendRequest()** function sends a request message.

### Syntax

```
ret = amSendRequest(hSession, senderName, policyName, data,  
sndMsgName, reasonCode)
```

## Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>hSession</i>	The session handle returned by amInitialize. [IN]
<i>senderName</i>	The name of a sender service. If you specify "" (null), amSendRequest uses the system default sender name. [IN]
<i>policyName</i>	The name of a policy. If you specify "" (null), amSendRequest uses the system default policy name. [IN]
<i>responseName</i>	The name of the receiver service to which the response to this send request should be sent. Specify "" (null) if no response is required. [IN]
<i>dataLen</i>	The length of the message data, in bytes. [IN]
<i>data</i>	The message data to be sent. [IN]
<i>sndMsgName</i>	The name of a message object for the message being sent. If you specify "" (null), amSendRequest uses the system default message name (constant AMSD SND_MSG). [IN]
<i>reasonCode</i>	Holds an AMI Reason Code when the function returns a status indicating an AMI warning or an AMI error occurred. You can use the AMI Reason Code to obtain more information about the cause of the warning or error. See Appendix A, " <a href="#">Appendix A: MQSeries API for UniData and UniVerse reason codes</a> ," for a list of AMI Reason Codes and their descriptions. [OUT]

### amSendRequest Parameters

The following table describes the status of each return code.

Return Code	Status
0 - AMCC_SUCCESS	Function completed successfully.
1 - AMCC_WARNING	A warning was returned from AMI. The reasonCode output parameter contains an AMI reason code with further details about the warning.
2 - AMCC_FAILED	An error was returned from AMI. The reasonCode output parameter contains an AMI reason code with further details about the error.
Other	A non-AMI error occurred.

#### Return Code Status

You can also use the BASIC STATUS() function to obtain the return status from the function.

## Sending a response

The **amSendResponse()** function sends a request message.

### Syntax

```
ret = amSendResponse(hSession, senderName, policyName,  
rcvMsgName, data, sndMsgName, reasonCode)
```

## Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>hSession</i>	The session handle returned by <b>amInitialize</b> . [IN]
<i>senderName</i>	The name of the sender service. You must set this parameter to the senderName specified for the <b>amReceiveRequest</b> function. [IN]
<i>policyName</i>	The name of a policy. If you specify “” (null), amSendResponse uses the system default policy name. [IN]
<i>rcvMsgName</i>	The name of the received message to which this message is a response. You must set this parameter to the <b>rcvMsgName</b> specified for the <b>amReceiveRequest</b> function. [IN]
<i>dataLen</i>	The length of the message data, in bytes. [IN]
<i>data</i>	The message data to be sent. [IN]
<i>sndMsgName</i>	The name of a message object for the message being sent. If you specify “” (null), amSendResponse uses the system default message name (constant AMSD_SND_MSG). [IN]
<i>reasonCode</i>	Holds an AMI Reason Code when the function returns a status indicating an AMI warning or an AMI error occurred. You can use the AMI Reason Code can be used to obtain more information about the cause of the warning error. See Appendix A, “ <a href="#">Appendix A: MQSeries API for UniData and UniVerse reason codes</a> ,” for a list of AMI Reason Codes and their descriptions.[OUT]

### amSendResponse Parameters

The following table describes the status of each return code.

Return Code	Status
0 - AMCC_SUCCESS	Function completed successfully.
1 - AMCC_WARNING	A warning was returned from AMI. The <i>reasonCode</i> output parameter contains an AMI reason code with further details about the warning.
2 - AMCC_FAILED	An error was returned from AMI. the <i>reasonCode</i> output parameter contains an AMI reason code with further details about the error.
Other	A non-AMI error occurred.

#### Return Code Status

You can also use the BASIC STATUS() function to obtain the return status from the function.

## Terminating a session

The **amTerminate()** function closes a session.

### Syntax

```
ret = amTerminate(hSession, policyName, reasonCode)
```

## Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>hSession</i>	The session handle returned by <b>amInitialize</b> . [IN/OUT]
<i>policyName</i>	The name of a policy. If you specify "" (null), amTerminate uses the system default policy name. [IN]
<i>reasonCode</i>	Holds an AMI Reason Code when the function returns a status indicating an AMI warning or an AMI error occurred. You can use the AMI Reason Code to obtain more information about the cause of the warning or error. See Appendix A, " <a href="#">Appendix A: MQSeries API for UniData and UniVerse reason codes</a> ," for a list of AMI Reason Codes and their descriptions. [OUT]

### amTerminate Parameters

The following table describes the status of each return code.

Return Code	Status
0 - AMCC_SUCCESS	Function completed successfully.
1 - AMCC_WARNING	A warning was returned from AMI. The <i>reasonCode</i> output parameter contains an AMI reason code with further details about the warning.
2 - AMCC_FAILED	An error was returned from AMI. The <i>reasonCode</i> output parameter contains an AMI reason code with further details about the error.
Other	A non-AMI error occurred.

### Return Code Status

You can also use the BASIC STATUS() function to obtain the return status from the function.

# CallAMI programming examples

## *Sending a message*

The following example demonstrates sending a message to a message queue. The message is sent using the policy and service rules set forth in "AMT.SAMPLE.POLICY" and "AMT.SAMPLE.SERVICE".

```
$INCLUDE INCLUDE U2AMI.H

ret = amInitialize(hSess, "SAMPLE.APP", "AMT.SAMPLE.POLICY",
reasonCode)

msg = "Sample Message sent at " : TIMEDATE()
ret = amSendMsg(hSess, "AMT.SAMPLE.SERVICE", "AMT.SAMPLE.POLICY",
msg, "", reasonCode)

ret = amTerminate(hSess, "AMT.SAMPLE.POLICY", reasonCode)
```

## *Retrieving a message*

The following example demonstrates retrieving a message from a message queue. The message, of maximum length 1024, is retrieved into the variable *msg* using the policy and service rules set forth in "AMT.SAMPLE.POLICY" and "AMT.SAMPLE.SERVICE". The message selection criteria is "", meaning the next available message will be retrieved from the queue.

```
$INCLUDE INCLUDE U2AMI.H

retCode = amInitialize(hSess, "SAMPLE.APP", "AMT.SAMPLE.POLICY",
reasonCode)

retCode = amReceiveMsg(hSess, "AMT.SAMPLE.SERVICE",
"AMT.SAMPLE.POLICY", "", 1024, dataLen, msg, "", reasonCode)

retCode = amTerminate(hSess, "AMT.SAMPLE.POLICY", reasonCode)
```

## *Request/response messaging*

The following client and server samples demonstrate use of the Request/Response messaging style. The client sends a message to the server, and the server then echoes that message back to the client.

Note that the repository definitions named in the program can be created via the amtsamp.tst script that ships with the WebSphere MQ AMI SupportPac. See the chapter “Installation and Sample Programs” in the *WebSphere MQ Application Messaging Interface* manual for more details.

### *Sample request/response client*

```
$INCLUDE INCLUDE U2AMI.H

APP_NAME          = "SAMPLE.CLIENT"
SENDER_SERVICE    = "AMT.SAMPLE.REQUEST.SERVICE"
RECEIVER_SERVICE  = "AMT.SAMPLE.RESPONSE.RECEIVER"
SEND_MESSAGE_NAME = "AMT.SAMPLE.SEND.MESSAGE"
RECEIVE_MESSAGE_NAME = "AMT.SAMPLE.RECEIVE.MESSAGE"
POLICY           = "AMT.SAMPLE.POLICY"

hSession      = 0
retCode       = 0
reasonCode   = 0

reqMessage    = ""
respMessage   = ""

respMsgLen   = 0
MAX_MSG_LEN  = 1024

*****
* Initialize the Session
*
*****


retCode = amInitialize(hSession, APP_NAME, POLICY, reasonCode)
IF retCode # AMCC_OK THEN
    function = "amInitialize"
    GOTO ErrorExit
END

*****


* Send the Request Message
*
*
* amSendRequest INPUTS
*
*      hSession          - Session handle.
*      SENDER_SERVICE    - Service definition used to send
*                           the request message.
*      POLICY           - Policy definition used to send
*                           the request message.
*      RECEIVER_SERVICE  - Service definition describing where
*                           the response message should be sent
*      reqMessage        - Request message.
*      SEND_MESSAGE_NAME - Name we're giving to the the
```

```

*
* underlying
* message object used to send the
* request message.
*
* amSendRequest OUTPUTS
*
*      retCode          - Holds return code indicating status of
*                           function:
*                           AMCC_OK      (0) - Success
*                           AMCC_WARNING (1) - WebSphere MQ AMI
*                           Warning
*                           AMCC_FAILED  (2) - WebSphere MQ AMI
*                           Error
*                           Other        - Other error
*      reasonCode       - Holds additional information if
*                           WebSphere MQ AMI error or warning
*                           occurs.
*
*****  

PRINT "Please enter a message to send to the server. "
PRINT "The server will echo the message back: "
INPUT reqMessage  

PRINT "Sending Request Message: "
PRINT "    << " : reqMessage
PRINT ""  

retCode = amSendRequest(hSession, SENDER_SERVICE, POLICY,
RECEIVER_SERVICE, reqMessage, SEND_MESSAGE_NAME, reasonCode)
IF retCode # AMCC_OK THEN
    function = "amSendRequest"
    GOTO ErrorExit
END  

*****  

*   Receive the Response Message
*  

*  

* amReceiveMsg INPUTS
*
*      hSession          - Session handle.
*      RECEIVER_SERVICE - Service definition describing where
*                           the response message is to be
*                           retrieved from.
*      POLICY           - Policy definition used to retrieve
*                           the response message.
*      SEND_MESSAGE_NAME - Name we gave to the the underlying
*                           message object used to send the
*                           request
*                           message. This message object is now

```

```

*
*                                used as the message selection criteria
*                                to ensure that we retrieve only the
*                                message
*                                that is sent in response to our
*                                request.
*      MAX_MSG_LEN           - Max allowed length for retrieved
*                                message.
*      RECEIVE_MESSAGE_NAME - Name we're giving to the the
*                                underlying
*                                message object used to receive the
*                                response message.
*
*      amReceiveMsg OUTPUTS
*
*      retCode                - Holds return code indicating status of
*                                function:
*                                AMCC_OK      (0) - Success
*                                AMCC_WARNING (1) - WebSphere MQ AMI
*                                Warning
*                                AMCC_FAILED  (2) - WebSphere MQ AMI
*                                Error
*                                Other        - Other error
*      respMsgLen            - Length of received message.
*      respMessage            - Response message.
*      reasonCode             - Holds additional information if
*                                WebSphere MQ AMI error or warning
*                                occurs.
*
*****  

retCode = amReceiveMsg(hSession, RECEIVER_SERVICE, POLICY,
SEND_MESSAGE_NAME, MAX_MSG_LEN, respMsgLen, respMessage,
RECEIVE_MESSAGE_NAME, reasonCode)
IF retCode # AMCC_OK THEN
    function = "amReceiveMsg"
    GOTO ErrorExit
END

PRINT "Received Response Message: "
PRINT "    >> " : respMessage
PRINT ""

*****  

*   Terminate the Session
*
*****  

retCode = amTerminate(hSession, POLICY, reasonCode)
IF retCode # AMCC_OK THEN
    function = "amTerminate"
    GOTO ErrorExit
END

```

```
STOP
```

```
ErrorExit:  
    PRINT "Error/Warning encountered in function " : function  
    PRINT "    retCode      = " : retCode  
    IF retCode = AMCC_WARNING OR retCode = AMCC_FAILED THEN  
        PRINT "    reasonCode = " : reasonCode  
    END
```

```
END
```

### *Sample request/response server*

```
$INCLUDE INCLUDE U2AMI.H

APP_NAME          = "SAMPLE.SERVER"
SENDER_SERVICE    = "AMT.SAMPLE.RESPONDER"
RECEIVER_SERVICE  = "AMT.SAMPLE.REQUEST.SERVICE"
SEND_MESSAGE_NAME = "AMT.SAMPLE.SEND.MESSAGE"
RECEIVE_MESSAGE_NAME = "AMT.SAMPLE.RECEIVE.MESSAGE"
POLICY           = "AMT.SAMPLE.POLICY"

hSession      = 0
retCode       = 0
reasonCode   = 0

MAX_REQUESTS  = 5
msgCount     = 0

reqMessage    = ""
respMessage   = ""

reqMsgLen    = 0
MAX_MSG_LEN  = 1024

*****
* Initialize the Session
*
*****
retCode = amInitialize(hSession, APP_NAME, POLICY, reasonCode)
IF retCode # AMCC_OK THEN
    function = "amInitialize"
    GOTO ErrorExit
END

*****
* Service up to MAX_REQUESTS
*
*****
FOR msgCount = 1 TO MAX_REQUESTS

*****
* Receive the Request Message
*
*
* amReceiveRequest INPUTS
*
* hSession           - Session handle.
```

```

* RECEIVER_SERVICE      - Service definition describing where
*                         the request message is to be
*                         retrieved from.
* POLICY                - Policy definition used to retrieve
*                         the request message.
* MAX_MSG_LEN           - Max allowed length for retrieved
*                         message.
* RECEIVE_MESSAGE_NAME  - Name we're giving to the the
*                         underlying
*                         message object used to receive the
*                         request message.
* SENDER_SERVICE         - Name we're giving to the special
*                         internally-defined "response sender"
*                         service. Note that because of the
*                         special status of
*                         response-senders, the
*                         name we give the service must not
*                         be defined in the AMI repository.
*
* amReceiveRequest OUTPUTS
*
* retCode                 - Holds return code indicating status of
*                           function:
*                           AMCC_OK      (0) - Success
*                           AMCC_WARNING (1) - WebSphere MQ AMI
*                                         Warning
*                           AMCC_FAILED  (2) - WebSphere MQ AMI
*                                         Error
*                           Other        - Other error
* reqMsgLen               - Length of received message.
* reqMessage              - Request message.
* reasonCode              - Holds additional information if
*                           WebSphere MQ AMI error or warning occurs.
*
*****  

retCode = amReceiveRequest(hSession, RECEIVER_SERVICE, POLICY,
MAX_MSG_LEN, reqMsgLen, reqMessage, RECEIVE_MESSAGE_NAME,
SENDER_SERVICE, reasonCode)
    IF retCode # AMCC_OK THEN
        function = "amReceiveRequest"
        GOTO ErrorExit
    END

    PRINT "Received Request Message: "
    PRINT "    >> " : reqMessage
    PRINT ""

*****  

*   Send the Response Message

```

```

*
*
* amSendResponse INPUTS
*
*      hSession          - Session handle.
*      SENDER_SERVICE    - The response-sender we named
*                           previously
*                           in the call to amReceiveRequest.
*      POLICY            - Policy definition used to send
*                           the response message.
*      RECEIVE_MESSAGE_NAME - Name we gave to the the underlying
*                           message object used to receive the
*                           request message. This message object
*                           is
*                           now used to correlate the response
*                           message with the initial request
*                           message.
*      respMessage        - Response message.
*      SEND_MESSAGE_NAME - Name we're giving to the the
*                           underlying
*                           message object used to send the
*                           response message.
*
* amSendResponse OUTPUTS
*
*      retCode           - Holds return code indicating status of
*                           function:
*                           AMCC_OK          (0) - Success
*                           AMCC_WARNING     (1) - WebSphere MQ AMI
*                           Warning
*                           AMCC_FAILED      (2) - WebSphere MQ AMI
*                           Error
*                           Other             - Other error
*      reasonCode        - Holds additional information if
*                           WebSphere MQ AMI error or warning
*                           occurs.
*
*****
respMessage = reqMessage

retCode = amSendResponse(hSession, SENDER_SERVICE, POLICY,
RECEIVE_MESSAGE_NAME, respMessage, SEND_MESSAGE_NAME, reasonCode)
IF retCode # AMCC_OK THEN
    function = "amSendResponse"
    GOTO ErrorExit
END

PRINT "Sent Response Message: "
PRINT "    << " : respMessage
PRINT ""

```

NEXT

```
*****
* Terminate the Session
*
*****  
  
retCode = amTerminate(hSession, POLICY, reasonCode)  
IF retCode # AMCC_OK THEN  
    function = "amTerminate"  
    GOTO ErrorExit  
END  
  
  
STOP  
ErrorExit:  
    PRINT "Error/Warning encountered in function " : function  
    PRINT "    retCode      = " : retCode  
    IF retCode = AMCC_WARNING OR retCode = AMCC_FAILED THEN  
        PRINT "    reasonCode = " : reasonCode  
    END  
END
```

## **Additional reading**

Interested readers are encouraged to refer to the following publications, which are available for download from  
<http://www.ibm.com/shop/publications/order>.

WebSphere MQ Primer

WebSphere MQ Application Messaging Interface

WebSphere MQ Application Programming Guide

WebSphere MQ Clients

# Chapter 4: Creating XML documents

XML for U2 UniVerse . . . . .	4-2
Document type definitions . . . . .	4-3
XML schema . . . . .	4-3
The Document Object Model (DOM) . . . . .	4-3
Well-formed and valid XML documents . . . . .	4-4
Creating an XML document from RetrieVe . . . . .	4-5
Create the &XML& file . . . . .	4-5
Mapping modes . . . . .	4-5
XML configuration file . . . . .	4-13
xmlconfig parameters . . . . .	4-15
The Mapping File . . . . .	4-24
Distinguishing elements . . . . .	4-27
Root element attributes . . . . .	4-27
Association elements . . . . .	4-34
Mapping file example . . . . .	4-35
How data is mapped . . . . .	4-39
Mapping example . . . . .	4-41
TCL commands for XML . . . . .	4-43
Session-level TCL commands . . . . .	4-43
XMLSETOPTIONS . . . . .	4-43
XMLGETOPTIONS . . . . .	4-45
XMLGETOPTIONVALUE . . . . .	4-46
Existing TCL command affected by XMLSETOPTIONS command or XMLSetOptions() API . . . . .	4-47
Creating an XML document using RetrieVe . . . . .	4-49
Examples . . . . .	4-50
Creating an XML document with UniVerse SQL . . . . .	4-59

Processing rules for UniVerse SQL SELECT statements . . . . .	4-60
XML limitations in UniVerse SQL. . . . .	4-62
Examples . . . . .	4-62
Creating an XML document through UniVerse BASIC . . . . .	4-71
Using the XMLExecute() function. . . . .	4-72
XMLSetOptions . . . . .	4-75
XMLGetOptions. . . . .	4-77
XMLGetOptionValue . . . . .	4-78
Existing APIs affected by XML options . . . . .	4-93
UniVerse BASIC example . . . . .	4-121

## XML for U2 UniVerse

The Extensible Markup Language (XML) is a markup language used to define, validate, and share document formats. It enables you to tailor document formats to specifications unique to your application by defining your own elements, tags, and attributes.



*Note: XML describes how a document is structured, not how a document is displayed.*

XML was developed by the World Wide Web Consortium (W3C), who describe XML as:

The Extensible Markup Language (XML) is the universal format for structured documents and data on the Web.

XML documents are text documents, intended to be processed by an application, such as a web browser.

An XML document consists of a set of tags that describe the structure of data. Unlike HTML, you can write your own tags. You can use XML to describe any type of data so that it is cross-platform and machine independent.

For detailed information about XML, see the W3C Web site at <http://www.w3.org/TR/REC-xml>.

UniVerse enables you to receive and create XML documents, and process them through UniVerse Basic, UniVerse SQL, or RetrievVe. In order to work with the XML documents in UniVerse, you will need to know some key terms:

- Document Type Definitions
- XML Schema
- Document Object Model
- Well-Formed and Valid Documents

## Document type definitions

You must define the rules of the structure of your XML document. These rules may be part of the XML document, and are called the Document Type Definition, or DTD. The DTD provides a list of elements, tags, attributes, and entities contained in the document, and describes their relationship to each other.

A DTD can be external or internal.

- External DTD — An external DTD is a separate document from the XML document, residing outside of your XML document. External DTDs can be applied to many different XML documents.
- Internal DTD — An internal DTD resides in the XML document as part of the header of the document, and applies only to that XML document.

You can combine external DTDs with internal DTDs in an XML document, and you can create DTDs in an XML document.

## XML schema

The structure of the XML document can also be defined using XML Schema, which is an XML-based alternative to the DTD. An XML Schema defines a class of XML documents, including the structure, content and meaning of the XML document. XML Schema is useful because it is written in XML and is extensible to future additions. You can create schema with XML, and you can use schema to validate XML. The XML Schema language can also be referred to as XML Schema Definition (XSD).

## The Document Object Model (DOM)

The Document Object Model (DOM) is a platform- and language-independent interface that enables programs and scripts to dynamically access and update the content, structure, and style of documents. A DOM is a formal way to describe an XML document to another application or programming language. You can describe the XML document as a tree, with nodes representing elements, attributes, entities, and text.

## **Well-formed and valid XML documents**

An XML document is either well-formed or valid:

- Well-formed XML documents must follow XML rules. All XML documents must be well-formed.
- Valid XML documents are both well-formed, and follow the rules of a specific DTD or schema. Not all XML documents must be valid.

For optimum exchange of data, you should try to ensure that your XML documents are valid.

## Creating an XML document from RetrieVe

You can create an XML document from UniVerse files through RetrieVe. To create an XML document through RetrieVe, complete the following steps:

1. If you are the originator of the DTD or XML Schema, use RetrieVe to create the DTD or XML Schema.  
If you are not the originator of the DTD or XML Schema, analyze the DTD or XML Schema associated with the application to which you are sending the XML file. Determine which of your dictionary attributes correspond to the DTD or XML Schema elements. You can also refer to [Mapping to an external schema](#) at the end of this section.
2. Create an XML mapping file, if necessary. The mapping file will enable users to create many different forms of XML.
3. List the appropriate fields using the LIST command.
4. Add the TOXML command to generate the XML document.

## Create the &XML& file

UniVerse stores XML mapping files in the &XML& directory file. This directory is automatically created with new accounts. If you have an older account, you can create this file for PICK flavor accounts using the following command:

```
CREATE.FILE &XML& 3,1,18 1,1,19
```

To create the &XML& account in other flavor accounts, use the following command:

```
CREATE.FILE &XML& 19
```

## Mapping modes

UniVerse supports three modes for mapping data to XML files. These modes are:

- Attribute-centric
- Element-centric
- Mixed

## **Attribute-centric mode**

In the attribute-centric mode, which is the default mode, each record displayed in the query statement becomes an XML element. The following rules apply to the record fields:

- Each singlevalued field becomes an attribute within the element.
- Each multivalued or multi-subvalued field becomes a sub-element of the record element. The name of the sub-element, if there is no association, is *fieldname*\_MV or \_MS.
- Within a sub-element, each multivalued field becomes an attribute of the sub-element.
  - Associated multi-subvalued fields become another nested sub-element of the sub-element. The name of this nested sub-element is *association\_name*-MS.
  - If there are no associated multi-subvalued fields, the sub-element name is *field\_name*-MV/MS.

This is the default mapping scheme. You can change the default by defining maps in the &XML& directory.

The following example shows data created in attribute mode:

```
>LIST STUDENT.F LNAME CGA TOXML SAMPLE 1

<?xml version="1.0" encoding="UTF-8"?>
<ROOT>
<STUDENT.F _ID = "424325656" LNAME = "Martin">
    <CGA_MV SEMESTER = "SP94" COURSE_HOURS = "3" TEACHER = "Masters" COURSE_HOURS = "3" TEACHER = "Fisher">
        <CGA_MS COURSE_GRD = "C" COURSE_NBR = "PY100" COURSE_NAME = "Introduction to Psychology"/>
        <CGA_MS COURSE_GRD = "C" COURSE_NBR = "PE100" COURSE_NAME = "Golf - I"/>
    </CGA_MV>
</STUDENT.F>
</ROOT>
>
```

## **Element-centric mode**

In the element-centric mode, as in the attribute-centric mode, each record becomes an XML element. The following rules apply:

- Each singlevalued field becomes a simple sub-element of the element, containing no nested sub-elements. The value of the field becomes the value of the sub-element.
- Each association whose multivalued and multi-subvalued fields are included in the query statement form a complex sub-element. In the sub-element, each multivalued field belonging to the association becomes a sub-element that may contain multi-subvalued sub-elements. There are two ways to display empty values in multivalued fields belonging to an association. For detailed information, see [Displaying empty values in multivalued fields in an association](#).
- By default, UniVerse converts text marks to an empty space.

Specify that you want to use element-centric mapping by using the ELEMENTS keyword in the RetriVe statement. You can also define treated-as = "ELEMENT" in the U2XMLOUT.map file, so that all XML will be created in element mode.

The following example shows data created in element mode:

```
>LIST STUDENT.F LNAME CGA TOXML ELEMENTS SAMPLE 1

<?xml version="1.0" encoding="UTF-8"?>
<ROOT>
<STUDENT.F>
  <_ID>424325656</_ID>
  <LNAME>Martin</LNAME>
  <CGA_MV>
    <SEMESTER>SP94</SEMESTER>
    <COURSE_HOURS>3</COURSE_HOURS>
    <TEACHER>Masters</TEACHER>
    <CGA_MS>
      <COURSE_GRD>C</COURSE_GRD>
      <COURSE_NBR>PY100</COURSE_NBR>
      <COURSE_NAME>Introduction to Psychology</COURSE_NAME>
    </CGA_MS>
    <COURSE_HOURS>3</COURSE_HOURS>
    <TEACHER>Fisher</TEACHER>
    <CGA_MS>
      <COURSE_GRD>C</COURSE_GRD>
      <COURSE_NBR>PE100</COURSE_NBR>
      <COURSE_NAME>Golf - I</COURSE_NAME>
    </CGA_MS>
  </CGA_MV>
</STUDENT.F>
</ROOT>
>
```

## ***Displaying empty values in multivalued fields in an association***

UniVerse displays empty values in multivalued fields belonging to an association depending on the setting of the Matchelement field in the U2XMLOUT.map file.

### ***Emptyattribute***

This attribute determines how to display the empty attributes for multivalued fields belonging to an association in the generated XML document and in the associated DTD or XML Schema. This option can be specified in the U2XMLOUT.map file, or in an individual mapping file.

- 0 - Hides the empty attributes in the multivalued fields.
- 1 - Shows the empty attributes in the multivalued fields.

### ***Matchelement***

This element specifies how UniVerse displays empty values in multivalued fields belonging to an association in the XML output.

If Matchelement is set to 1 (the default), matching values or subvalues belonging to the same association display as empty elements for matching pairs.

Consider the following example:

```
LIST STUDENT.F LNAME FNAME COURSE_NBR COURSE_GRD COURSE_NAME
SEMESTER
01:04:52pm 06 Jan 2009 PAGE 1

STUDENT.... 424-32-5656
Last Name.. Martin
First Name. Sally
Crs #..... GD. Course Name.... Term
PY100 C Introduction to SP94
Psychology
PE100 Golf - I

STUDENT.... 521-81-4564
Last Name.. Smith
First Name. Harry
Crs #..... GD. Course Name.... Term
CS130 A Intro to FA93
Operating
Systems
CS100 Intro to
Computer
Science
PY100 B Introduction to
Psychology

CS131 B Intro to SP94
Operating
Systems
CS101 B Intro to
Computer
Science
PE220 Racquetball

...
>
```

Notice that three of the GRADE fields are empty, while their associated values for COURSE # and COURSE NAME are not.

When Matchelement is set to 1, the missing values for COURSE \_GRD, <COURSE\_GRD></COURSE\_GRD>, display as empty values in the XML document, as shown in the following example:

```
>LIST STUDENT.F LNAME CGA TOXML ELEMENTS

<?xml version="1.0" encoding="UTF-8"?>
<ROOT>
<STUDENT.F>
  <_ID>424325656</_ID>
  <LNAME>Martin</LNAME>
  <CGA_MV>
    <SEMESTER>SP94</SEMESTER>
    <COURSE_HOURS>3</COURSE_HOURS>
    <TEACHER>Masters</TEACHER>
    <CGA_MS>
      <COURSE_GRD>C</COURSE_GRD>
      <COURSE_NBR>PY100</COURSE_NBR>
      <COURSE_NAME>Introduction to Psychology</COURSE_NAME>
    </CGA_MS>
    <COURSE_HOURS>3</COURSE_HOURS>
    <TEACHER>Fisher</TEACHER>
    <CGA_MS>
      <COURSE_GRD/> ← empty value
      <COURSE_NBR>PE100</COURSE_NBR>
      <COURSE_NAME>Golf - I</COURSE_NAME>
    </CGA_MS>
  </CGA_MV>
</STUDENT.F>
<STUDENT.F>
  <_ID>521814564</_ID>
  <LNAME>Smith</LNAME>
  <CGA_MV>
    <SEMESTER>FA93</SEMESTER>
    <COURSE_HOURS>5</COURSE_HOURS>
    <TEACHER>James</TEACHER>
    <CGA_MS>
      <COURSE_GRD>A</COURSE_GRD>
      <COURSE_NBR>CS130</COURSE_NBR>
      <COURSE_NAME>Intro to Operating Systems</COURSE_NAME>
    </CGA_MS>
    <COURSE_HOURS>3</COURSE_HOURS>
    <TEACHER>Gibson</TEACHER>
    <CGA_MS>
      <COURSE_GRD/> ← empty value
      <COURSE_NBR>CS100</COURSE_NBR>
      <COURSE_NAME>Intro to Computer Science</COURSE_NAME>
    </CGA_MS>
    <COURSE_HOURS>3</COURSE_HOURS>
    <TEACHER>Masters</TEACHER>
    <CGA_MS>
      <COURSE_GRD>B</COURSE_GRD>
      <COURSE_NBR>PY100</COURSE_NBR>
```

```
<COURSE_NAME>Introduction to Psychology</COURSE_NAME>
</CGA_MS>
</CGA_MV>
<CGA_MV>
  <SEMESTER>SP94</SEMESTER>
  <COURSE_HOURS>5</COURSE_HOURS>
  <TEACHER>Aaron</TEACHER>
<CGA_MS>
  <COURSE_GRD>B</COURSE_GRD>
  <COURSE_NBR>CS131</COURSE_NBR>
  <COURSE_NAME>Intro to Operating Systems</COURSE_NAME>
</CGA_MS>
<COURSE_HOURS>4</COURSE_HOURS>
<TEACHER>Gibson</TEACHER>
<CGA_MS>
  <COURSE_GRD>B</COURSE_GRD>
  <COURSE_NBR>CS101</COURSE_NBR>
  <COURSE_NAME>Intro to Computer Science</COURSE_NAME>
</CGA_MS>
<COURSE_HOURS>3</COURSE_HOURS>
<TEACHER>Fisher</TEACHER>
<CGA_MS>
  <COURSE_GRD/> ← empty value
  <COURSE_NBR>PE220</COURSE_NBR>
  <COURSE_NAME>Racquetball</COURSE_NAME>
</CGA_MS>
</CGA_MV>
</STUDENT.F>
...

```

This is the default behavior.

When Matchelement is set to 0, the missing value for COURSE\_GRD, <COURSE\_GRD></COURSE\_GRD>, is ignored in the XML document, as shown in the following example:

```
LIST STUDENT.F LNAME CGA TOXML ELEMENTS
<?xml version="1.0" encoding="UTF-8"?>
<ROOT>
<STUDENT.F>
  <_ID>424325656</_ID>
  <LNAME>Martin</LNAME>
  <CGA_MV>
    <SEMESTER>SP94</SEMESTER>
    <COURSE_HOURS>3</COURSE_HOURS>
    <TEACHER>Masters</TEACHER>
    <CGA_MS>
      <COURSE_GRD>C</COURSE_GRD>
      <COURSE_NBR>PY100</COURSE_NBR>
      <COURSE_NAME>Introduction to Psychology</COURSE_NAME>
    </CGA_MS>
    <COURSE_HOURS>3</COURSE_HOURS>
    <TEACHER>Fisher</TEACHER>
    <CGA_MS>
      ← missing value
      <COURSE_NBR>PE100</COURSE_NBR>
      <COURSE_NAME>Golf - I</COURSE_NAME>
    </CGA_MS>
  </CGA_MV>
</STUDENT.F>
<STUDENT.F>
  <_ID>521814564</_ID>
  <LNAME>Smith</LNAME>
  <CGA_MV>
    <SEMESTER>FA93</SEMESTER>
    <COURSE_HOURS>5</COURSE_HOURS>
    <TEACHER>James</TEACHER>
    <CGA_MS>
      <COURSE_GRD>A</COURSE_GRD>
      <COURSE_NBR>CS130</COURSE_NBR>
      <COURSE_NAME>Intro to Operating Systems</COURSE_NAME>
    </CGA_MS>
    <COURSE_HOURS>3</COURSE_HOURS>
    <TEACHER>Gibson</TEACHER>
    <CGA_MS>
      ← missing value
      <COURSE_NBR>CS100</COURSE_NBR>
      <COURSE_NAME>Intro to Computer Science</COURSE_NAME>
    </CGA_MS>
    <COURSE_HOURS>3</COURSE_HOURS>
    <TEACHER>Masters</TEACHER>
    <CGA_MS>
      <COURSE_GRD>B</COURSE_GRD>
      <COURSE_NBR>PY100</COURSE_NBR>
      <COURSE_NAME>Introduction to Psychology</COURSE_NAME>
```

```

        </CGA_MS>
    </CGA_MV>
    <CGA_MV>
        <SEMESTER>SP94</SEMESTER>
        <COURSE_HOURS>5</COURSE_HOURS>
        <TEACHER>Aaron</TEACHER>
        <CGA_MS>
            <COURSE_GRD>B</COURSE_GRD>
            <COURSE_NBR>CS131</COURSE_NBR>
            <COURSE_NAME>Intro to Operating Systems</COURSE_NAME>
        </CGA_MS>
        <COURSE_HOURS>4</COURSE_HOURS>
        <TEACHER>Gibson</TEACHER>
        <CGA_MS>
            <COURSE_GRD>B</COURSE_GRD>
            <COURSE_NBR>CS101</COURSE_NBR>
            <COURSE_NAME>Intro to Computer Science</COURSE_NAME>
        </CGA_MS>
        <COURSE_HOURS>3</COURSE_HOURS>
        <TEACHER>Fisher</TEACHER>
        <CGA_MS> ← missing value
            <COURSE_NBR>PE220</COURSE_NBR>
            <COURSE_NAME>Racquetball</COURSE_NAME>
        </CGA_MS>
    </CGA_MV>
</STUDENT.F>

</MAIN>

```

### **Mixed mode**

In the mixed-mode, you create your own map file, where you specify which fields are treated as attribute-centric and which fields are treated as element-centric.

Field-level mapping overrides the mode you specify in the Retrieve statement.

### **XML configuration file**

The xmlconfig file allows you to specify the encoding used in XML documents and to set other XML options. There are two levels of XML configuration files:

- System level. The default location of the system-level xmlconfig file is the UniVerse home directory.

- Account level. The default location of the account-level xmlconfig file is under the account directory. Configuration parameters in the account-level xmlconfig file override the settings in the system-level xmlconfig file.

The xmlconfig file is a text file in which each line is a key/value pair, as shown in the following example:

```
encoding=utf-8
in-encoding=EUC-JP
out-encoding=uft-8
out-xml-declaration=true
out-pretty-format=true
out-newline=CR-LF
```

Keys and values are case-insensitive.

When you start a UniVerse session, UniVerse loads the xmlconfig files and validates the options. If it finds an error in an xmlconfig file, it reports one of the following error messages in errlog, if errlog exists in the UniVerse home directory:

- 38 Invalid XML config key 'key\_name' at line line\_number in xmlconfig file(xmlconfig\_filename)
- 39 Invalid XML config value 'value\_string' at line line\_number in xmlconfig file(xmlconfig\_filename)
- 40 Invalid XML config format 'name\_value\_string' at line line\_number in xmlconfig file(xmlconfig\_filename)

If an error is found in an xmlconfig file, no part of its content is loaded.

## xmlconfig parameters

The following XML options are available in the system-level and account-level xmlconfig files.

Parameter	Description	Default
encoding	Specifies a setting for the encoding to be used in general in XML documents, such as UTF-8.  Beginning at UniVerse 11.1.15, the encoding option defines the information in the XML declaration when TOXML is used with a LIST/SORT command. You can turn off this behavior by setting the encoding option to none.  If the setting is <b>default</b> , XML documents use the operating system's default encoding.	default
	Valid encoding settings can be found at <a href="http://www.iana.org/assignments/character-sets">http://www.iana.org/assignments/character-sets</a>	
in-encoding	Specifies a setting for the encoding to be used for strings that are imported to XML libraries. If NULL, strings imported to XML libraries use the same encoding as specified in the general encoding parameter (see above).	default
out-encoding	Specifies a setting for the encoding to be used for strings exported from XML libraries. If NULL, strings exported from XML libraries use the same encoding as specified in the general encoding parameter.	default
xdomopen-encoding	Specifies an encoding to use when the XML document does not have one declared in the declaration element.	Not set ( UTF-8 ) assumed.

### xmlconfig Parameters

---

Parameter	Description	Default
version	Specifies the XML version number. Currently, 1.0 is the only version supported.	1.0
standalone	A flag that specifies whether the XML document is dependent on another XML document.  yes – The XML document is standalone no – The XML document is dependent on another XML document	yes
out-newline	Specifies the newline character to be used in XML documents.  NULL – Uses the operating system default (CR-LF for Windows or CR for UNIX) CR – Carriage return character (xD). CR-LF – Carriage return and linefeed characters (xD xA). LF – Linefeed character (xA).	NULL
out-xml-declaration	Specifies whether the XML declaration is to appear in output.  true – The XML declaration appears in output. false – The XML declaration does not appear in output.	true

---

#### **xmlconfig Parameters (Continued)**

Parameter	Description	Default
out-format-pretty-print	<p>Specifies whether to add white space to produce an indented, human-readable XML document. The exact nature of the transformations is not specified by this parameter.</p> <p>true – Pretty-prints XML documents. Setting this state also sets the out-format-canonical parameter to false.</p> <p>false – Does not pretty-print XML documents. Setting this state also sets the out-format-canonical parameter to true.</p>	true
out-normalize-characters	<p>Specifies whether to perform W3C text normalization of the characters in the document at the time they are written in output. Only those characters that are written are subject to change in the normalization process. The DOM document itself remains unchanged.</p> <p>true – Performs W3C text normalization.</p> <p>false – Does not perform text normalization.</p>	true

---

#### **xmlconfig Parameters (Continued)**

Parameter	Description	Default
out-split-cdata-sections	<p>Specifies whether to split character data (CDATA) sections of XML documents containing the CDATA termination marker <code>]]&gt;</code> or characters that cannot be represented in the output encoding.</p> <p>true – Splits CDATA sections containing the termination marker <code>]]&gt;</code> or characters that cannot be represented in the output encoding, and output the characters as their Unicode numeric character references. If a CDATA section is split, a warning is issued.</p> <p>false – Does not split CDATA sections. Issues an error if a CDTA section contains an unrepresentable character.</p>	true
out-validation	<p>Specifies whether to validate the XML document against the abstract schema while the document is being serialized.</p> <p>true – Validates the XML document while it is being serialized. If validation errors are found during the serialization process, the error handler is notified. Setting this state also sets the use-abstract-schema parameter to true.</p> <p>false – Does not validate the XML document while it is being serialized.</p>	false
out-expand-entity-references	<p>Specifies whether to expand EntityReference nodes while an XML document is being serialized.</p> <p>true – Expands EntityReference nodes in the XML document while it is being serialized.</p> <p>false – Does not expand EntityReference nodes in the XML document while it is being serialized.</p>	false

#### **xmlconfig Parameters (Continued)**

Parameter	Description	Default
out-whitespace-in-element-content	<p>Specifies whether to output all white space in an XML document.</p> <p>true – Outputs all white space.</p> <p>false – Outputs only the white space that is not within element content.</p>	true
out-discard-default-content	<p>Specifies whether to suppress output of default content in the Attr nodes of an XML document.</p> <p>true – Suppresses output of default content in Attr nodes.</p> <p>false – Outputs all attributes and all content.</p>	true
out-format-canonical	<p>Specifies whether the formatting process writes the XML document according to the rules specified in the Canonical XML specification, rather than pretty-prints the document.</p> <p>true – Prints XML documents in canonical format. Setting this state also sets the out-format-pretty-print parameter to false.</p> <p>false – Pretty-prints XML documents. Setting this state also sets the out-format-pretty-print parameter to true.</p>	false

#### **xmlconfig Parameters (Continued)**

Parameter	Description	Default
out-write-BOM	<p>A nonstandard extension was added in Xerces-C++ 2.2 (or XML4C 5.1) to enable writing the byte order mark (BOM) in the XML stream.</p> <p>Specifies whether to enable writing the byte order mark in the XML stream under certain conditions.</p> <p>true – Enables writing the byte order mark if a DOMDocumentNode is rendered for serialization and if the output encoding is one of the following:</p> <ul style="list-style-type: none"> <li>UTF-16</li> <li>UTF-16LE</li> <li>UTF-16BE</li> <li>UCS-4</li> <li>UCS-4LE</li> <li>UCS-4BE</li> </ul> <p>false – Disables writing the byte order mark.</p>	false
matchelement	<p>Specifies how UniVerse displays empty values in multivalued fields belonging to an association in XML output.</p> <p>0 – For matching values or subvalues belonging to the same association, the empty value is ignored in the generated XML document.</p> <p>1 – For matching values or subvalues belonging to the same association, the second value is displayed as an empty element in the generated XML document.</p>	1

---

#### xmlconfig Parameters (Continued)

---

---

Parameter	Description	Default
elementmode	<p>Specifies whether to create XML documents in element mode.</p> <p>0 – Create XML documents in attribute mode. This mode does not produce an extra level of element tags. The display name of each field is used as an element tag. The content of the field is shown as attribute/value pairs within the field's element tag. For example:</p> <pre>:LIST MYSTUDENT '1111' TOXML &lt;?xml version="1.0"?&gt; &lt;ROOT&gt; &lt;MYSTUDENT _ID = "1111"/&gt; &lt;/ROOT&gt;</pre> <p>1 – Create XML documents in element mode. This mode produces an extra level of element tags. All fields referenced in the query are represented as XML elements. The display name of the field is used for the field's element tags. The display name of each field is used as a nested element tag. The content of the field is shown within the element tags. For example:</p> <pre>:LIST MYSTUDENT '1111' TOXML &lt;?xml version="1.0"?&gt; &lt;ROOT&gt; &lt;MYSTUDENT&gt;   &lt;_ID&gt;1111&lt;/_ID&gt; &lt;/MYSTUDENT&gt; &lt;/ROOT&gt;</pre>	1

---

#### xmlconfig Parameters (Continued)

---

Parameter	Description	Default
schematype	<p>Specifies the format of the schema used in XML output.</p> <p>inline – Only the top-level element (the record element) is defined globally. All other elements are children of the record element. A drawback of this format is that it is difficult for other schemas to reference child elements.</p> <p>ref – All elements are global; every element is a child of the root element. This format is somewhat restrictive: all elements are at the same level, so the element names must be unique. However, this format includes a ref tag for every element, which makes it easier for other schemas to reference elements.</p>	ref
hidemv	<p>Specifies whether to hide &lt;MV&gt; and &lt;/MV&gt; tags for multivalued fields belonging to an association in the generated XML document and in the associated DTD or XMLSchema. This parameter applies only if the XML document is created in element mode.</p> <p>By default, the MV tag is generated as "association name_MV".</p> <p>0 – Show MV tags for multivalued fields.</p> <p>1 – Hide MV tags for multivalued fields.</p>	0

---

#### xmlconfig Parameters (Continued)

Parameter	Description	Default
hidems	<p>Specifies whether to hide &lt;MS&gt; and &lt;/MS&gt; tags for multi-subvalued fields belonging to an association in the generated XML document and in the associated DTD or XMLSchema. This parameter applies only if the XML document is created in element mode.</p> <p>0 – Show MS tags for multi-subvalued fields.</p> <p>1 – Hide MS tags for multi-subvalued fields.</p>	0
collapsemv	<p>Specifies whether to collapse &lt;MV&gt; and &lt;/MV&gt; tags, using only one set of these tags for multivalued fields belonging to an association in the generated XML document and in the associated DTD or XMLSchema. This parameter applies only if the XML document is created in element mode.</p> <p>0 – Expand MV tags for multivalued fields.</p> <p>1 – Collapse MV tags multivalued fields.</p>	0
collapsemv	<p>Specifies whether to collapse &lt;MS&gt; and &lt;/MS&gt; tags, using only one set of these tags for multi-subvalued fields belonging to an association in the generated XML document and in the associated DTD or XMLSchema. This parameter applies only if the XML document is created in element mode.</p> <p>0 – Expand MS tags for multi-subvalued fields.</p> <p>1 – Collapse MS tags multi-subvalued fields.</p>	0

#### **xmlconfig Parameters (Continued)**

Parameter	Description	Default
hideroot	Specifies whether to create the entire XML document or only a section of it.  0 – UniVerse creates the entire XML document as well as a DTD and an XMLSchema.  1 – UniVerse creates only the record portion of the XML document; it does not create a DTD or an XMLSchema. For example, you may want only a section of the XML document if you are using the SAMPLE keyword and other conditional clauses	0
root	Specifies the name of the root element in XML output.	NULL
encode	Specifies the ASCII code for the character to be encoded.	NULL
target	Specifies the URL of the target namespace for XML output.	NULL
xmlns:namespace	Specifies the URL of the XML namespace for XML output.	NULL

#### xmlconfig Parameters (Continued)

## The Mapping File

You can create the U2XMLOUT.map file in \$UVHOME/&XML& to define commonly used global settings for creating XML documents. UniVerse reads and processes this mapping file each time UniVerse is started. For example, if you normally create element-centric output, and display empty elements for missing values or subvalues belonging to the same association, you can define these settings in the U2XMLOUT.map file, as shown in the following example:

```
<U2
  matchelement = "1"
  treated-as = "element"
/>
```

Defining these settings in the mapping file eliminates the need to specify them in each RetrievE statement.

UniVerse processes XML options as follows:

1. Reads options defined in the U2XMLOUT.map file when UniVerse starts.
2. Reads the \$UVHome\xmlconfig file.
3. Reads the account information from the XMLConfig file.
4. Reads any options defined in a mapping file. This mapping file resides in the &XML& directory in the current account, and is specified in the RetrievE statement, as shown in the following example:

```
LIST STUDENT.F SEMESTER TOXML XMLMAPPING mystudent.map
```

5. Processes any options you specify in the RetrievE statement.

Options you specify in the RetrievE statement override options defined in the mapping file. Options defined in the mapping file override options defined in the U2XMLOUT.map file.

A mapping file has the following format:

```
<?XML version="1.0"?>
<!--there can be multiple <U2xml:mapping> elements -->
<U2xml:mapping file="file_name">
```

**Note:** We suggest that you only put individual field options in the mapping file. Global options should be set in either the U2XMLOUT.map file or the xmlconfig file.

```

        hidemv="0"
        hidems="0"
        hideroot="0"
        collapsemv="0"
        collapsesms="0"
        emptyattribute="0"
        hastm="yes" | "1"
        matchelement="0" | "1"
        schematype="ref"
        targetnamespace="targetURL"
        xmlns:NAME="URL"
        field="dictionary_display_name"
        map-to="name_in_xml_douniversec"
        type="MV" | "MS"
        treated-as="attribute" | "element"
        root="root_element_name"
        record="record_element_name"
        association-mv="mv_level_assoc_name"
        association-ms="ms_level_assoc_name"
        format (or Fmt)= "format -pattern"..
        conversion (or Conv)= "conversion code"
        encode="encoding characters"
    />

...
</U2xml-mapping>

```

The XML mapping file is, in itself, in XML format. There are three types of significant elements: the root element, the field element, and the association element.

- **The root Element** – The root element describes the global options that control different output formats, such as the schema type, targetNamespace, hideroot, hidemv, and hidems. You can also use the root element to change the default root element name, or the record element name. You should have only one root element in the mapping file.
- **The field Element** – UniVerse uses the field element to change the characteristics of a particular field's XML output attributes, such as the display name, the format, or the conversion.

- **The association Element** – UniVerse uses the association element to change the display name of an association. By default, this name is the association phrase name, together with “-MV” or “-MS.”

## Distinguishing elements

You can distinguish the root element from the field and association elements because the root element does not define a field or association element.

Both the field element and the association element must have the file and field attribute to define the file name and the field name in the file that has been processed. Generally, the field name is a data-descriptor or I-descriptor defined in the dict file, making it a field element. If the field name is an association phrase, it is an association element.

The [Mapping file example](#) section shows this in more detail.

## Root element attributes

The default root element name in an XML document is ROOT. You can change the name of the root element, as shown in the following example:

```
root="root-element-name"
```

### Record name attribute

The default record name is FILENAME. The record attribute in the root element changes the record name. The following example illustrates the record attribute:

```
record="record-element-name"
```

### Hideroot attribute

The Hideroot attribute allows you to specify whether to create the entire XML document or only a section of it. For example, using the SAMPLE keyword or other conditional clauses. If Hideroot is set to 1, UniVerse only creates the record portion of the XML document, it does not create a DTD or XMLSchema. The default value is 0.

```
Hideroot="1" / "0"
```

### **Hidemv attribute**

This attribute specifies whether to hide <MV> and </MV> tags for multivalued fields belonging to an association in the generated XML document and in the associated DTD or XML Schema. This parameter applies only if the XML document is created in element mode.

- 0 - Show MV tags for multivalued fields.
- 1 - HideMV tags for multivalued fields.

You can also use this option with XMLEXECUTE().



*Note: If the document is created in attribute mode, it is not possible to eliminate the extra level of element tags.*

### **Hidems attribute**

This attribute specifies whether to hide <MS> and </MS> tags for multivalued fields belonging to an association in the generated XML document and in the associated DTD or XML Schema. This parameter applies only if the XML document is created in element mode.

- 0 - ShowMS tags for multi-subvalued fields.
- 1 - Hide MS tags for multi-subvalued fields.

You can also use this option with XMLEXECUTE().

*Note: If the document is created in attribute mode, it is not possible to eliminate the extra level of element tags.*

### **Collapsemv attribute**

This attribute specifies whether to collapse <MV> and </MV> tags, using only one set of these tags for multivalued fields belonging to an association in the generated XML document and in the associated DTD or XMLSchema. This parameter applies only if the XML document is created in element mode.

- 0 - Expand MV tags for each set of multivalued fields.
- 1 - CollapseMV tags for each set of multivalued fields.

### ***CollapseMS attribute***

This attribute specifies whether to collapse <MS> and </MS> tags, using only one set of these tags for multi-subvalued fields belonging to an association in the generated XML document and in the associated DTD or XMLSchema. This parameter applies only if the XML document is created in element mode.

- 0 - Expand MS tags for multi-subvalued fields.
- 1 - Collapse MS tags for multi-subvalued fields.

### ***Emptyattribute***

This attribute determines how to display the empty attributes for multivalued fields belonging to an association in the generated XML document. This option can be specified in the U2XMLOUT.map file, or in an individual mapping file.

- 0 - Ignores the empty attributes in the multivalued fields.
- 1 - Shows the empty attributes in the multivalued fields.

### ***Namespace attributes***

UniVerse provides the following attributes for defining namespaces:

- xmlns:name-space-name="URL"
- targetnamespace="URL"

UniVerse displays the targetnamespace attribute in the XMLSchema as targetNamespace, and uses the URL you define in the XML document to define the schema location.

If you define the targetnamespace and other explicit namespace definitions, UniVerse checks if the explicitly defined namespace has the same URL as the targetnamespace. If it does, UniVerse uses the namespace name to qualify the schema element, and the XML document element name.

In this case, UniVerse does not qualify the schema element or the XML document element.

UniVerse uses the namespace attributes and xmlns:name-space-name together to define the namespace. All namespaces defined in the root element are for global element namespace qualifiers only.



**Note:** Namespace is used primarily for XMLSchema. If you do not specify XMLSchema in the command line, UniVerse will not use a global namespace to qualify any element in the document.

The following example shows the output if the TARGETNAMESPACE attribute is set to “www.rocketU2.com”:

```
XMLSETOPTIONS TARGETNAMESPACE = "http://www.rocketU2.com"

>LIST STUDENT.F LNAME COURSE_NBR COURSE_GRD COURSE_NAME SEMESTER
FNAME TOXML WITHSCHEMA XMLMAPPING student.map
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="www.rocketU2.com"
    xmlns:rocketU2="http://www.rocketU2.com"
    xmlns="www.rocketU2.com"
    elementFormDefault="qualified">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
            account: C:\U2\ud72\XMLDemo\udxml
            command: LIST STUDENT.F LNAME COURSE_NBR COURSE_GRD
COURSE_NAME SEMESTER FNAME TOXML WITHSCHEMA XMLMAPPING student.map
        </xsd:documentation>
    </xsd:annotation>
    <xsd:element name="MAIN">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="STUDENT" type="STUDENTType" minOccurs="0"
maxOccurs="un
bounded"/>
                </xsd:sequence>
                <xsd:complexType>
            </xsd:element>
            <xsd:complexType name="STUDENTType">
                <xsd:sequence>
                    <xsd:element name="_ID" type="xsd:string"/>
                    <xsd:element name="LNAME" type="xsd:string"/>
                    <xsd:sequence minOccurs="0" maxOccurs="unbounded">
                        <xsd:element name="SEMESTER" type="xsd:string"/>
                        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
                            <xsd:element name="COURSE_GRD" type="xsd:string"/>
                            <xsd:element name="COURSE_NAME" type="xsd:string"/>
                            <xsd:element name="COURSE_NBR" type="xsd:string"/>
                        </xsd:sequence>
                    </xsd:sequence>
                    <xsd:element name="FNAME" type="xsd:string"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:schema>
        <?xml version="1.0"?>
    <MAIN
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns="www.rocketU2.com"
        xmlns:rocketU2="http://www.rocketU2.com"
    >
    <STUDENT>
        <_ID>123456789</_ID>
        <LNAME>Martin</LNAME>
```

```

<SEMESTER>SP94</SEMESTER>
    <COURSE_GRD></COURSE_GRD> ← empty value
    <COURSE_NAME>Introduction to Psychology</COURSE_NAME>
    <COURSE_NBR>PY100</COURSE_NBR>
    <COURSE_GRD>C</COURSE_GRD>
    <COURSE_NAME>Golf - I</COURSE_NAME>
    <COURSE_NBR>PE100</COURSE_NBR>
    <FNAME>Sally</FNAME>
</STUDENT>

<STUDENT>
    <_ID>987654321</_ID>
    <LNAME>Miller</LNAME>
        <SEMESTER>FA93</SEMESTER>
            <COURSE_GRD>C</COURSE_GRD>
            <COURSE_NAME>Engineering Principles</COURSE_NAME>
            <COURSE_NBR>EG110</COURSE_NBR>
            <COURSE_GRD></COURSE_GRD> ← empty value
            <COURSE_NAME>Calculus- I</COURSE_NAME>
            <COURSE_NBR>MA220</COURSE_NBR>
            <COURSE_GRD>B</COURSE_GRD>
            <COURSE_NAME>Introduction to Psychology</COURSE_NAME>
            <COURSE_NBR>PY100</COURSE_NBR>
        <SEMESTER>SP94</SEMESTER>
            <COURSE_GRD>B</COURSE_GRD>
            <COURSE_NAME>Fluid Mechanics</COURSE_NAME>
            <COURSE_NBR>EG140</COURSE_NBR>
            <COURSE_GRD>B</COURSE_GRD>
            <COURSE_NAME>Circuit Theory</COURSE_NAME>
            <COURSE_NBR>EG240</COURSE_NBR>
            <COURSE_GRD></COURSE_GRD> ← empty value
            <COURSE_NAME>Calculus - II</COURSE_NAME>
            <COURSE_NBR>MA221</COURSE_NBR>
        <FNAME>Susan</FNAME>
    </STUDENT>
</MAIN>
>

```

## **Schema attribute**

The default schema format is ref type schema. You can use the schema attribute to define a different schema format.

```
schema="inline" | "ref" | "type"
```

## **Elementformdefault and Attributeformdefault attributes**

UniVerse uses the elementformdefault and attributeformdefault attributes in the XML Schema. If you use them together with the namespace attribute in the root element, you can indicate all of the local elements and local attributes that need to be qualified with the namespace.

### **File attribute**

UniVerse uses the File attribute to process both RetrievVe and UniVerse SQL commands. If you do not define the file attribute exactly as it is used on the command line, the field element will not be properly processed.

```
File="filename"
```

### **Field attribute**

The Field attribute defines the field name. The field can be either a data-descriptor, an I-descriptor, or an ‘association phrase name’. For more information, see [Association elements](#).

```
Field="field-name"
```

**Note:** The file and field attributes are used to identify the query file and field needed to change the default directions. Use these attributes in the same element of the XML mapping file to pinpoint the database file and field.

### **Map-to attribute**

The Map-to attribute allows you to define a new attribute tag or element tag name for the field. By default, UniVerse uses the dictionary display field name for the element or attribute name tag.

### **Type attribute**

The Type attribute defines how to treat the field in the XML document, either as a multivalued field or a multi-subvalued field.

```
type="MV" | "MS"
```

### ***Treated-as attribute***

The Treated-as attribute determines if the field should be treated as an element or an attribute in the generated XML document.

### ***Matchelement attribute***

The Matchelement attribute specifies whether to display empty elements for missing values or subvalues belonging to the same association, or to ignore the missing values.

### ***Encode attribute***

The Encode attribute encodes unprintable characters, or characters that have special meanings in XML, such as { : }, with a macro.

```
encode="0x7B 0x7D"
```

### ***Conv attribute***

The Conv attribute changes the conversion defined in the dictionary record to the conversion you define.

```
conv="new conv code" | conversion = "new conversion code"
```

### ***Fmt attribute***

The Fmt attribute changes the format defined in the dictionary record to the format you define.

```
fmt="new format code" | format = "new format code"
```

## **Association elements**

An association element contains the following four attributes:

- file = “file name”
- field = “association phrase name”
- association-mv = “new multivalue element tag”
- association-ms = “new multi-subvalue element tag”

## Mapping file example

The following example illustrates the student.map mapping file:

```
<!-- this is for STUDENT.F file -->
<U2
    root="main"
    collapsemv='1'
    collapsesms='1'
    schema="ref"
    hidemv="1"
    hidems="1"
    hideroot="0"
    elementformdefault="qualified"
    attributeformdefault="qualified"
    treated-as="element"
/>
<U2 file="STUDENT.F"
    field = "CGA"
    association-mv="Term"
    association-ms="Courses_Taken"
/>
<U2 file="STUDENT.F"
    field = "COURSE_NBR"
    type="MS"
    treated-as="element"
/>
<U2 file="STUDENT.F"
    field = "SEMESTER"
    map-to="SEMESTER"
    type="MV"
    treated-as="element"
/>
<U2 file="STUDENT.F"
    field = "COURSE_GRD"
    map-to="COURSE_GRD"
    type="ms"
    treated-as="element"
/>
<U2 file="STUDENT.F"
    field = "COURSE_NAME"
    type="ms"
    treated-as="element"
/>
<U2 file="STUDENT.F"
    field = "TEACHER"
    type="ms"
    treated-as="element"
/>
```

Notice that the SEMESTER, COURSE\_NBR, COURSE\_GRD, and COURSE\_NAME fields are to be treated as elements. When you create the XML document, these fields will produce element-centric XML data. Any other fields listed in the query statement will produce attribute-centric XML data, since attribute-centric is the default mode.

Additionally, COURSE\_NBR, COURSE\_GRD, and COURSE\_NAME are defined as multi-subvalued fields. If they were not, UniVerse would create the XML data as if they were multivalued attributes.

*Note:* The global attributes listed above are not defined because they are set to "1".

The next example illustrates an XMLSchema using the mapping file in the previous example.

Use the following command to create the .xsd schema:

```
>LIST STUDENT.F LNAME SEMESTER COURSE_NBR TOXML XMLMAPPING  
student.map SCHEMAONLY

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="www.rocketU2.com"
    xmlns:intf="www.rocketU2.com"
    xmlns="www.rocketU2.com"
    elementFormDefault="qualified"
    attributeFormDefault="qualified">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
            account: C:\U2\UV
            command: LIST STUDENT.F LNAME SEMESTER COURSE_NBR TOXML
            XMLMAPPING student.map SCHEMAONLY
        </xsd:documentation>
        </xsd:annotation>
        <xsd:element name="main">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element ref="intf:STUDENT.F" minOccurs="0"
maxOccurs="unbounded"/>
                </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="STUDENT.F">
                <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="_ID" minOccurs="0" type="xsd:string"/>
                    <xsd:element name="LNAME" minOccurs="0"
type="xsd:string"/>
                    <xsd:sequence minOccurs="0" maxOccurs='unbounded'>
                        <xsd:element name="SEMESTER" minOccurs="0"
type="xsd:string"/>
                        <xsd:sequence minOccurs="0" maxOccurs='unbounded'>
                            <xsd:element name="COURSE_NBR" minOccurs="0"
type="xsd:string"/>
                        </xsd:sequence>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:schema>
    >
```

The next example illustrates an XML document created using the mapping file in the previous example. Use the following command to display the XML to the screen:

```
>LIST STUDENT.F LNAME SEMESTER COURSE_NBR TOXML XMLMAPPING
student.map

<?xml version="1.0" encoding="UTF-8"?>
<main>
<STUDENT.F>
    <_ID>424325656</_ID>
    <LNAME>Martin</LNAME>
    <SEMESTER>SP94</SEMESTER>
    <COURSE_NBR>PY100</COURSE_NBR>
    <COURSE_NBR>PE100</COURSE_NBR>
</STUDENT.F>
<STUDENT.F>
    <_ID>521814564</_ID>
    <LNAME>Smith</LNAME>
    <SEMESTER>FA93</SEMESTER>
    <COURSE_NBR>CS130</COURSE_NBR>
    <COURSE_NBR>CS100</COURSE_NBR>
    <COURSE_NBR>PY100</COURSE_NBR>
    <SEMESTER>SP94</SEMESTER>
    <COURSE_NBR>CS131</COURSE_NBR>
    <COURSE_NBR>CS101</COURSE_NBR>
    <COURSE_NBR>PE220</COURSE_NBR>
</STUDENT.F>
...
</main>
```

## ***Conversion code considerations***

UniVerse uses the following rules when extracting data from database files:

- If the dictionary record of a field you are extracting contains a conversion code, UniVerse uses that conversion code when extracting data from database files.
- If you specify a conversion code in the mapping file, the conversion code in the mapping file overrides the conversion code specified in the dictionary record.
- If you specify a conversion code using the CONV keyword during the execution of a RetriVe statement, that conversion code overrides both the conversion code specified in the mapping file and the conversion code specified in the dictionary record.

## ***Formatting considerations***

UniVerse does not generally apply the dictionary format pattern to the extracted data. To specify a format, define it in the mapping file. If you specify a format using the FMT keyword in a RetriVe statement, that format will override the format defined in the mapping file.

## ***Mapping file encoding***

For special characters encountered in data, UniVerse uses the default XML entities to encode the data. For example, '<' becomes &lt;, '>' becomes &gt;, '&' becomes &amp;, and '\"' becomes &quot;. However, UniVerse does not convert ' to &apos;, unless you specify it in attribute encode. (&lt;, &gt;, &amp;, &apos;, and &quot; are all built-in entities for the XML parser).

Use the encode field in the mapping file to add flexibility to the output. You can define special characters to encode in hexadecimal form. UniVerse encodes these special characters to &#x##. For example, if you want the character '{' to be encoded for field FIELD1, specify the following encode value in the mapping file for FIELD1:

```
encode="0x7B"
```

In this case, UniVerse will convert '{' found in the data of FIELD1 to &#x7B.

You can also use this type of encoding for any nonprintable character. If you need to define more than one character for a field, add a space between the hexadecimal definitions. For example, if you want to encode both '{' and '}', the encode value in the mapping file should look like the following example:

```
encode="0x7B 0x7D"
```

## **How data is mapped**

Regardless of the mapping mode you choose, the outer-most element in the XML document is created as <ROOT>, by default. The name of each record element defaults to <*file\_name*>.

You can change these mapping defaults in the mapping file, as shown in the following example:

```
<U2xml:mapping root="root_name"  
record="record_name" />
```

## Mapping example

The following example illustrates the creation of XML documents. These examples use the STUDENT.F file, which contains the following fields:

```
>LIST DICT STUDENT.F

DICT STUDENT.F      11:39:32am  11 Sep 2007  Page     1
          Type &
Field..... Field. Field..... Conversion.. Column..... Output
Depth &
Name..... Number Definition... Code..... Heading..... Format
Assoc..

@ID           D   0                      STUDENT      10L    S
ID            D   0                      STUDENT      12R### S
                  -##-##
# #
LNAME          D   1                      Last Name    40T    S
FNAME          D   2                      First Name   10L    S
MAJOR          D   3                      Major        20L    S
MINOR          D   4                      Minor        4L     S
ADVISOR         D   5                      Advisor      8L     S
SEMESTER        D   6                      Term         4L     S
CGA
TESTSEME        D   6                      Term         4L     S
COURSE_NBR      D   7                      Crs #       10L    S
CGA
TESTCOURSE      D   7                      Crs #       5L     S
COURSE_GRD      D   8                      GD           3L     S
CGA
?%
? ` ?
GPA1           I   SUBR('GPA1',C MD3)      GPA          5R     S

          Type &
Field..... Field. Field..... Conversion.. Column..... Output
Depth &
Name..... Number Definition... Code..... Heading..... Format
Assoc..

                               OURSE_HOURS,C
                               OURSE_GRD)
TEACHER          I   TRANS('COURSE'           Teacher      10L    M
CGA
                  S',COURSE_NBR
                  , 'TEACHER', 'X'
                  ')
COURSE_NAME      I   TRANS('COURSE'           Course Name  15T    S
CGA
                  S',COURSE_NBR
                  , 'NAME', 'X')
COURSE_HOURS    I   TRANS('COURSE'           Hours        5R     M
CGA
                  S',COURSE_NBR
                  , CREDITS, 'X')
@                PH  LNAME FNAME
                  MAJOR MINOR
                  ADVISOR
```

```
          SEMESTER
          COURSE_NBR
          Type &
Field..... Field. Field..... Conversion.. Column..... Output
Depth &
Name..... Number Definition... Code..... Heading..... Format
Assoc..

          COURSE_GRD
CGA           PH      SEMESTER
                  COURSE_NBR
                  COURSE_NAME
                  COURSE_GRD
                  COURSE_HOURS
                  TEACHER
@ORIGINAL     S  @ID
@SYNONYM       S   ID

M
M

22 records listed.
```

## TCL commands for XML

You can enter TCL commands to specify the encoding and other parameters for use in XML documents during the current session.

This section discusses the following TCL commands:

- [Session-level TCL commands](#)
- [Existing TCL command affected by XMLSETOPTIONS command or XMLSetOptions\(\) API](#)

### Session-level TCL commands

Three XML commands are available from the TCL command line:

- [XMLSETOPTIONS](#)
- [XMLGETOPTIONS](#)
- [XMLGETOPTIONVALUE](#)

## XMLSETOPTIONS

### *Syntax*

**XMLSETOPTIONS** <*options*>

### *Description*

Use this command to set the encoding parameter and other options for XML documents in the current session. XML settings entered with this command override the settings in the system-level and account-level xmlconfig files during the current UniVerse session.

## Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>options</i>	A string in the format of space-delimited key/value pairs. The XML options are the same as those in the xmlconfig file and accept the same values. Keys and values are case-insensitive. For a complete list of valid UniVerse XML options and settings, see <a href="#">xmlconfig parameters</a> . The XMLSETOPTIONS command also accepts three special strings as the <i>options</i> parameter. A special string must be entered as the only option: <ul style="list-style-type: none"><li>■ <b>defaults</b> – Sets all XML options to their default settings in the current session.</li><li>■ <b>reload</b> – Reloads the current system-level and account-level xmlconfig files, since they may have changed after you started your UniVerse session.</li><li>■ <b>reset</b> – Resets XML options to the original settings that were loaded when you started the UniVerse session.</li></ul> <p><i>Note:</i> If UniVerse encounters a problem such as a syntax error or an invalid value in the options string, it displays an error message and none of the XML parameters are changed.</p>

### XMLSETOPTIONS Parameters

## Examples

The following example shows the format for entering the XML options as key/value pairs in the TCL command.

```
XMLSETOPTIONS matchelement=1
```

The next example shows the format for entering a special string as the *options* parameter:

```
XMLSETOPTIONS defaults
```

## XMLGETOPTIONS

### Syntax

**XMLGETOPTIONS** <*delimiterString*>

### Description

Use this command to return the values of the encoding parameter and other XML options in effect in the current UniVerse session.

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>delimiterString</i>	Specifies the string to be used to separate the key/value pairs returned by the command.

#### XMLGETOPTIONS Parameters

### Examples

The following example shows the format for entering *delimiterString* as the string used to separate the key/value pairs returned by the command. Key/value pairs can be separated by a space or by any string, such as <>, as shown in this example:

```
XMLGETOPTIONS <>

standalone=yes<>out-xml-declaration=true<>out-format-pretty-
print=true<>out-norm
alize-characters=true<>out-split-cdata-sections=true<>out-
validation=false<>out-
expand-entity-references=false<>out-whitespace-in-element-
content=true<>out-disc
ard-default-content=true<>out-format-canonical=false<>out-write-
bom=false
matchelement=1<>emptyattribute=0<>elementmode=0<>schematype=ref<>h
idemv=0<>hidem
s=0<>collapsemv=0<>collapsemv=0<>hideroot=0<>
```

If you enter the `XMLSETOPTIONS` command with no `delimiterString`, the key/value pairs are separated by a space, as shown in the next example:

```
XMLGETOPTIONS  
standalone=yes out-xml-declaration=true  
out-format-pretty-print=true out-normalize-characters=true  
out-split-cdata-sections=true out-validation=false  
out-expand-entity-references=false  
out-whitespace-in-element-content=true  
out-discard-default-content=true out-format-canonical=false out-  
write-bom=false matchelement=1 emptyattribute=0  
elementmode=0 schematype=ref hidemv=0 hidems=0 collapsemv=0  
collapsems=0 hideroot=0
```

For a complete list of the standard UniVerse XML options and values returned by this command, see [xmlconfig parameters](#).

## XMLGETOPTIONVALUE

### Syntax

```
XMLGETOPTIONVALUE <optionName>
```

### Description

Use this command to return the value of the encoding parameter or any other XML option in effect in the current UniVerse session.

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<code>optionName</code>	Specifies the name of the XML option for which you want to return the current value.  For a complete list of valid UniVerse XML options, see <a href="#">xmlconfig parameters</a> .

#### XMLGETOPTIONVALUE Parameters

## **Example**

The following example shows the format for entering *optionName* to specify the XML parameter for which you want to return the current value.

```
XMLGETOPTIONVALUE encoding
```

This command returns the value of the encoding option, as shown below:

```
XMLGETOPTIONVALUE encoding
```

```
UTF-8
```

## **Existing TCL command affected by XMLSETOPTIONS command or XMLSetOptions() API**

The syntax of the following TCL command remains unchanged. However, the command is affected by the XML options you set previously at the session level through the XMLSETOPTIONS command or through the XMLSetOptions() API.

## **DB.TOXML**

### *Syntax*

```
DB.TOXML "xml_doc_filename" "xmap_filename" "condition"
```

### *Description*

Use the DB.TOXML command to create an XML document from the UniVerse database.



**Note:** The XML options set previously at the session level through the XMLSETOPTIONS command or through the XMLSetOptions() API are used when you run the DB.TOXML command in the current UniVerse session.



**Note:** You can use an active select list with the DB.TOXML command.

### *Parameters*

The following table describes each parameter of the syntax.

Parameter	Description
<i>xml_doc_filename</i>	The name of the XML document to create. If you do not enter a full path, the file is written to the &XML& directory.
<i>xmap_filename</i>	The file name for the XMAP file.
<i>condition</i>	A RetriVe condition string, for example, WITH CLASSID = "Class002"

### **DB.TOXML Parameters**

### *Example*

The following example illustrates using DB.TOXML from TCL to create an XML document.

```
DB.TOXML SCHOOL_STUDENT.XML SCHOOL.MAP WITH CLASSID = "Class002"
```

## Creating an XML document using RetrievE

To create an XML document using RetrievE, use the LIST command.

```
LIST [DICT | USING [DICT] dictname] filename ... [TOXML  
[ELEMENTS] [WITHDTD] [WITHSCHEMA | SCHEMAONLY]  
[XMLMAPPING mapping_file] [TO xmlfile]]
```

The following table describes each parameter of the syntax.

Parameter	Description
DICT	Lists records in the file dictionary of <i>filename</i> . If you do not specify DICT, records in the data file are listed.
USING [DICT] <i>dictname</i>	If DICT is not specified, uses the data portion of <i>dictname</i> as the dictionary of <i>filename</i> . If DICT is specified, the dictionary of <i>dictname</i> is used as the dictionary of <i>filename</i> .
<i>filename</i>	The file whose records you want to list. You can specify <i>filename</i> anywhere in the sentence. LIST uses the first word in the sentence that has a file descriptor in the VOC file as the file name.
TOXML	Outputs LIST results in XML format.
ELEMENTS	Outputs results in element-centric format.
WITHDTD	Output produces a DTD corresponding to the XML output.
WITHSCHEMA	The output produces an XML schema corresponding to the XML output.
SCHEMAONLY	The output produces a schema for the corresponding query.
XMLMAPPING <i>mapping_file</i>	Specifies a mapping file containing transformation rules for display. This file must exist in the &XML& file.
TO <i>xmlfile</i>	This option redirects the query xml output from the screen to the &XML& file. This file has a .xml suffix. If you specify WITHSCHEMA in the query, UniVerse creates an <i>xmlfile.xsd</i> in the &XML& directory. If you specify WITHDTD, UniVerse creates an <i>xmlfile.dtd</i> .

### LIST Parameters

For detailed information about the LIST command, see the *Using RetrievE*.

## Examples

### ***Creating an attribute-centric XML document***

Using the mapping file described in the [Mapping file example](#), the following example creates an attribute-centric XML document. To use a mapping file, specify the XMLMAPPING keyword in the RetriVe statement.

```
>LIST STUDENT.F LNAME FNAME SEMESTER COURSE_NBR COURSE_GRD  
COURSE_NAME TOXML XMLMAPPING student.map

<?xml version="1.0" encoding="UTF-8"?>
<main>
<STUDENT _ID = "987654321" LNAME = "Miller" FNAME = "Susan">
    <Term SEMESTER = "FA93">
        <Courses_Taken COURSE_NBR = "EG110" COURSE_GRD = "C"
        COURSE_NAME = "Engineering Principles"/>
        <Courses_Taken COURSE_NBR = "MA220" COURSE_NAME = "Calculus-I"/>
        <Courses_Taken COURSE_NBR = "PY100" COURSE_GRD = "B"
        COURSE_NAME = "Introduction to Psychology"/>
    </Term>
    <Term SEMESTER = "SP94">
        <Courses_Taken COURSE_NBR = "EG140" COURSE_GRD = "B"
        COURSE_NAME = "Fluid Mechanics"/>
        <Courses_Taken COURSE_NBR = "EG240" COURSE_GRD = "B"
        COURSE_NAME = "Circuit Theory"/>
        <Courses_Taken COURSE_NBR = "MA221" COURSE_NAME = "Calculus-II"/>
    </Term>
</STUDENT>
<STUDENT _ID = "123456789" LNAME = "Martin" FNAME = "Sally">
    <Term SEMESTER = "SP94">
        <Courses_Taken COURSE_NBR = "PY100" COURSE_NAME =
        "Introduction to Psychology"/>
        <Courses_Taken COURSE_NBR = "PE100" COURSE_GRD = "C"
        COURSE_NAME = "Golf - I"/>
    </Term>
</STUDENT>
</main>
>
```

### ***Creating an XML document with a DTD or XML schema***

If you only include the TOXML keyword in the RetrieVe statement, the resulting XML document does not include a DTD or XML Schema. To create an XML document that includes a DTD, use the WITHDTD keyword. To create an XML document that includes an XML Schema, use the WITHSCHEMA keyword.

The following example illustrates an XML document that includes a DTD:

```
>LIST STUDENT.F SEMESTER COURSE_NBR COURSE_GRD COURSE_NAME TOXML  
WITHDTD  
  
<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE ROOT [  
  <!ELEMENT ROOT (STUDENT.F*)>  
  <!ELEMENT STUDENT.F ( CGA_MV* )>  
  <!ATTLIST STUDENT.F  
        _ID CDATA #REQUIRED  
>  
  <!ELEMENT CGA_MV ( CGA_MS* )>  
  <!ATTLIST CGA_MV  
        SEMESTER CDATA #IMPLIED  
>  
  <!ELEMENT CGA_MS EMPTY>  
  <!ATTLIST CGA_MS  
        COURSE_NBR CDATA #IMPLIED  
        COURSE_GRD CDATA #IMPLIED  
        COURSE_NAME CDATA #IMPLIED  
>  
]>  
  
<ROOT>  
<STUDENT.F _ID = "424325656">  
  <CGA_MV SEMESTER = "SP94">  
    <CGA_MS COURSE_NBR = "PY100" COURSE_NAME = "Introduction to  
Psychology"/>  
    <CGA_MS COURSE_NBR = "PE100" COURSE_GRD = "C"  
      COURSE_NAME = "Golf - I"/>  
  </CGA_MV>  
</STUDENT.F>  
<STUDENT.F _ID = "521814564">  
  <CGA_MV SEMESTER = "FA93">  
    <CGA_MS COURSE_NBR = "CS130" COURSE_GRD = "A"  
      COURSE_NAME = "Intro to Operating Systems"/>  
    <CGA_MS COURSE_NBR = "CS100" COURSE_GRD = "B"  
      COURSE_NAME = "Intro to Computer Science"/>  
    <CGA_MS COURSE_NBR = "PY100" COURSE_GRD = "B"  
      COURSE_NAME = "Introduction to Psychology"/>  
  </CGA_MV>  
  <CGA_MV SEMESTER = "SP94">  
    <CGA_MS COURSE_NBR = "CS131" COURSE_GRD = "B"  
      COURSE_NAME = "Intro to Operating Systems"/>  
    <CGA_MS COURSE_NBR = "CS101" COURSE_GRD = "B"  
      COURSE_NAME = "Intro to Computer Science"/>  
    <CGA_MS COURSE_NBR = "PE220" COURSE_GRD = "A"  
      COURSE_NAME = "Racquetball"/>  
  </CGA_MV>  
</STUDENT.F>  
...  
>
```

## **Using WITHSCHEMA**

Use the WITHSCHEMA keyword with the RetrievVe LIST command to create an XML schema.

The syntax for the LIST command is:

```
LIST [DICT | USING [DICT] dictname] filename ... [TOXML  
[ELEMENTS][WITHSCHEMA][WITHDTD] [SCHEMAONLY] TO filename  
[XMLMAPPING mapping_file] [TO xmlfile]]...
```



*Note:* If you specify both WITHDTD and WITHSCHEMA in the same RetrievVe statement, UniVerse does not produce an XML schema.

WITHSCHEMA creates an XML schema *filename.xsd*. By default, UniVerse writes this file to the &XML& directory when the TO *xmlfile* command is used. The information is displayed on the screen if the TO *xmlfile* command is not used. If you do not specify a targetNamespace in the mapping file, the *filename.xml*'s root element contains the following command to define the schema location:

```
noNamespaceSchemaLocation=filename.xsd
```

If you specify the targetNamespace in the mapping file, UniVerse generates the following:

```
schemaLocation="namespaceURL filename.xsd"
```

In both of these cases, you can validate the files using the XML schema validator, or the UniVerse Basic API XDOMValidate() function.

## **Mapping to an external schema**

A mapping file enables users to define how the dictionary attributes correspond to the DTD or XML Schema elements. This allows you to create many different forms of XML. Defining settings in the mapping file eliminates the need to specify them in each RetrievVe statement. The following example illustrates how to map to an external schema.

Assume you are trying to map to the following schema:

```
:<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:rocketU2="http://www.rocketU2.com"
  elementFormDefault="qualified">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">
      This is a sample schema
    </xsd:documentation>
  </xsd:annotation>
  <xsd:element name="transcript">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="student" type="studentType" minOccurs="0"
maxOccurs="unbounded"/>
      </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:complexType name="studentType">
      <xsd:sequence>
        <xsd:element name="semesterReport"
type="semesterReportType" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="ref" type="xsd:string"/>
      <xsd:attribute name="firstName" type="xsd:string"/>
      <xsd:attribute name="lastName" type="xsd:string"/>
    </xsd:complexType>
    <xsd:complexType name="semesterReportType">
      <xsd:sequence>
        <xsd:element name="results" type="resultsType"
minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="term" type="xsd:string"/>
    </xsd:complexType>
    <xsd:complexType name="resultsType">
      <xsd:sequence>
        <xsd:element name="courseGrade" type="xsd:string"/>
        <xsd:element name="courseHours" type="xsd:string"/>
      </xsd:sequence>
      <xsd:attribute name="courseNumber" type="xsd:string"/>
      <xsd:attribute name="courseName" type="xsd:string"/>
      <xsd:attribute name="courseInstructor"
type="xsd:string"/>
    </xsd:complexType>
  </xsd:schema>
```

The following map illustrates how to map your student file to this schema.  
Use the steps shown below to create the map:

1. Set the default settings for the map.
2. Rename singlevalued fields to match the schema names.
3. Rename the element tags used for the association.
4. Rename the multivalued fields.
5. Rename the multi-subvalued fields.

The following mapping file is the transcript.map file.

```
>
<u2
!-- First set the default settings for the map -->
  root="transcript"
  record="student"
  schema="type"
  xmlns:rocketU2="http://www.rocketU2.com"
  treated-as="element"
  collaosemv="1"
/>

!-- Rename singlevalued fields to match the schema names -->

<u2
file="STUDENT.F"
field="@ID"
map-to="ref"
type="S"
treated-as="attribute"
/>

<u2
file="STUDENT.F"
field="FNAME"
map-to="firstName"
type="S"
treated-as="attribute"
/>

<u2
file="STUDENT.F"
field="LNAME"
map-to="lastName"
type="S"
treated-as="attribute"
/>

!-- Rename the element tags used for the association -->
<u2
file="STUDENT.F"
field="CGA"
association-mv="semesterReport"
association-ms="results"
/>

!-- Rename the multivalued fields -->
<u2
file="STUDENT.F"
field="SEMESTER"
map-to="term"
type="MV"
treated-as="attribute"
```

```
/>

<!-- Rename the multi-subvalued fields -->

<u2
file="STUDENT.F"
field="COURSE_NBR"
map-to="courseNumber"
type="MSV"
treated-as="attribute"
/>

<u2
file="STUDENT.F"
field="COURSE_NAME"
map-to="courseName"
treated-as="attribute"
type="MSV"
/>

<u2
file="STUDENT.F"
field="COURSE_GRD"
map-to="courseGrade"
type="MSV"
/>

<u2
file="STUDENT.F"
field="COURSE_HOURS"
map-to="courseHours"
type="MSV"
/>

<u2
file="STUDENT.F"
field="TEACHER"
map-to="courseInstructor"
type="MSV"
treated-as="attribute"
/>>
```

You can now view the output from the schema using the following command:

```
>LIST STUDENT.F FNAME LNAME CGA SAMPLE 1 TOXML XMLMAPPING  
transcript.map

<?xml version="1.0" encoding="UTF-8"?>
<transcript
    xmlns:rocketU2="http://www.rocketU2.com"
>
<student ref = "424325656" firstName = "Sally" lastName =
"Martin">
    <semesterReport term = "SP94" courseNumber = "PY100"
        courseName = "Introduction to Psychology"
        courseInstructor = "Masters"
        courseNumber = "PE100"
        courseName= "Golf - I"
        courseInstructor = "Fisher">
        <courseGrade>C</courseGrade>
        <courseHours>3</courseHours>
        <courseHours>3</courseHours>
    </semesterReport>
</student>
</transcript>
>
```

## Creating an XML document with UniVerse SQL

In addition to RetrieVe, you can also create XML documents using UniVerse SQL. To create an XML document through UniVerse SQL, complete the following steps:

1. Analyze the DTD or XML schema associated with the application to which you are sending the XML file. Determine which of your dictionary nsattributes correspond to the DTD or XML schema elements.
2. Create an XML mapping file, if necessary.
3. List the appropriate fields using the UniVerse SQL SELECT command.

To create an XML document from UniVerse SQL, use the UniVerse SQL SELECT command.

The following table describes each parameter of the syntax.

Parameter	Description
SELECT clause	Specifies the columns to select from the database.
FROM clause	Specifies the tables containing the selected columns.
WHERE clause	Specifies the criteria that rows must meet to be selected.
WHEN clause	Specifies the criteria that values in a multivalued column must meet for an association row to be output.
GROUP BY clause	Groups rows to summarize results.
HAVING clause	Specifies the criteria that grouped rows must meet to be selected.
ORDER BY clause	Sorts selected rows.
<i>report_qualifiers</i>	Formats a report generated by the SELECT statement.
<i>processing_qualifiers</i>	Modifies or reports on the processing of the SELECT statement.

### SELECT Parameters

Parameter	Description
TOXML	Outputs SELECT results in XML format.
ELEMENTS	Outputs results in element-centric format.
WITHDTD	Output produces a DTD corresponding to the query.
WITHSCHEMA	Output produces an XML schema corresponding to the query.
SCHEMAONLY	The output will produce a schema for the corresponding query.
XMLMAPPING <i>'mapping_file'</i>	Specifies a mapping file containing transformation rules for display. This file must exist in the &XML& file.
XMLDATA <i>'extraction_mapping_file'</i>	Specifies the file containing the extraction rules for the XML document. This file is used for receiving an XML file.
TO <i>'xmlfile'</i>	This option redirects the query xml output from the screen to the &XML& file. This file has a .xml suffix. If you specify WITHSCHEMA in the query, UniVerse creates an <i>xmlfile.xsd</i> in the &XML& directory. If you specify WITHDTD, UniVerse creates an <i>xmlfile.dtd</i> as well.

#### **SELECT Parameters (Continued)**

You must specify clauses in the SELECT statement in the order shown in the syntax.

For a full discussion of the UniVerse SQL SELECT statement clauses, see *Using UniVerse SQL*.

## **Processing rules for UniVerse SQL SELECT statements**

UniVerse processes SELECT statements much the same as it processes LIST statements, with a few exceptions.

The processing rules for a UniVerse SQL SELECT statement against a single table are the same as the RetrievE LIST rules. For a discussion of how UniVerse SQL processes these statements, see [Creating an XML document from RetrievE](#).

## ***Processing multiple tables***

When processing a UniVerse SQL SELECT statement involving multiple files, UniVerse attempts to keep the nesting inherited in the query in the resulting XML document. Because of this, the order in which you specify the fields in the UniVerse SQL SELECT statement is important for determining how the elements are nested.

### ***Processing in attribute-centric mode***

As with RetrievVe, the attribute-centric mode is the default mapping mode. For more information about the attribute-centric mode, see the [Attribute-centric mode](#) section.

- In this mode, UniVerse uses the name of the file containing the first field you specify in the SELECT statement as the outer-most element in the XML output. Any singlevalued fields you specify in the SELECT statement that belong to this file become attributes of this element.
- UniVerse processes the SELECT statement in the order you specify. If it finds a field that belongs to another file, UniVerse creates a sub-element. The name of this sub-element is the new file name. All singlevalued fields found in the SELECT statement that belong to this file are created as attributes for the sub-element.
- If UniVerse finds a multivalued or multi-subvalued field in the SELECT statement, it creates a sub-element. The name of this element is the name of the association of which this field is a member.
- When you execute UNNEST against an SQL table, it flattens the multivalues into single values.

UniVerse processes the ELEMENTS, WITHDTD, WITHSCHEMA, SCHEMAONLY and XMLMAPPING keywords in the same manner as it processes them for the RetrievVe LIST command.

### ***Processing in Element-centric mode***

When using the element-centric mode, UniVerse automatically prefixes each file name to the association name. For example, the CGA association in the STUDENT file is named STUDENT\_CGA in the resulting XML file.

## **XML limitations in UniVerse SQL**

The TOXML keyword is not allowed in the following cases:

- In a sub-query
- In a SELECT statement that is part of an INSERT statement.
- In a SELECT statement that is part of a UNION definition.
- In a SELECT statement that is part of a VIEW definition.

## **Examples**

This section illustrates XML output from the UniVerse SQL SELECT statement. The examples use sample CUSTOMER, TAPES, and STUDENT files.

The following example lists the dictionary records from the CUSTOMER file that are used in the examples:

```
>LIST DICT CUSTOMER.F

DICT CUSTOMER      02:11:01pm 11 Sep 2007  Page      1

          Type &
Field..... Field. Field..... Conversion.. Column..... Output
Depth &
Name..... Number Definition... Code..... Heading..... Format
Assoc..

@ID        D    0           CUSTOMER      10L     S
NAME       D    1           Customer Name 15T     S
TAPES_RENTED D    7           Tapes         10L     M

TAPE_INFO      PH    TAPES_RENTED
          Type &
          DATE_OUT
          DATE_DUE
          DAYS_BETWEEN
          TAPE_COST
          TAPE_NAME
          UP_NAMES

29 records listed.

>LIST DICT TAPES.F

DICT TAPES.F      07:51:38am 07 Jan 2009  Page      1

          Type &
Field..... Field. Field..... Conversion.. Column..... Output
Depth &
Name..... Number Definition... Code..... Heading..... Format
Assoc..

@ID        D    0           TAPES      10L     S
ID         D    0           TAPES      10L     S
NAME       D    1           Tape Name  20T     S
RENTAL_PRICE D   2           MD2      Retail Charge 8R     S
COPIES     D    3           MD2      Copies Owned 4R     S
COPIES_OUT  D   4           MD0      Rented      4R     S
NUM_RENTALS D   5           MD0      Times Rented 6R     S
COST        D   6           MD2      Tape Cost   6R2$, S
ACTORS      D   7           MD2      Actors      12L     S
DIRECTOR    D   8           MD2      Director    12L     S
CATEGORIES   D   9           MD2      Type of Video 12L     S
CATS        C   1

NEW_PRICE     I      IF @ID[1,1] = MD2
                  'B' THEN 0
                  ELSE
                  NUM_RENTALS;I
                  F @1 > 10

...
>
```

## ***Using WITHSCHEMA***

The syntax for the UniVerse SQL SELECT command is:

```
SELECT command.  
SELECT clause FROM clause  
[WHERE clause]  
[WHEN clause [WHEN clause]...]  
[GROUP BY clause]  
[HAVING clause]  
[ORDER BY clause]  
[report_qualifiers]  
[processing_qualifiers]  
[TOXML [ELEMENTS] [WITHDTD] [WITHSCHEMA] [SCHEMAONLY]  
[XMLMAPPING 'mapping_file']]  
[XMLDATA extraction_mapping_file]  
[TO 'xmlfile'];
```

When the TOXML command is used in SQL, both the mapping file and the TO xml file need to be quoted

## ***Creating an XML document from multiple files with a multivalued field***

The next example illustrates creating an XML document from multiple files with a multivalued field. In the example, TAPES\_RENTED is multivalued and belongs to the TAPE\_INFO association in the CUSTOMER.F file. In the XML document, TAPES\_RENTED appears in the CUSTOMER\_TAPE\_INFO\_MV element.

```
>SELECT CUSTOMER.F.NAME, TAPES.F.CAT_NAME,TAPES_RENTED FROM
CUSTOMER.F,TAPES.F WHERE TAPES_RENTED = TAPES.F.@ID ORDER BY
CUSTOMER.F.NAME TOXML;

<?xml version="1.0" encoding="UTF-8"?>
<ROOT>
<CUSTOMER.F NAME = "Barrie, Dick">
  <TAPES.F>
    <TAPES.F_CATS_MV CAT_NAME = "Old Classic"/>
    <TAPES.F_CATS_MV CAT_NAME = "Drama"/>
    <TAPES.F_CATS_MV CAT_NAME = "Horror"/>
  </TAPES.F>
  <CUSTOMER.F_TAPE_INFO_MV TAPES_RENTED = "V996" />
</CUSTOMER.F>
<CUSTOMER.F NAME = "Best, George">
  <TAPES.F>
    <TAPES.F_CATS_MV CAT_NAME = "Romance"/>
    <TAPES.F_CATS_MV CAT_NAME = "Tear Jerker"/>
  </TAPES.F>
  <CUSTOMER.F_TAPE_INFO_MV TAPES_RENTED = "B2297" />
</CUSTOMER.F>
<CUSTOMER.F NAME = "Bowie, David">
  <TAPES.F>
    <TAPES.F_CATS_MV CAT_NAME = "Avant Garde"/>
    <TAPES.F_CATS_MV CAT_NAME = "Science Fiction"/>
  </TAPES.F>
  <CUSTOMER.F_TAPE_INFO_MV TAPES_RENTED = "V9961" />
</CUSTOMER.F>
...
</ROOT>
```

## ***Creating an XML document From multiple files with a DTD***

The following example illustrates creating an XML document from multiple files with a DTD. To include the DTD, use the WITHDTD keyword.

```
>SELECT CUSTOMER.F.NAME, TAPES.F.CAT_NAME, TAPES.RENTED FROM
CUSTOMER.F, TAPES.F WHERE TAPES.RENTED = TAPES.F.@ID ORDER BY
CUSTOMER.F.NAME TOXML WITHDTD;
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ROOT[
<!ELEMENT ROOT (CUSTOMER.F*)>
<!ELEMENT CUSTOMER.F ( TAPES.F*, CUSTOMER.F_TAPE_INFO_MV* )>
<!ATTLIST CUSTOMER.F
    NAME CDATA #REQUIRED
>
<!ELEMENT TAPES.F ( TAPES.F_CATS_MV* )>
<!ELEMENT TAPES.F_CATS_MV EMPTY>
<!ATTLIST TAPES.F_CATS_MV
    CAT_NAME CDATA #IMPLIED
>
<!ELEMENT CUSTOMER.F_TAPE_INFO_MV EMPTY>
<!ATTLIST CUSTOMER.F_TAPE_INFO_MV
    TAPES_RENTED CDATA #IMPLIED
>
]>

<ROOT>
<CUSTOMER.F NAME = "Barrie, Dick">
    <TAPES.F>
        <TAPES.F_CATS_MV CAT_NAME = "Old Classic"/>
        <TAPES.F_CATS_MV CAT_NAME = "Drama"/>
        <TAPES.F_CATS_MV CAT_NAME = "Horror"/>
    </TAPES.F>
    <CUSTOMER.F_TAPE_INFO_MV TAPES_RENTED = "V996"/>
</CUSTOMER.F>
<CUSTOMER.F NAME = "Best, George">
    <TAPES.F>
        <TAPES.F_CATS_MV CAT_NAME = "Romance"/>
        <TAPES.F_CATS_MV CAT_NAME = "Tear Jerker"/>
    </TAPES.F>
    <CUSTOMER.F_TAPE_INFO_MV TAPES_RENTED = "B2297"/>
</CUSTOMER.F>
<CUSTOMER.F NAME = "Bowie, David">
    <TAPES.F>
        <TAPES.F_CATS_MV CAT_NAME = "Avant Garde"/>
        <TAPES.F_CATS_MV CAT_NAME = "Science Fiction"/>
    </TAPES.F>
    <CUSTOMER.F_TAPE_INFO_MV TAPES_RENTED = "V9961"/>
</CUSTOMER.F>
<CUSTOMER.F NAME = "Chase, Carl">
    <TAPES.F>
```

```
<TAPES.F_CATS_MV CAT_NAME = "Musical" />
<TAPES.F_CATS_MV CAT_NAME = "Drama" />
</TAPES.F>
<CUSTOMER.F_TAPE_INFO_MV TAPES_RENTED = "V8481" />
<CUSTOMER.F_TAPE_INFO_MV TAPES_RENTED = "V1254" />
<CUSTOMER.F_TAPE_INFO_MV TAPES_RENTED = "V4951" />
</CUSTOMER.F>
<CUSTOMER.F NAME = "Chase, Carl">
<TAPES.F>
    <TAPES.F_CATS_MV CAT_NAME = "Comedy" />
    <TAPES.F_CATS_MV CAT_NAME = "Childrens Movie" />
</TAPES.F>
<CUSTOMER.F_TAPE_INFO_MV TAPES_RENTED = "V8481" />
<CUSTOMER.F_TAPE_INFO_MV TAPES_RENTED = "V1254" />
<CUSTOMER.F_TAPE_INFO_MV TAPES_RENTED = "V4951" />
</CUSTOMER.F>
...
</ROOT>
>
```

## ***Creating an XML document from multiple files using a mapping file***

As with RetrievE, you can create a mapping file to define transformation rules differing from the defaults. For information about creating the mapping file, see [The Mapping File](#) section.

The following mapping file defines rules for the CUSTOMER and TAPES file.

```
>ED &XML& CUST_TAPES.map
Top of "CUST_TAPES.map" in "&XML&", 22 lines, 259 characters.
*--: p
001: <U2xml
002:   file="TAPES.F"
003:   field = "CAT_NAME"
004:   map-to="Cat_name"
005:   type="mv"
006: />
007: <u2
008:   file="CUSTOMER.F"
009:   field="TAPES_RENTED"
010:   map-to="Tapes_rented"
011:   TYPE="mv"
012: />
013: <u2
014:   file="CUSTOMER.F"
015:   field="DATE_OUT"
016:   TYPE="mv"
017: />
018: <u2
019:   file="CUSTOMER.F"
020:   field="DATE_DUE"
021:   TYPE="mv"
022: />
```

To use this mapping file in the SELECT statement, specify the XMLMAPPING keyword, as shown in the following example:

**Note:** You must surround the name of the mapping file in single quotation marks.

```
>SELECT CUSTOMER.F.NAME, TAPES.F.NAME, CAT_NAME, DATE_OUT,
DATE_DUE FROM CUSTOMER.F, TAPES.F WHERE TAPES.RENTED = TAPES.F.@ID
ORDER BY CUSTOMER.F.NAME TOXML XMLMAPPING 'CUST_TAPES.MAP';

<?xml version="1.0" encoding="UTF-8"?>
<ROOT>
<CUSTOMER.F NAME = "Barrie, Dick">
<TAPES.F NAME = "Citizen Kane">
<TAPES.F_CATS_MV Cat_name = "Old Classic"/>
<TAPES.F_CATS_MV Cat_name = "Drama"/>
<TAPES.F_CATS_MV Cat_name = "Horror"/>
</TAPES.F>
<CUSTOMER.F_TAPE_INFO_MV DATE_OUT = "03/29/94" DATE_DUE =
"03/31/94"/>
</CUSTOMER.F>
<CUSTOMER.F NAME = "Best, George">
<TAPES.F NAME = "Love Story">
<TAPES.F_CATS_MV Cat_name = "Romance"/>
<TAPES.F_CATS_MV Cat_name = "Tear Jerker"/>
</TAPES.F>
<CUSTOMER.F_TAPE_INFO_MV DATE_OUT = "03/29/94" DATE_DUE =
"03/31/94"/>
</CUSTOMER.F>
<CUSTOMER.F NAME = "Bowie, David">
<TAPES.F NAME = "The Stalker">
<TAPES.F_CATS_MV Cat_name = "Avant Garde"/>
<TAPES.F_CATS_MV Cat_name = "Science Fiction"/>
</TAPES.F>
<CUSTOMER.F_TAPE_INFO_MV DATE_OUT = "04/15/94" DATE_DUE =
"04/17/94"/>
</CUSTOMER.F>
<CUSTOMER.F NAME = "Chase, Carl">
<TAPES.F NAME = "'Round Midnight">
<TAPES.F_CATS_MV Cat_name = "Musical"/>
<TAPES.F_CATS_MV Cat_name = "Drama"/>
</TAPES.F>
<CUSTOMER.F_TAPE_INFO_MV DATE_OUT = "04/20/94" DATE_DUE =
"04/22/94"/>
<CUSTOMER.F_TAPE_INFO_MV DATE_OUT = "04/20/94" DATE_DUE =
"04/22/94"/>
<CUSTOMER.F_TAPE_INFO_MV DATE_OUT = "04/21/94" DATE_DUE =
"04/23/94"/>
</CUSTOMER.F>
<CUSTOMER.F NAME = "Chase, Carl">
<TAPES.F NAME = "American Graffiti ">
<TAPES.F_CATS_MV Cat_name = "Comedy"/>
<TAPES.F_CATS_MV Cat_name = "Childrens Movie"/>
</TAPES.F>
<CUSTOMER.F_TAPE_INFO_MV DATE_OUT = "04/20/94" DATE_DUE =
"04/22/94"/>
```

```
<CUSTOMER.F_TAPE_INFO_MV DATE_OUT = "04/20/94" DATE_DUE =
"04/22/94"/>
<CUSTOMER.F_TAPE_INFO_MV DATE_OUT = "04/21/94" DATE_DUE =
"04/23/94"/>
</CUSTOMER.F>
<CUSTOMER.F NAME = "Chase, Carl">
<TAPES.F NAME = "Flash Gordon">
<TAPES.F_CATS_MV Cat_name = "Science Fiction"/>
<TAPES.F_CATS_MV Cat_name = "Childrens Movie"/>
</TAPES.F>
<CUSTOMER.F_TAPE_INFO_MV DATE_OUT = "04/20/94" DATE_DUE =
"04/22/94"/>
<CUSTOMER.F_TAPE_INFO_MV DATE_OUT = "04/20/94" DATE_DUE =
"04/22/94"/>
<CUSTOMER.F_TAPE_INFO_MV DATE_OUT = "04/21/94" DATE_DUE =
"04/23/94"/>
</CUSTOMER.F>
<CUSTOMER.F NAME = "Faber, Harry">
<TAPES.F NAME = "To Kill A Mockingbird">
<TAPES.F_CATS_MV Cat_name = "Horror"/>
<TAPES.F_CATS_MV Cat_name = "Political"/>
<TAPES.F_CATS_MV Cat_name = "Drama"/>
</TAPES.F>
<CUSTOMER.F_TAPE_INFO_MV DATE_OUT = "04/19/94" DATE_DUE =
"04/21/94"/>
</CUSTOMER.F>
...
</ROOT>
```

## **Creating an XML document through UniVerse BASIC**

Use the UniVerse Basic commands described in this section to create XML documents through UniVerse Basic.

## Using the XMLExecute() function

### Syntax

**XMLExecute**(*cmd*, *options*, *xmlvar*, *xsdvar*)

### Description

The XMLExecute function enables you to create an XML document using RetrievVe from UniVerse Basic and returns the xml and xsd schema in BASIC variables. By default, the XMLExecute command generates an XML Schema. The options can also be separated by a [space] command, using “=” to assign option values.

The following table describes each parameter of the syntax.

Parameter	Description
<i>cmd</i>	Holds the text string of the RetrievVe LIST statement or the UniVerse SQL SELECT statement. [IN]
<i>options</i>	Each XML-related option is separated by a field mark (@FM), or space (“ ”). If the option requires a value, the values are contained in the same field, separated by value marks (@VM), or equal signs (“=”).
WITHDTD	Creates a DTD and binds it with the XML document. By default, UniVerse creates an XML schema. However, if you include WITHDTD in your RetrievVe or UniVerse SQL statement, UniVerse does not create an XML schema, but only produces the DTD.
ELEMENTS	The XML output is in element-centric format.
XMLMAPPING = ' <i>mapping_file_name</i> '	Specifies the mapping file containing transformation rules for display. This file must exist in the &XML& directory.
SCHEMA = ' <i>type</i> '	The default schema format is ref type schema. You can use the SCHEMA attribute to define a different schema format.

#### XMLExecute Parameters

Parameter	Description
HIDEMV, HIDEMS	Normally, when UniVerse processes multivalued or multi-subvalued fields, UniVerse adds another level of elements to produce multiple levels of nesting. You have the option of disabling this additional level by adding the HIDEMV and HIDEMS attributes. When these options are on, the generated XML document and the associated DTD or XML schema have fewer levels of nesting.
HIDEROOT = 1	Allows you to specify to only create a segment of an XML document, for example, using the SAMPLE keyword and other conditional clauses. If you specify HIDEROOT, UniVerse only creates the record portion of the XML document, it does not create a DTD or XML schema.
RECORD = ' <i>newrecords</i> '	The default record name is FILENAME_record. The record attribute in the ROOT element changes the record name.
ROOT = ' <i>newroot</i> '	The default root element name in an XML document is ROOT. You can change the name of the root element as shown in the following example:
	<code>root="root_element_name"</code>
TARGETNAMESPACE = ' <i>namespaceURL</i> '	UniVerse displays the targetnamespace attribute in the XMLSchema as <i>targetNamespace</i> , and uses the URL you specify to define <i>schemaLocation</i> . If you define the targetnamespace and other explicit namespace definitions, UniVerse checks if the explicitly defined namespace has the same URL and the targetnamespace. If it does, UniVerse uses the namespace name to qualify the schema element, and the XML document element name.

---

#### XMLExecute Parameters (Continued)

---

Parameter	Description
COLLAPSEMV, COLLAPSEMS	Normally, when UniVerse processes multivalued or multi-subvalued fields, UniVerse adds another level of elements to produce multiple levels of nesting. You have the option of disabling this additional level by adding the COLLAPSEMV and COLLAPSEMS attributes. When these options are on, the generated XML document and the associated DTD or XML Schema have fewer levels of nesting.
MATCHELEMENT	The Matchelement attribute specifies whether to display empty elements for missing values or subvalues belonging to the same association, or to ignore the missing values. When this option is on, the generated XML document and the associated DTD or XML Schema have fewer levels of nesting.
EMPTYATTRIBUTE	This attribute determines how to display the empty attributes for multivalued fields belonging to an association in the generated XML document and in the associated DTD or XML Schema. When this option is on, the generated XML document and the associated DTD or XML Schema have fewer levels of nesting.
<i>XmlVar</i>	The name of the variable to which to store the generated XML document [OUT]
<i>XsdVar</i>	The name of the variable in which to store the XML Schema if one is generated along with the XML document.

---

#### XMLExecute Parameters (Continued)

## XMLSetOptions

### Syntax

```
XMLSetOptions("options")
```

### Description

Use this function in UniVerse Basic programs to set the encoding parameter and other XML options in the current UniVerse session. The settings specified in this API override the settings in the system-level and account-level xmlconfig files and in TCL commands during the current session.

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>options</i>	A string in the format of space-delimited key/value pairs. [IN]  The XML options are the same as those in the xmlconfig file and accept the same values. Keys and values are case-insensitive.  For a complete list of valid UniVerse XML options and settings, see <a href="#">xmlconfig parameters</a> .  The XMLSetOptions function also accepts three special strings as the <i>options</i> parameter. <ul style="list-style-type: none"><li>■ <b>defaults</b> – Sets all XML options to their default settings in the current session.</li><li>■ <b>reload</b> – Reloads the current system-level and account-level xmlconfig files, since they may have changed after you started your UniVerse session.</li><li>■ <b>reset</b> – Resets XML options to the original settings that were loaded when you started the UniVerse session.</li></ul> A special string must be entered as the only option.

#### XMLSetOptions Parameters

## **Examples**

The following example shows the format for entering the XML options in this API.

```
XMLSetOptions("encoding=iso-8859-1 newline=CR-LF")
```

The next example shows the format for entering a special string as the *options* parameter:

```
XMLSetOptions("defaults")
```

## **Return codes**

The return code indicates success or failure. The following table describes each return code.

Return Code	Description
XML.SUCCESS	Function completed successfully.
XML.ERROR	When the return status is XML.ERROR, XMLGetError() returns one of the following error codes: <ul style="list-style-type: none"><li>■ 38 Invalid XML config key: 'key_name'</li><li>■ 39 Invalid XML config value: 'value_string'</li><li>■ 40 Invalid XML config format: 'name_value_string'41     Invalid XML config key 'key_name' at line line_number     in xmlconfig file(xmlconfig_filename)</li><li>■ 42 Invalid XML config value 'value_string' at line     line_number in xmlconfig file(xmlconfig_filename)</li><li>■ 43 Invalid XML config format 'name_value_string' at     line line_number in xmlconfig file(xmlconfig_filename)</li></ul>

*Note:* When the return code is XML.ERROR, none of the XML parameters are changed.

### **XMLSetOptions (defaults) Return Codes**

The next example shows the format for entering a special string as the *options* parameter:

```
XMLSetOptions("reload")
```

## **XMLGetOptions**

### **Syntax**

**XMLGetOptions(*outOptions*[, *delimiterString*])**

### **Description**

Use this function in UniVerse Basic programs to return the values for encoding and other XML options in effect in the current UniVerse session.

### **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>outOptions</i>	A variable used to retrieve the XML options key/value pairs in effect in the current UniVerse session. [OUT]
<i>delimiterString</i>	Optional. Specifies the string to be used to separate the key/value pairs returned by the command. By default, <i>delimiterString</i> is a space. [IN]

#### **XMLGetOptions Parameters**

### **Examples**

The following example shows the format for entering *delimiterString* as the string used to separate the key/value pairs returned by the function. Key/value pairs can be separated by a space or by any string, such as <>, as shown in this example:

```
(xmlOptions, "<>")  
encoding=default<>in-encoding=UTF-8<>version=1.1
```

If you enter the **XMLGetOptions** function with no *delimiterString*, the key/value pairs are separated by a space, as shown in the next example:

```
XMLGetOptions(xmlOptions)  
encoding=default in-encoding=UTF-8 version=1.1
```

For a complete list of the standard UniVerse XML options and values returned by this function, see [xmlconfig parameters](#).

### ***Return codes***

The return code indicates the status on completion of this function. The following table describes each return code.

Return Code	Description
XML.SUCCESS	Function completed successfully.

**XMLGetOptions Return Codes**

## **XMLGetOptionValue**

### ***Syntax***

**XMLGetOptionValue**(*optionName*, *outOptionValue*)

### ***Description***

Use this function in UniVerse Basic programs to return the value of encoding or any other XML option in effect in the current UniVerse session.

### ***Parameters***

The following table describes each parameter of the syntax.

Parameter	Description
<i>optionName</i>	The name of the XML option for which you want to retrieve the current value. [IN]  The XML options are the same as those in the <a href="#">xmlconfig</a> file. For a complete list of valid UniVerse XML options, see <a href="#">xmlconfig parameters</a> .
<i>outOptionValue</i>	A variable used to retrieve the value of the specified XML option in the current UniVerse session. [OUT]

### **XMLGetOptionValue Parameters**

## **Example**

The following example shows the format for entering the *optionName* and *outOptionValue* variables:

```
XMLGetOptionValue("encoding", xmlOptionValue)
```

This function returns the value of the encoding option in the second argument, *xmlOptionValue*.

## **Return codes**

The return code indicates success or failure. The following table describes each return code.

Return Code	Description
XML.SUCCESS	Function completed successfully.
XML.ERROR	When the return status is XML.ERROR, XMLGetError() returns the following error code: <ul style="list-style-type: none"><li>■ 38 Invalid XML config key: 'key_name'</li></ul>

### **XMLGETOPTIONVALUE Return Codes**

The following XML APIs accept additional parameters for encoding and other XML options, or other new parameters:

- [XDOMCreateRoot](#)
- [XDOMTransform](#)
- [XDOMValidate](#)
- [XDOMValidateDom](#)
- [XDOMWrite](#)
- [XMAPToXMLDoc](#)

## XDOMCreateRoot

### Syntax

**XDOMCreateRoot**(*domHandle*[, *xmlOptions*])

### Description

XDOMCreateRoot creates a new DOM structure with root only. You can use the result handle in other functions where a DOM handle or node handle is needed. The default declaration for the XML document created by this function is as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
```

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>domHandle</i>	Handle to the opened DOM structure. [OUT]
<i>xmlOptions</i>	A string specifying the key/value pairs for the encoding, standalone, and version options to be declared in the XML document created by this function. [IN]  The string can be entered in either of the following formats:  <code>"version=1.0 encoding=ISO-8859-1 standalone=yes"</code> or <code>'version="1.0" encoding="iso-8859-1" standalone="no'"</code>  The encoding, standalone, and version options are the same as those in the <code>xmlconfig</code> file and accept the same values. For details, see the <a href="#">xmlconfig parameters</a> section. Keys and values are case-insensitive.  Valid encoding settings can be found at <a href="http://www.iana.org/assignments/character-sets">http://www.iana.org/assignments/character-sets</a>

### XDOMCreateRoot Parameters

### *Return codes*

The return code indicates success or failure. The following table describes each return code.

<b>Return Code</b>	<b>Description</b>
XML.SUCCESS	Function completed successfully.
XML.ERROR	An error occurred. If an error is encountered in <i>xmlOptions</i> , XMLGetError() returns one of the following error codes: <ul style="list-style-type: none"><li>■ 38 Invalid XML config key: 'key_name'</li><li>■ 39 Invalid XML config value: 'value_string'</li><li>■ 40 Invalid XML config format: 'name_value_string'</li></ul>

#### **XDOMCreateRoot Return Codes**

## XDOMTransform

### Syntax

**XDOMTransform**(*domHandle*, *styleSheet*, *ssLocation*, *outHandle*[, *outFormat*])

### Description

XDOMTransform transforms the input DOM structure using the style sheet specified by *styleSheet* to output the DOM structure, file, or string.

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>domHandle</i>	Handle to the DOM structure. [IN]
<i>styleSheet</i>	Handle to the context. [IN]
<i>ssLocation</i>	A flag to specify whether <i>styleSheet</i> contains style sheet itself, or is just the style sheet file name. Value values are: XML.FROM.FILE XML.FROM.STRING [IN]
<i>outHandle</i>	Handle to the resulting DOM structure, file, or string. [OUT]
<i>outFormat</i>	Specifies one of the following values as the output format for the DOM structure: <ul style="list-style-type: none"><li>■ XML.TO.DOM – Transforms the input DOM structure using the style sheet specified by <i>styleSheet</i>, and outputs the resulting document into a DOM structure.</li><li>■ XML.TO.FILE – Transforms the input DOM structure using the style sheet specified by <i>styleSheet</i>, and outputs the resulting document into a file.</li><li>■ XML.TO.STRING – Transforms the input DOM structure using the style sheet specified by <i>styleSheet</i>, and outputs the resulting document into a string.</li></ul>

### XDOMTransform Parameters

### *Return codes*

The return code indicates success or failure. The following table describes each return code.

<b>Return Code</b>	<b>Description</b>
XML.SUCCESS	The function completed successfully.
XML.ERROR	An error occurred.
XML.INVALID.HANDLE	An invalid DOM handle was returned to the function.

**XDOMTransform Return Codes**

## **XDOMValidate**

### *Syntax*

**XDOMValidate**(*xmlDocument*, *docLocation*, *noNsSchema*,  
*noNsSchemaLocation*[, *NsSchemas*])

### *Description*

XDOMValidate validates the DOM document using an external no-namespace schema specified by *noNsSchema* and, optionally, external namespace schemas specified by *NsSchemas*.

### *Parameters*

The following table describes each parameter of the syntax.

Parameter	Description
<i>xmlDocument</i>	The name of the XML document. [IN]
<i>docLocation</i>	A flag to specify whether <i>xmlDocument</i> is the document itself, or the document file name. Valid values are: XML.FROM.FILE XML.FROM.STRING XML.FROM.DOM [IN]
<i>noNsSchema</i>	The external no-namespace schema file. [IN]
<i>noNsSchemaLocation</i>	A flag to specify whether <i>noNsSchema</i> is the schema itself, or the schema file name. Valid values are: XML.FROM.FILE (default) XML.FROM.STRING [IN]
<i>NsSchemas</i>	The external namespace schema files. [IN]

### **XDOMValidate Parameters**

### *Return codes*

The return code indicates success or failure. The following table describes each return code.

<b>Return Code</b>	<b>Description</b>
XML.SUCCESS	Function completed successfully.
XML.ERROR	An error occurred.
XML.INVALID.HANDLE	An invalid DOM handle was passed to the function.

**XDOMValidate Return Codes**

## **XDOMValidateDom**

### *Syntax*

**XDOMValidateDom**(*domHandle*, *noNsSchema*, *noNsSchemaLocation*[,  
*NsSchemas*])

### *Description*

XDOMValidateDom validates the DOM document using an external no-namespace schema specified by *noNsSchema* and, optionally, external namespace schemas specified by *NsSchemas*.

### *Parameters*

The following table describes each parameter of the syntax.

Parameter	Description
<i>domHandle</i>	Handle to the DOM structure. [IN]
<i>noNsSchema</i>	The external no-namespace schema file. [IN]
<i>noNsSchemaLocation</i>	A flag to specify whether <i>noNsSchema</i> is the schema itself, or the schema file name. Valid values are: XML.FROM.FILE (default) XML.FROM.STRING [IN]
<i>NsSchemas</i>	The external namespace schema files. [IN]

### **XDOMValidateDom Parameters**

### *Return codes*

The return code indicates success or failure. The following table describes each return code.

Return Code	Description
XML.SUCCESS	Function completed successfully.
XML.ERROR	An error occurred.
XML.INVALID.HANDLE	An invalid DOM handle was passed to the function.

**XDOMValidateDom Return Codes**

## **XDOMWrite**

### *Syntax*

**XDOMWrite**(*domHandle*, *xmlDocument*, *docLocation*[, *xmlOptions*])

### *Description*

XDOMWrite writes the DOM structure to *xmlDocument*. *xmlDocument* can be a string or a file, depending on the value of the *docLocation* flag.

### *Parameters*

The following table describes each parameter of the syntax.

Parameter	Description
<i>domHandle</i>	Handle to the opened DOM structure. [IN]
<i>xmlDocument</i>	The XML document [OUT]
<i>docLocation</i>	A flag to specify whether <i>xmlDocument</i> is an output string that should hold the XML document, or is a file to which the XML document should be written. Valid values are: XML.TO.FILE XML.TO.STRING [IN]

### **XDOMWrite Parameters**

---

Parameter	Description
<i>xmlOptions</i>	<p>A string specifying the key / value pairs of the XML options to be used in the XML document written by this function. [IN]</p> <p>The string can be entered in either of the following formats:</p> <pre>"encoding=ISO-8859-1 standalone=yes newline=CR-LF" or 'encoding="iso-8859-1" standalone="no'"</pre> <p>The XML options are the same as those in the <code>xmlconfig</code> file and accept the same values. Keys and values are case-insensitive.</p> <p>For a complete list of valid UniVerse XML options and settings, see <a href="#">xmlconfig parameters</a>.</p> <p>Valid encoding settings can be found at:</p> <p style="text-align: center;"><a href="http://www.iana.org/assignments/character-sets">http://www.iana.org/assignments/character-sets</a></p>

---

#### XDOMWrite Parameters (Continued)

#### *Return codes*

The return code indicates success or failure. The following table describes each return code.

---

Return Code	Description
XML.SUCCESS	Function completed successfully.
XML.ERROR	<p>An error occurred.</p> <p>If an error is encountered in <i>xmlOptions</i>, XMLGetError() returns one of the following error codes:</p> <ul style="list-style-type: none"> <li>■ 38 Invalid XML config key: 'key_name'</li> <li>■ 39 Invalid XML config value: 'value_string'</li> <li>■ 40 Invalid XML config format: 'name_value_string'</li> </ul>
XML.INVALID.HANDLE	Invalid DOM handle passed to the function.

---

#### XDOMWrite Return Codes

## **XMAPToXMLDoc**

### *Syntax*

**XMAPToXMLDoc(XMAPhandle, xmlfile, doc\_flag[, xmlOptions])**

### *Description*

The XMAPToXMLDoc function generates an XML document from the data in the U2XMAP dataset using the mapping rules you define. The XML document can be either an XML DOM handle or an XML document. UniVerse writes the data to a file or a UniVerse Basic variable.

### *Parameters*

The following table describes each parameter of the syntax.

Parameter	Description
<i>XMAPhandle</i>	The handle to the U2XMAP dataset.
<i>xmlfile</i>	The name of the XML file, or the name of a UniVerse Basic variable to hold the XML document.
<i>doc_flag</i>	Indicates where to write the XML document. Valid values are: <ul style="list-style-type: none"><li>■ XML.TO.DOM - Writes the XML document to an XML DOM handle.</li><li>■ XML.TO.FILE - Writes the XML document to a file.</li><li>■ XML.TO.STRING - Writes the XML document to a UniVerse Basic variable.</li></ul>

### **XMAPToXMLDoc Parameters**

---

Parameter	Description
<i>xmlOptions</i>	<p>A string specifying the key/value pairs of the XML options to be used in the XML document generated by this function. [IN]</p> <p>The string can be entered in either of the following formats:</p> <pre>"encoding=ISO-8859-1 standalone=yes newline=CR-LF"</pre> <p>or</p> <pre>'encoding="iso-8859-1" standalone="no"'</pre> <p>The XML options are the same as those in the <code>xmlconfig</code> file and accept the same values. Keys and values are case-insensitive.</p> <p>For a complete list of valid UniVerse XML options and settings, see <a href="#">xmlconfig parameters</a>.</p> <p>Valid encoding settings can be found at:</p> <p><a href="http://www.iana.org/assignments/character-sets">http://www.iana.org/assignments/character-sets</a></p>

---

#### XMAPToXMLDoc Parameters (Continued)

#### *Return codes*

The return code indicates success or failure. The following table describes each return code.

---

Return Code	Description
XML_SUCCESS	The XML document was opened successfully.
XML_ERROR	An error occurred opening the XML document. If an error is encountered in <i>xmlOptions</i> , XMLGetError() returns one of the following error codes: <ul style="list-style-type: none"><li>■ 38 Invalid XML config key: 'key_name'</li><li>■ 39 Invalid XML config value: 'value_string'</li><li>■ 40 Invalid XML config format: 'name_value_string'</li></ul>
XML_INVALID_HANDLE	The XMAP dataset was invalid.

---

#### XMAPToXMLDoc Return Codes

## Existing APIs affected by XML options

One or more XML options set in the xmlconfig files, the XMLSETOPTIONS command, or the XMLSetOptions() API are used by the following APIs:

- [DBTOXML](#)
- [XDOMAddChild](#)
- [XDOMAppend](#)
- [XDOMCreateNode](#)
- [XDOMEvaluate](#)
- [XDOMGetAttribute](#)
- [XDOMGetnodeName](#)
- [XDOMGetnodeValue](#)
- [XDOMInsert](#)
- [XDOMLocate](#)
- [XDOMRemove](#)
- [XDOMReplace](#)
- [XDOMSetnodeValue](#)
- [XMAPAppendRec](#)
- [XMAPOpen](#)
- [XMAPReadNext](#)

## **DBTOXML**

### *Syntax*

**DBTOXML**(*xml\_document*, *doc\_location*, *u2xmap\_file*, *u2xmap\_location*,  
*condition*, *status*)

### *Description*

Creates an XML document from the UniVerse database.

### *Parameters*

The following table describes each parameter of the syntax.

Parameter	Description
<i>xml_document</i>	The name of the XML document to create.
<i>doc_flag</i>	A flag defining the type of <i>xml_document</i> . Valid values are: <ul style="list-style-type: none"><li>■ XML.FROM.FILE – <i>xml_document</i> is a file name.</li><li>■ XML.FROM.STRING – <i>xml_document</i> is the name of a variable containing the XML document.</li></ul>
<i>u2xmap_file</i>	The name of the U2XMAP file to use to produce the XML document.
<i>u2xmap_location</i>	The location of the U2XMAP file. <ul style="list-style-type: none"><li>■ XML.FROM.FILE – <i>u2xmap_file</i> is a file name.</li><li>■ XML.FROM.STRING – <i>u2xmap_file</i> is the name of a variable containing the mapping rules.</li></ul>
<i>condition</i>	A query condition for selecting data from the UniVerse file, for example, WHERE SCHOOL = "CO002"
Status	XML.SUCCESS or XML.FAILURE.

### **DBTOXML Parameters**



**Note:** The XML options set previously at the session level through the XMLSETOPTIONS command or through the XMLSetOptions() API are used when you run the DBTOXML API in the current UniVerse session.

## XDOMAddChild

### Syntax

**XDOMAddChild**(*xmlHandle*, *xpathString*, *nsMap*, *nodeHandle*, *dupFlag*)

### Description

Finds the *xpathString* in the context *xmlHandle* in the DOM structure, and inserts a node as the last child of the found node. If the inserted node type is XDOM.ATTR.NODE, this node is inserted as an attribute.

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>xmlHandle</i>	The handle to the context. [IN]
<i>xpathString</i>	Relative or absolute xpath string. [IN]  The <i>xpathString</i> parameter uses the <i>in-encoding</i> parameter set in the system-level or account-level xmlconfig file, the XMLSETOPTIONS command, or the XMLSetOptions() API.
<i>nsMap</i>	The map of namespaces that resolves the prefixes in the xpath string.  Format is "xmlns=default_url xmlns:prefix1=prefix1_url xmlns:prefix2=prefix2_url"  For example: "xmlns=http://myproject.mycompany.com xmlns:a_prefix=a.mycompany.com" [IN]  The <i>nsMap</i> parameter uses the <i>in-encoding</i> parameter set in the system-level or account-level xmlconfig file, the XMLSETOPTIONS command, or the XMLSetOptions() API.
<i>nodeHandle</i>	Handle to a DOM subtree. If <i>nodeHandle</i> points to a DOM document, all of its children are inserted, in the same order. [IN]

### XDOMAddChild Parameters

---

<b>Parameter</b>	<b>Description</b>
<i>dupFlag</i>	XDOM.DUP: Clones nodeHandle, and inserts the duplicate node. XDOM.NODUP: Inserts the original node. The subtree is also removed from its original location. [IN]

---

**XDOMAddChild Parameters (Continued)**

*Return codes*

The return code indicates success or failure. The following table describes each return code.

---

<b>Return Code</b>	<b>Description</b>
XML.SUCCESS	The function completed successfully.
XML.ERROR	An error occurred.
XML.INVALID.HANDLE	An invalid DOM handle was returned to the function.

---

**XDOMAddChild Return Codes**

## XDOMAppend

### Syntax

**XDOMAppend**(*xmlHandle*, *xpathString*, *nsMap*, *nodeHandle*, *dupFlag*)

### Description

Finds the *xpathString* in the context *xmlHandle* in the DOM structure, and inserts *nodeHandle* into the DOM structure as the next sibling of the found node. If the inserted node type is XDOM.ATTR.NODE, this node is inserted as an attribute.

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>xmlHandle</i>	The handle to the context. [IN]
<i>xpathString</i>	Relative or absolute XPath string. [IN] The <i>xpathString</i> parameter uses the <i>in-encoding</i> parameter set in the system-level or account-level xmlconfig file, the XMLSETOPTIONS command, or the XMLSetOptions() API.
<i>nsMap</i>	The map of namespaces that resolves the prefixes in the <i>xpathString</i> . Format is “ <code>xmlns=default_url xmlns:prefix1=prefix1_url xmlns:prefix2=prefix2_url</code> ” For example: “ <code>xmlns=http://myproject.mycompany.com xmlns:a_prefix=a.mycompany.com</code> ” [IN] The <i>nsMap</i> parameter uses the <i>in-encoding</i> parameter set in the system-level or account-level xmlconfig file, the XMLSETOPTIONS command, or the XMLSetOptions() API.
<i>nodeHandle</i>	Handle to a DOM subtree. If <i>nodeHandle</i> points to a DOM document, all of its children are inserted, in the same order. [IN]

### XDOMAppend Parameters

---

Parameter	Description
<i>dupFlag</i>	XDOM.DUP: Clones <i>nodeHandle</i> , and inserts the duplicate node. XDOM.NODUP: Inserts the original node. The subtree is also removed from its original location. [IN]

---

#### **XDOMAppend Parameters (Continued)**

#### *Return codes*

The return code indicates success or failure. The following table describes each return code.

---

Return Code	Description
XML.SUCCESS	The function completed successfully.
XML.ERROR	An error occurred.
XML.INVALID.HANDLE	An invalid DOM handle was returned to the function.

---

#### **XDOMAppend Return Codes**

## XDOMCreateNode

### Syntax

**XDOMCreateNode**(*xmlHandle*, *nodeName*, *nodeValue*, *nodeType*, *nodeHandle*)

### Description

XDOMCreateNode creates a new node in the DOM structure.

### Parameters

The following table describes each parameter of the syntax.

Parameters	Description
<i>xmlHandle</i>	A handle to the DOM structure. This handle acts as the context when resolving the namespace_uri from the prefix or resolving the prefix from the namespace_uri. [IN]
<i>nodeName</i>	The name of the node to be created. [IN] The name can be in any of the following formats: <ul style="list-style-type: none"><li>■ Local_name</li><li>■ prefix: local_name:namespace_uri</li><li>■ prefix:local_name</li><li>■ :local_name:namespace_uri</li></ul> The <i>nodeName</i> parameter uses the <i>in-encoding</i> parameter set in the system-level or account-level xmlconfig file, the XMLSETOPTIONS command, or the XMLSetOptions() API.
<i>nodeValue</i>	The string to hold the node value. [IN] The <i>nodeValue</i> parameter uses the <i>in-encoding</i> parameter set in the system-level or account-level xmlconfig file, the XMLSETOPTIONS command, or the XMLSetOptions() API.

### XDOMCreateNode Parameters

---

Parameters	Description
<i>nodeType</i>	The type of the node to be created. Valid values are: XDOM.ELEMENT.NODE XDOM.ATTR.NODE XDOM.TEXT.NODE XDOM.CDATA.NODE XDOM.ENTITY.REF.NODE XDOM.ENTITY.NODE XDOM.PROCINST.NODE XDOM.COMMENT.NODE XDOM.DOC.NODE XDOM.DOC.TYPE.NODE XDOM.DOC.FRAG.NODE XDEOM.NOTATION.NODE XDOM.XML.DECL.NODE  [IN]
<i>nodeHandle</i>	A handle to the node to be created in the DOM structure. [IN]

---

#### **XDOMCreateNode Parameters (Continued)**

#### *Return codes*

The return code indicates success or failure. The following table describes each return code.

---

Return Code	Description
XML.SUCCESS	Function completed successfully.
XML.ERROR	An error occurred.

---

#### **XDOMCreateNode Return Codes**

## **XDOMEvaluate**

### *Syntax*

**XDOMEvaluate**(*xmlHandle*, *xpathString*, *nsMap*, *aValue*)

### *Description*

XDOMEvaluate returns the value of *xpathString* in the context *xmlHandle* in the DOM structure.

### *Parameters*

The following table describes each parameter of the syntax.

Parameter	Description
<i>xmlHandle</i>	Handle to the context. [IN]
<i>xpathString</i>	The relative or absolute XPath string. [IN]  The <i>xpathString</i> parameter uses the <i>in-encoding</i> parameter set in the system-level or account-level xmlconfig file, the XMLSETOPTIONS command, or the XMLSetOptions() API.
<i>nsMap</i>	The map of namespaces that resolves the prefixes in the <i>xpathString</i> .  Format is “xmlns=default_url xmlns:prefix1=prefix1_url xmlns:prefix2=prefix2_url”  For example: “xmlns=http://myproject.mycompany.com xmlns:a_prefix=a.mycompany.com” [IN]  The <i>nsMap</i> parameter uses the <i>in-encoding</i> parameter set in the system-level or account-level xmlconfig file, the XMLSETOPTIONS command, or the XMLSetOptions() API.

### **XDOMEvaluate Parameters**

---

Parameter	Description
<i>aValue</i>	The value of <i>xpathString</i> . [OUT] The <i>aValue</i> parameter uses the <i>out-encoding</i> parameter set in the system-level or account-level xmlconfig file, the XMLSETOPTIONS command, or the XMLSetOptions() API.

---

#### XDOMEvaluate Parameters (Continued)

#### *Return codes*

The return code indicates success or failure. The following table describes each return code.

---

Return Code	Description
XML.SUCCESS	The function completed successfully.
XML.ERROR	An error occurred.
XML.INVALID.HANDLE	An invalid DOM handle was returned to the function.

---

#### XDOMEvaluate Return Codes

## XDOMGetAttribute

### Syntax

**XDOMGetAttribute**(*nodeHandle*, *attrName*, *nodeHandle*)

### Description

XDOMGetAttribute gets the node's attribute node, whose attribute name is *attrName*.

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>nodeHandle</i>	Handle to the DOM node. [IN]
<i>attrName</i>	Attribute name. [IN]  The <i>attrName</i> parameter uses the <i>in-encoding</i> parameter set in the system-level or account-level xmlconfig file, the XMLSETOPTIONS command, or the XMLSetOptions() API.
<i>nodeHandle</i>	Handle to the found attribute node. [OUT]

### XDOMGetAttribute Parameters

### Return codes

The return code indicates success or failure. The following table describes each return code.

Return Code	Description
XML.SUCCESS	The function completed successfully.
XML.ERROR	An error occurred.
XML.INVALID.HANDLE	An invalid DOM handle was returned to the function.

### XDOMGetAttribute Return Codes

## **XDOMGetNodeName**

### *Syntax*

**XDOMGetNodeName**(*nodeHandle*, *nodeName*)

### *Description*

XDOMGetNodeName returns the node name.

### *Parameters*

The following table describes each parameter of the syntax.

Parameter	Description
<i>nodeHandle</i>	Handle to the DOM node. [IN]
<i>nodeName</i>	String to store the node name. [OUT] The <i>nodeName</i> parameter uses the <i>out-encoding</i> parameter set in the system-level or account-level xmlconfig file, the XMLSETOPTIONS command, or the XMLSetOptions() API.

### **XDOMGetNodeName Parameters**

### *Return codes*

The return code indicates success or failure. The following table describes each return code.

Return Code	Description
XML.SUCCESS	The function completed successfully.
XML.ERROR	An error occurred.
XML.INVALID.HANDLE	An invalid DOM handle was returned to the function.

### **XDOMGetNodeName Return Codes**

## XDOMGetNodeValue

### Syntax

**XDOMGetNodeValue**(*nodeHandle*, *nodeValue*)

### Description

XDOMGetNodeValue gets the node value.

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>nodeHandle</i>	The handle to the DOM node. [IN]
<i>nodeValue</i>	The string to hold the node value. [OUT] The <i>nodeValue</i> parameter uses the <i>out-encoding</i> parameter set in the system-level or account-level xmlconfig file, the XMLSETOPTIONS command, or the XMLSetOptions() API.

### XDOMGetNodeValue Parameters

### Return codes

The return code indicates success or failure. The following table describes each return code.

Return Code	Description
XML.SUCCESS	The function completed successfully.
XML.ERROR	An error occurred.
XML.INVALID.HANDLE	An invalid DOM handle was returned to the function.

### XDOMGetNodeValue Return Codes

## XDOMInsert

### Syntax

**XDOMInsert** (*xmlHandle*, *xpathString*, *nsMap*, *nodeHandle*, *dupFlag*)

### Description

XDOMInsert finds the *xpathString* in the context *xmlHandle* in the DOM structure, and inserts *nodeHandle* into the DOM structure as the previous sibling of the found node. If the inserted node type is XDOM.ATTR.NODE, this node is inserted as an attribute.

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>xmlHandle</i>	The handle to the context. [IN]
<i>xpathString</i>	Relative or absolute XPath string. [IN]  The <i>xpathString</i> parameter uses the <i>in-encoding</i> parameter set in the system-level or account-level xmlconfig file, the XMLSETOPTIONS command, or the XMLSetOptions() API.
<i>nsMap</i>	The map of namespaces that resolves the prefixes in the <i>xpathString</i> .  Format is “xmlns=default_url xmlns:prefix1=prefix1_url xmlns:prefix2=prefix2_url”  For example:  “xmlns=http://myproject.mycompany.com xmlns:a_prefix=a.mycompany.com” [IN]  The <i>nsMap</i> parameter uses the <i>in-encoding</i> parameter set in the system-level or account-level xmlconfig file, the XMLSETOPTIONS command, or the XMLSetOptions() API.
<i>nodeHandle</i>	The handle to a DOM subtree. If <i>nodeHandle</i> points to a DOM document, all of its children are inserted, in the same order. [IN]

### XDOMInsert Parameters

---

<b>Parameter</b>	<b>Description</b>
<i>dupFlag</i>	XDOM.DUP: Clones nodeHandle, and inserts the duplicate node. XDOM.NODUP: Inserts the original node. The subtree is also removed from its original location.

---

**XDOMInsert Parameters (Continued)**

*Return codes*

The return code indicates success or failure. The following table describes each return code.

---

<b>Return Code</b>	<b>Description</b>
XML.SUCCESS	The function completed successfully.
XML.ERROR	An error occurred.
XML.INVALID.HANDLE	An invalid DOM handle was returned to the function.

---

**XDOMInsert Return Codes**

## XDOMLocate

### Syntax

**XDOMLocate**(*xmlHandle*, *xpathString*, *nsMap*, *nodeHandle*)

### Description

XDOMLocate finds a starting point for relative XPath searching in context *xmlHandle* in the DOM structure. The *xpathString* should specify only one node; otherwise, this function will return an error.

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>xmlHandle</i>	A handle to the DOM structure. [IN]
<i>xpathString</i>	A string to specify the starting point. [IN] The <i>xpathString</i> parameter uses the <i>in-encoding</i> parameter set in the system-level or account-level xmlconfig file, the XMLSETOPTIONS command, or the XMLSetOptions() API.
<i>nsMAP</i>	The map of namespaces that resolves the prefixes in the <i>xpathString</i> . The format is: “ <i>xmlns=default_url xmlns:prefix1=prefix1_url xmlns:prefix2=prefix2_url</i> ” For example: “ <i>xmlns=”http://myproject.mycompany.com xmlns:a_prefix=a.mycompany.com</i> ” [IN] The <i>nsMap</i> parameter uses the <i>in-encoding</i> parameter set in the system-level or account-level xmlconfig file, the XMLSETOPTIONS command, or the XMLSetOptions() API.
<i>noteHandle</i>	Handle to the found node. [OUT]

### XDOMLocate Parameters

### *Return codes*

The return code indicates success or failure. The following table describes each return code.

<b>Return Code</b>	<b>Description</b>
XML.SUCCESS	Function completed successfully.
XML.ERROR	An error occurred.
XML.INVALID.HANDLE	An invalid handle was returned to the function.

#### **XDOMLocate Return Codes**



*Note:* In this document, `xmlHandle` is a generic type, it can be `domHandle` or `nodeHandle`. `DomHandle` stands for a whole document, while `nodeHandle` stands for a subtree. `DomHandle` is also a `nodeHandle`.

## **XDOMRemove**

### *Syntax*

**XDOMRemove**(*xmlHandle*, *xpathString*, *nsMap*, *attrName*, *nodeHandle*)

### *Description*

XDOMRemove finds the *xpathString* in the context *xmlHandle* in the DOM structure, and then removes the found node or its attribute with name *attrName*.

### *Parameters*

The following table describes each parameter of the syntax.

Parameter	Description
<i>xmlHandle</i>	Handle to the context. [IN]
<i>xpathString</i>	Relative or absolute xpath string. [IN] The <i>xpathString</i> parameter uses the <i>in-encoding</i> parameter set in the system-level or account-level xmlconfig file, the XMLSETOPTIONS command, or the XMLSetOptions() API.

### **XDOMRemove Parameters**

---

Parameter	Description
<i>nsMap</i>	<p>The map of namespaces that resolves the prefixes in the <i>xpathString</i>. Format is “<code>xmlns=default_url xmlns:prefix1_url xmlns:prefix2=prefix2_url</code>” For example: “<code>xmlns=http://myproject.mycompany.com xmlns:a_prefix=a.mycompany.com</code>” [IN] The <i>nsMap</i> parameter uses the <i>in-encoding</i> parameter set in the system-level or account-level <code>xmlconfig</code> file, the <code>XMLSETOPTIONS</code> command, or the <code>XMLSetOptions()</code> API.</p>
<i>attrName</i>	<p>The attribute name. [IN] The <i>attrName</i> parameter uses the <i>in-encoding</i> parameter set in the system-level or account-level <code>xmlconfig</code> file, the <code>XMLSETOPTIONS</code> command, or the <code>XMLSetOptions()</code> API.</p>
<i>nodeHandle</i>	The removed node, if <i>nodeHandle</i> is not NULL. [OUT]

---

#### **XDOMRemove Parameters (Continued)**

#### *Return codes*

The return code indicates success or failure. The following table describes each return code.

---

Return Code	Description
<code>XML.SUCCESS</code>	The function completed successfully.
<code>XML.ERROR</code>	An error occurred.
<code>XML.INVALID.HANDLE</code>	An invalid DOM handle was returned to the function.

---

#### **XDOMRemove Return Codes**

## XDOMReplace

### Syntax

**XDOMReplace**(*xmlHandle*, *xpathString*, *nsMap*, *nodeHandle*, *dupFlag*)

### Description

XDOMReplace finds the *xpathString* in the context *xmlHandle* in the DOM structure, and replaces the found node with *nodeHandle*.

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>xmlHandle</i>	The handle to the context. [IN]
<i>xpathString</i>	Relative or absolute XPath string. [IN]  The <i>xpathString</i> parameter uses the <i>in-encoding</i> parameter set in the system-level or account-level xmlconfig file, the XMLSETOPTIONS command, or the XMLSetOptions() API.
<i>nsMap</i>	The map of namespaces that resolves the prefixes in the <i>xpathString</i> .  Format is “xmlns=default_url xmlns:prefix1=prefix1_url xmlns:prefix2=prefix2_url”  For example: “xmlns=http://myproject.mycompany.com xmlns:a_prefix=a.mycompany.com” [IN]  The <i>nsMap</i> parameter uses the <i>in-encoding</i> parameter set in the system-level or account-level xmlconfig file, the XMLSETOPTIONS command, or the XMLSetOptions() API.
<i>nodeHandle</i>	Handle to a DOM subtree. If <i>nodeHandle</i> points to a DOM document, the found node is replaced by all of <i>nodeHandle</i> children, which are inserted in the same order. [IN]

### XDOMReplace Parameters

---

Parameter	Description
<i>dupFlag</i>	XDOM.DUP: Clones nodeHandle, and replaces it with the duplicate node. XDOM.NODUP: Replaces with the original node. The subtree is also removed from its original location. [IN]

---

**XDOMReplace Parameters (Continued)**

*Return codes*

The return code indicates success or failure. The following table describes each return code.

---

Return Code	Description
XML.SUCCESS	The function completed successfully.
XML.ERROR	An error occurred.
XML.INVALID.HANDLE	An invalid DOM handle was returned to the function.

---

**XDOMReplace Return Codes**

## XDOMSetNodeValue

### Syntax

**XDOMSetNodeValue**(*nodeHandle*, *nodeValue*)

### Description

XDOMSetNodeValue sets the node value.

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>nodeHandle</i>	The handle to the DOM node. [IN]
<i>nodeValue</i>	The string to hold the node value. [IN]  The <i>nodeValue</i> parameter uses the <i>in-encoding</i> parameter set in the system-level or account-level xmlconfig file, the XMLSETOPTIONS command, or the XMLSetOptions() API.

### XDOMSetNodeValue Parameters

### Return codes

The return code indicates success or failure. The following table describes each return code.

Return Code	Description
XML.SUCCESS	The function completed successfully.
XML.ERROR	An error occurred.
XML.INVALID.HANDLE	An invalid DOM handle was returned to the function.

### XDOMSetNodeValue Return Codes

## **XMAPAppendRec**

### *Syntax*

**XMAPAppendRec(*XMAPhandle*, *file\_name*, *record*)**

### *Description*

The XMAPAppendRec function formats the specified record from the UniVerse file as a U2XMAP dataset record and appends it to the U2XMAP dataset.

### *Parameters*

The following table describes each parameter of the syntax.

Parameter	Description
<i>XMAPhandle</i>	The handle to the U2XMAP dataset. The <i>XMAPhandle</i> parameter uses the <i>in-encoding</i> parameter set in the system-level or account-level xmlconfig file, the XMLSETOPTIONS command, or the XMLSetOptions() API for the input record value.
<i>file_name</i>	The name of the UniVerse file that is being mapped in the U2 XMAP dataset.
<i>record</i>	The data record formatted according to the dictionary record of the UniVerse file.

### **XMAPAppendRec Parameters**

### *Return codes*

The return code indicates success or failure. The following table describes each return code.

<b>Return Code</b>	<b>Description</b>
XML_SUCCESS	The XML document was opened successfully.
XML_ERROR	An error occurred opening the XML document.
XML_INVALID_HANDLE	The XMAP dataset was invalid.

**XMAPAppendRec Return Codes**

## XMAPOpen

### Syntax

```
XMAPOpen(xml_document, doc_flag, u2xmapping_rules, u2xmap_flag,  
XMAPhandle)
```

### Description

The XMAPOpen function opens an XML document as a U2XMAP data set.

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>xml_document</i>	The name of the XML document.
<i>doc_flag</i>	A flag defining the type of <i>xml_document</i> . Valid values are: <ul style="list-style-type: none"><li>■ XML.FROM.DOM - <i>xml_document</i> is a DOM handle.</li><li>■ XML.FROM.FILE - <i>xml_document</i> is a file name.</li><li>■ XML.FROM.STRING - <i>xml_document</i> is the name of a variable containing the XML document.</li></ul>
<i>u2xmapping_rules</i>	The name of the U2XMAP file, or the UniVerse Basic variable containing the XML to Database mapping rules.
<i>u2xmap_flag</i>	A flag indicating if the mapping file is the U2XMAP file itself or a string located within the UniVerse Basic program. Valid values are: <ul style="list-style-type: none"><li>■ XMAP.FROM.FILE - the mapping rules are contained in a U2XMAP file.</li><li>■ XMAP.FROM.STRING - <i>u2xmap_flag</i> is the name of the variable containing the mapping rules.</li></ul>
<i>XMAPhandle</i>	The handle to the XMAP dataset.  This API registers the current <i>in-encoding</i> and <i>out-encoding</i> parameters in the <i>XMAPhandle</i> . These parameters are used throughout the life of the <i>XMAPhandle</i> .

### XMAPOpen Parameters

### *Return codes*

The return code indicates success or failure. The following table describes each return code.

<b>Return Code</b>	<b>Description</b>
XML_SUCCESS	The XML document was opened successfully.
XML_ERROR	An error occurred opening the XML document.
<b>XMAPOpen Return Codes</b>	

## **XMAPReadNext**

### *Syntax*

**XMAPReadNext(XMAPhandle, file\_name, record)**

### *Description*

The XMAPReadNext function retrieves the next record from the U2XMAP dataset and formats it as a record of the UniVerse file that is being mapped.

### *Parameters*

The following table describes each parameter of the syntax.

Parameter	Description
<i>XMAPhandle</i>	The U2XMAP dataset handle.  The <i>XMAPhandle</i> parameter uses the <i>out-encoding</i> parameter set in the system-level or account-level xmlconfig file, the XMLSETOPTIONS command, or the XMLSetOptions() API for the record value.
<i>file_name</i>	The name of the UniVerse file that is being mapped in the U2XMAP dataset.
<i>record</i>	The data record formatted according to the dictionary record of the file.

### **XMAPReadNext Parameters**

### *Return codes*

The return code indicates success or failure. The following table describes each return code.

<b>Return Code</b>	<b>Description</b>
XML_SUCCESS	The XMAPReadNext was executed successfully.
XML_ERROR	An error occurred in executing XMAPReadNext.
XML_IN- VALID_HANDL E	U2 XMAP dataset handle was invalid.
XML_EOF	The end of the U2XMAP dataset has been reached.

**XMAPReadNext Return Codes**

## UniVerse BASIC example

The following example illustrates a UniVerse Basic program that generates an XML document:

```
CT BP XML2
BP:

$INCLUDE UNIVERSE.INCLUDE XML.H
*
* Here we test different Options for HIDEMS, and also ELEMENT mode

CMD = "LIST STUDENT LNAME COURSE_NBR COURSE_GRD COURSE_NAME
SEMESTER"
OPTIONS ="XMLMAPPING=student.map"
OPTIONS = OPTIONS:' HIDEMS=1 ELEMENTS'
PRINT OPTIONS
STATUS = XMLExecute(CMD,OPTIONS,XMLVAR,XSDVAR)
IF STATUS = 0 THEN
    STATUS =
    XDOMValidate(XMLVAR,XML.FROM.STRING,XSDVAR,XML.FROM.STRING)
    IF STATUS <> XML.SUCCESS THEN
        STATUS = XMLGetError(code,msg)
        PRINT code,msg
        PRINT "Validate FAILED."
        PRINT XSDVAR
        PRINT XMLVAR
    END ELSE
    PRINT "Options ": OPTIONS
    PRINT "XML output"
    PRINT XMLVAR
    PRINT ""
END
ELSE
    STATUS = XMLGetError(code,msg)
    PRINT code,msg
    PRINT "XMLExecute() failed"
END
```

The next example illustrates the output from the program described in the previous example:

```
>RUN BP XML2
XMLMAPPING=student.map HIDEMS=1 ELEMENTS
Options XMLMAPPING=student.map HIDEMS=1 ELEMENTS
XML output
<?xml version="1.0" encoding="UTF-8"?>
<STUDENT
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:rocketU2="http://www.rocketU2.com"
>
<STUDENT>
    <_ID>221345665</_ID>
    <LNAME>Miller</LNAME>
    <Term>
        <SEMESTER>FA93</SEMESTER>
        <COURSE_GRD>C</COURSE_GRD>
        <COURSE_NAME>Engineering Principles</COURSE_NAME>
        <COURSE_NBR>EG110</COURSE_NBR>
        <COURSE_GRD>B</COURSE_GRD>
        <COURSE_NAME>Calculus- I</COURSE_NAME>
        <COURSE_NBR>MA220</COURSE_NBR>
        <COURSE_GRD>B</COURSE_GRD>
        <COURSE_NAME>Introduction to Psychology</COURSE_NAME>
        <COURSE_NBR>PY100</COURSE_NBR>
    </Term>
    <Term>
        <SEMESTER>SP94</SEMESTER>
        <COURSE_GRD>B</COURSE_GRD>
        <COURSE_NAME>Fluid Mechanics</COURSE_NAME>
        <COURSE_NBR>EG140</COURSE_NBR>
        <COURSE_GRD>B</COURSE_GRD>
        <COURSE_NAME>Circuit Theory</COURSE_NAME>
        <COURSE_NBR>EG240</COURSE_NBR>
        <COURSE_GRD>B</COURSE_GRD>
        <COURSE_NAME>Calculus - II</COURSE_NAME>
        <COURSE_NBR>MA221</COURSE_NBR>
    </Term>
</STUDENT>
...
</MAIN>
```

The following example shows the output if the HIDEMS attribute is set equal to "0":

```
>LIST STUDENT SEMESTER COURSE_NBR TOXML XMLMAPPING student.map
ELEMENTS

<?xml version="1.0" encoding="UTF-8"?>
<STUDENT
    xmlns:rocketU2="http://www.rocketU2.com"
>
<STUDENT>
    <_ID>221345665</_ID>
    <Term>
        <SEMESTER>FA93</SEMESTER>
        <Courses_Taken>
            <COURSE_NBR>EG110</COURSE_NBR>
        </Courses_Taken>
        <Courses_Taken>
            <COURSE_NBR>MA220</COURSE_NBR>
        </Courses_Taken>
        <Courses_Taken>
            <COURSE_NBR>PY100</COURSE_NBR>
        </Courses_Taken>
    </Term>
    <Term>
        <SEMESTER>SP94</SEMESTER>
        <Courses_Taken>
            <COURSE_NBR>EG140</COURSE_NBR>
        </Courses_Taken>
        <Courses_Taken>
            <COURSE_NBR>EG240</COURSE_NBR>
        </Courses_Taken>
        <Courses_Taken>
            <COURSE_NBR>MA221</COURSE_NBR>
        </Courses_Taken>
    </Term>
</STUDENT>
```

The following example shows the output if the HIDEMS attribute is set equal to "1":

```
XMLSETOPTIONS HIDEMS=1

>LIST STUDENT SEMESTER COURSE_NBR TOXML XMLMAPPING student.map
ELEMENTS

<?xml version="1.0" encoding="UTF-8"?>
<STUDENT
  xmlns:rocketU2="http://www.rocketU2.com"
>
<STUDENT>
  <_ID>221345665</_ID>
  <Term>
    <SEMESTER>FA93</SEMESTER>
    <COURSE_NBR>EG110</COURSE_NBR>
    <COURSE_NBR>MA220</COURSE_NBR>
    <COURSE_NBR>PY100</COURSE_NBR>
  </Term>
  <Term>
    <SEMESTER>SP94</SEMESTER>
    <COURSE_NBR>EG140</COURSE_NBR>
    <COURSE_NBR>EG240</COURSE_NBR>
    <COURSE_NBR>MA221</COURSE_NBR>
  </Term>
</STUDENT>
<STUDENT>
  <_ID>414446545</_ID>
  <Term>
    <SEMESTER>FA93</SEMESTER>
    <COURSE_NBR>CS104</COURSE_NBR>
    <COURSE_NBR>MA101</COURSE_NBR>
    <COURSE_NBR>FA100</COURSE_NBR>
  </Term>
  <Term>
    <SEMESTER>SP94</SEMESTER>
    <COURSE_NBR>CS105</COURSE_NBR>
    <COURSE_NBR>MA102</COURSE_NBR>
    <COURSE_NBR>PY100</COURSE_NBR>
  </Term>
</STUDENT>
<STUDENT>
  <_ID>424325656</_ID>
  <Term>
    <SEMESTER>SP94</SEMESTER>
    <COURSE_NBR>PY100</COURSE_NBR>
    <COURSE_NBR>PE100</COURSE_NBR>
  </Term>
</STUDENT>
...
>
```

### ***Collapsemv option***

This option specifies whether to collapse <MV> and </MV> tags, using only one set of these tags for multivalued fields belonging to an association in the generated XML document and in the associated DTD or XMLSchema. This parameter applies only if the XML document is created in element mode.

- 0 - Expand MV tags for multivalued fields.
- 1 - CollapseMV tags for multivalued fields.

### ***Collapsems option***

This attribute specifies whether to collapse <MS> and </MS> tags, using only one set of these tags for multi-subvalued fields belonging to an association in the generated XML document and in the associated DTD or XMLSchema. This parameter applies only if the XML document is created in element mode.

- 0 - Expand MS tags for multi-subvalued fields.
- 1 - Collapse MS tags for multi-subvalued fields.

The following example shows the output if the COLLAPSEMS attribute is set to "1":

```
XMLSETOPTIONS COLLAPSEMS=1

>LIST STUDENT SEMESTER COURSE_NBR TOXML XMLMAPPING student.map
ELEMENTS

<?xml version="1.0" encoding="UTF-8"?>
<STUDENT
  xmlns:rocketU2="http://www.rocketU2.com"
>
<STUDENT>
  <_ID>221345665</_ID>
  <Term>
    <SEMESTER>FA93</SEMESTER>
    <Courses_Taken>
      <COURSE_NBR>EG110</COURSE_NBR>
      <COURSE_NBR>MA220</COURSE_NBR>
      <COURSE_NBR>PY100</COURSE_NBR>
    </Courses_Taken>
  </Term>
  <Term>
    <SEMESTER>SP94</SEMESTER>
    <Courses_Taken>
      <COURSE_NBR>EG140</COURSE_NBR>
      <COURSE_NBR>EG240</COURSE_NBR>
      <COURSE_NBR>MA221</COURSE_NBR>
    </Courses_Taken>
  </Term>
</STUDENT>
<STUDENT>
  <_ID>414446545</_ID>
  <Term>
    <SEMESTER>FA93</SEMESTER>
    <Courses_Taken>
      <COURSE_NBR>CS104</COURSE_NBR>
      <COURSE_NBR>MA101</COURSE_NBR>
      <COURSE_NBR>FA100</COURSE_NBR>
    </Courses_Taken>
  </Term>
  <Term>
    <SEMESTER>SP94</SEMESTER>
    <Courses_Taken>
      <COURSE_NBR>CS105</COURSE_NBR>
      <COURSE_NBR>MA102</COURSE_NBR>
      <COURSE_NBR>PY100</COURSE_NBR>
    </Courses_Taken>
  </Term>
</STUDENT>
...
>
```

# Chapter 5: Receiving XML documents

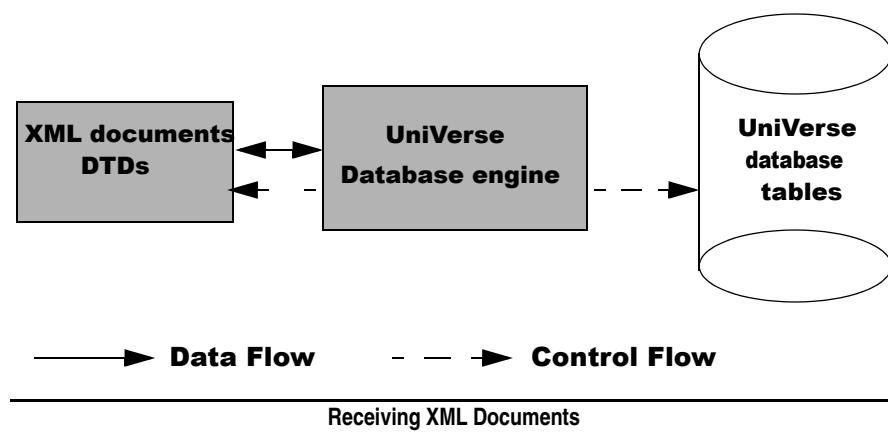
Receiving an XML document through UniVerse BASIC . . . . .	5-2
Defining extraction rules . . . . .	5-2
Defining the XPath . . . . .	5-4
Extracting XML data through UniVerse BASIC . . . . .	5-13
Displaying an XML document through RetrieVe . . . . .	5-18
Displaying an XML document through UniVerse SQL . . . . .	5-22

## Receiving an XML document through UniVerse BASIC

XML documents are text documents, intended to be processed by an application, such as a web browser. UniVerse enables you to receive and create XML documents, and process them through UniVerse BASIC, UniVerse SQL, or RetrievE.

You can receive an XML document, then read the document through UniVerse BASIC, and execute UniVerse BASIC commands against the XML data.

The following example illustrates the UniVerse implementation of receiving XML documents:



### Defining extraction rules

You must define the extraction rules for each XML document you receive. This extraction file defines where to start extracting data from the XML document, how to construct UniVerse data file fields from the data, the name of the data file dictionary to use, and how to treat a missing value.

*Note: The extraction file can reside anywhere. We recommend that it reside in the &XML& file, and have a file extension of .ext.*



## ***Extraction file syntax***

An extraction file has the following format:

```
<?XML version = "1.0"?>
<U2xml-extraction xmlns:U2xml="http://www.rocketU2.com/U2-xml">
    <!-- there must be one and only one <U2xml:extraction> element with
        mode/start/dictionary -->
    <U2xml:extraction
        start="xpath_expression"
        dictionary="dict1 filename ..."
        null="NULL" | "EMPTY"
    />
    <! -- there can be zero or multiple <U2xml:extraction> elements with
        field/path/format -->
    <U2xml:field_extraction
        field="field name"
        path="xpath_expression"
    />
    ...
</U2xml_extraction>
```



***Note:*** UniVerse supports multiple cases of the U2XML\_extraction tag. Valid cases are:

- U2XML\_extraction
- U2xml\_extraction
- u2xml\_extraction

The following tables describes the elements of the extraction file.

Element	Description
XML version	The XML version number.
Namespace	The name of the namespace. A namespace is a unique identifier that links an XML markup element to a specific DTD. They indicate to the processing application, for example, a browser, which DTD you are using.
start	Defines the starting node in the XML file. This specifies where UniVerse should begin extracting data from the XML file.
dictionary	Specifies the UniVerse dictionary of the file name to use when viewing the XML data.
null	Determines how to treat a missing node. If null is set to "NULL," a missing node will be result in the null value in the resulting output. If null is set to EMPTY, a missing node will be replaced with an empty string.
field	The field name.
path	The XPath definition for the field you are extracting.
by_name	This element will map files to the field by name. Otherwise, the files will be mapped by location.

#### Extraction File Elements

## Defining the XPath

In XML, the XPath language describes how to navigate an XML document, and describes a section of the document that needs to be transformed. It also enables you to point to certain part of the document.



*Note:* For the full XPath specification, see <http://www.w3.org/TR/xpath>.

At this release, UniVerse supports the following XPath syntax:

Parameter	Description
/	Node path divider.
.	Current node.
..	Parent node.
@	Attributes
text()	The contents of the element.
xmldata()	The remaining, unparsed, portion of the selected node.
,	Node path divider, and also specifies multivalue or multi-subvalued field.

#### Extraction File Parameters

Consider the following DTD and XML document:

```
<?xml version="1.0"?>
<!DOCTYPE ROOT[
  <!ELEMENT ROOT (STUDENT_record*)>
  <!ELEMENT STUDENT_record ( STUDENT , Last_Name , CGA-MV* )>
  <!ELEMENT STUDENT (#PCDATA) >
  <!ELEMENT Last_Name (#PCDATA) >
  <!ELEMENT CGA-MV ( Term* , CGA-MS* )>
  <!ELEMENT Term (#PCDATA) >
  <!ELEMENT CGA-MS ( Crs_* , GD* , Course_Name* )>
  <!ELEMENT Crs_* (#PCDATA) >
  <!ELEMENT GD (#PCDATA) >
  <!ELEMENT Course_Name (#PCDATA) >
]>

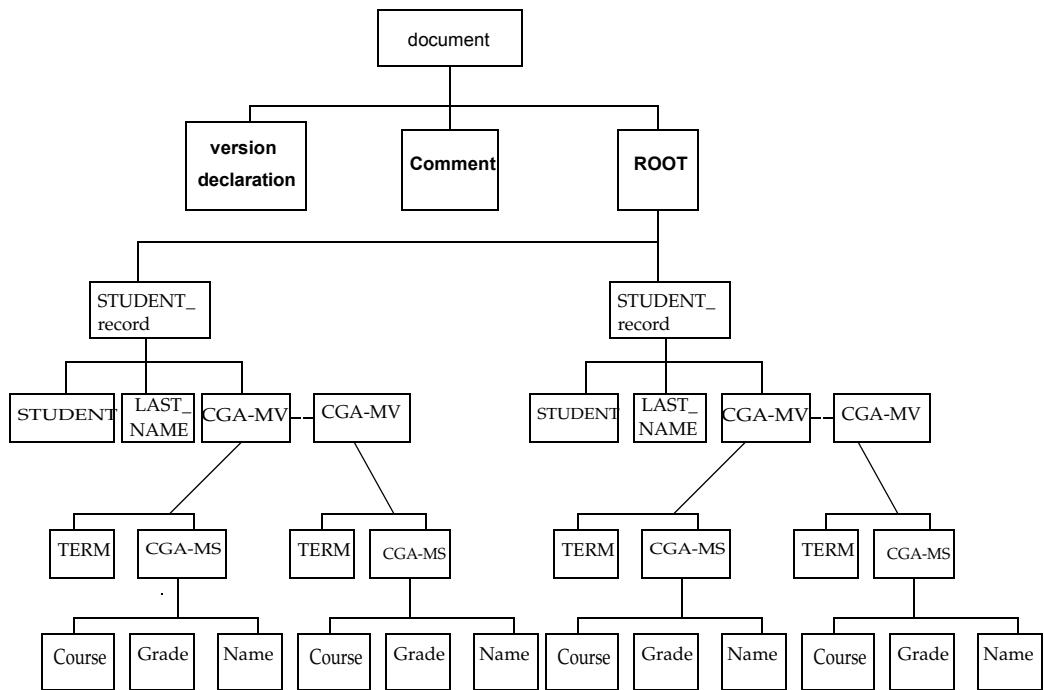
<ROOT>
<STUDENT_record>
  <STUDENT>424-32-5656</STUDENT>
  <Last_Name>Martin</Last_Name>
  <CGA-MV>
    <Term>SP94</Term>
    <CGA-MS>
      <Crs_*>PY100</Crs_*>
      <GD>C</GD>
      <Course_Name>Introduction to Psychology</Course_Name>
    </CGA-MS>
    <CGA-MS>
      <Crs_*>PE100</Crs_*>
      <GD>C</GD>
      <Course_Name>Golf - I </Course_Name>
    </CGA-MS>
  </CGA-MV>
</STUDENT_record>
<STUDENT_record>
  <STUDENT>414-44-6545</STUDENT>
  <Last_Name>Offenbach</Last_Name>
  <CGA-MV>
    <Term>FA93</Term>
    <CGA-MS>
      <Crs_*>CS104</Crs_*>
      <GD>D</GD>
      <Course_Name>Database Design</Course_Name>
    </CGA-MS>
    <CGA-MS>
      <Crs_*>MA101</Crs_*>
      <GD>C</GD>
      <Course_Name>Math Principles </Course_Name>
    </CGA-MS>
    <CGA-MS>
      <Crs_*>FA100</Crs_*>
      <GD>C</GD>
      <Course_Name>Visual Thinking </Course_Name>
    </CGA-MS>
```

```

    </CGA-MV>
<CGA-MV>
    <Term>SP94</Term>
    <CGA-MS>
        <Crs__>CS105</Crs__>
        <GD>B</GD>
        <Course_Name>Database Design</Course_Name>
    <CGA-MS>
        <Crs__>MA102</Crs__>
        <GD>C</GD>
        <Course_Name>Introduction of Psychology</Course_Name>
    </CGA-MS>
</CGA-MV>
<STUDENT_record>
    <STUDENT>221-34-5665</STUDENT>
    <Last_Name>Miller</Last_Name>
    <CGA-MV>
        <Term>FA93</Term>
        <CGA-MS>
            <Crs__>EG110</Crs__>
            <GD>C</GD>
            <Course_Name>Engineering Principles</Course_Name>
        </CGA-MS>
        <CGA-MS>
            <Crs__>PY100</Crs__>
            <GD>B</GD>
            <Course_Name>Introduction to Psychology</Course_Name>
        </CGA-MS>
    </CGA-MV>
        <Term>SP94</Term>
        <CGA-MS>
            <Crs__>EG140</Crs__>
            <GD>B</GD>
            <Course_Name>Fluid Mechanics</Course_Name>
        </CGA-MS>
        <CGA-MS>
            <Crs__>MA221</Crs__>
            <GD>B</GD>
            <Course_Name>Calculus -- II</Course_Name>
        </CGA-MS>
    </CGA-MV>
</STUDENT_record>
</ROOT>

```

This document could be displayed as a tree, as shown in the following example:



In the previous example, each element in the XML document appears in a box. These boxes are called nodes when using XPath terminology. As shown in the example, nodes are related to each other. The relationships in this example are:

- The document node contains the entire XML document.
- The document node contains three children: the version declaration, the comment node, and the ROOT node. These three children are siblings.
- The ROOT node contains two STUDENT nodes, which are children of ROOT, and are siblings of each other.
- The STUDENT node contains three nodes: the ID, NAME, and CGA-MV. These nodes are children of the STUDENT node, and are siblings of each other.
- The CGA-MV node contains TERM nodes and CGA-MS nodes. These nodes are children of the CGA-MV node, and are siblings of each other.
- Finally, the CGA-MS node contains three nodes: the Course, Grade, and Name nodes. These three nodes are children of the CGA-MS node, and are siblings of each other.

When you define the XPath in the extraction file, you must indicate how to treat these different nodes.

### ***Defining the starting location***

The first thing to define in the extraction file is the starting node in the XML document from which you want to begin extracting data. In our example, we want to start at the STUDENT\_record node. You can also define the dictionary file to use when executing RetrievVe LIST statements or UniVerse SQL SELECT statements against the data.

The following example illustrates how to specify the STUDENT\_record node as the starting node, and use the STUDENT dictionary file:

```
<file_extraction start = "ROOT/STUDENT_record" dictionary =
"STUDENT"/>
```

If you want to start the extraction at the CGA-MV node, specify the file extraction node as follows:

```
<file_extraction start = "ROOT/STUDENT_record/CGA-MV" dictionary =
"STUDENT"/>
```

### ***Specifying field equivalents***

Next, you specify the rules for extracting fields from the XML document. In this example, there are six fields to extract (@ID, NAME, TERM, COURSE, GRADE and NAME).

### ***Extracting singlevalued fields***

The following example illustrates how to define the extraction rule for two singlevalued fields:

```
<field_extraction field = "@ID" path = "STUDENT/text()", />
<field_extraction field = "LNAME" path = "Last_Name/text()", />
```

In the first field extraction, the @ID value in the UniVerse record will be extracted from the STUDENT node. The text in the STUDENT node will be the value of @ID.

In the next field extraction rule, the LNAME field will be extracted from the text found in the Last\_Name node in the XML document.

### ***Extracting Multivalued Fields***

To access multivalued data in the XML document, you must specify the location path relative to the start node (full location path).

UniVerse uses the “/” character to specify levels of the XML document. The “/” tells the xmlparser to go to the next level when searching for data.

Use a comma (”,”) to tell the xmlparser where to place marks in the data.

The following example illustrates how to define the path for a multivalued field (SEMESTER) in the XML document:

```
<field_extraction field = "SEMESTER" path = "CGA-MV,Term/text()" />
```

In this example, the value of the SEMESTER field in the UniVerse data file will be the text in the Term node. The “/” in the path value specifies multiple levels in the XML document, as follows:

1. Start at the CGA-MV node in the XML document.
2. From the CGA-MV node, go to the next level, the Term node.
3. Return the text from the Term node as the first value of the SEMESTER field in the UniVerse data file.
4. Search for the next CGA-MV node under the same STUDENT, and extract the text from the Term node belonging to that CGA-MV node, and make it the next multivalue. The comma tells the xmlparser to get the node preceding the command for the next sibling.
5. Continue processing all the CGA-MV nodes belonging to the same parent.

The SEMESTER field will appear in the following manner:

Term<Value mark>Term<Value Mark>...

### *Extracting multisubvalued fields*

As with multivalued fields, UniVerse uses the “//” character to specify levels of the XML document. The “/” tells the xmlparser to go to the next level when searching for data.

Use the comma (“,”) to define where to place marks in the data. You can specify 2 levels of marks, value marks and subvalue marks.

Consider the following example of a field extraction XPath definition:

```
<field_extraction field = "COURSE_NBR" path = "CGA-MV, CGA-MS,  
Course_Name/ text()" />
```

In this case, the resulting data will appear as follows:

<Value Mark>Course\_Name <subvalue mark>Course\_Name<subvalue  
mark>Course\_Name...<Value Mark>...

Suppose the XPath definition contains another level of data, as shown in the next example:

```
<field_extraction field = "COURSE_NBR" path = "CGA-MV/CGA-MS/  
Course_Name/Comment/text()" />
```

You must determine where you want the marks to appear in the resulting data. If you want Comment to represent the multi-subvalue, begin inserting commas after CGA-MS, since the Comment is three levels below CGA-MS.

```
<field_extraction field = "COURSE_NBR" path = "CGA-MV/CGA-MS,  
Course_Name,Comment/text()" />
```

Suppose we add yet another level of data to XPath definition:

```
<field_extraction field = "COURSE_NBR" path = "CGA-MV/CGA-MS,  
Course_Name,Comment,activities/text()" />
```

This is not a valid XPath, since there are more than three levels of XML data. If you want your data to have subvalue marks between Comment and activities, change the XPath definition as follows:

```
<field_extraction field = "COURSE_NBR" path = "CGA-MV/CGA-MS/  
Course_Name,Comment,activities/text()" />
```

The “/” and the “,” characters are synonymous when defining the navigation path, UniVerse still uses the “/” AND the “,” to define the navigation path of the data, but only the “,” to determine the location of the marks in the resulting data.

Like multivalued fields, you must start at the XPath with the parent node of the multivalue.

The next example illustrates how to extract data for a multi-subvalued field:

```
<field_extraction field = "COURSE_NBR" path = "CGA-MV, CGA-MS,  
Crs__/text()" />
```

The COURSE\_NBR field in the UniVerse data file will be extracted as follows:

1. Start at the CGA-MV node in the XML document, under the start node (ROOT/STUDENT\_record).
2. From the first CGA-MV node, go to the next level, the CGA-MS node.
3. From the first CGA-MS node, go to the Crs\_\_ node. Return the text from the Crs\_\_node, and make that text the first multi-subvalue of COURSE\_NBR.
4. Go back to the CGA-MS node, and search the siblings of the CGA-MS nodes to see if there are any more CGA-MS nodes of the same name. If any are found, return the Crs\_\_/text() under these nodes, and make them the next multi-subvalues of COURSE\_NBR.

5. Go back to the CGA-MV node and search for siblings of the CGA-MS node that have the same CGA-MV node name. If any are found, repeat steps 3 and 4 to get the values for these CGA-MV nodes, and make them multivalues.

The COURSE\_NBR field will look like this:

```
<Field Mark>Crs__text() value under 1st CGA-MS node of 1st CGA-MV
node<multi-subvalue mark>Crs__text() under 2nd CGA-MS node of 1st
CGA-MV node<multi-subvalue mark>...<multivalue mark>Crs__text()
under 1st CGA-MS node of the 2nd CGA-MV node<multi-subvalue
mark>Crs__text() under 2nd
CGA-MS node of the 2nd CGA-MV node<multi-subvalue mark>Crs__text()
value under the 3rd CGS-MS node of the 2nd CGA-MV node>...<Field
Mark>
```

The following example illustrates the complete extraction file for the above examples:

```
<U2XML_extraction>
  <file_extraction start = "/ROOT/STUDENT_record" dictionary =
  "D_MYSTUDENT"
    <!--field extraction rule in element mode-->
    <field_extraction field = "@ID" path = "STUDENT/text()"/>
    <field_extraction field = "LNAME" path = "Last_Name/text()"/>
    <field_extraction field = "SEMESTER" path = "CGA-MV/Term/text()"/>
    <field_extraction field = "COURSE_NBR" path = "CGA-MV, CGA-MS,
    Crs__/text"/>
    <field_extraction field = "COURSE_GRD" path = "CGA-MV, CGA-MS,
    GD/text()"/>
    <field_extraction field = "COURSE_NAME" path = "CGA-MV, CGA-MS,
    Course_Name/text()"/>
  </U2XML_extraction>
```

## Extracting XML data through UniVerse BASIC

Complete the following steps to access the XML data through UniVerse BASIC:

1. Familiarize yourself with the elements of the DTD associated with the XML data you are receiving.
2. Create the extraction file for the XML data.
3. Prepare the XML document using the UniVerse BASIC PrepareXML function.

4. Open the XML document using the UniVerse BASIC OpenXMLData function.
5. Read the XML data using the UniVerse BASIC ReadXMLData function.
6. Close the XML document using the UniVerse BASIC CloseXMLData function.
7. Release the XML document using the UniVerse BASIC ReleaseXML function.

### ***Preparing the XML document***

You must first prepare the XML document in the UniVerse BASIC program. This step allocates memory for the XML document, opens the document, determines the file structure of the document, and returns the file structure.

`Status=PrepareXML(xml_file,xml_handle)`

The following table describes each parameter of the syntax.

Parameter	Description
<code>xml_file</code>	The path to the file where the XML document resides.
<code>xml_handle</code>	The return value. The return value is the UniVerse BASIC variable for <code>xml_handle</code> . Status is one of the following return values:
<code>XML.SUCCESS</code>	Success
<code>XML.ERROR</code>	Error

#### **PrepareXML Parameters**

### ***Example***

The following example illustrates use of the PrepareXML function:

```
STATUS = PrepareXML ("&XML& /MYSTUDENT.XML" , STUDENT_XML)
IF STATUS=XML.ERROR THEN
    STATUS = XMLError(errmsg)
    PRINT "error message ":errmsg
    STOP "Error when preparing XML document "
END
```

## ***Opening the XML document***

After you prepare the XML document, open it using the OpenXMLData function.

```
Status=OpenXMLData(xml_handle,xml_data_extraction_rule,  
xml_data_handle)
```

The following table describes each parameter of the syntax.

Parameter	Description
<i>xml_handle</i>	The XML handle generated by the PrepareXML() function.
<i>xml_data_extraction_rule</i>	The path to the XML extraction rule file.
<i>xml_data_handle</i>	The XML data file handle. The following are the possible return values:  XML.SUCCESS                 Success. XML.ERROR                     Failed XML.INVALID.HANDLE         Invalid XML handle

### **OpenXMLData Parameters**

## ***Example***

The following example illustrates use of the OpenXMLData function:

```
status = OpenXMLData("STUDENT_XML",  
"&XML&/MYSTUDENT.ext", STUDENT_XML_DATA)  
If status = XML.ERROR THEN  
    STOP "Error when opening the XML document. "  
END  
IF status = XML.INVALID.HANDLE THEN  
    STOP "Error: Invalid parameter passed."  
END
```

## ***Reading the XML document***

After opening the XML document, read the document using the ReadXMLData function. UniVerse BASIC returns the XML data as a dynamic array.

```
Status=ReadXMLData(xml_data_handle, rec)
```

The following table describes each parameter of the syntax.

Parameter	Description								
<i>xml_data_handle</i>	A variable that holds the XML data handle created by the OpenXMLData function.								
<i>rec</i>	A mark-delimited dynamic array containing the extracted data. Status if one of the following: <table><tr><td>XML.SUCCESS</td><td>Success</td></tr><tr><td>XML.ERROR</td><td>Failure</td></tr><tr><td>XML.INVALID.HANDLE 2</td><td>Invalid <i>xml_data_handle</i></td></tr><tr><td>XML.EOF</td><td>End of data</td></tr></table>	XML.SUCCESS	Success	XML.ERROR	Failure	XML.INVALID.HANDLE 2	Invalid <i>xml_data_handle</i>	XML.EOF	End of data
XML.SUCCESS	Success								
XML.ERROR	Failure								
XML.INVALID.HANDLE 2	Invalid <i>xml_data_handle</i>								
XML.EOF	End of data								

#### ReadXMLData Parameters

After you read the XML document, you can execute any UniVerse BASIC statement or function against the data.

#### Example

The following example illustrates use of the ReadXMLData function:

```
MOREDATA=1
LOOP WHILE (MOREDATA=1)
    status = ReadXMLData(STUDENT_XML,rec)
    IF status = XML.ERROR THEN
        STOP "Error when preparing the XML document. "
    END ELSE IF status = XML.EOF THEN
        PRINT "No more data"
        MOREDATA = 0
    END ELSE
        PRINT "rec = ":rec
    END
REPEAT
```

#### Closing the XML document

After you finish using the XML data, use CloseXMLData to close the dynamic array variable.

```
Status=CloseXMLData(xml_data_handle)
```

where *xml\_data\_handle* is the name of the XML data file handle created by the OpenXMLData() function.

The return value is one of the following:

XML.SUCCESS	Success
XML.ERROR	Failure
XML.INVALID.HANDLE2	Invalid <i>xml_data_handle</i>

### *Example*

The following example illustrates use of the CloseXMLData function:

```
status = CloseXMLData(STUDENT_XML)
```

### ***Releasing the XML document***

Finally, release the dynamic array variable using the ReleaseXML function.

```
ReleaseXML(XMLhandle)
```

where *XMLhandle* is the XML handle created by the PrepareXML() function.

ReleaseXML destroys the internal DOM tree and releases the associated memory.

### ***Getting error messages***

Use the XMLError function to get the last error message.,

```
XMLError(errmsg)
```

Where errmsg is the error message string, or one of the following return values:

XML.SUCCESS	Success
XML.ERROR	Failure

## ***Example***

The following example illustrates a UniVerse BASIC program that prepares, opens, reads, closes, and releases an XML document:

```
# INCLUDE UNIVERSE.INCLUDE XML.H
STATUS=PrepareXML ("&XML&/MYSTUDENT.XML", STUDENT_XML)
IF STATUS=XML.ERROR THEN
    STATUS = XMLError(errmsg)
    PRINT "error message ":errmsg
    STOP "Error when preparing XML document "
END

STATUS =
OpenXMLData ("STUDENT_XML", "&XML&/MYSTUDENT.ext", STUDENT_XML_DATA)

IF STATUS = XML.ERROR THEN
    STOP "Error when opening the XML document. "
END

IF STATUS = XML.INVALID.HANDLE THEN
    STOP "Error: Invalid parameter passed." END

MOREDATA=1
LOOP WHILE (MOREDATA=1)
    STATUS=ReadXMLData(STUDENT_XML_DATA,rec)
    IF STATUS = XML.ERROR THEN
        STOP "Error when preparing the XML document. "
    END ELSE IF STATUS = XML.EOF THEN
        PRINT "No more data"
        MOREDATA = 0
    END ELSE
        PRINT "rec = ":rec
        PRINT "rec = ":rec
    END
REPEAT
STATUS = CloseXMLData(STUDENT_XML_DATA)
STATUS = ReleaseXML(STUDENT_XML)
```

## **Displaying an XML document through RetriVe**

You can display the contents of an XML file through RetriVe by defining an extraction file, preparing the XML document, then using LIST to display the contents.

## ***Preparing the XML document***

Before you execute the LIST statement against the XML data, you must first prepare the XML file using the PREPARE XML command.

```
PREPARE.XML xml_file xml_data
```

*xml\_file* is the path to the location of the XML document.

*xml\_data* is the name of the working file you assign to the XML data.

The following example illustrates preparing the MYSTUDENT.XML document:

```
PREPARE.XML "&XML&/MYSTUDENT.XML" STUDENT_XML  
PREPARE.XML successful.
```

## ***Listing the XML data***

Use the RetrievVe LIST command with the XMLDATA option to list the XML data.

```
LIST XMLDATA xml_data "extraction_file" [fields]
```

The following table describes each parameter of the syntax.

Parameter	Description
XMLDATA <i>xml_data</i>	Specifies to list the records from the <i>xml_data</i> you prepared.
<i>extraction_file</i>	The full path to the location of the extraction file. You must surround the path in quotation marks.
<i>fields</i>	The fields from the dictionary you specified in the extraction file that you want to display.

### **LIST Parameters for Listing XML Data**

When you list an XML document, RetriVe uses the dictionary you specify in the extraction file. The following example lists the dictionary records for the MYSTUDENT dictionary:

```
>LIST DICT MYSTUDENT

DICT MYSTUDENT      10:25:32am 19 Oct 2001  Page      1

          Type &
Field..... Field. Field..... Conversion.. Column..... Output
Depth &
Name..... Number Definition.. Code..... Heading..... Format
Assoc..

@ID        D    0           MYSTUDENT      10L     S
LNAME      D    1           Last Name    15T     S
SEMESTER   D    2           Term        4L      M
CGA
COURSE_NBR D    3           Crs #       5L      M
CGA
COURSE_GRD D    4           GD         3L      M
CGA

5 records listed.
```

The fields in the dictionary record must correspond to the position of the fields in the XML extraction file. In the following extraction file, @ID is position 0, LNAME is position 1, SEMESTER is position 2, COURSE\_NBR is position 3, COURSE\_GRD is position 4, and COURSE\_NAME is position 5. The dictionary of the MYSTUDENT file matches these positions.

The following example illustrates listing the fields from the MYSTUDENT XML document, using the MYSTUDENT.EXT extraction file:

```

LIST XMLDATA STUDENT_XML "&XML&/MYSTUDENT.EXT" LNAME SEMESTER COURSE_NBR
COURSE
_GRD COURSE_NAME 11:58:01am 19 Oct 2001 PAGE 1
MYSTUDENT. Last Name..... Term Crs # GD. Course
Name.....
424-32-565 Martin SP94 PY100 C Introduction to
Psycholog 6 Y
414-44-654 Offenbach FA93 PE100 C Golf - I
5 CS104 D Database Design
MA101 C Math Principals
FA100 C Visual Thinking
SP94 CS105 B Database Design
MA102 C Algebra
PY100 C Introduction to
Psycholog Y
221-34-566 Miller FA93 EG110 C Engineering Principles
5 MA220 B Calculus- I
PY100 B Introduction to
Psycholog Y
SP94 EG140 B Fluid Mechanics
EG240 B Circut Theory
MA221 B Calculus - II
978-76-667 Muller FA93 FA120 A Finger Painting
6 FA230 C Photography Principals
HY101 C Western Civilization
SP94 FA121 A Watercorlors
FA231 B Photography Practicum
HY102 I Western Civilization -
15 00 to 1945
521-81-456 Smith FA93 CS130 A Intro to Operating
System 4 S
Science CS100 B Intro to Computer
Psycholog PY100 B Introduction to
System SP94 CS131 B Intro to Operating
Science CS101 B Intro to Computer
291-22-202 Smith PE220 A Racquetball
1 SP94 FA100 B Visual Thinking
6 records listed.
>

```

## ***Release the XML document***

When you finish with the XML document, release it using RELEASE.XML.

```
RELEASE.XML xml_data
```

## **Displaying an XML document through UniVerse SQL**

You can display an XML document through UniVerse SQL using the SELECT statement.

### ***Preparing the XML document***

Before you execute the SELECT statement against the XML data, you must first prepare the XML file using the PREPARE XML command.

```
PREPARE.XML xml_file xml_data
```

*xml\_file* is the path to the location of the XML document.

*xml\_data* is the name of the working file you assign to the XML data.

The following example illustrates preparing the MYSTUDENT.XML document:

```
PREPARE.XML "&XML&/MYSTUDENT.XML" STUDENT_XML  
PREPARE.XML successful.
```

### ***Listing the XML data***

Use the UniVerse SQL SELECT command with the XMLDATA option to list the XML data.

```
SELECT clause FROM XMLDATA xml_data extraction_file
```

```
[WHERE clause]  
[WHEN clause [WHEN clause]...]  
[GROUP BY clause]  
[HAVING clause]  
[ORDER BY clause]  
[report_qualifiers]  
[processing_qualifiers]
```

The following table describes each parameter of the syntax.

Parameter	Description
SELECT clause	Specifies the columns to select from the database.
FROM XMLDATA <i>xml_data</i>	Specifies the XML document you prepared from which you want to list data.
<i>extraction_file</i>	Specifies the file containing the extraction rules for the XML document.
WHERE clause	Specifies the criteria that rows must meet to be selected.
WHEN clause	Specifies the criteria that values in a multivalued column must meet for an association row to be output.
GROUP BY clause	Groups rows to summarize results.
HAVING clause	Specifies the criteria that grouped rows must meet to be selected.
ORDER BY clause	Sorts selected rows.
<i>report_qualifiers</i>	Formats a report generated by the SELECT statement.
<i>processing_qualifiers</i>	Modifies or reports on the processing of the SELECT statement.

#### SELECT Parameters

You must specify clauses in the SELECT statement in the order shown in the syntax. You can use the SELECT statement with type 1, type 19, and type 25 files only if the current isolation level is 0 or 1.

For a full discussion of the UniVerse SQL SELECT statement clauses, see the *UniVerse SQL Reference*.

The following example illustrates displaying the XML document using the UniVerse SQL SELECT statement:

```
>SELECT * FROM XMLDATA STUDENT_XML "&XML&./MYSTUDENT.EXT";
MYSTUDENT.    Last Name.....   Term   Crs #   Course Name..... .
GD.

        424-32-565    Martin      SP94    PY100  Introduction to Psycholog C
6                                         Y
                                         PE100  Golf - I          C
                                         CS104  Database Design D
                                         5
                                         MA101  Math Principals C
                                         FA100  Visual Thinking C
                                         SP94   CS105  Database Design B
                                         MA102  Algebra          C
                                         PY100  Introduction to Psycholog C
                                         Y
                                         EG110  Engineering Principles C
                                         5
                                         MA220  Calculus- I       B
                                         PY100  Introduction to Psycholog B
                                         Y
                                         SP94   EG140  Fluid Mechanics B
                                         EG240  Circuit Theory   B
                                         MA221  Calculus - II     B
                                         FA120  Finger Painting A
                                         6
                                         Press any key to continue...

MYSTUDENT.    Last Name.....   Term   Crs #   Course Name..... .
GD.

                                         FA230  Photography Principals C
                                         HY101  Western Civilization C
                                         SP94   FA121  Watercolor          A
                                         FA231  Photography Practicum B
                                         HY102  Western Civilization - 15 I
                                         00 to 1945
                                         521-81-456    Smith      FA93    CS130  Intro to Operating System A
4                                         CS100  Intro to Computer Science B
                                         PY100  Introduction to Psycholog B
                                         Y
                                         SP94   CS131  Intro to Operating System B
                                         CS101  Intro to Computer Science B
                                         PE220  Racquetball          A
                                         291-22-202    Smith      SP94   FA100  Visual Thinking B
1

6 records listed.
>
```

### ***Release the XML document***

When you finish with the XML document, release it using the RELEASE.XML.

```
RELEASE.XML xml_data
```

# Chapter 6: The Simple Object Access Protocol

SOAP components . . . . .	6-2
The SOAP API for UniVerse BASIC . . . . .	6-4
Sending a SOAP request . . . . .	6-4
SOAP API for UniVerse BASIC programmatic interfaces . . . . .	6-5
SOAPSetDefault . . . . .	6-5
SOAPGetDefault . . . . .	6-7
SOAPCreateRequest . . . . .	6-8
SOAPCreateSecureRequest . . . . .	6-10
SOAPSetParameters. . . . .	6-12
SOAPSetRequestHeader . . . . .	6-14
SOAPSetRequestBody . . . . .	6-15
SOAPSetRequestContent . . . . .	6-16
SOAPRequestWrite . . . . .	6-17
SOAPSubmitRequest . . . . .	6-19
SOAPGetResponseHeader . . . . .	6-20
SOAPGetFault. . . . .	6-21
protocolLogging . . . . .	6-23
SOAP API for UniVerse BASIC example . . . . .	6-25

The Simple Object Access Protocol (SOAP) is an XML-based protocol for exchanging structured information in a distributed environment. It allows the sender and receiver of XML documents over the web to support a common data transfer protocol, and is language and platform independent. Its most common method of operation is as a means of issuing Remote Procedure Calls across a network. However, it can also be used in other manners, such as the posting of XML documents to servers for processing.

## **SOAP components**

A SOAP message contains three major blocks, the envelope, the header, and the body.

- Envelope – Defines the start and end of a SOAP message.
- Header – Carries application-defined information associated with the message, such as security tokens, message correlation mechanisms, and transaction identifiers. A header is optional in a SOAP message.
- Body – One or more body blocks containing the SOAP message itself.

### ***The envelope***

The envelope is the outermost element in a SOAP message, and is the root element of the message. Use the *env* namespace prefix to specify the envelope, and the *Envelope* element. The envelope specifies the version of SOAP you are using. If you are using a v1.1-compliant SOAP processor, it generates a fault if it receives a message containing a v1.2 envelope namespace.

The envelope element namespace for the difference SOAP versions follows:

- v1.1 – <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2001/12/soap-envelope"
- v1.1 – SOAP-ENV:"http://schemas.xmlsoap.org/soap/envelope/"

You can also specify an encoding namespace in the SOAP envelope.

## ***SOAP header***

The header is an optional part of a SOAP message, and is the first immediate child element of the envelope. You can have multiple SOAP headers.

SOAP headers are used to include additional functionality, such as security, and other attributes associated with the message. A SOAP header can have attributes for intermediary processing, such as a user name and password, that may not be processed at the final SOAP processor destination.

You can include the *mustUnderstand* attribute in a SOAP header to require the receiver of the message to understand a header. If you include the *mustUnderstand* attribute with a value of *true*, the SOAP node you target must process the SOAP block using the requirements defined in the header, or not process the message and return an error code.

## ***SOAP body***

The SOAP body contains the XML data defined by the application. You must define the SOAP body within the envelope, and it must appear after any SOAP headers.

For complete information about SOAP, see the W3C technical publications website at <http://www.w3.org/TR>.

## The SOAP API for UniVerse BASIC

The SOAP API for UniVerse BASIC provides the capability of issuing requests to SOAP servers from UniVerse BASIC through the standard HTTP protocol. The SOAP API for UniVerse BASIC makes use of UniVerse CallHTTP to send and receive SOAP messages. SOAP responses retrieved from a SOAP server can be parsed through the XML API for UniVerse BASIC.

### Sending a SOAP request

Sending a SOAP message to a server from the SOAP API for UniVerse BASIC typically involves some variation of the following procedure:

1. **Set Protocol Defaults** – You must define the SOAP default settings, including the version of SOAP you are using. Use [SOAPSetDefault](#) to define the SOAP version, and SetHTTPDefault to define any HTTP headers you may need.
2. **Create the SOAP Request** – Use the [SOAPCreateRequest](#) function to define the SOAP request.
3. **Set the SOAP Request Content** – Use one of the following functions to set the SOAP request content:
  - [SOAPSetRequestHeader](#) – You can use this function to set any SOAP header blocks for the SOAP request.
  - [SOAPSetParameters](#) – Sets the SOAP body according to the RPC style of communication, based on the input parameters to the function,
  - [SOAPSetRequestBody](#) – This function is similar to SOAPSetParameters, but you can also use this function for non-RPC styles of communication, and add multiple body elements to the SOAP request.
  - [SOAPSetRequestContent](#) – Sets the entire SOAP message content, either from an input string or from the contents of a file.
4. **Submit the SOAP Request** – Use the [SOAPSubmitRequest](#) to submit the SOAP request. The output parameters *respData* and *respHeaders* contain the response from the SOAP server. You can use the XML API for UniVerse BASIC to parse the resulting SOAP response.

---

## **SOAP API for UniVerse BASIC programmatic interfaces**

This section describes the SOAP API for UniVerse BASIC functions.

### **SOAPSetDefault**

#### **Syntax**

**SOAPSetDefault**(*option, value*)

#### **Description**

Use the **SOAPSetDefault** function to define default SOAP settings, such as the SOAP version. By default, the SOAP version is 1.1, although you can specify version 1.2.

For SOAP version 1.1, the namespace prefixes "env" and "enc" are associated with the SOAP namespace names

<http://schemas.xmlsoap.org/soap/envelope/> and

<http://schemas.xmlsoap.org/soap/encoding/> respectively. The namespace prefixed "xsi" and "xsd" are associated with the namespace names

<http://www.w3.org/1999/XMLSchema-instance> and

<http://www.w3.org/1999/XMLSchema> respectively.

The SOAP version can be set to 1.2 to support the newer SOAP 1.2 protocol.

The namespace prefixes "env" and "enc" are associated with the SOAP namespace names "<http://www.w3.org/2001/12/soap-envelope>" and "<http://www.w3.org/2001/12/soap-encoding>" respectively. The namespace prefixes "xsd" and "xsi" will be associated with the namespace names "<http://www.w3.org/2001/XMLSchema>" and "<http://www.w3.org/2001/XMLSchema-instance>" respectively.



*Note: All defaults set by SOAPSetDefault remain in effect until the end of the current UniVerse session. If you do not want the setting to affect subsequent programs, clear it before exiting the current program.*

Along with SOAPSetDefault, you can use the CallHTTP function setHTTPDefault to set HTTP-specific settings or headers, if the HTTP default settings are not sufficient.

### **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>option</i>	A string containing an option name. UniVerse currently only supports the "VERSION" option. [IN]
<i>value</i>	A string containing the appropriate option value. For the VERSION option, the string should be 1.0, 1.1, or 1.2. [IN]

#### **SOAPSetDefault Parameters**

### **Return codes**

The return code indicating success or failure. The following table describes the value of each return code.

Return Code	Description
0	Function completed successfully.
1	Invalid option (currently, UniData only supports VERSION).
2	Invalid value. If you do not specify a value, UniData uses the default of 1.1.

#### **SOAPSetDefault Return Codes**

You can also use the UniVerse BASIC STATUS() function to obtain the return status from the function.

## **SOAPGetDefault**

### **Syntax**

**SOAPGetDefault**(*option*, *value*)

### **Description**

The **SOAPGetDefault** function retrieves default SOAP settings, such as the SOAP version.

### **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>option</i>	A string containing an option name. UniVerse currently only supports the VERSION option. [IN]
<i>value</i>	A string returning the option value. [OUT]

**SOAPGetDefault Parameters**

### **Return codes**

The return code indicating success or failure. The following table describes the value of each return code.

Return Code	Description
0	Function completed successfully.
1	Invalid option (currently, UniVerse only supports the VERSION option).

**SOAPGetDefault Return Codes**

You can also use the UniVerse BASIC STATUS() function to obtain the return status from the function.

## **SOAPCreateRequest**

### **Syntax**

**SOAPCreateRequest**(*URL*, *soapAction*, *Request*)



### **Description**

The **SOAPCreateRequest** creates a SOAP request and returns a handle to the request.

### **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
URL	A string containing the URL where the web service is located. UniVerse sends the SOAP request to this URL. For information about the format of the URL, see “ <a href="#">URL format</a> ” on page 8. [IN]
soapAction	A string UniVerse uses as the SOAPAction HTTP header for this SOAP request. [IN]
Request	The returned handle to the SOAP request. You can use this handle can be used in subsequent calls to the SOAP API for UniVerse BASIC. [OUT]

#### **SOAPCreateRequest Parameters**

### **URL format**

The URL you specify must follow the syntax defined in RFS 1738. The general format is:

`http://<host>:<port>/path>?<searchpart>`

The following table describes each parameter of the syntax.

Parameter	Description
<i>host</i>	Either a name string or an IP address of the host system.
<i>port</i>	The port number to which you want to connect. If you do not specify <i>port</i> , UniVerse BASIC defaults to 80. Omit the preceding colon if you do not specify this parameter.
<i>path</i>	Defines the file you want to retrieve on the web server. If you do not specify <i>path</i> , UniVerse BASIC defaults to the home page.
<i>searchpart</i>	Use <i>searchpart</i> to send additional information to a web server.

#### URL Parameters



*Note:* If the URL you define contains a *searchpart*, you must define it in its encoded format. For example, a space is converted to +, and other non-alphanumeric characters are converted to %HH format.

You do not need to specify the *host* and *path* parameters in their encoded formats. UniVerse BASIC encodes these parameters prior to communicating with the web server.

### Return code

The return code indicating success or failure. The following table describes the value of each return code.

Return Code	Description
0	Function complete successfully.
1	Invalid URL syntax.
2	Invalid HTTP method (indicates the POST method is not supported by the HTTP server).

#### SOAPCreateRequest Return Codes

You can also use the UniVerse BASIC STATUS() function to obtain the return status from the function.

## **Example**

The following code segment illustrates the SOAPCreateRequest function:

```
* Create the Request
Ret = SoapCreateRequest(URL, SoapAction, SoapReq)
IF Ret <> 0 THEN
    STOP "Error in SoapCreateRequest: " : Ret
END
.
.
```

## **SOAPCreateSecureRequest**

### **Syntax**

**SOAPCreateSecureRequest**(*URL*, *soapAction*, *Request*, *security\_context*)

### **Description**

The **SOAPCreateSecureRequest** function creates a secure SOAP request and returns a handle to the request.

### **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>URL</i>	A string containing the URL where the web service is located. UniVerse sends the SOAP request to this URL. For information about the format of the URL, see <a href="#">URL format</a> . [IN]

#### **SOAPCreateSecureRequest Parameters**

---

Parameter	Description
<i>soapAction</i>	A string UniVerse uses as the SOAPAction HTTP header for this SOAP request. [IN]
<i>Request</i>	The returned handle to the SOAP request. You can use this handle can be used in subsequent calls to the SOAP API for UniVerse BASIC. [OUT]
<i>security_context</i>	A handle to the security context.

---

#### **SOAPCreateSecureRequest Parameters (Continued)**

## URL format

The URL you specify must follow the syntax defined in RFS 1738. The general format is:

`http://<host>:<port>/path>?<searchpart>`

The following table describes each parameter of the syntax.

---

Parameter	Description
<i>host</i>	Either a name string or an IP address of the host system.
<i>port</i>	The port number to which you want to connect. If you do not specify <i>port</i> , UniVerse defaults to 80. Omit the preceding colon if you do not specify this parameter.
<i>path</i>	Defines the file you want to retrieve on the web server. If you do not specify <i>path</i> , UniVerse defaults to the home page.
<i>searchpart</i>	Use <i>searchpart</i> to send additional information to a web server.

---

#### **URL Parameters**

**Note:** If the URL you define contains a *searchpart*, you must define it in its encoded format. For example, a space is converted to +, and other nonalphanumeric characters are converted to %HH format.

You do not need to specify the *host* and *path* parameters in their encoded formats. UniVerse BASIC encodes these parameters prior to communicating with the web server.

## Return code

The return code indicating success or failure. The following table describes the value of each return code.

Return Code	Description
0	Function complete successfully.
1	Invalid URL syntax.
2	Invalid HTTP method (indicates the POST method is not supported by the HTTP server).
101	Invalid security context handle.

### **SOAPCreateSecureRequest Return Codes**

You can also use the UniVerse BASIC STATUS() function to obtain the return status from the function.

## SOAPSetParameters

### **Syntax**

**SOAPSetParameters**(*Request*, *URI*, *serviceName*, *paramArray*)

### **Description**

The **SOAPSetParameters** function sets up the SOAP request body, specifying a remote method to call along with the method's parameter list.

## Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>Request</i>	Handle to the request created with <a href="#">SOAPCreateRequest</a> . [IN]
<i>namespace</i>	A string is used as the namespace URI for the SOAP call. [IN]
<i>serviceName</i>	The name of the SOAP service. [IN]
<i>paramArray</i>	A dynamic array containing the method parameters for the SOAP call. Each method parameter consists of the following values: <ul style="list-style-type: none"><li>■ A parameter name</li><li>■ A parameter value</li><li>■ A parameter type (if type is omitted, <i>xsd:string</i> will be used). name, value, and type are separated by @VM. Additional parameters are separated by @AM, as shown in the following example:</li></ul> <pre>&lt;param1Name&gt;@VM&lt;param1Value&gt;@VM&lt;param1Type&gt;@AM &lt;param2Name&gt;@VM&lt;param2Value&gt;@VM&lt;param2Type&gt;...[IN]</pre>

### SOAPSetParameters Parameters

## Return codes

The return code indicating success or failure. The following table describes the value of each return code.

Return Code	Description
0	Function completed successfully.
1	Invalid request handle was passed to the function.

### SOAPSetParameters Return Codes

You can also use the UniVerse BASIC STATUS() function to obtain the return status from the function.

## **Example**

As an example, the following inputs:

Input	Description
<i>serviceName</i>	“getStockQuote”
<i>namespace</i>	“http://host/#StockQuoteService”
<i>paramArray</i>	“symbol”:@VM:“rocketU2”:@VM:“xsd:string”

**SOAPSetParameter Example**

set the SOAP body as follows:

```
<SOAP-ENV:Body>
  <nsl:getStockQuote
    xmlns:nsl="http://host/#StockQuoteService">
    <symbol xsi:type="xsd:string">rocketU2</symbol>
  </nsl:getStockQuote>
<SOAP-ENV:Body>
```

The following code example illustrates the SOAPSetParameters function:

```
* Set up the Request Body
Ret = SoapSetParameters(SoapReq, NameSpace, Method, MethodParms)
IF Ret <> 0 THEN
  STOP "Error in SoapSetParameters: " : Ret
END
```

## **SOAPSetRequestHeader**

### **Syntax**

**SOAPSetRequestHeader**(*Request*, *value*)

### **Description**

The **SOAPSetRequestHeader** sets up a SOAP request header. By default, there is no SOAP header.

## Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>Request</i>	Handle to the request created with <a href="#">SOAPCreateRequest</a> . [IN]
<i>value</i>	A dynamic array containing SOAP header blocks, for example: <code>&lt;header block&gt;@AM&lt;header block&gt;...[IN]</code>

### SOAPSetRequestHeader Parameters

## Return codes

The return code indicating success or failure. The following table describes the value of each return code.

Return Code	Description
0	Function completed successfully.
1	Invalid request handle.

### SOAPSetRequestHeader Return Codes

You can also use the UniVerse BASIC STATUS() function to obtain the return status from the function.

## SOAPSetRequestBody

### Syntax

**SOAPSetRequestBody**(*Request*, *value*)

### Description

The **SOAPSetRequestBody** function sets up a SOAP request body directly, as opposed to having it constructed via the [SOAPSetParameters](#) function. With this function, you can also attach multiple body blocks to the SOAP request.

Each SOAP request should include at least one body block.

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
Request	Handle to the request created with <a href="#">SOAPCreateRequest</a> . [IN]
value	A dynamic array containing SOAP body blocks, for example: <code>&lt;body block&gt;@AM&lt;body block&gt;... [IN]</code>

**SOAPSetRequestBody Parameters**

### Return codes

The return code indicating success or failure. The following table describes the value of each return code.

Return Code	Description
0	Function completed successfully.
1	Invalid request handle.

**SOAPSetRequestBody Return Codes**

You can also use the UniVerse BASIC STATUS() function to obtain the return status from the function.

## SOAPSetRequestContent

### Syntax

**SOAPSetRequestContent**(*Request*, *reqDoc*, *docTypeFlag*)

### Description

The **SOAPSetRequestContent** function sets the entire SOAP request's content from an input string or from a file.

## Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>Request</i>	Handle to the request created with <a href="#">SOAPCreateRequest</a> . [IN]
<i>reqDoc</i>	The input document to use as the SOAP request content. [IN]
<i>docTypeFlag</i>	A flag indicating whether <i>reqDoc</i> is a string holding the actual content, or the path to a file holding the content. <ul style="list-style-type: none"><li>■ 0 – <i>reqDoc</i> is a file holding the request content.</li><li>■ 1 – <i>reqDoc</i> is a string holding the request content.</li></ul> [IN]

### SOAPSetRequestContent Parameters

## Return codes

The return code indicating success or failure. The following table describes the status of each return code.

Return Code	Description
0	Function completed successfully.
1	Invalid request handle.
2	Unable to open the file named by <i>reqDoc</i> .
3	Unable to read the file named by <i>reqDoc</i> .

### SOAPSetRequestContent Return Codes

You can also use the UniVerse BASIC STATUS() function to obtain the return status from the function.

## SOAPRequestWrite

### Syntax

**SOAPRequestWrite**(*Request*, *reqDoc*, *docTypeFlag*)

## Description

The **SOAPRequestWrite** function outputs the SOAP request, in XML format, to a string or to a file.

## Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>Request</i>	Handle to the request created with <a href="#">SOAPCreateRequest</a> . [IN]
<i>reqDoc</i>	Depending on <i>docTypeFlag</i> , either an output string containing the SOAP request content, or a path to a file where the SOAP request content will be written. [OUT]
<i>docTypeFlag</i>	A flag indicating whether <i>reqDoc</i> is an output string that is to hold the request content, or a path to a file where the SOAP request content will be written. <ul style="list-style-type: none"><li>■ 0 – <i>reqDoc</i> is a file where the request content will be written upon successful completion.</li><li>■ 1 – <i>reqDoc</i> is a string that will hold the request upon successful completion. [IN]</li></ul>

**SOAPRequestWrite Parameters**

## Return codes

The return code indicating success or failure. The following table describes the value of each return code.

Return Code	Description
0	Function completed successfully.
1	Invalid request handle.
2	Unable to open the file named by <i>reqDoc</i> .
3	Unable to write to the file named by <i>reqDoc</i> .

**SOAPRequestWrite Return Codes**

You can also use the UniVerse BASIC STATUS() function to obtain the return status from the function.

## SOAPSubmitRequest

### Syntax

**SOAPSubmitRequest**(*Request*, *timeout*, *respHeaders*, *respData*, *soapStatus*)

### Description

The **SOAPSubmitRequest** function submits a request and gets the response.

Internally, **SOAPSubmitRequest** utilizes CallHTTP's submitRequest() function to send the SOAP message. The *soapStatus* variable holds the status from the underlying CallHTTP function. If an error occurs on the SOAP server while processing the request, *soapStatus* will indicate an HTTP 500 "Internal Server Error", and **respData** will be a SOAP Fault message indicating the server-side processing error.

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>Request</i>	Handle to the request created with <a href="#">SOAPCreateRequest</a> . [IN]
<i>timeout</i>	Timeout, in milliseconds, to wait for a response. [IN]
<i>respHeaders</i>	Dynamic array of HTTP response headers and their associated values. [OUT]
<i>respData</i>	The SOAP response message. [OUT]
<i>soapStatus</i>	Dynamic array containing status code and explanatory text. [OUT]

**SOAPSubmitRequest Parameters**

## **Return codes**

The return code indicating success or failure. The following table describes the value of each return code.

Return Code	Description
0	Function completed successfully.
1	Invalid request handle.
2	Request timed out.
3	Network error occurred.
4	Other error occurred.

### **SOAPSubmitRequest Return Codes**

You can also use the UniVerse BASIC STATUS() function to obtain the return status from the function.

## **Example**

The following code sample illustrates the SOAPSSubmitRequest function:

```
* Submit the Request
Ret = SoapSubmitRequest(SoapReq, Timeout, RespHeaders, RespData,
SoapStatus)
IF Ret <> 0 THEN
    STOP "Error in SoapSubmitRequest: " : Ret
END

PRINT "Response status : " : SoapStatus
PRINT "Response headers: " : RespHeaders
PRINT "Response data   : " : RespData
.
.
```

## **SOAPGetResponseHeader**

### **Syntax**

**SOAPGetResponseHeader(Request, headerName, headerValue)**

## Description

The **SOAPGetResponseHeader** function gets a specific response header after issuing a SOAP request.

## Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>Request</i>	Handle to the request created with <a href="#">SOAPCreateRequest</a> . [IN]
<i>headerName</i>	The header name whose value is being queried. [IN]
<i>headerValue</i>	The header value, if present in the response, or empty string if not (in which case the return status of the function is 2). [OUT]

### SOAPGetResponseHeader Parameters

## Return codes

The return code indicating success or failure. The following table describes the value of each return code.

Return Code	Description
0	Function completed successfully.
1	Invalid request handle.
2	Header not found in set of response headers.

### SOAPGetResponseHeader Return Codes

You can also use the UniVerse BASIC STATUS() function to obtain the return status from the function.

## SOAPGetFault

### Syntax

**SOAPGetFault**(*respData*, *soapFault*)

## Description

If the [SOAPSubmitRequest](#) function receives a SOAP Fault, the [SOAPGetFault](#) function parses the response data from SOAPSubmitRequest into a dynamic array of SOAP Fault components.



## Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>respData</i>	Response data from SOAPSubmitRequest after receiving a SOAP fault. [IN]
<i>soapFault</i>	Dynamic array consisting of Fault Code, Fault String, and optional Fault Detail, for example: <code>&lt;faultcode&gt;@AM&lt;faultstring&gt;@AM&lt;faultdetail&gt;@AM&lt;faultactor&gt;</code> Fault code values are XML-qualified names, consisting of: <ul style="list-style-type: none"><li>■ VersionMismatch</li><li>■ MustUnderstand</li><li>■ DTDNotSupported</li><li>■ DataEncoding Unknown</li><li>■ Sender</li><li>■ Receiver</li></ul>

### SOAPGetFault Parameters

## Return codes

The return code indicating success or failure. The following table describes the value of each return code.

Return Code	Description
0	Function completed successfully.
1	Invalid response data, possibly not a valid XML document.
2	SOAP Fault not found in response data.

### SOAPGetFault Return Codes

You can also use the UniVerse BASIC STATUS() function to obtain the return status from the function.

## protocolLogging

### Syntax

**protocolLogging**(*logFile*, *logAction*, *logLevel*)

### Description

The **protocolLogging** function starts or stops logging. By default, no logging takes place. The function parses the response data from [SOAPSubmitRequest](#), after receiving a SOAP Fault, into a dynamic array of SOAP Fault components.



### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>logFile</i>	The name of the log file [IN]
<i>logAction</i>	Flag (values “ON” or “OFF”) indicating whether logging should be turned on or off. [IN]
<i>logLevel</i>	Controls the level of log detail. <ul style="list-style-type: none"><li>■ 0 – No logging takes place.</li><li>■ 1 – Only errors are logged.</li><li>■ 3 – Warnings and errors are logged.</li><li>■ 7 – All SOAP actions are logged.</li></ul>
	[IN]

**protocolLogging Parameters**

## ***Return codes***

The return code indicating success or failure. The following table describes the value of each return code.

<b>Return Code</b>	<b>Description</b>
0	Function completed successfully.
1	An error occurred starting or stopping logging.

### **protocol Logging Return Codes**

You can also use the UniVerse BASIC STATUS() function to obtain the return status from the function.

## SOAP API for UniVerse BASIC example

```
*  
* Example of Sending a SOAP Request from UniBasic  
*  
  
URL          =  
"http://u2.rocketU2.com/webservices/AddressBookService"  
SoapAction   =  
"http://u2.rocketU2.com/AddressBook#getAddressFromName"  
NameSpace    = "http://u2.rocketU2.com/AddressBook"  
Method       = "getAddressFromName"  
MethodParms  = "name":@VM:"John Doe":@VM:"xsd:string"  
Timeout      = 30000  
  
* Create the Request  
Ret = SoapCreateRequest(URL, SoapAction, SoapReq)  
IF Ret <> 0 THEN  
    STOP "Error in SoapCreateRequest: " : Ret  
END  
  
* Set up the Request Body  
Ret = SoapSetParameters(SoapReq, NameSpace, Method, MethodParms)  
IF Ret <> 0 THEN  
    STOP "Error in SoapSetParameters: " : Ret  
END  
  
* Submit the Request  
Ret = SoapSubmitRequest(SoapReq, Timeout, RespHeaders, RespData,  
SoapStatus)  
IF Ret <> 0 THEN  
    STOP "Error in SoapSubmitRequest: " : Ret  
END  
  
PRINT "Response status : " : SoapStatus  
PRINT "Response headers: " : RespHeaders  
PRINT "Response data   : " : RespData
```

# Chapter 7: The Document Object Model

7

XPath and the Document Object Model . . . . .	7-3
A sample XML document . . . . .	7-3
Opening and closing a DOM document . . . . .	7-4
Navigating the DOM tree . . . . .	7-4
Building DOM trees from scratch . . . . .	7-5
Transforming XML documents . . . . .	7-7
XML for UniVerse BASIC API programmatic interfaces . . . . .	7-11
XDOMOpen . . . . .	7-11
XDOMCreateNode . . . . .	7-12
XDOMCreateRoot . . . . .	7-13
XDOMWrite . . . . .	7-14
XDOMClose . . . . .	7-15
XDOMValidate . . . . .	7-16
XDOMLocate . . . . .	7-18
XDOMLocateNode . . . . .	7-19
XDOMRemove . . . . .	7-25
XDOMAppend . . . . .	7-26
XDOMInsert . . . . .	7-28
XDOMReplace . . . . .	7-29
XDOMAddChild . . . . .	7-31
XDOMClone . . . . .	7-32
XDOMTransform . . . . .	7-33
XDOMGetNodeValue . . . . .	7-35
XDOMGetNodeType . . . . .	7-36
XDOMGetAttribute . . . . .	7-37
XDOMGetOwnerDocument . . . . .	7-38
XDOMGetUserData . . . . .	7-39

XDOMSetNodeValue . . . . .	7-40
XDOMSetUserData . . . . .	7-41
XMLGetError . . . . .	7-42

The Document Object Model (DOM) provides a standard way for you to manipulate XML documents. You can use the DOM API to delete, remove, and update an XML document, as well as create new XML documents.

The DOM represents a document as a tree of Nodes. Each node has a parent (except for the "root" node), and optional children. The DOM provides functions to traverse and manipulate these nodes. Another technology, XPath, (also supported in the XML for UniVerse BASIC API) provides the ability to locate nodes in the DOM based on search criteria.

The DOM defines different types of nodes, with the Element type being the most commonly used:

- Document
- DocumentFragment
- DocumentType
- EntityReference
- Element
- Attr
- ProcessingInstruction
- Comment
- Text
- CDATASection
- Entity
- Notation

---

## XPath and the Document Object Model

XPath is a language that gives the ability to address specific parts of an XML document. It allows you to execute node searches through expressions such as "find the node whose name is 'LASTNAME' and whose value starts with 'Smith' and is a child node of 'Employee.'"

The XML for UniVerse BASIC API provides support for XPath by allowing you to specify XPath search strings in some of its API calls. For example, XDOMLocate takes an Xpath string to locate a particular node in the DOM tree. XDOMRemove also takes an Xpath search string to evaluate which node(s) to remove.

### A sample XML document

The following document, "sample.xml," will be used to help illustrate usage of the XML for UniVerse BASIC API. It is a simple address book with two entries:

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

<ADDRBOOK>
    <ENTRY ID="id1">
        <NAME>Name One</NAME>
        <ADDRESS>101 Some Way</ADDRESS>
        <PHONENUM DESC="Work">303-111-1111</PHONENUM>
        <PHONENUM DESC="Fax">303-111-2222</PHONENUM>
        <PHONENUM DESC="Pager">303-111-3333</PHONENUM>
        <EMAIL>name.one@some.com</EMAIL>
    </ENTRY>
    <ENTRY ID="id2">
        <NAME>Name Two</NAME>
        <ADDRESS>202 Some Way</ADDRESS>
        <PHONENUM DESC="Work">303-222-1111</PHONENUM>
        <PHONENUM DESC="Fax">303-222-2222</PHONENUM>
        <PHONENUM DESC="Home">303-222-3333</PHONENUM>
        <EMAIL>name.two@some.com</EMAIL>
    </ENTRY>
</ADDRBOOK>
```

## Opening and closing a DOM document

Use XDOMOpen to load an XML document and build a DOM tree in memory.

```
XDOMOpen( "sample.xml", XML.FROM.FILE, mydom)
```

If the first parameter is the XML document itself, the second parameter is XML.FROM.STRING.

You can reference the DOM handle, *mydom*, later in other XML for UniVerse BASIC API calls.

When finished with an XML document, use XDOMClose to destroy the DOM tree in memory:

```
XDOMClose(mydom)
```

## Navigating the DOM tree

Use XDOMLocate to locate the context node.

```
XDOMLocate(xmlHandle, xpathString, nsMap, nodeHandle)
```

For example,

```
XDOMLocate(mydom, "/ADDRBOOK", "", myctx)
```

puts the element ADDRBOOK in variable *myctx*. You can use the context handle, which is also a node handle and more generically an xml handle, in other API calls later.

Use XDOMLocateNode to navigate the DOM tree without bothering with XPath:

```
XDOMLocateNode(nodeHandle, direction, childIndex, nodeType,  
newNodeHandle)
```

For example,

```
XDOMLocateNode(mydom, XDOM.CHILD, XDOM.FIRST.CHILD,  
XDOM.ELEMENT.NODE, thefirst)
```

puts the ADDRBOOK element, which is the first child of the root in handle *thefirst*.

And,

```
XDOMLocateNode(theFirst, XDOM.CHILD, 2, XDOM.ELEMENT.NODE,  
entrynode2)
```

puts the second ENTRY element in handle entrynode2.

Then,

```
XDOMLocateNode(entrynode2, XDOM.PREV.SIBLING, 0,  
XDOM.ELEMENT.NODE, entrynode1)
```

puts the first ENTRY element in handle entrynode1.

## Building DOM trees from scratch

There are two API calls available to build a DOM tree from scratch:

```
XDOMCreateRoot(domHandle)
```

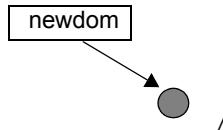
```
XDOMCreateNode(xmlHandle, nodeName, nodeValue, nodeType,  
nodeHandle)
```

XDOMCreateRoot creates a DOM tree with one root, whose type is DOCUMENT. XDOMCreateNode creates a node that bears the name, value and type you specify with the parameters.

For example,

```
XDOMCreateRoot(newdom)
```

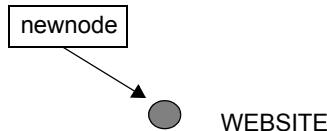
creates this tree in memory:



And

```
XDOMCreateNode(newdom, "Website", "", XDOM.ELEMENT.NODE, newnode)
```

creates a new element node with name Website, which you can reference through the handle newnode.

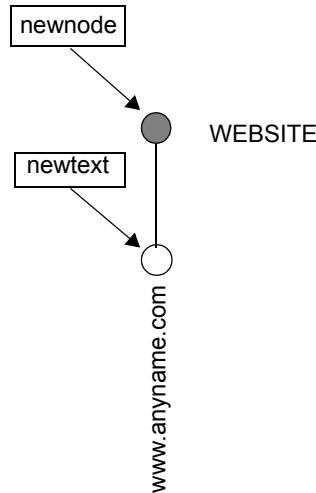


Now we create a text node

```
XDOMCreateNode(newnode, "", "www.anyname.com", XDOM.TEXT.NODE,  
newtext)
```

```
XDOMAddChild(newdom, "", "", newnode, XDOM.NODUP)  
XDOMAddChild(newdom, "/Website", "", newtext, XDOM.NODUP)
```

The subtree becomes:



## Transforming XML documents

Use XDOMTransform to transform an XML document via an XSL document. For example, using the style sheet defined in the following example, "sample.xsl":

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">

    <xsl:output method='html' indent='yes' />

    <!-- Template 1 -->
    <xsl:template match="/">
        <HTML><HEAD><TITLE>Address Book</TITLE></HEAD>
        <BODY>
            <H1>Address Book</H1>
            <xsl:apply-templates/>
        </BODY>
        </HTML>
    </xsl:template>

    <!-- Template 2 -->
    <xsl:template match="ENTRY">
        <TABLE>
            <TR>
                <TD COLSPAN='2'>
                    Contact: <xsl:value-of select='NAME' /> (ID: <xsl:value-of
select='@ID' />)
                </TD>
            </TR>
            <xsl:apply-templates select='ADDRESS|PHONENUM|EMAIL' />
        </TABLE>
        <xsl:if test='not(position()=last())'><HR/></xsl:if>
    </xsl:template>

    <!-- Template 3 -->
    <xsl:template match="ADDRESS">
        <TR><TD>Address</TD><TD><xsl:apply-templates/></TD></TR>
    </xsl:template>

    <!-- Template 4 -->
    <xsl:template match="PHONENUM">
        <TR>
            <TD><xsl:value-of select='@DESC' /> Phone Number</TD>
            <TD><xsl:apply-templates/></TD>
        </TR>
    </xsl:template>

    <!-- Template 5 -->
    <xsl:template match="EMAIL">
```

```
<TR><TD>E-mail Address</TD><TD><xsl:apply-  
templates/></TD></TR>  
</xsl:template>  
  
</xsl:stylesheet>
```

The XML for UniVerse BASIC API function XDOMTransform can transform the sample Address Book DOM document (referenced by *mydom*) into a new DOM document using the following command:

```
XDOMTransform(mydom, "sample.xsl", XML.FROM.FILE, newdom)
```

The handle newdom will point to the transformed document, which will be:

```
<HTML>
<HEAD>
<META http-equiv="Content-Type" content="text/html; charset=UTF-
8">
<TITLE>Address Book</TITLE>
</HEAD>
<BODY>
<H1>Address Book</H1>
<TABLE>
<TR>
<TD COLSPAN="2">
    Contact: Name One (ID: id1)
</TD>
</TR>
<TR>
<TD>Address</TD><TD>101 Some Way</TD>
</TR>
<TR>
<TD>Work Phone Number</TD><TD>303-111-1111</TD>
</TR>
<TR>
<TD>Fax Phone Number</TD><TD>303-111-2222</TD>
</TR>
<TR>
<TD>Pager Phone Number</TD><TD>303-111-3333</TD>
</TR>
<TR>
<TD>E-mail Address</TD><TD>name.one@some.com</TD>
</TR>
</TABLE>
<HR>
<TABLE>
<TR>
<TD COLSPAN="2">
    Contact: Name Two (ID: id2)
</TD>
</TR>
<TR>
<TD>Address</TD><TD>202 Some Way</TD>
</TR>
<TR>
<TD>Work Phone Number</TD><TD>303-222-1111</TD>
</TR>
<TR>
<TD>Fax Phone Number</TD><TD>303-222-2222</TD>
</TR>
<TR>
<TD>Home Phone Number</TD><TD>303-222-3333</TD>
</TR>
<TR>
<TD>E-mail Address</TD><TD>name.two@some.com</TD>
```

```
</TR>
</TABLE>
<HR>
</BODY>
</HTML>
```

---

## XML for UniVerse BASIC API programmatic interfaces

This section describes the XML for UniVerse BASIC API functions.

### XDOMOpen

#### Syntax

**XDOMOpen**(*xmlDocument*, *docLocation*, *domHandle*)

#### Description

The XDOMOpen function reads an *xmlDocument* and creates DOM structure. If the DTD is included in the document, UniVerse validates the document. The *xmlDocument* can be from a string or from a file, depending on the *docLocation* flag.

#### Parameters

The following table describes each parameter of the syntax.

---

Parameter	Description
<i>xmlDocument</i>	The XML document. [IN]
<i>docLocation</i>	A flag to specify whether <i>xmlDocument</i> is a string holding the XML document, or it is a file containing the XML document. Valid values are: <ul style="list-style-type: none"><li>■ XML.FROM.FILE</li><li>■ XML.FROM.STRING</li></ul> [IN]
<i>domHandle</i>	Handle to the opened DOM structure. [OUT]

---

XDOMOpen Parameters

### ***Return codes***

The return code indicating success or failure. The following table describes the value of each return code.

Return Code	Description
XML.SUCCESS	Function completed successfully.
XML.ERROR	An error occurred.
XML.INVALID.HANDLE	Invalid DOM handle passed to the function.

**XDOMOpen Return Codes**

## **XDOMCreateNode**

### ***Syntax***

**XDOMCreateNode**(*xmlhandle*, *nodeName*, *nodeValue*, *nodeType*, *nodeHandle*)

### ***Description***

The XDOMCreateNode function creates a new node, whose name and value and *nodeName* and *nodeValue*, respectively. Valid values for *nodeType* are:

- XDOM.ELEMENT.NODE
- XDOM.ATTR.NODE
- XDOM.TEXT.NODE
- XDOM.CDATA.NODE
- XDOM.ENTITY.REF.NODE
- XDOM.PROC.INST.NODE
- XDOM.COMMENT.NODE
- XDOM.DOC.NODE
- XDOM.DOC.TYPE.NODE
- XDOM.DOC.FRAG.NODE
- XDOM.NOTATION.NODE

## **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>xmlHandle</i>	Handle to the DOM structure. [IN]
<i>nodeName</i>	The name for the new node. [IN]
<i>nodeValue</i>	The value for the new node. [IN]
<i>nodeType</i>	The type of the new node. [IN]
<i>nodeHandle</i>	The handle to the new node. [OUT]

### **XDOMCreateNode Parameters**

## **Return codes**

The return code indicating success or failure. The following table describes the value of each return code.

Return Code	Description
XML.SUCCESS	Function completed successfully.
XML.ERROR	An error occurred.

### **XDOMCreateNode Return Codes**

## **XDOMCreateRoot**

### **Syntax**

**XDOMCreateRoot(*domHandle*)**

### **Description**

The XDOMCreateRoot function creates a new DOM structure with root only. You can use the result handle in other functions where a DOM handle or node handle is needed.

## **Parameters**

The following table describes the parameter of the syntax.

Parameter	Description
<i>domHandle</i>	Handle to the opened DOM structure. [OUT]
<b>XDOMCreateRoot Parameter</b>	

## **Return codes**

The return code indicating success or failure. The following table describes the value of each return code.

Return Code	Description
XML.SUCCESS	Function completed successfully.
XML.ERROR	An error occurred.
<b>XDOMCreateRoot Return Codes</b>	

## **XDOMWrite**

### **Syntax**

**XDOMWrite**(*domHandle*, *xmlDocument*, *docLocation*)

### **Description**

The XDOMWrite function writes the DOM structure to *xmlDocument*. *xmlDocument* can be a string or a file, depending on the value of the *docLocation* flag.

## **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>domHandle</i>	The handle to the opened DOM structure. [IN]
<i>xmlDocument</i>	The XML document [OUT]
<i>docLocation</i>	A flag to specify whether <i>xmlDocument</i> is an output string which should hold the XML document, or it is a file where the XML document should be written. Valid values are: <ul style="list-style-type: none"><li>■ XML.TO.FILE</li><li>■ XML.TO.STRING</li></ul> [IN]

## **Return codes**

The return code indicating success or failure. The following table describes the value of each return code.

Return Code	Description
XML.SUCCESS	Function completed successfully.
XML.ERROR	An error occurred.
XML.INVALID.HANDLE	Invalid DOM handle passed to the function.

### **XDOMWrite Return Codes**

## **XDOMClose**

### **Syntax**

**XDOMClose**(*domHandle*)

### **Description**

The XDOMClose function frees the DOM structure.

### **Parameters**

The following table describes the parameter of the syntax.

Parameter	Description
<i>domHandle</i>	Handle to the DOM structure. [IN]
<b>XDOMClose Parameter</b>	

### **Return codes**

The return code indicating success or failure. The following table describes the value of each return code.

Return Code	Description
XML.SUCCESS	Function completed successfully.
XML.ERROR	An error occurred.
XML.INVALID.HANDLE	Invalid DOM handle passed to the function.
<b>XDOMClose Return Codes</b>	

## **XDOMValidate**

### **Syntax**

**XDOMValidate**(*xmlDocument*, *docLocation*, *schFile*, *schLocation*)

### **Description**

The XDOMValidate function validates the DOM document using the schema specified by *schFile*.

## **Parameters**

The following table describes each parameter of the syntax.

<b>Parameter</b>	<b>Description</b>
<i>xmlDocument</i>	The name of the XML document. [IN]
<i>docLocation</i>	A flag to specify whether <i>xmlDocument</i> is the document itself, or the document file name. Valid values are: <ul style="list-style-type: none"><li>■ XML.FROM.FILE (default)</li><li>■ XML.FROM.STRING</li><li>■ XML.FROM.DOM</li></ul> [IN]
<i>schFile</i>	The schema file.
<i>schLocation</i>	A flag to specify whether <i>schFile</i> is the schema itself, or the schema file name. Valid values are: <ul style="list-style-type: none"><li>■ XML.FROM.FILE (default)</li><li>■ XML.FROM.STRING</li></ul> [IN]

### **XDOMValidate Parameters**

## **Return codes**

The return code indicating success or failure. The following table describes the value of each return code.

<b>Return Code</b>	<b>Description</b>
XML.SUCCESS	Function completed successfully.
XML.ERROR	An error occurred.
XML.INVALID.HANDLE	An invalid DOM handle was passed to the function.

### **XDOMValidate Return Codes**

## XDOMLocate

### Syntax

**XDOMLocate**(*xmlHandle*, *xpathString*, *nsMap*, *nodeHandle*)

### Description

The XDOMLocation function finds a starting point for relative XPath searching in context *xmlHandle* in the DOM structure. The *xpathString* should specify only one node; otherwise, this function returns an error.

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>xmlHandle</i>	A handle to the DOM structure. [IN]
<i>xpathString</i>	A string to specify the starting point. [IN]
<i>nsMAP</i>	The map of namespaces which resolve the prefixes in the <i>xpathString</i> . The format is: “xmlns=default_url xmlns:prefix1=prefix1_url xmlns:prefix2=prefix2_url  For example: “xmlns=”http://myproject.mycompany.com xmlns:a_prefix=a.mycompany.com [IN]
<i>nodeHandle</i>	Handle to the found node. [OUT]

### XDOMLocate Parameters

## **Return codes**

The return code indicating success or failure. The following table describes the value of each return code.

Return Code	Description
XML.SUCCESS	Function completed successfully.
XML.ERROR	An error occurred.
XML.INVALID.HANDLE	An invalid handle was returned to the function.

### **XDOMLocate Return Codes**



*Note:* In this document, *xmlHandle* is a generic type, it can be *domHandle* or *nodeHandle*. *DomHandle* stands for a whole document, while *nodeHandle* stands for a subtree. *DomHandle* is also a *nodeHandle*.

## **XDOMLocateNode**

### **Syntax**

**XDOMLocateNode**(*nodeHandle*, *direction*, *childIndex*, *nodeType*,  
*newNodeHandle*)

### **Description**

The XDOMLocateNode function traverses from *nodeHandle* and gets the next node according to *direction* and *childIndex*.

## **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>nodeHandle</i>	The handle to the starting node. [IN]
<i>direction</i>	Direction to traverse. Valid values are: <ul style="list-style-type: none"><li>■ XDOM.PREV.SIBLING</li><li>■ XDOM.NEXT.SIBLING</li><li>■ XDOM.NEXT.SIBLING.WITH.SAME.NAME</li><li>■ XDOM.PREV.SIBLING.WITH.SAME.NAME</li><li>■ XDOM.PARENT</li><li>■ XDOM.CHILD</li></ul> [IN]

### **XDOMLocateNode Parameters**

Parameter	Description
<i>childIndex</i>	The index in the child array. Valid values are: <ul style="list-style-type: none"><li>■ XDOM.FIRST.CHILD</li><li>■ XDOM.LAST.CHILD</li><li>■ Positive Integer</li></ul> [IN]

---

**XDOMLocateNode Parameters (Continued)**

---

---

Parameter	Description
<i>nodeType</i>	<p>The type of node to be located. Valid values are:</p> <ul style="list-style-type: none"> <li>■ XDOM.NONE</li> <li>■ XDOM.ELEMENT.NODE</li> <li>■ XDOM.ATTR.NODE</li> <li>■ XDOM.TEXT.NODE</li> <li>■ XDOM.CDATA.NODE</li> <li>■ XDOM.ENTITY.REF.NODE</li> <li>■ XDOM.ENTITY.NODE</li> <li>■ XDOM.PROC.INST.NODE</li> <li>■ XDOM.COMMENT.NODE</li> <li>■ XDOM.DOC.NODE</li> <li>■ XDOM.DOC.TYPE.NODE</li> <li>■ XDOM.DOC.FRAG.NODE</li> <li>■ XDEOM.NOTATION.NODE</li> </ul> <p>If <i>nodeType</i> is not XDOM.NONE, UniVerse uses this argument, along with <i>direction</i> and <i>childIndex</i>, to get the right typed node. For example, if <i>direction</i> is XDOM.PREV.SIBLING, and <i>nodeType</i> is XDOM.ELEMENT.NODE, UniVerse finds the element node which is the first previous sibling of <i>nodeHandle</i>. If <i>direction</i> is XDOM.CHILD, <i>childIndex</i> is XDOM.FIRST.CHILD, and <i>nodeType</i> is XDOM.ELEMENT.NODE, UniVerse finds the element node which is the first element child of <i>nodeHandle</i>. If the <i>direction</i> is XDOM.CHILD, <i>childIndex</i> is 2, and <i>nodeType</i> is XDOM.ELEMENT.NODE, UniVerse finds the element node which is the second element child of <i>nodeHandle</i>.</p> <p>When the <i>direction</i> is XDOM.NEXT.SIBLING.WITH.SAME.NAME, XDOM.PREV.SIBLING.WITH.SAME.NAME, or XDOM.PARENT, this argument is not used. [IN]</p>
<i>newNodeHandle</i>	Handle to the found node. [OUT]

---

#### XDOMLocateNode Parameters (Continued)

### ***Return codes***

The return code indicating success or failure. The following table describes the value of each return code.

<b>Return Code</b>	<b>Description</b>
XML.SUCCESS	The function completed successfully.
XML.ERROR	An error occurred.
XM.INVALID.HANDLE	An invalid handle was returned to the function.

#### **XDOMLocateNode Return Codes**

## **XDOMEvaluate**

### ***Syntax***

**XDOMEvaluate**(*xmlHandle*, *xpathString*, *nsMap*, *aValue*)

### ***Description***

The XDOMEvaluate function returns the value of the *xpathString* in the context *xmlHandle* in the DOM structure.

## **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>xmlHandle</i>	Handle to the context. [IN]
<i>xpathString</i>	Relative or absolute Xpath string. [IN]
<i>nsMap</i>	The map of namespaces which resolves the prefixes in the <i>xpathString</i> . Format is “xmlns=default_url xmlns:prefix1=prefix1_url xmlns:prefix2=prefix2_url” For example: “xmlns=http://myproject.mycompany.com xmlns:a_prefix=a.mycompany.com” [IN]
<i>aValue</i>	The value of <i>xpathString</i> . [OUT]

### **XDOMEvaluate Parameters**

## **Return codes**

The return code indicating success or failure. The following table describes the value of each return code.

Return Code	Description
XML.SUCCESS	The function completed successfully.
XML.ERROR	An error occurred.
XML.INVALID.HANDLE	An invalid DOM handle was returned to the function.

### **XDOMEvaluate Return Codes**

## XDOMRemove

### Syntax

**XDOMRemove**(*xmlHandle*, *xpathString*, *nsMap*, *attrName*, *nodeHandle*)

### Description

The XDOMRemove function finds the *xpathString* in the context *xmlHandle* in DOM structure, removes the found node or its attribute with name *attrName*.

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>xmlHandle</i>	Handle to the context. [IN]
<i>xpathString</i>	Relative or absolute Xpath string. [IN]
<i>nsMap</i>	The map of namespaces which resolve the prefixes in the <i>xpathString</i> . Format is “xmlns=default_url xmlns:prefix1_url xmlns:prefix2=prefix2_url” For example: “xmlns=http://myproject.mycompany.com xmlns:a_prefix=a.mycompany.com” [IN]
<i>attrName</i>	The attribute name. [IN]
<i>nodeHandle</i>	The removed node, if <i>nodeHandle</i> is not NULL. [OUT]

XDOMRemove Parameters

### ***Return codes***

The return code indicating success or failure. The following table describes the value of each return code.

Return Code	Description
XML.SUCCESS	The function completed successfully.
XML.ERROR	An error occurred.
XML.INVALID.HANDLE	An invalid DOM handle was returned to the function.

### **XDOMRemove Return Codes**

## **XDOMAppend**

### ***Syntax***

**XDOMAppend**(*xmlHandle*, *xpathString*, *nsMap*, *nodeHandle*, *dupFlag*)

### ***Description***

The XDOMAppend function finds the *xpathString* in the context *xmlHandle* in the DOM structure, and inserts *nodeHandle* into the DOM structure as next sibling of found node. If the inserted node type is XDOM.ATTR.NODE, this node is inserted as an attribute.

## Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>xmlHandle</i>	The handle to the context. [IN]
<i>xpathString</i>	Relative or absolute XPath string. [IN]
<i>nsMap</i>	The map of namespaces which resolve the prefixes in the <i>xpathString</i> . Format is “xmlns=default_url xmlns:prefix1=prefix1_url xmlns:prefix2=prefix2_url” For example: “xmlns=http://myproject.mycompany.com xmlns:a_prefix=a.mycompany.com” [IN]
<i>nodeHandle</i>	Handle to a DOM subtree. If <i>nodeHandle</i> points to a DOM document, all of its children are inserted, in the same order. [IN]
<i>dupFlag</i>	XDOM.DUP: Clones <i>nodeHandle</i> , and insert the duplicate node. XDOM.NODUP: Inserts the original node. The subtree is also removed from its original location. [IN]

### XDOMAppend Parameters

## Return codes

The return code indicating success or failure. The following table describes the value of each return code.

Return Code	Description
XML.SUCCESS	The function completed successfully.
XML.ERROR	An error occurred.
XML.INVALID.HANDLE	An invalid DOM handle was returned to the function.

### XDOMAppend Return Codes

## XDOMInsert

### Syntax

**XDOMInsert** (*xmlHandle*, *xpathString*, *nsMap*, *nodeHandle*, *dupFlag*)

### Description

The XDOMInsert function finds the *xpathString* in the context *xmlHandle* in the DOM structure and inserts *nodeHandle* into the DOM structure as a previous sibling of the found node. If the inserted node type is XDOM.ATTR.NODE, this node is inserted as an attribute.

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>xmlHandle</i>	The handle to the context. [IN]
<i>xpathString</i>	Relative or absolute xpath string. [IN]
<i>nsMap</i>	The map of namespaces which resolves the prefixes in the <i>xpathString</i> . Format is “xmlns=default_url xmlns:prefix1=prefix1_url xmlns:prefix2=prefix2_url” For example: “xmlns=http://myproject.mycompany.com xmlns:a_prefix=a.mycompany.com” [IN]
<i>nodeHandle</i>	The handle to a DOM subtree. If <i>nodeHandle</i> points to a DOM document, all of its children are inserted, in the same order. [IN]
<i>dupFlag</i>	XDOM.DUP: Clones <i>nodeHandle</i> , and inserts the duplicate node. XDOM.NODUP: Inserts the original node and removes the subtree from its original location.

### XDOMInsert Parameters

## ***Return codes***

The return code indicating success or failure. The following table describes the value of each return code.

<b>Return Code</b>	<b>Description</b>
XML.SUCCESS	The function completed successfully.
XML.ERROR	An error occurred.
XML.INVALID.HANDLE	An invalid DOM handle was returned to the function.

### **XDOMInsert Return Codes**

## **XDOMReplace**

### **Syntax**

**XDOMReplace**(*xmlHandle*, *xpathString*, *nsMap*, *nodeHandle*, *dupFlag*)

### **Description**

The XDOMReplace function finds the *xpathString* in the context *xmlHandle* in the DOM structure, and replaces the found node with *nodeHandle*.

## **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>xmlHandle</i>	The handle to the context. [IN]
<i>xpathString</i>	Relative or absolute Xpath string. [IN]
<i>nsMap</i>	The map of namespaces which resolve the prefixes in the xpathString. Format is “xmlns=default_url xmlns:prefix1=prefix1_url xmlns:prefix2=prefix2_url” For example: “xmlns=http://myproject.mycompany.com xmlns:a_prefix=a.mycompany.com” [IN]
<i>nodeHandle</i>	Handle to a DOM subtree. If nodeHandle points to a DOM document, the found node is replaced by all of nodeHandle children, which are inserted in the same order. [IN]
<i>dupFlag</i>	XDOM.DUP: Clones nodeHandle, and replaces it with the duplicate node. XDOM.NODUP: Replaces with the original node. The subtree is also removed from its original location. [IN]

### **XDOMReplace Parameters**

## **Return codes**

The return code indicating success or failure. The following table describes the value of each return code.

Return Code	Description
XML.SUCCESS	The function completed successfully.
XML.ERROR	An error occurred.
XML.INVALID.HANDLE	An invalid DOM handle was returned to the function.

### **XDOMReplace Return Codes**

## XDOMAddChild

### Syntax

**XDOMAddChild**(*xmlHandle*, *xpathString*, *nsMap*, *nodeHandle*, *dupFlag*)

### Description

The XDOMAddChild function finds the *xpathString* in the context *xmlHandle* in the DOM structure and inserts a node *nodeHandle* as the last child of the found node. If the inserted node type is XDOM.ATTR.NODE, this node is inserted as an attribute.

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>xmlHandle</i>	The handle to the context. [IN]
<i>xpathString</i>	Relative or absolute Xpath string. [IN]
<i>nsMap</i>	The map of namespaces which resolve the prefixes in the xpath string. Format is “xmlns=default_url xmlns:prefix1=prefix1_url xmlns:prefix2=prefix2_url” For example: “xmlns=http://myproject.mycompany.com xmlns:a_prefix=a.mycompany.com” [IN]
<i>nodeHandle</i>	Handle to a DOM subtree. If <i>nodeHandle</i> points to a DOM document, all of its children are inserted, in the same order. [IN]
<i>dupFlag</i>	XDOM.DUP: Clones <i>nodeHandle</i> , and inserts the duplicate node. XDOM.NODUP: Inserts the original node. The subtree is also removed from its original location. [IN]

### XDOMAddChild Parameters

The return code indicating success or failure. The following table describes the value of each return code.

Return Code	Description
XML.SUCCESS	The function completed successfully.
XML.ERROR	An error occurred.
XML.INVALID.HANDLE	An invalid DOM handle was returned to the function.

#### XDOMAddChild Return Codes

## XDOMClone

### Syntax

XDOMClone (*xmlHandle*, *newXmlHandle*, *depth*)

### Description

The XDOMClone function duplicates the DOM subtree specified by *xmlHandle* to a new subtree *newXmlHandle*. The duplicate node has no parent (*parentNode* returns null.).

Cloning an element copies all attributes and their values, including those generated by the XML processor, to represent defaulted attributes, but this method does not copy any text it contains unless it is a deep clone, since the text is contained in a child text node. Cloning any other type of node simply returns a copy of this node.

## **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>xmlHandle</i>	Handle to the DOM subtree. [IN]
<i>newXmlHandle</i>	Handle to the new DOM subtree. [IN]
<i>depth</i>	XDOM.FALSE: Clone only the node itself. XDOM.TRUE: Recursively clone the subtree under the specified node. [IN]

### **XDOMClone Parameters**

## **Return codes**

The return code indicating success or failure. The following table describes the value of each return code.

Return Code	Description
XML.SUCCESS	The function completed successfully.
XML.ERROR	An error occurred.
XML.INVALID.HANDLE	An invalid DOM handle was returned to the function.

### **XDOMClone Return Codes**

## **XDOMTransform**

### **Syntax**

**XDOMTransform**(*domHandle*, *styleSheet*, *ssLocation*, *outDomHandle*)

### **Description**

The XDOMTransform function transforms input DOM structure using the style sheet specified by *styleSheet* to output DOM structure.

## **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>domHandle</i>	Handle to the DOM structure. [IN]
<i>styleSheet</i>	Handle to the context [IN]
<i>ssLocation</i>	A flag to specify whether styleSheet contains style sheet itself, or is just the style sheet file name. Value values are: <ul style="list-style-type: none"><li>■ XML.FROM.FILE (default)</li><li>■ XML.FROM.STRING</li></ul> [IN]
<i>outDomHandle</i>	Handle to the resulting DOM structure. [OUT]

### **XDOMTransform Parameters**

## **Return codes**

The return code indicating success or failure. The following table describes the value of each return code.

Return Code	Description
XML.SUCCESS	The function completed successfully.
XML.ERROR	An error occurred.
XML.INVALID.HANDLE	An invalid DOM handle was returned to the function.

### **XDOMTransform Return Codes**

## **XDOMGetNodeName**

### **Syntax**

**XDOMGetNodeName**(*nodeHandle*, *nodeName*)

### ***Description***

The XDOMGetNodeName function returns the node name.

### ***Parameters***

The following table describes each parameter of the syntax.

Parameter	Description
<i>nodeHandle</i>	Handle to the DOM node. [IN]
<i>nodeName</i>	String to store the node name. [OUT]
<b>XDOMGetNodeName Parameters</b>	

### ***Return codes***

The return code indicating success or failure. The following table describes the value of each return code.

Return Code	Description
XML.SUCCESS	The function completed successfully.
XML.ERROR	An error occurred.
XML.INVALID.HANDLE	An invalid DOM handle was returned to the function.
<b>XDOMGetNodeName Return Codes</b>	

## **XDOMGetValue**

### ***Syntax***

**XDOMGetValue**(*nodeHandle*, *nodeValue*)

### ***Description***

The XDOMGetValue returns the node value.

## **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
nodeHandle	The handle to the DOM node. [IN]
nodeValue	The string to hold the node value. [OUT]

**XDOMGetNodeValue Parameters**

## **Return codes**

The return code indicating success or failure. The following table describes the value of each return code.

Return Code	Description
XML.SUCCESS	The function completed successfully.
XML.ERROR	An error occurred.
XML.INVALID.HANDLE	An invalid DOM handle was returned to the function.

**XDOMGetNodeValue Return Codes**

## **XDOMGetNodeType**

### **Syntax**

**XDOMGetNodeType**(*nodeHandle*, *nodeType*)

### **Description**

The XDOMGetNodeType function returns the node type.

## **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>nodeHandle</i>	The handle to the DOM node. [IN]
<i>nodeType</i>	An integer to store the node type. [OUT]
<b>XDOMGetNodeType Parameters</b>	

## **Return codes**

The return code indicating success or failure. The following table describes the value of each return code.

Return Code	Description
XML.SUCCESS	The function completed successfully.
XML.ERROR	An error occurred.
XML.INVALID.HANDLE	An invalid DOM handle was returned to the function.
<b>XDOMGetNodeType Return Codes</b>	

## **XDOMGetAttribute**

### **Syntax**

**XDOMGetAttribute**(*nodeHandle*, *attrName*, *nodeHandle*)

### **Description**

The XDOMGetAttribute function returns the node's attribute node, whose attribute name is *attrName*.

## **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>nodeHandle</i>	Handle to the DOM node. [IN]
<i>attrName</i>	Attribute name. [IN]
<i>nodeHandle</i>	Handle to the found attribute node. [OUT]

### **XDOMGetAttribute Parameters**

## **Return codes**

The return code indicating success or failure. The following table describes the value of each return code.

Return Code	Description
XML.SUCCESS	The function completed successfully.
XML.ERROR	An error occurred.
XML.INVALID.HANDLE	An invalid DOM handle was returned to the function.

### **XDOMGetAttribute Return Codes**

## **XDOMGetOwnerDocument**

### **Syntax**

**XDOMGetOwnerDocument**(*nodeHandle*, *domHandle*)

### **Description**

The XDOMGetOwnerDocument function returns the DOM handle to which *nodeHandle* belongs.

## **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>nodeHandle</i>	Handle to the DOM node. [IN]
<i>domHandle</i>	Handle to the DOM structure. [OUT]
<b>XDOMGetOwnerDocument Parameters</b>	

## **Return codes**

The return code indicating success or failure. The following table describes the value of each return code.

Return Code	Description
XML.SUCCESS	The function completed successfully.
XML.ERROR	An error occurred.
XML.INVALID.HANDLE	An invalid DOM handle was returned to the function.
<b>XDOMGetOwnerDocument Return Codes</b>	

## **XDOMGetUserData**

### **Syntax**

**XDOMGetUserData(*nodeHandle*, *userData*)**

### **Description**

The XDOMGetUserData function returns the user data associated with the node.

## **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>nodeHandle</i>	The handle to the DOM node. [IN]
<i>userData</i>	String to hold the user data. [OUT]
<b>XDOMGetUserData Parameters</b>	

## **Return codes**

The return code indicating success or failure. The following table describes the value of each return code.

Return Code	Description
XML.SUCCESS	The function completed successfully.
XML.ERROR	An error occurred.
XML.INVALID.HANDLE	An invalid DOM handle was returned to the function.
<b>XDOMGetUserData Return Codes</b>	

## **XDOMSetNodeValue**

### **Syntax**

**XDOMSetNodeValue(*nodeHandle*, *nodeValue*)**

### **Description**

XDOMSetNodeValue sets the node value.

## **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>nodeHandle</i>	The handle to the DOM node. [IN]
<i>nodeValue</i>	String to hold the node value. [IN]
<b>XDOMSetNodeValue Parameters</b>	

## **Return codes**

The return code indicating success or failure. The following table describes the value of each return code.

Return Code	Description
XML.SUCCESS	The function completed successfully.
XML.ERROR	An error occurred.
XML.INVALID.HANDLE	An invalid DOM handle was returned to the function.
<b>XDOMSetNodeValue Return Codes</b>	

## **XDOMSetUserData**

### **Syntax**

**XDOMSetUserData**(*nodeHandle*, *userData*)

### **Description**

The XDOMSetUserData function sets the user data associated with the node.

## **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>nodeHandle</i>	Handle to the DOM node. [IN]
<i>userData</i>	String to hold the user data. [IN]

**XDOMSetUserData Parameters**

## **Return codes**

The return code indicating success or failure. The following table describes the value of each return code.

Return Code	Description
XML.SUCCESS	The function completed successfully.
XML.ERROR	An error occurred.
XML.INVALID.HANDLE	An invalid DOM handle was returned to the function.

**XDOMSetUserData Return Codes**

## **XMLGetError**

### **Syntax**

**XMLGetError(errorCode, errorMessage)**

### **Description**

The XMLGetError function returns the error code and error message after the previous XML API failed.

## **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>errorCode</i>	The error code. [OUT]
<i>errorMessage</i>	The error message. [OUT]
<b>XMLGetError Parameters</b>	

## **Return codes**

The return code indicating success or failure. The following table describes the value of each return code.

Return Code	Description
XML.SUCCESS	The function completed successfully.
XML.ERROR	An error occurred.
<b>XDOMGetError Return Codes</b>	

# Chapter 8: Data transfer between XML documents and UniVerse files

Transferring Data from XML to the database . . . . .	8-3
Populating the database. . . . .	8-11
Populating the database from TCL. . . . .	8-11
Populating the database using the UniVerse BASIC XMAP API. . . . .	8-13
The XMAP API. . . . .	8-14
XMAPOpen function . . . . .	8-14
XMAPClose function . . . . .	8-16
XMAPCreate function . . . . .	8-16
XMAPReadNext function . . . . .	8-17
XMAPAppendRec function . . . . .	8-18
XMAPToXMLDoc function . . . . .	8-19
Examples . . . . .	8-20
Transferring data from the database to XML. . . . .	8-23
Creating an XML document from TCL . . . . .	8-23

The XMLDB data transfer capability extends the existing XML support in UniVerse. It consists of the data transfer utilities and the UniVerse BASIC XMAP API. The data transfer utilities consist of two TCL commands, XML.TODB and DB.TOXML, and two UniVerse BASIC functions, XMLTODB() and DBTOXML(). The UniVerse BASIC XMAP API consists of the following six UniVerse BASIC functions:

- XMAPOpen()
- XMAPClose()
- XMAPCreate()
- XMAPReadNext()
- XMAPAppendRec()
- XMAPToXMLDoc()

These TCL commands and UniVerse BASIC functions enable data transfer between XML documents and UniVerse files.

## Transferring Data from XML to the database

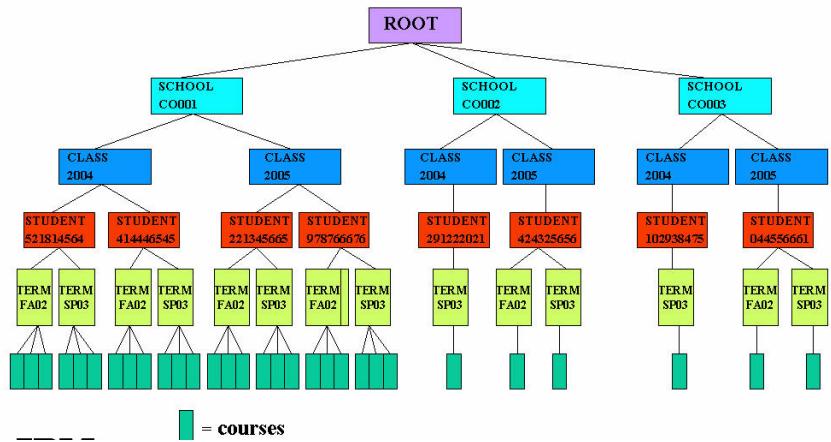
You can store data contained in an XML document in UniVerse files. This process is called shredding. You can also create XML documents from data contained in UniVerse files.

You can transfer an unlimited levels of XML data to UniVerse files, although one UniVerse file can contain no more than three levels of data. If you are extracting more than three levels of data, you must write the data to more than one UniVerse file.

The following example shows a sample XML document containing information about students in different school districts:

```
<?xml version="1.0" ?>
<ROOT>
  <SCHOOL SCHOOLID="CO001" NAME="Fairview"
    DISTRICT="BVSD">
    <CLASS CLASSOF="2004"
      <STUDENT ID="521814564" NAME="Harry Smith" DOB="1985-02-08">
        <TERM SEMESTER="FA02">
          <COURSE NAME="MA130" GRADE="A" />
          <COURSE NAME="CH100" GRADE="B" />
          <COURSE NAME="PY100" GRADE="B" />
        </TERM>
        <TERM SEMESTER="SP03">
          <COURSE NAME="MA131" GRADE="B" />
          <COURSE NAME="CH101" GRADE="B" />
          <COURSE NAME="PE220" GRADE="A" />
        </TERM>
      </STUDENT>
      <STUDENT ID="414446545" NAME="Karl Offenbach" DOB="1984-12-
26">
        ...
      </STUDENT>
    </CLASS>
    <CLASS CLASSOF="2005">
      ...
    </SCHOOL>
  <SCHOOL SCHOOLID="CO002" NAME="Golden" DISTRICT="ACSD"
    <CLASS CLASSOF="2004">
      <STUDENT ID="291222021" NAME="Jojo Smith" DOB="1985-08-06">
        <TERM SEMESTER="SP03">
          <COURSE NAME="FR101" GRADE="B" />
        </TERM>
      </STUDENT>
    </CLASS>
    <CLASS CLASSOF="2005">
      <STUDENT ID="424325656" NAME="Sally Martin" DOB="1985-12-01"
        <TERM SEMESTER="FA02">
          <COURSE NAME="PY100" GRADE="C" />
          <COURSE NAME="PE100" GRADE="C" />
        </TERM>
      </STUDENT>
    </CLASS>
  <SCHOOL SCHOOLID="CO003" NAME="Cherry Creek" DISTRICT="CCSD"
    ...
  </SCHOOL>
</ROOT>
```

This document can be represented by the following XML document tree:



The following is an overview of the steps required to retrieve data from an XML database and store it in UniVerse files.

### ***Generate database schema***

First, you must understand the structure of the data in the incoming XML document. You can understand this structure by reviewing the DTD or Schema of the XML document.

### ***Create UniVerse data files***

After you review the DTD or Schema from the incoming XML document, you must create the corresponding UniVerse data and dictionary files, and create a dictionary record for each element or attribute in the corresponding XML document.

If you want to transfer all the data from this sample XML document to UniVerse, you must deal with five levels of nested elements, SCHOOL, CLASS, STUDENT, TERM and COURSES. Therefore, you have to map this XML document to two UniVerse files. Logically, you would map the bottom three elements, STUDENT, TERM, and COURSES, to one UniVerse file. We will call this file the STUDENT file. We will map the top two elements, SCHOOL and CLASS, to another UniVerse file. We will call this file the SCHOOL file. The ID, NAME, and DOB attributes of the element STUDENT will map to singlevalued fields in the STUDENT file, the attribute SEMESTER of the subelement TERM will map to a multivalued field, and the attributes NAME and GRADE of the third-level nested element COURSES will map to multivalued fields as well. However, we will separate the fields that correspond to the NAME and GRADE attributes with subvalue marks.

Since the STUDENT file fields corresponding to XML attributes SEMESTER, NAME, and GRADE are all related, they will be combined into one association, called CGA.

The element CLASS serves as a link between the two UniVerse files, and therefore the field CLASS\_OF appears in both UniVerse files.

The contents of the dictionary for each file follows:

```
DICT STUDENT      01:25:17pm  16 Sep 2003  Page     1
                                         Type &
Field..... Field. Field..... Conversion.. Column..... Output Depth &
Name..... Number Definition... Code..... Heading..... Format Assoc..
@ID       D   0                      STUDENT      10L   S
NAME      D   1                      Name        10L   S
DOB       D   2                      D2          DOB        10L   S
CLASS_OF  D   3                      Class Of    10L   S
SEMESTER  D   4                      Semester     10L   M CGA
COURSE_NBR D   5                      Course No   10L   M CGA
COURSE_GRD D   6                      Grade       10L   M CGA
7 records listed.
>

DICT SCHOOL      01:27:14pm  16 Sep 2003  Page     1
                                         Type &
Field..... Field. Field..... Conversion.. Column..... Output Depth &
Name..... Number Definition... Code..... Heading..... Format Assoc..
@ID       D   0                      SCHOOL       10L   S
SCHOOLID D   0                      SchoolId    10L   S
SCHOOL_NAME D  1                      Name        10L   S
SCHOOL_DISTRIC D  2                  District    10L   S
T
CLASS_OF  D   3                      Class Of    10L   S
5 records listed.
```

## Create the U2XMAP file

The rules for transferring data between an XML document and database files are recorded in a separate file, referred to as the U2XMAP file. This file contains such information as the starting node of the XML document, names and relationships of database files that are being used to exchange data with a specified XML document, the mapping of XML attribute names to database field names, and other optional information, such as the mapping of NULL values and date format conversions.

The following example illustrates a U2XMAP:

```
<?xml version="1.0" ?>
<!--!DOCTYPE U2XMAP SYSTEM "U2XMAP.DTD"    --&gt;
&lt;U2XMAP Version="1.0" Name="XMAP1"&gt;
  &lt;!-- Table/Class Map    --&gt;
  &lt;TABLECLASSMAP MapName="M1" StartNode="/ROOT/SCHOOL" TableName="SCHOOL"&gt;
    &lt;ColumnMap Node="@SCHOOLID" Column="SCHOOLID" /&gt;
    &lt;ColumnMap Node="@NAME" Column="SCHOOL_NAME" /&gt;
    &lt;ColumnMap Node="@DISTRICT" Column="SCHOOL_DISTRICT" /&gt;
    &lt;ColumnMap Node="CLASS, @CLASSOF" Column="CLASS_OF" /&gt;
    &lt;TableMap Node="CLASS/STUDENT" MapName="M2" /&gt;
  &lt;/TABLECLASSMAP&gt;
  &lt;TABLECLASSMAP MapName="M2" StartNode="CLASS/STUDENT" TableName="STUDENT"&gt;
    &lt;ColumnMap Node="@ID" Column="@ID" /&gt;
    &lt;ColumnMap Node="@NAME" Column="NAME" /&gt;
    &lt;ColumnMap Node="@DOB" Column="DOB" /&gt;
    &lt;ColumnMap Node="TERM, @SEMESTER" Column="SEMESTER" /&gt;
    &lt;ColumnMap Node="TERM, COURSES, @NAME" Column="COURSE_NBR" /&gt;
    &lt;ColumnMap Node="TERM, COURSES, @GRADE" Column="COURSE_GRD" /&gt;
  &lt;/TABLECLASSMAP&gt;</pre>
```

Each TABLECLASSMAP element defines where to find the data in the XML document, and where to place it in the UniVerse data file based on the dictionary definition of the field.

Syntax:

```
TABLECLASSMAP MapName = "xx" StartNode = "startnode"  
TableName = "UniVerse file name"
```

The following table describes each parameter of the syntax:

Parameter	Description
MapName	The name of the relationship between the portion of the XML document that starts with StartNode and the UniVerse data file.
StartNode	The XPath expression defining the starting position in the XML document.
TableName	The name of the target UniVerse file.

#### TABLECLASSMAP Parameters

To map a particular XML attribute to a UniVerse field, use the ColumnMap element.

Syntax:

```
ColumnMap Node=XPath expression, Column = UniVerse record field
```

The ColumnMap node defines the location of the node in the XML document. The ColumnMap Column defines the field in the UniVerse file to which you want to map the XML data. The UniVerse file must exist, and the dictionary record for the field must be defined.

#### ***Mapping XML data to multivalued fields***

If you want to map an XML attribute to a multivalued field in the UniVerse record, specify a comma (",") before the name of the XML attribute, as shown in the following example:

```
ColumnMap Node = "CLASS, @CLASSOF" Column = "CLASS_OF"
```

If you want the values of the corresponding UniVerse data files to be separated by subvalue marks, such as COURSE\_NBR and COURSE\_GRADE in the STUDENT file, specify a comma before the attribute of the next level subelement and another comma before the attribute of the next level subelement, as shown in the following example:

```
ColumnMap Node="TERM, COURSES, @NAME"  
Column="COURSE_NBR"  
ColumnMap Node="TERM, COURSES, @GRADE"  
Column="COURSE_GRD"
```

### ***Defining a map relationship***

If you are mapping the XML attributes to more than one UniVerse data file, you must define a dependent map using the TableMap Node element.

```
TableMap Node="CLASS/STUDENT" MapName="M2"
```

In this example, MapName M2 is defined within the MapName M1 element as a dependent map to M1.

```
<TABLECLASSMAP MapName="M1" StartNode="/ROOT/SCHOOL" TableName="SCHOOL">  
    <ColumnMap Node="@SCHOOLID" Column="SCOOLID" />  
    <ColumnMap Node="@NAME" Column="SCHOOL_NAME" />  
    <ColumnMap Node="@DISTRICT" Column="SCHOOL_DISTRICT" />  
    <ColumnMap Node="CLASS, @CLASSOF" Column="CLASS_OF" />  
    <TableMap Node = "CLASS/STUDENT" MapName="M2" />  
</TABLECLASSMAP>
```

### ***Defining related tables***

If you are mapping more than three levels of data, you must map the data to more than one UniVerse file, since a UniVerse file can support no more than three levels of data. In the U2XMAP file, you define the files that are related to each other using the RelatedTable element.

Use the MapParentKey element to define the parent file (the file corresponding to the top portion of the XML subtree being transformed). Use the MapChildKey to define each child file of the parent file, as shown in the following example:

```
<RelatedTable>
    <MapParentKey TableName="SCHOOL" Column="CLASS_OF" Key
Generate="No" />
    <MapChildKey TableName="STUDENT" Column="CLASS_OF" />
</RelatedTable>
```

In this example, SCHOOL is the parent UniVerse file which contains one child file, STUDENT. You must define a field that appears in both UniVerse files using the Column element. In this case, CLASS\_OF appears in both the SCHOOL and STUDENT files.

The KeyGenerate element determines if UniVerse generates the parent/child key or not.

## **Populating the database**

After you define the U2XMAP file, you can populate the UniVerse database from TCL or UniVerse BASIC.

### **Populating the database from TCL**

Use the XML.TODB command to populate the UniVerse database from TCL.

Syntax:

```
XML.TODB <XML Document> <U2XMAP File>
```

The following example assumes that the XML document STUDENT.XML and the U2XMAP STUDENT.MAP are located in the &XML& file.

#### **XML.TODB STUDENT.XML STUDENT.MAP**

##### **LIST SCHOOL**

SCHOOL.....	Name.....	District.....	Class Of...
CO001	Fairview	BVSD	2004 2005
CO002	Golden	ACSD	2004 2005
CO003	Cherry Creek	CCSD	2004 2005

##### **LIST STUDENT**

STUDENT.....	Name.....	DOB...	Class Of	Semester..	Course NO.	Grade
414446545	Karl Offenbach	24 DEC 84	2004	FA02	HY104	D
					MA101	C
					FR100	C
				SP03	HY105	B
					MA102	C
					FR101	C
4243255656	Sally Martin	01 DEC 85	2005	FA02	PY100	C
				.....		

## Populating the database using the UniVerse BASIC XMLTODB() function

You can also populate the UniVerse database by calling the UniVerse BASIC XMLTODB function. XMLTODB does the same thing as the TCL XML.TODB command. If you want to transform specific data, use the XMAP API.

Syntax:

```
XMLTODB(xml_document, doc_flag, u2xmapping_rules, u2xmap_flag,  
status)
```

The following table describes each parameter of the syntax.

Parameter	Description
<i>xml_document</i>	The name of the XML document.
<i>doc_flag</i>	A flag defining the type of <i>xml_document</i> . Valid values are: <ul style="list-style-type: none"><li>■ XML.FROM.DOM - <i>xml_document</i> is a DOM handle.</li><li>■ XML.FROM.FILE - <i>xml_document</i> is a file name.</li><li>■ XML.FROM.STRING - <i>xml_document</i> is the name of a variable containing the XML document</li></ul>
<i>u2xmapping_rules</i>	The mapping rules for the XML document.
<i>u2xmap_flag</i>	A flag indicating if the mapping file is the U2XMAP file itself or a string located within the UniVerse BASIC program. Valid values are: <ul style="list-style-type: none"><li>■ XMAP.FROM.FILE - the mapping rules are contained in a U2XMAP file.</li><li>■ XMAP.FROM.STRING - <i>u2xmap_flag</i> is the name of the variable containing the mapping rules.</li></ul>
<i>Status</i>	The return status.

### XMAPOpen Parameters

## **Populating the database using the UniVerse BASIC XMAP API**

While the TCL command XML.TODB and the UniVerse BASIC function XMLTODB() provide easy ways of transferring data from an XML document to a set of related database files, you may want to have greater control over which part of the XML document you want to use for transferring data. For example, neither XML.TODB or XMLTODB() let you start the data transfer from a particular sibling of the start node. An example of such finer control is transferring only the second school data and its dependent subtree to the database from the sample XML document. You can accomplish this using a combination of the DOM API functions and the XMAP API functions.

In order to provide a record-by-record mapping between the XML document and the corresponding UniVerse files, the UniVerse engine generates an internal structure, called U2XMAP dataset. This internal structure contains information about the mapped XML elements and attributes, as well as how they relate to the fields in the corresponding UniVerse files. The U2XMAP dataset is not directly accessible to Basic programs, but instead referenced by its handle, called U2XMAP dataset handle.

## The XMAP API

The UniVerse XMAP API consists of the following UniVerse BASIC functions:

- XMAPOpen()
- XMAPClose()
- XMAPCreate()
- XMAPReadNext()
- XMAPAppendRec()
- XMAPToXMLDoc()

### XMAPOpen function

The XMAPOpen function opens an XML document as a U2XMAP data set.

*Syntax:*

XMAPOpen(*xml\_document, doc\_flag, u2xmapping\_rules, u2xmap\_flag,*  
*XMAPhandle*)

### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>xml_document</i>	The name of the XML document.
<i>doc_flag</i>	A flag defining the type of <i>xml_document</i> . Valid values are: <ul style="list-style-type: none"><li>■ XML.FROM.DOM - <i>xml_document</i> is a DOM handle.</li><li>■ XML.FROM.FILE - <i>xml_document</i> is a file name.</li><li>■ XML.FROM.STRING - <i>xml_document</i> is the name of variable containing the XML document.</li></ul>
<i>u2xmapping_rules</i>	The name of the U2XMAP file, or the UniVerse BASIC variable containing the XML to Database mapping rules.
<i>u2xmap_flag</i>	A flag indicating if the mapping file is the U2XMAP file itself or a string located within the UniVerse BASIC program. Valid values are: <ul style="list-style-type: none"><li>■ XMAP.FROM.FILE - the mapping rules are contained in a U2XMAP file.</li><li>■ XMAP.FROM.STRING - <i>u2xmap_flag</i> is the name of the variable containing the mapping rules.</li></ul>
<i>XMAPhandle</i>	The handle to the XMAP dataset.

### XMAPOpen Parameters

### Return values

The following table describes the return values for the XMAPOpen function.

Return Value	Description
XML_SUCCESS	The XML document was opened successfully.
XML_ERROR	An error occurred opening the XML document.

### XMAPOpen Return Values

## XMAPClose function

The XMAPClose function closes the U2XMAP dataset handle and frees all related structures and memory.

### Syntax

XMAPClose(*XMAPhandle*)

where *XMAPhandle* is the handle to the U2XMAP dataset.

### Return values

The following table describes the return values from the XMAPClose function.

Return Value	Description
XML_SUCCESS	The XML document was closed successfully.
XML_ERROR	An error occurred closing the XML document.
XML_INVALID_HANDLE	The XMAP dataset was invalid.

### XMAPClose Return Values

## XMAPCreate function

The XMAPCreate function creates an empty XML document for transferring data from the UniVerse database to XML according the mapping rules you define.

### Syntax

XMAPCreate(*u2xmapping\_rules*, *mapping\_flag*, *XMAPhandle*)

### *Parameters*

The following table describes each parameter of the syntax.

Parameter	Description
<i>u2xmapping_rules</i>	The name of the U2XMAP file, or the UniVerse BASIC variable containing the XML to Database mapping rules.
<i>mapping_flag</i>	A flag indicating if the mapping file is the U2XMAP file itself or a string located within the UniVerse BASIC program. Valid values are: <ul style="list-style-type: none"><li>■ XMAP.FROM.FILE - the mapping rules are contained in a U2XMAP file.</li><li>■ XMAP.FROM.STRING - <i>u2xmapping_rules</i> is the name of the variable containing the mapping rules.</li></ul>
<i>XMAPHandle</i>	The handle to the XMAP dataset.

### **XMAPCreate Parameters**

### *Return values*

The following table describes the return values for the XMAPCreate function.

Return Value	Description
XML_SUCCESS	The XML document was opened successfully.
XML_ERROR	An error occurred opening the XML document.
XML_INVALID_HANDLE	The XMAP dataset was invalid.

### **XMAPCreate Return Values**

## **XMAPReadNext function**

The XMAPReadNext function retrieves the next record from the U2XMAP dataset and formats it as a record of the UniVerse file that is being mapped.

### *Syntax*

XMAPReadNext(*XMAPHandle*, *file\_name*, *record*)

### *Parameters*

The following table describes each parameter of the syntax.

Parameter	Description
<i>XMAPhandle</i>	The U2XMAP dataset handle.
<i>file_name</i>	The name of the UniVerse file that is being mapped in the U2XMAP dataset.
<i>record</i>	The data record formatted according to the dictionary record of the file.

### **XMAPReadNext Parameters**

### *Return values*

The following table describes the return values for the XMAPReadNext function.

Return Value	Description
XML_SUCCESS	The XMAPReadNext was executed successfully.
XML_ERROR	Error in executing XMAPReadNext.
XML_INVALID_HANDLE	U2 XMAP dataset handle was invalid.
XML_EOF	The end of the U2XMAP dataset has been reached.

### **XMAPReadNext Return Values**

## **XMAPAppendRec function**

The XMAPAppendRec function formats the specified record from the UniVerse file as a U2XMAP dataset record and appends it to the U2XMAP dataset.

### *Syntax*

XMAPAppendRec(*XMAPhandle*, *file\_name*, *record*)

### *Parameters*

The following table describes each parameter of the syntax.

Parameter	Description
<i>XMAPhandle</i>	The handle to the U2XMAP dataset.
<i>file_name</i>	The name of the UniVerse file that is being mapped in the U2 XMAP dataset
<i>record</i>	The data record formatted according to the dictionary record of the UniVerse file.

#### **XMAPAppendRec Parameters**

### *Return values*

The following table describes the return values of the XMAPAppendRec function.

Return Value	Description
XML_SUCCESS	The XML document was opened successfully.
XML_ERROR	An error occurred opening the XML document.
XML_INVALID_HANDLE	The XMAP dataset was invalid.

#### **XMAPAppendRec Return Values**

## **XMAPToXMLDoc function**

The XMAPToXMLDoc function generates an XML document from the data in the U2XMAP dataset using the mapping rules you define. The XML document can be either an XML DOM handle or an XML document. UniVerse writes the data to a file or a UniVerse BASIC variable.

### *Syntax*

`XMAPToXMLDoc(XMAPhandle, xmlfile, doc_flag)`

### *Parameters*

The following table describes each parameter of the syntax.

Parameter	Description
<i>XMAPhandle</i>	The handle to the U2XMAP dataset.
<i>xmlfile</i>	The name of the XML file, or the name of a UniVerse BASIC variable to hold the XML document.
<i>doc_flag</i>	Indicates where to write the XML document. Valid values are: <ul style="list-style-type: none"><li>■ XML.TO.DOM - Writes the XML document to an XML DOM handle.</li><li>■ XML.TO.FILE - Writes the XML document to a file.</li><li>■ XML.TO.STRING - Writes the XML document to a UniVerse BASIC variable.</li></ul>

### **XMAPToXMLDoc Parameters**

### *Return values*

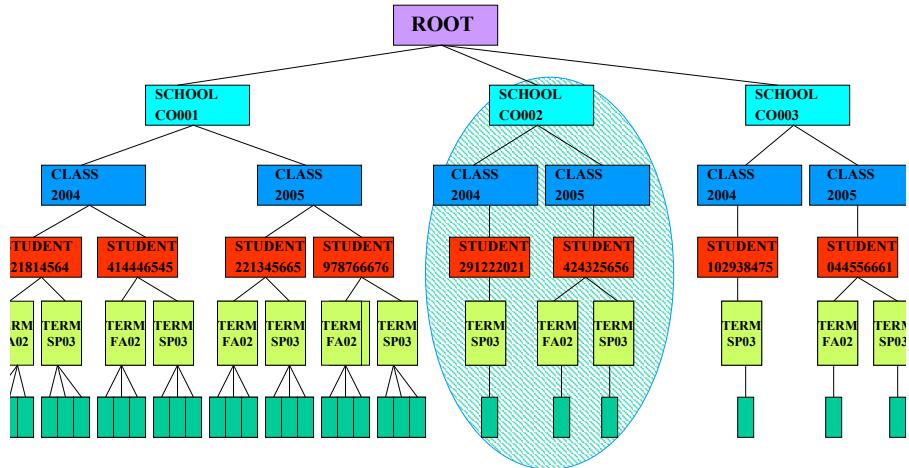
The following table describes the return values of the XMAPToXMLDoc function.

Return Value	Description
XML_SUCCESS	The XML document was opened successfully.
XML_ERROR	An error occurred opening the XML document.
XML_INVALID_HANDLE	The XMAP dataset was invalid.

### **XMAPToXMLDoc Return Values**

## **Examples**

The following example illustrates an XMLDOM tree containing information for three schools:



We will use the following U2XMAP to transfer the data to the UniVerse database:

```

<TABLECLASSMAP MapName="M2" StartNode="CLASS/STUDENT" TableName="STUDENT">
    <ColumnMap Node="@ID" Column="@ID" />
    <ColumnMap Node="@NAME" Column="NAME" />
    <ColumnMap Node="@DOB" Column="DOB" />
    <ColumnMap Node="TERM, @SEMESTER" Column="SEMESTER" />
    <ColumnMap Node="TERM, COURSES, @NAME" Column="COURSE_NBR" />
    <ColumnMap Node="TERM, COURSES, @GRADE" Column="COURSE_GRD" />
</TABLECLASSMAP>

```

The following UniVerse BASIC program segment illustrates extracting data for only SCHOOL CO002 to the STUDENT file in the UniVerse database:

```
$INCLUDE UNIVERSE.INCLUDE XML.H

*Parse XML document and build DOM tree in memory
STATUS = XDOMOpen("STUDENT.XML", XML.FROM.FILE, domH)

*Position at a specific node
STATUS = XDOMLocate(domH, "/ROOT/SCHOOL[2]", "", domHandle)

*Open XMAP dataset for reading
STATUS = XMAPOpen(domHandle, XML.FROM.DOM, "STUDENT.MAP",
XML.FROM.FILE, Xfile)

OPEN "STUDENT" TO F1 ELSE STOP "Error opening file STUDENT"

*Read records from XMAP dataset, write to STUDENT file
MOREDATA = 1
LOOP
    STATUS = XMAPReadNext(Xfile, "STUDENT", RECORD)
    IF STATUS = XML.EOF THEN
        MOREDATA = 0
    END
    WHILE MOREDATA DO
        ID = RECORD<1>
        REC = FIELD(RECORD, @FM, 2, 999)
        WRITE REC TO F1, ID ELSE STOP "Write to file STUDENT failed"
    REPEAT
    STATUS = XMAPClose(Xfile)
RETURN
END
```

---

## Transferring data from the database to XML

There are multiple methods available to transfer data from the UniVerse database to an XML document:

- TCL LIST command
- SQL SELECT statement
- TCL DB.TOXML command and UniVerse BASIC DBTOXML() function
- UniVerse BASIC XMAP API ([XMAPCreate function](#), [XMAPAppendRec function](#), and [XMAPToXMLDoc function](#))

For information about creating an XML document using the TCL LIST command or the SQL SELECT statement, see *Creating XML Documents* in the *UniVerse BASIC Extensions* manual.



**Note:** *XMLTODB()* and *DBTOXML()* are not part of the XMAP API, they are standalone UniVerse BASIC functions completely equivalent to the corresponding TCL commands.

### Creating an XML document from TCL

To create an XML document from TCL, use the DB.TOXML command.

#### Syntax

DB.TOXML “*xml\_doc\_filename*” “*xmap\_filename*” “*condition*”

## **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>xml_doc_filename</i>	The name of the XML document to create. If you do not enter a full path, the file is written to the &XML& directory.
<i>xmap_filename</i>	The file name for the U2XMAP file.
<i>condition</i>	A UniVerse RetrieVe condition string, for example, WITH SCHOOL = "CO002"

### **DB.TOXML Parameters**

## **Example**

The following example illustrates using DB.TOXML from TCL to create an XML document.

```
DB.TOXML SCHOOL_STUDENT.XML STUDENT.MAP WITH SCHOOLID = "CO002"
```

## **Creating an XML document from UniVerse BASIC**

To create an XML document from the UniVerse database using UniVerse BASIC, use the DBTOXML function.

Syntax:

```
DBTOXML(xml_document, doc_location, u2xmap_file, u2xmap_location,  
condition, status)
```

# Chapter 9: The XML/DB tool

Installing the XML/DB tool . . . . .	9-3
Create the DTD or XML schema . . . . .	9-8
Using the XML/DB tool. . . . .	9-9
Create server definition . . . . .	9-10
Connect to server. . . . .	9-12
Creating a DTD. . . . .	9-15
Creating or displaying an XML schema . . . . .	9-17
Create a mapping file. . . . .	9-19
Create relationship . . . . .	9-24
Mapping all matching elements. . . . .	9-26
Mapping to multiple UniVerse files . . . . .	9-28
Defining related tables . . . . .	9-30
Options . . . . .	9-34
Define how to treat empty strings . . . . .	9-34
Define date format . . . . .	9-35
Specify how to treat namespace. . . . .	9-35
Define namespace . . . . .	9-35
Define cascade rules. . . . .	9-35
Choose how to treat existing records . . . . .	9-36
Importing and exporting mapping files . . . . .	9-37
Importing a mapping file . . . . .	9-38
Exporting a mapping file . . . . .	9-40
XML/DB tool logging . . . . .	9-42

The XML/DB tool enables you to create a mapping file to use when creating XML documents from the UniVerse database, or when extracting data from an XML document and updating the UniVerse database.

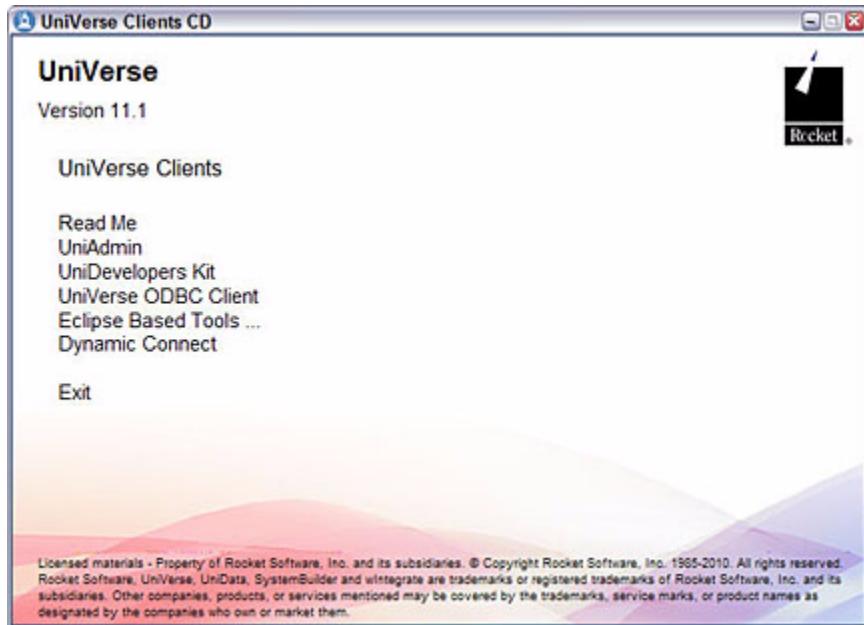
The XML/DB tool loads a DTD or XML Schema, validates the DTD or XML Schema, opens the associated data files, and produces an outline of the file structure. You can then map the DTD or XML Schema tags to the associated fields in the data file, and use this map with RetriVe, UniVerse SQL, UniVerse BASIC, or the XMAP API.

## Installing the XML/DB tool

Complete the following steps to install the XML/DB tool.

### **1. Load the UniData client CD**

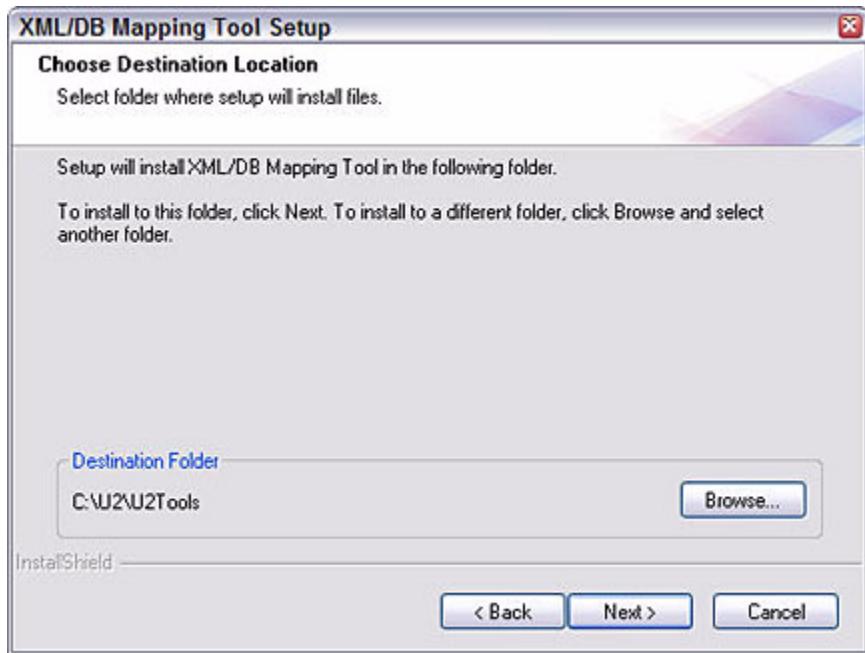
Place the UniData Client CD in your CD-Rom drive. The following menu appears:



From the menu select **Eclipse Based Tools**, then select **XML/DB Mapping Tool**. From the **Welcome** menu, click Next.

## **Choose destination**

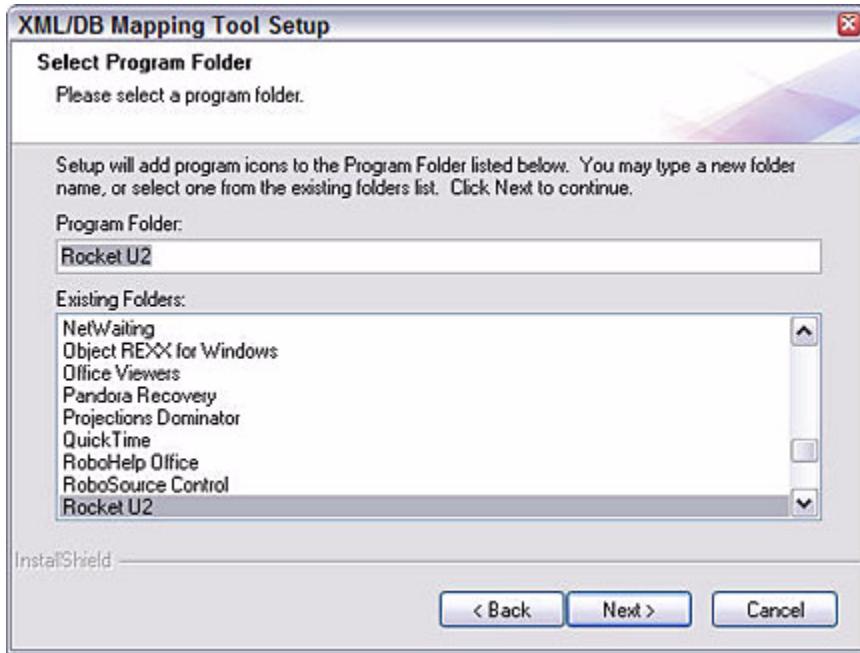
The **Choose Destination Location** dialog box appears, as shown in the following example:



By default, UniVerse installs the Web Tools in the C:\U2\U2Tools directory. If you want to choose a different directory, click **Browse** and choose the directory where you want to install the Web Tools. Click **Next** to continue with the installation.

## **Select program folder**

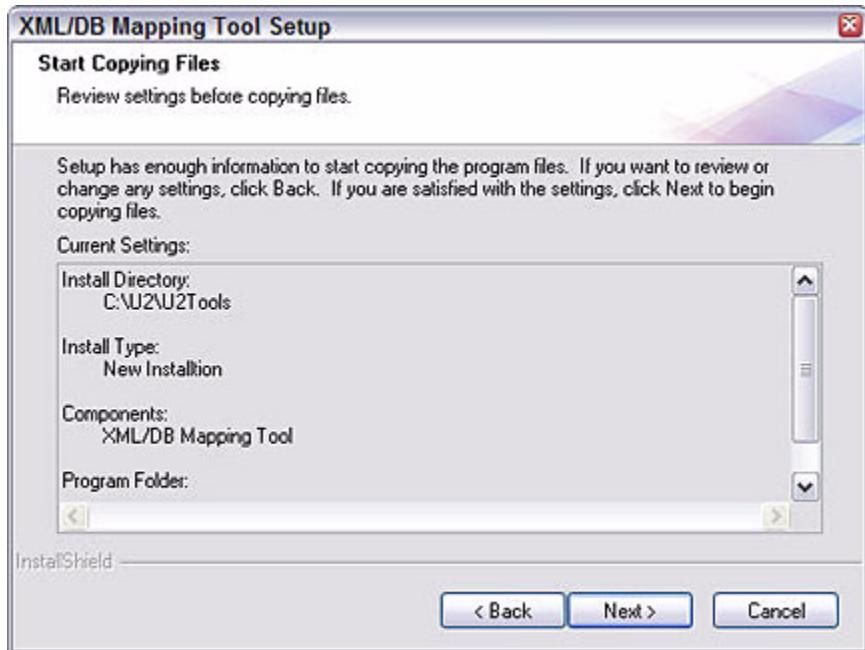
The installation next prompts you for a program folder for UniVerse Tools, as shown in the following example:



Enter the name of the program folder for UniVerse Tools if you do not want to accept the default, then click **Next** to continue the installation process.

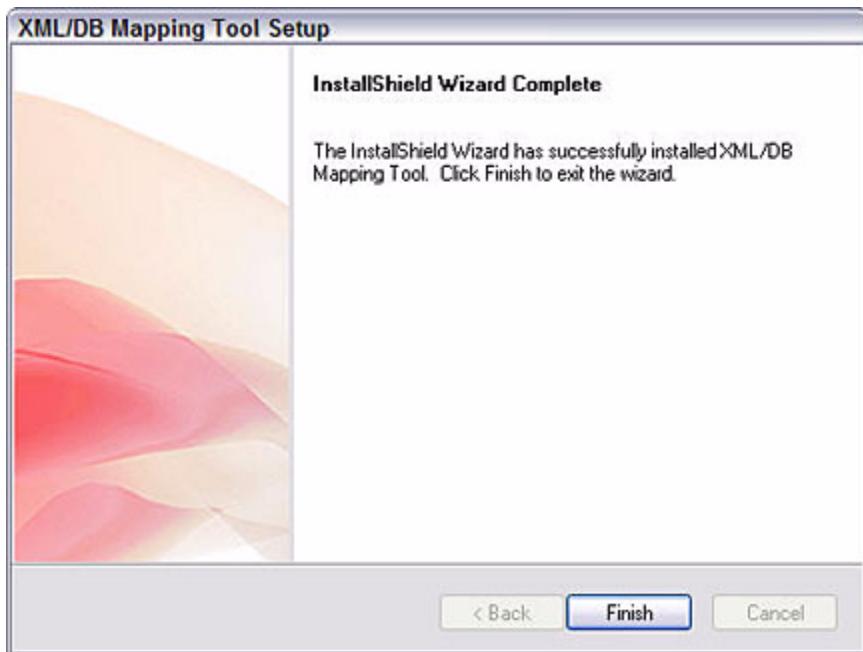
## ***Copy files***

The installation process now has enough information to begin copying files, as shown in the following example. If you want to change any settings, click **Back**. Click **Next** to begin copying files.



### ***Complete the installation***

Click **Finish**. You may want to restart your computer before you use the XML/DB tool. The installation process prompts if you want to restart your computer now or at a later time, as shown in the following example:



Choose when you want to restart your computer, then click **Finish**.

## Create the DTD or XML schema

The XML/DB Tool works with existing DTD or XML Schema files. You must create a DTD or XML Schema file before using the XML/DB mapping tool if one does not already exist. Use UniVerse SQL, UniVerse BASIC, or RetrievVe to create the file.

When you use the RetrievVe LIST statement or the UniVerse SQL SELECT statement, UniVerse creates an XML Schema or DTD if you specify TOXML and the TO keyword in the same statement. UniVerse writes the file to the &XML& directory, with an extension of .xsd for an XML schema file or .dtd for a DTD file.

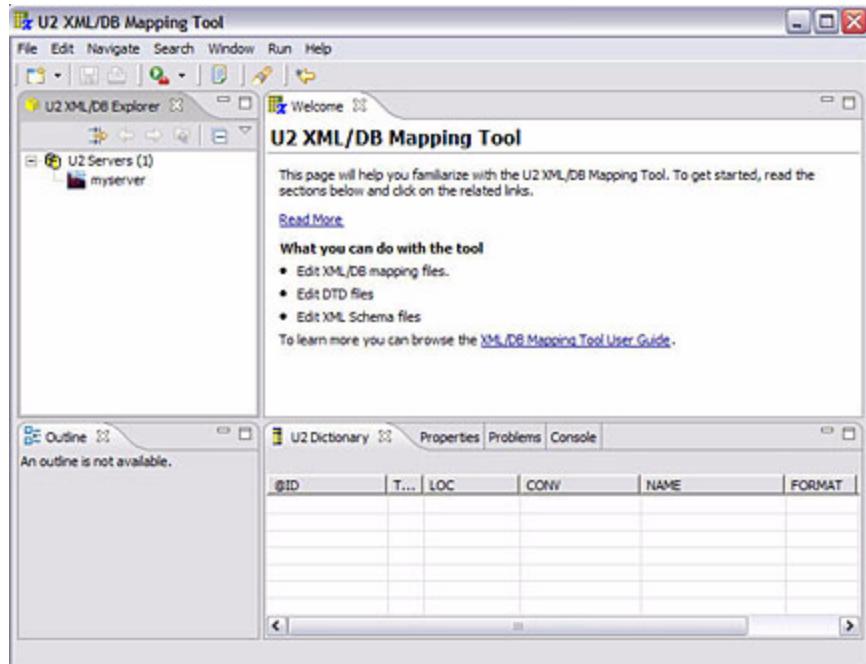
In the following example, UniVerse creates a STUDENT.xsd file in the &XML& directory:

```
:LIST STUDENT FNAME LNAME MAJOR MINOR ADVISOR TOXML WITHSCHEMA TO
STUDENT

:ED &XML& STUDENT.xsd
Top of "STUDENT.xsd" in "&XML&", 26 lines, 812 characters.
*--: p
001: <?xml version="1.0"?>
002: <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
003:   <xsd:annotation>
004:     <xsd:documentation xml:lang="en">
005:       account: C:\rocketU2\ud72\Demo
006:       command: LIST STUDENT FNAME LNAME MAJOR MINOR ADVISOR
TOXML WITHSCHEMA
TO STUDENT
007:   </xsd:documentation>
008:   </xsd:annotation>
009:   <xsd:element name="ROOT">
010:     <xsd:complexType>
011:       <xsd:sequence maxOccurs="unbounded">
012:         <xsd:element ref="STUDENT" maxOccurs="unbounded" />
013:       </xsd:sequence>
014:     </xsd:complexType>
015:   </xsd:element>
016:   <xsd:element name="STUDENT">
017:     <xsd:complexType>
018:       <xsd:attribute name="_ID" />
019:       <xsd:attribute name="FNAME" />
020:       <xsd:attribute name="LNAME" />
021:       <xsd:attribute name="MAJOR" />
022:       <xsd:attribute name="MINOR" />
```

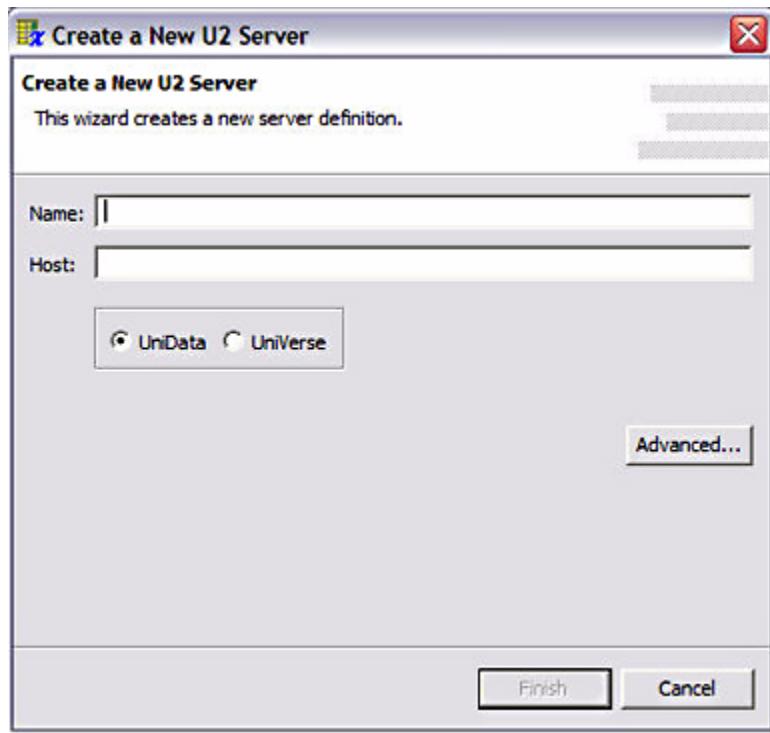
## Using the XML/DB tool

To start the XML/DB Tool, from the **Start** menu, select **Programs**, then select **Rocket U2**, then select **UniVerse Tools**, then click **XMLDB Tool**. The following dialog box appears:



## Create server definition

To establish a connection to a server, right-click **U2 Servers**, then click **New U2 Server**. The **Create New U2 Server** dialog box appears, as shown in the following example:

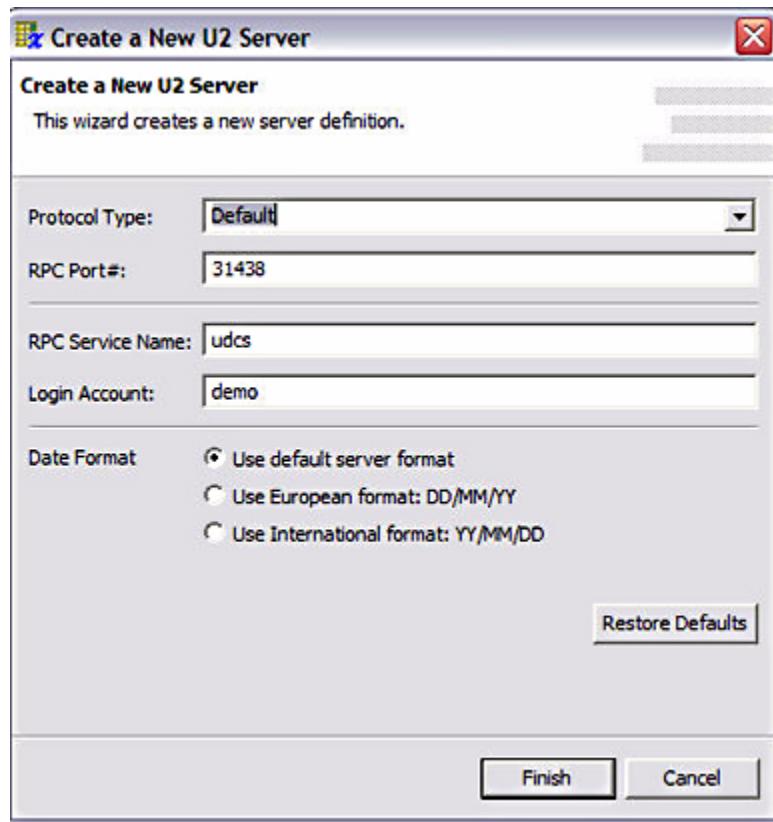


In the **Name** box, enter a unique name to identify the server. The name cannot contain a forward slash (“/”) or a back slash (“\”).

In the **Host** box, enter the hostname or IP address of the server.

In the **Database** box, select **UniData** or **UniVerse** as the underlying database you are using.

Click **Advanced** if you want to define the type of communication you are using, specify a port number that differs from default, define the RPC Service Name, or specify a login account. A dialog box similar to the following example appears:



In the **Transport Type** box, choose the type of communication you are using to the server. You can choose Default, TCP/IP, or Lan Manager. The default is TCP/IP.

In the **RPC Port#** box, enter the UniRPC port number.

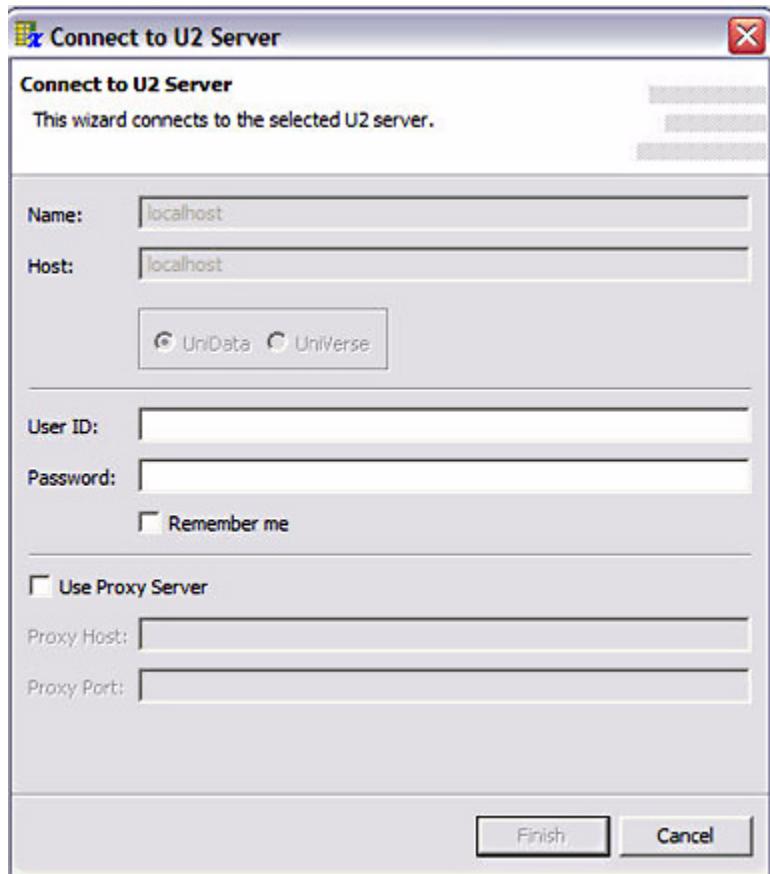
In the **RPC Service Name** box, enter the name of the RPC service you are using. The default service name is udcs for UniData, or uvcs for UniVerse.

Enter the name of the account to which you want to connect in the **Login Account** box.

Click **Finish** to register the server.

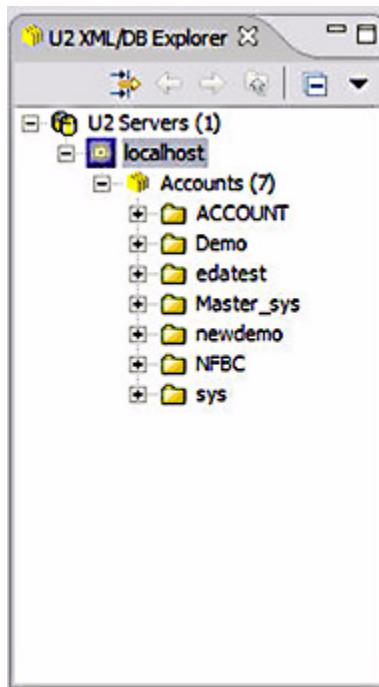
## Connect to server

To connect to a server, double-click the server to which you want to connect. A dialog box similar to the following example appears:



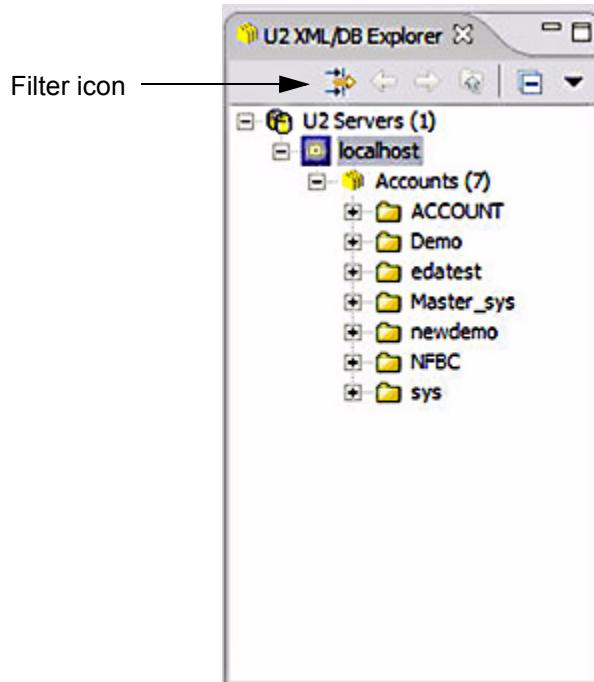
Enter your user ID and password in the appropriate boxes, then click **Finish**.

Once you connect to the server, the XML/DB Tool displays all available accounts, data files, DTD files, XMAP files, and XSD files, as shown in the following example:



### **Filtering accounts**

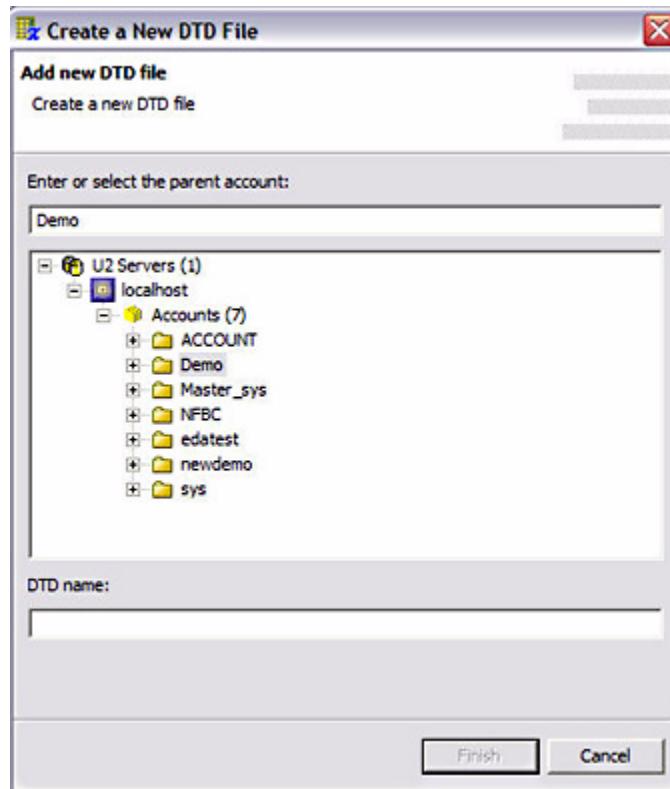
You can filter the accounts that are available in the XML/DB tool. To filter accounts, select the **Filter** icon, as shown in the following example:



## Creating a DTD

Use the DTD File wizard to create a new DTD. To display an existing DTD, double-click the file you want to display in the **U2 XML/DB Explorer** area.

To access the file wizard, expand the account where you want to create the DTD, then right-click **DTD Files**. The **Create a New DTD File** dialog box appears, as shown in the following example:



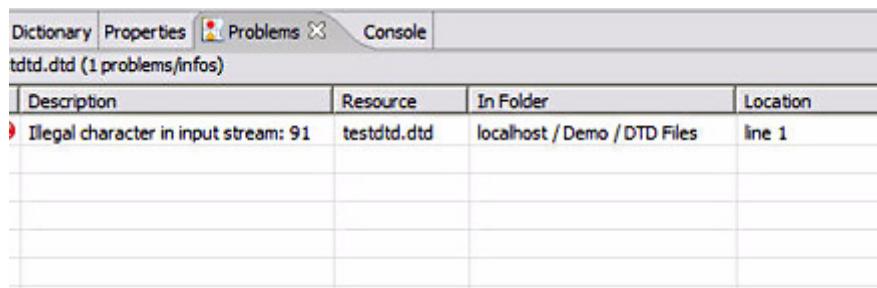
In the **DTD name** box, enter the name of the DTD, then click **Finish**.

Enter the DTD in the dialog box. The following example illustrates a DTD.



```
<?XML Version="1.0"?>
<!DOCTYPE ROOT[
  <!ELEMENT ROOT (CUSTOMER*)>
  <!ELEMENT CUSTOMER ( ADDRESS_MV* )>
  <!ATTLIST CUSTOMER
    _ID CDATA #REQUIRED
    NAME CDATA #IMPLIED
    STATE CDATA #IMPLIED
  >
  <!ELEMENT ADDRESS_MV EMPTY>
  <!ATTLIST ADDRESS_MV
    ADDRESS CDATA #IMPLIED
  >
]> I>
```

The XML/DB tool automatically validates the DTD. Click the **Problems** tab to view any problems detected with the DTD, as shown in the following example:

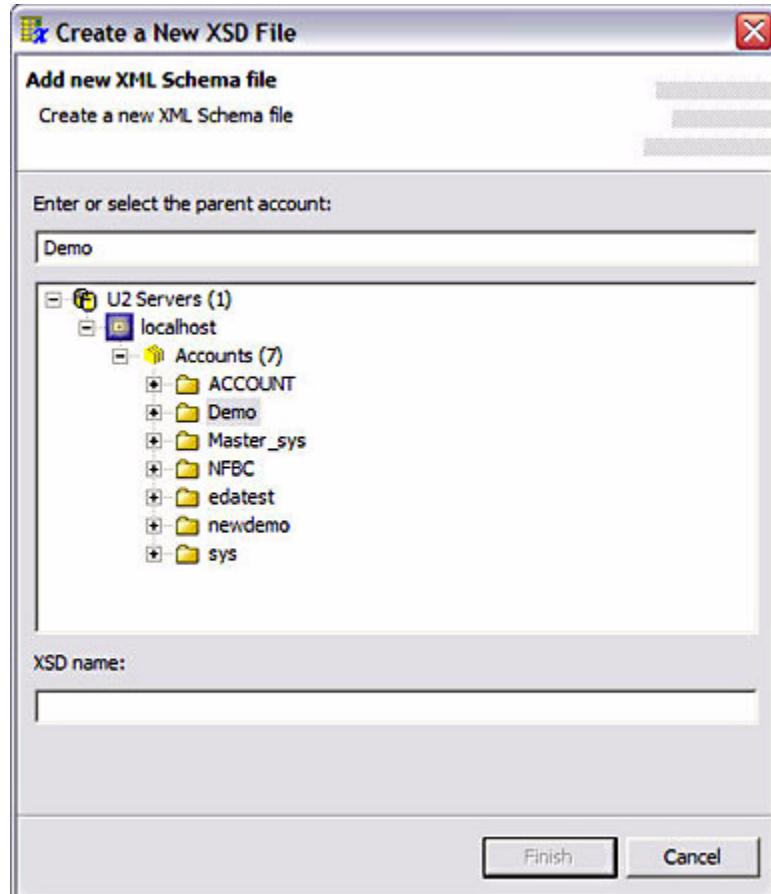


Description	Resource	In Folder	Location
Illegal character in input stream: 91	testdtd.dtd	localhost / Demo / DTD Files	line 1

## Creating or displaying an XML schema

Use the XSD File wizard to create a new XML Schema or display an existing XML Schema. To display an existing XML Schema, double-click the file you want to display in the Account Explorer area.

To access the file wizard, expand the account where you want to create the XML Schema, then right-click **XSD Files**, then click **New XSD File**. The **Create a New XSD File** dialog box appears, as shown in the following example:



In the **XSD name** box, enter the name of the XSD. Enter the XML schema in the dialog box that appears, as shown in the following example:



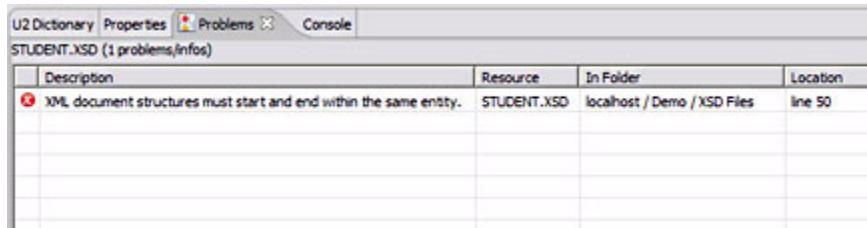
```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="ROOT">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="CLASS" minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="CLASS">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="STUDENT" minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="ClassID" type="xsd:string"/>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="STUDENT">
```

The XML/DB tool automatically validates the XML Schema. Click the **Problems** tab to view any problems detected with the XML Schema, as shown in the following example:



STUDENT.XSD (1 problems/infos)			
Description	Resource	In Folder	Location
XML document structures must start and end within the same entity.	STUDENT.XSD	localhost / Demo / XSD Files	line 50

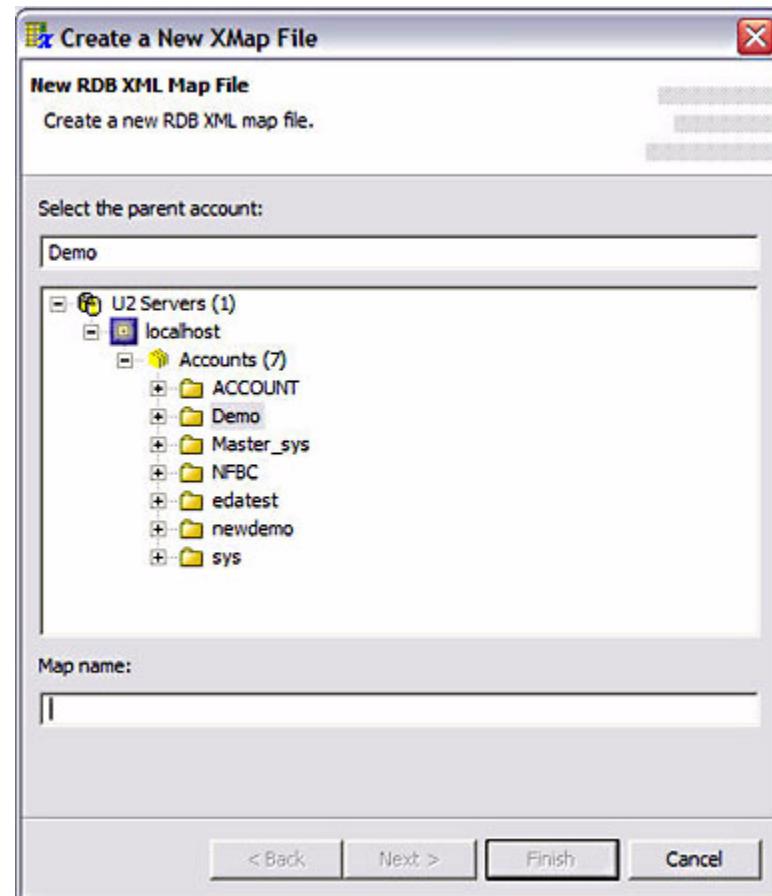
---

## Create a mapping file

The XML/DB Tool enables you to create a new XML mapping file based on an existing DTD or XML Schema file. Creating an XML mapping file includes the following steps:

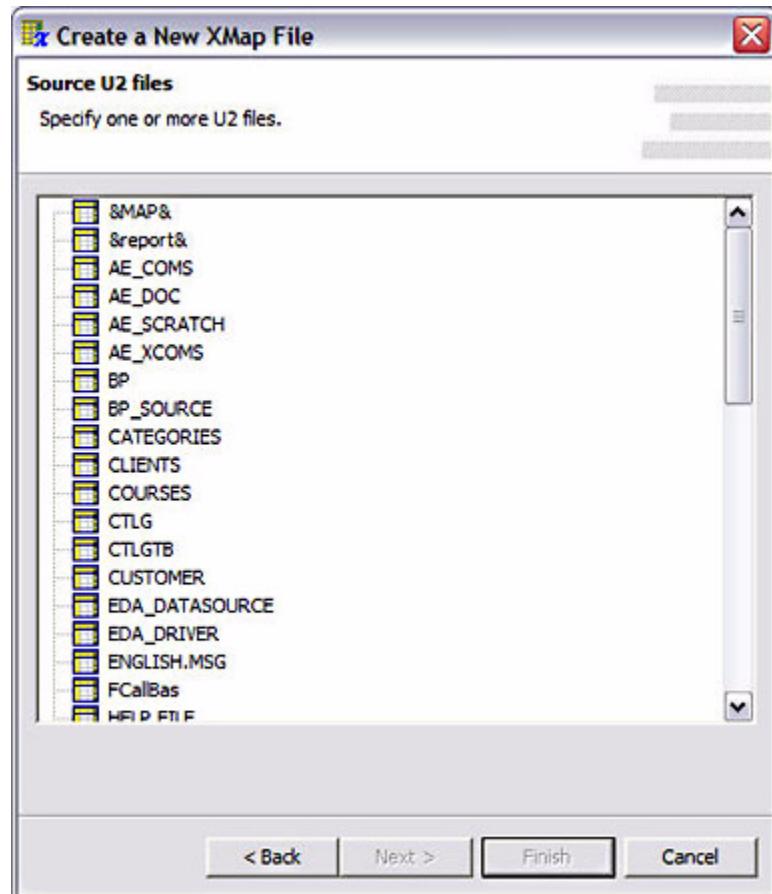
- Select the target account
- Specify the XMap file name
- Select the data files
- Select the DTD or XSD file
- Specify the root element

To create a new mapping file, expand the account where you want to create the XML Schema, then right-click **XMap Files**, then click **New XMap File**. The **Create a New XMap File** dialog box appears, as shown in the following example:

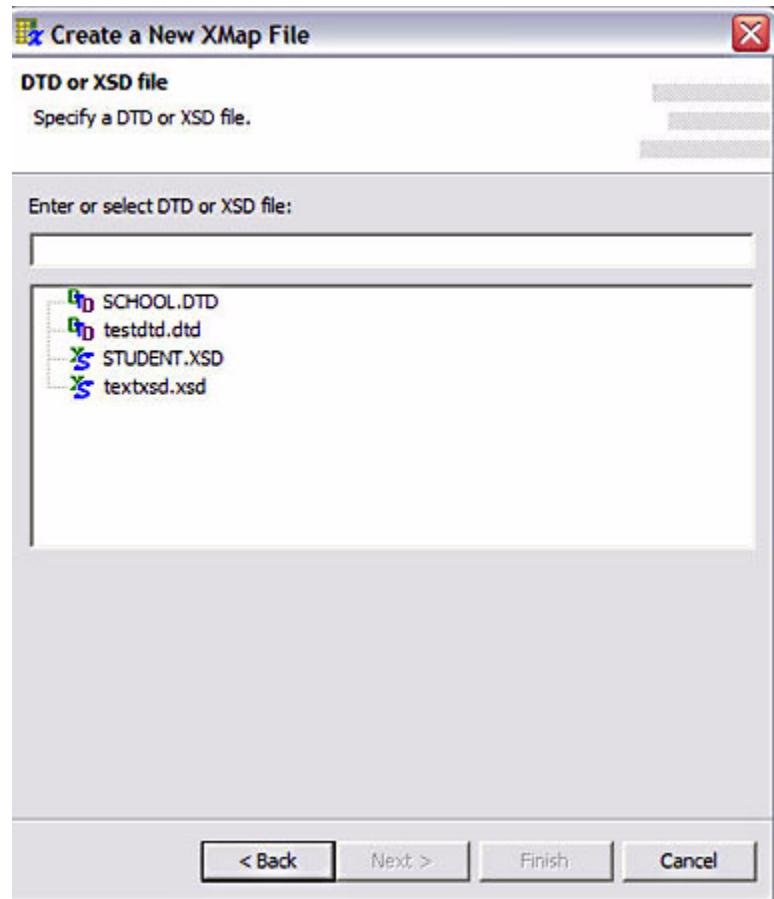


In the **Map name** box, enter the name for the XMap, then click **Next**.

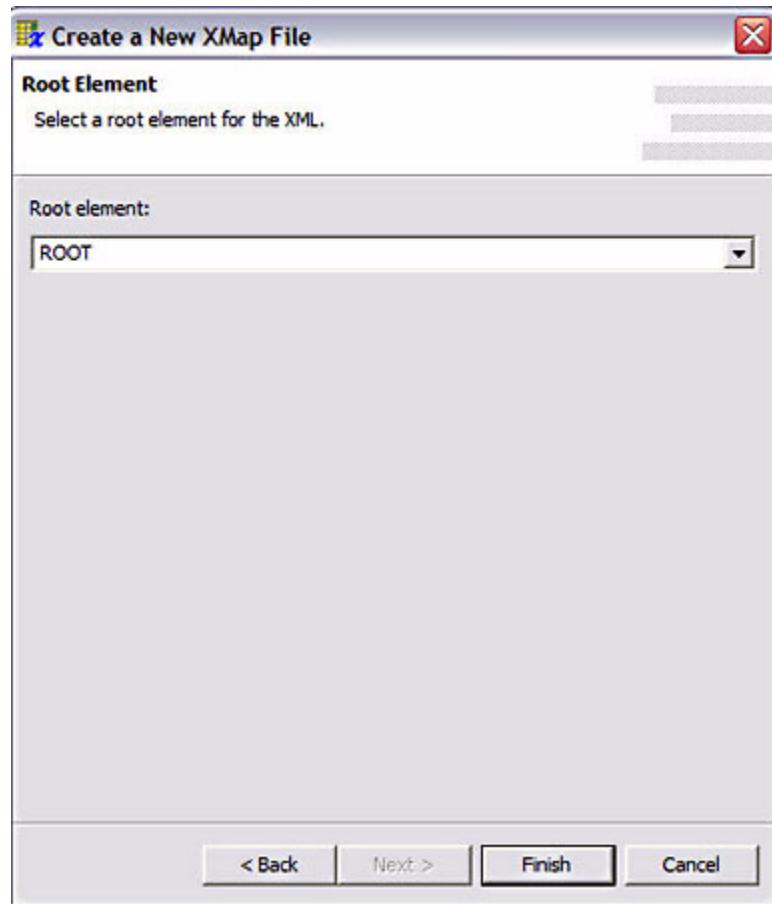
The **Source U2 files** dialog box appears, as shown in the following example:



Select one or more files for which you want to create a map from the DTD or XSD file, then click **Next**. A dialog box similar to the following example appears:

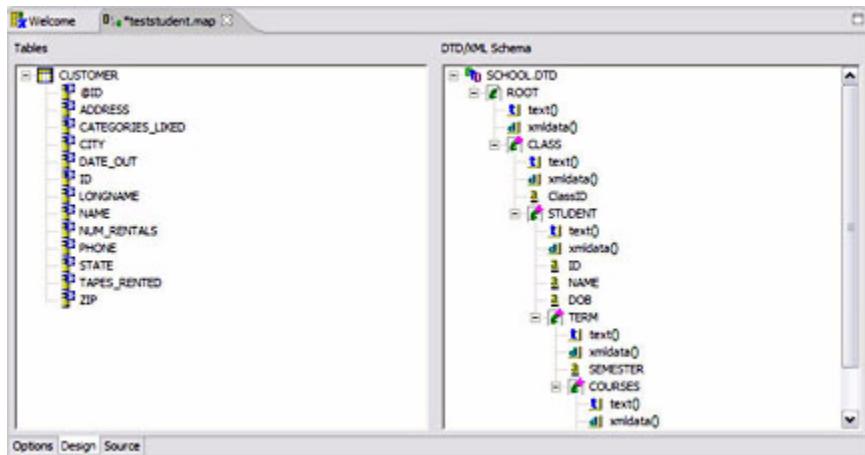


Select the DTD or XSD file for which you want to create a mapping file, then click **Next**. The **Root Element** dialog box appears, as shown in the following example:



In the **Root Element** box, select the name of the root element in the DTD or XSD file, then click **Finish**.

The Mapping Editor appears, as shown in the following example:



The names of the dictionary records for the file you specified appear in the **Tables** area of the editor. The elements of the DTD or XML Schema file you specified appear in the **DTD/XML Schema** area.

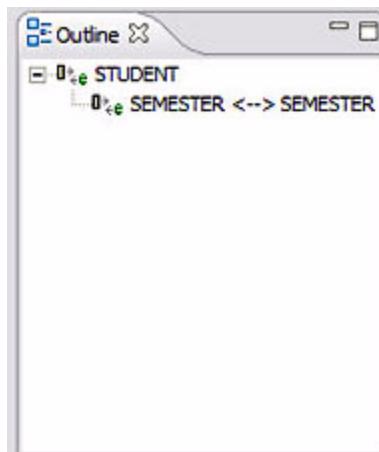
## Create relationship

To create a relationship between a dictionary item and a DTD or XML Schema element, click the dictionary record name. Detailed information about the dictionary records appears in the **U2 Dictionary** area, as shown in the following example:

The screenshot shows the "U2 Dictionary" interface with a table of student attributes. The columns are labeled: #ID, T..., LOC, CONV, NAME, FORMAT, SM, ASSOC, and Mapped To. The rows show attributes like STUDENT, ID, UNAME, FNAME, MAJOR, MINOR, and GRADE, each with its data type and length.

#ID	T...	LOC	CONV	NAME	FORMAT	SM	ASSOC	Mapped To
STUDENT				STUDENT	10L	S		
ID	D	0		STUDENT	12R # #...	S		
UNAME	D	1		Last Name	15T	S		
FNAME	D	2		First Name	10L	S		
MAJOR	D	3		Major	4L	S		
MINOR	D	4		Minor	4L	S		
GRADE	N	4		Grade	4L	C		

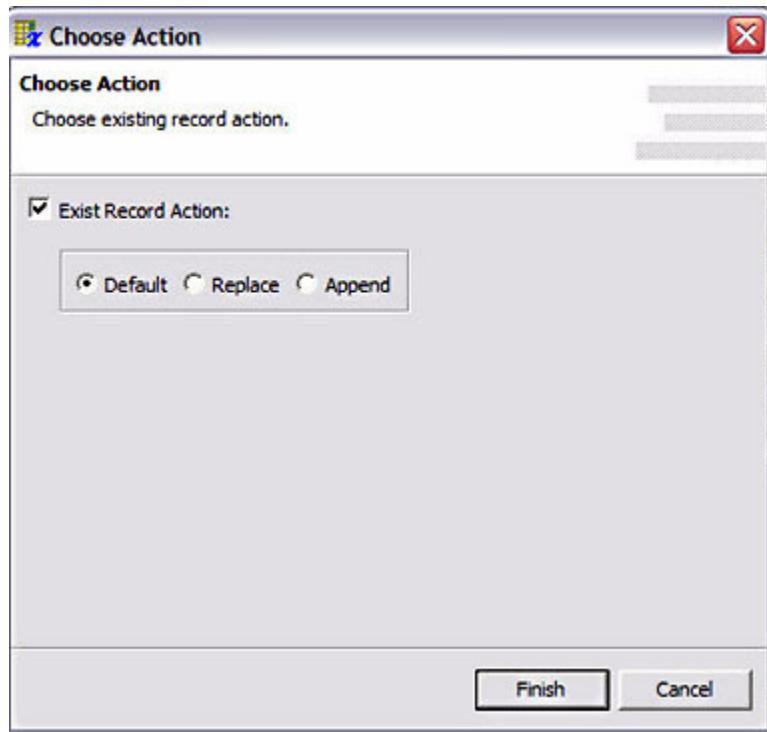
Right-click the element that corresponds to the dictionary item in the **DTD/XML Schema** area, then click **Create Mapping**. Arrows appear next to the items you selected for mapping, and the mapping relationship appears in the **Outline** area of the dialog box, as shown in the following example:



If you want to remove the mapping relationship, right-click the mapping definition in the **DTD/XML Schema** area, then click **Remove Mapping**.

## Mapping all matching elements

If you want to map all dictionary items that have a matching element name, click the file name in the **Tables** area of the dialog box, right-click the appropriate element in the DTD/XML Schema area, then click **Match Mapping**. The XML/DB tool prompts you to select how you want to update the database if a record in the XML document already exists in the database, as shown in the following example:

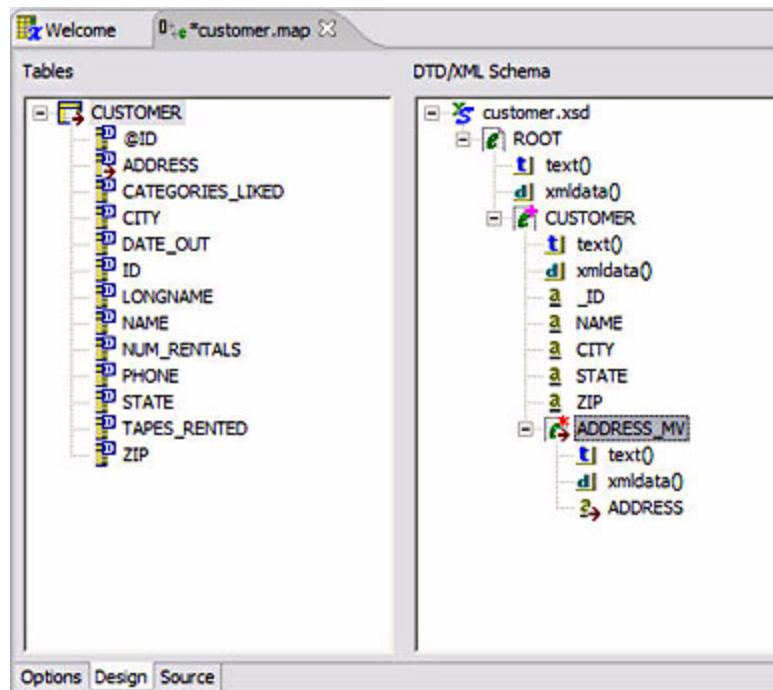


Following are the choices if a record already exists in the database:

- **Default** – If the record already exists in the database, do not update it.
- **Replace** – Replace the data in the existing record with the data from the XML document.

- **Append** – In the case of a multivalued field, append the new value to the existing multivalued field. If you have duplicate data, do not use the Append option, as the multivalued field may be updated with duplicate data.

Click **Finish**. The XML/DB Tool maps all like elements, as shown in the following example:



## Mapping to multiple UniVerse files

You can map an XML document to multiple UniVerse files.

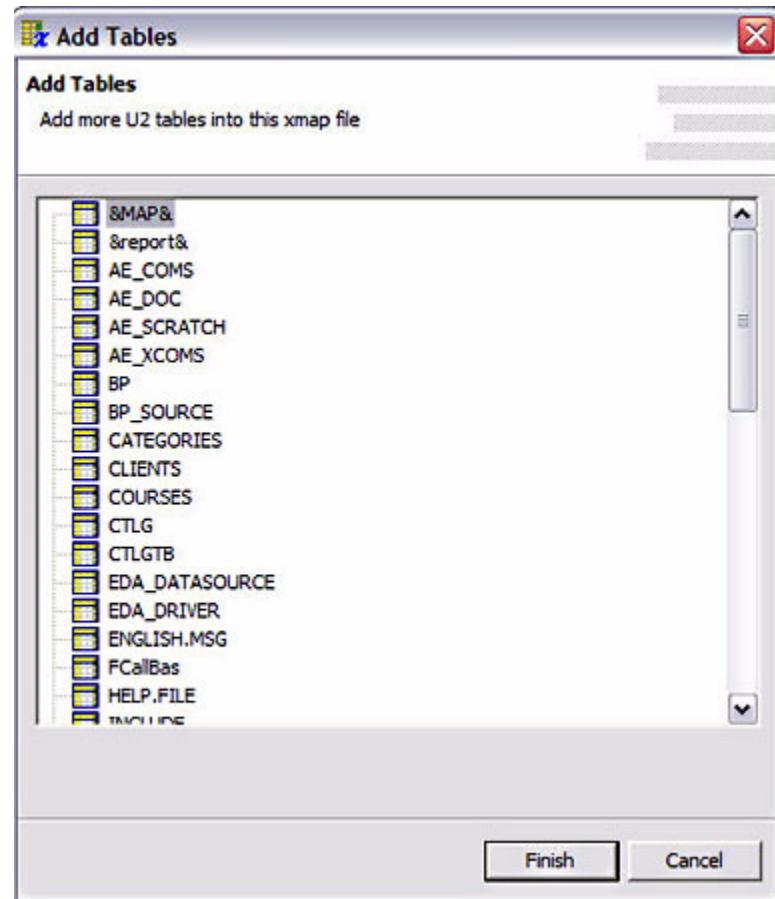
Choose one of the following methods to access the **Add Tables** dialog box:

- From the Mapping Editor, from the **XML/DB** menu, click **Add Tables**.
- Click the Add Tables icon from the toolbar, as shown in the following example:

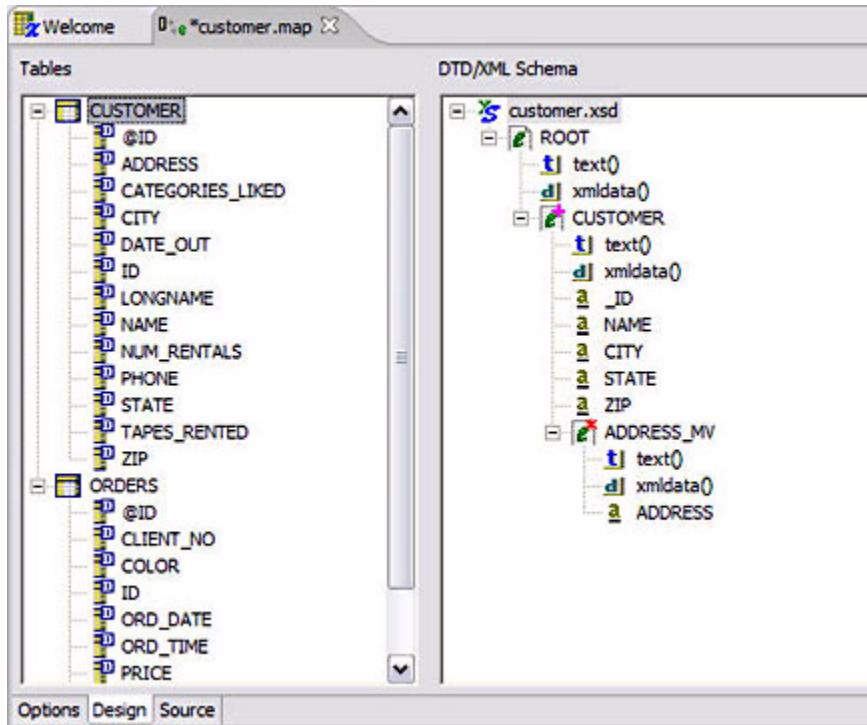


- Right-click an existing table in the **Tables** portion of the XML/DB Mapping Tool, then click **Add Tables**.

The **Source U2 files** dialog box appears, as shown in the following example:



Click the tables you want to add to the mapping file, then click **Finish**. The new table is now added to the mapping editor, as shown in the following example:



## Defining related tables

If you are mapping more than three levels of data, you must map the data to more than one UniVerse file, since a UniVerse file can support no more than three levels of data.

Choose on the following methods to access the **Define Related Tables** dialog box:

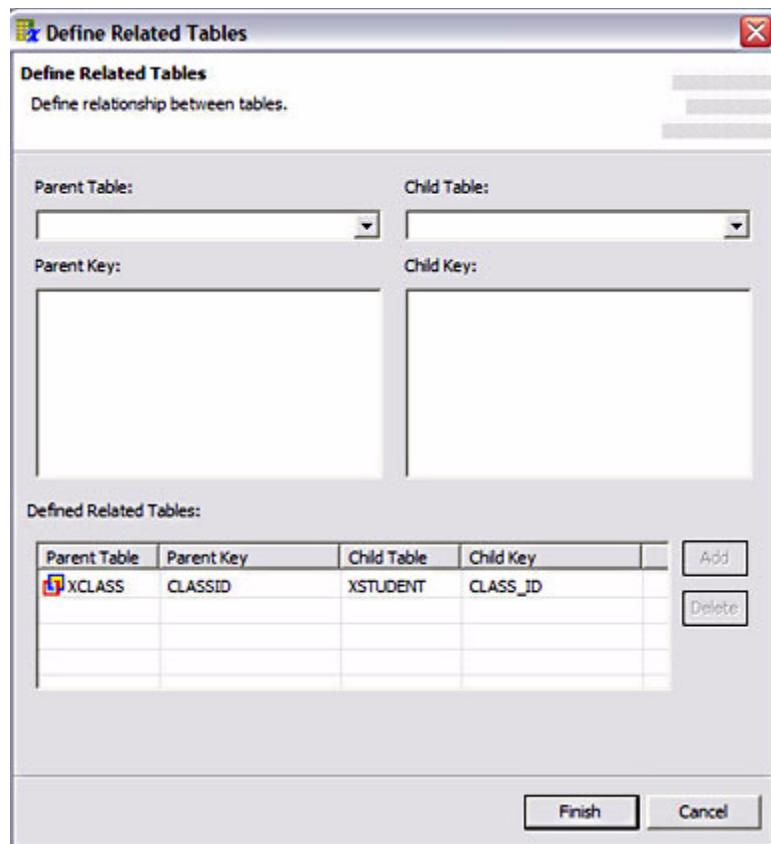
- From the Mapping Editor, from the **XML/DB** menu, click **Define Related Tables**.

- Click the **Define Related Tables** icon from the toolbar, as shown in the following example:



- Right-click an existing table in the **Tables** portion of the XML/DB Mapping Tool, then click **Define Related Table**.

The **Define Related Tables** dialog box appears, as shown in the following example:



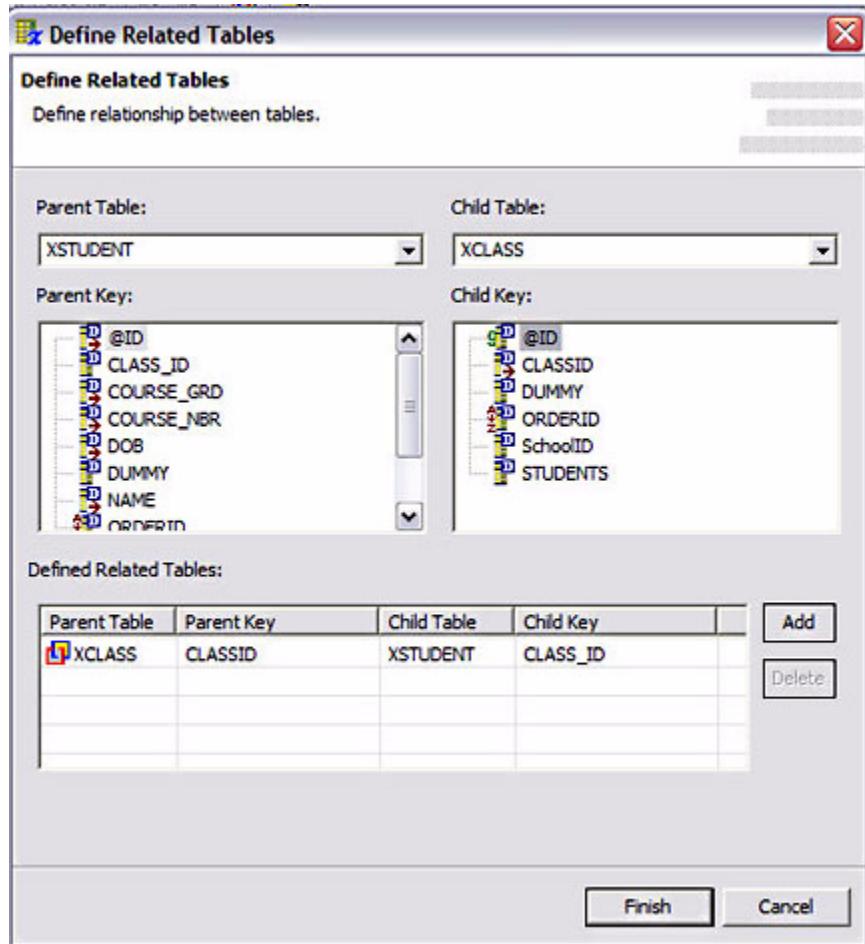
In the **Parent Table** box, select the parent table. The parent table is the file corresponding to the top portion of the XML subtree being transformed.

In the **Parent Key** area, click the dictionary item that represents the primary key for the parent table.

In the **Child Table** box, select the child table of the parent table.

In the **Child Key** box, click the dictionary item that represents the primary key for the child table.

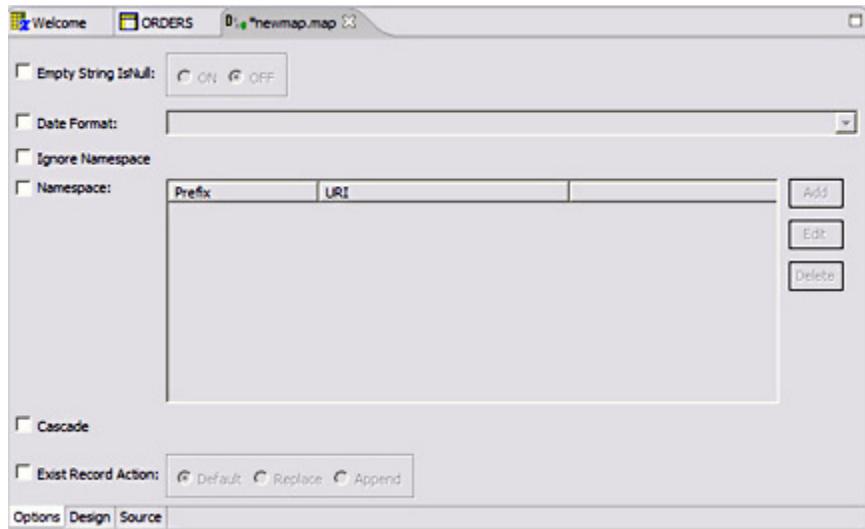
Click **Add**. The **Define Related Tables** area is now populated, as shown in the following example:



Click **Finish** to save the definition, or click **Cancel** to exit without saving changes.

## Options

From the Mapping Editor, click the **Options** tab. A dialog box similar to the following example appears:



### Define how to treat empty strings

If you select the **Empty String is Null** check box and click ON, when the value of an optional XML element is omitted, the corresponding database field value is set to NULL, otherwise it is considered missing. Similarly, when a database field value is NULL and you select ON, the corresponding value of an optional XML element is omitted, otherwise it is set to an empty string.

## Define date format

Data Format is a conversion code UniVerse uses to perform the ICONV() / OCONV() function when getting the data from the XML document or creating the XML document. This code can be any conversion code understood by UniVerse. You can also use the value of "XMLFormat" which converts the data into a standard data format for the XML document, or takes the standard XML date format (yyyy-mm-dd).

## Specify how to treat namespace

If you select the **Ignore Namespace** check box, UniVerse ignores all the Namespace information. This option applies when converting the XML document to the UniVerse database.

## Define namespace

Namespace elements are not required. If they are used, the same URI or prefix cannot be used more than once. Zero-length prefixes ("") are not currently supported.

If you do not define a Prefix, UniVerse uses the Default Name space, used only for generating the XML document. In this case, UniVerse puts a default NameSpace in the output document.

## Define cascade rules

This option only applies when an XML document is mapped to more than one UniVerse file. By default, the cascade mode is off, which means that the system takes care of the parent-child record relationship according to the RelateTable rules described in the U2XMAP file. Setting the cascade mode ON allows you to control the parent-child record relationship. When the cascade mode is ON, once the current parent table record is established, all child table records that are either read from or written to the child table are associated with the current parent table record. This affords you the freedom of associating parent and child table records the way you see fit.

## Choose how to treat existing records

Following are the choices if a record already exists in the database:

- **Default** – If the record already exists in the database, do not update it.
- **Replace** – Replace the data in the existing record with the data from the XML document.
- **Append** – In the case of a multivalued field, append the new value to the existing multivalued field. If you have duplicate data, do not use the Append option, as the multivalued field may be updated with duplicate data.

For detailed information about the mapping file, see Appendix B, “[Appendix B: The U2XMAP file](#).”

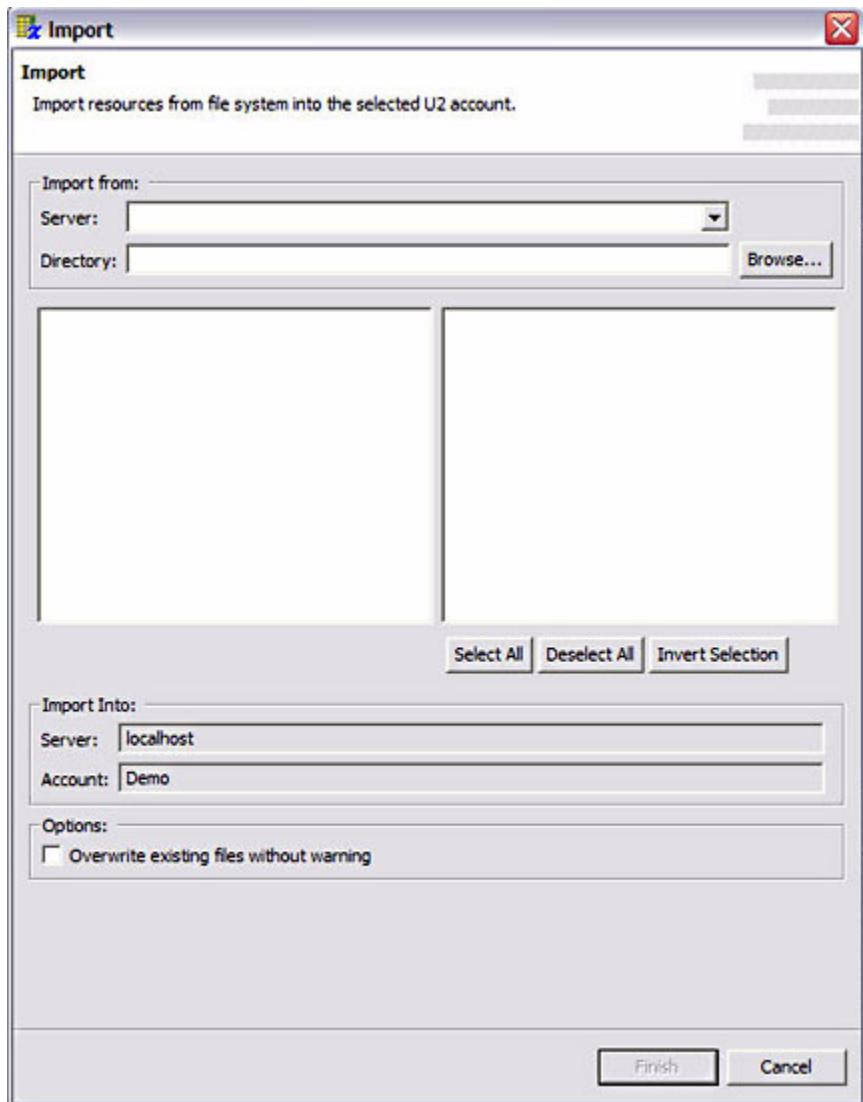
---

## **Importing and exporting mapping files**

You can import a mapping file, DTD, or XSD from another system-level directory to your account, or export a mapping file to another system-level account.

## Importing a mapping file

To import an operating system-level mapping file, right-click **XMap Files**, then click **Import**. The **Import** dialog box appears, as shown in the following example:



From the **Import From** area, in the **Server** box, enter the name of the server where the mapping file resides.

In the **Directory** box, enter the full path to the location of the mapping file, or click **Browse** to search for the location.

In the **Import Into** area, in the **Server** box, enter the name of the server where you want to write the mapping file.

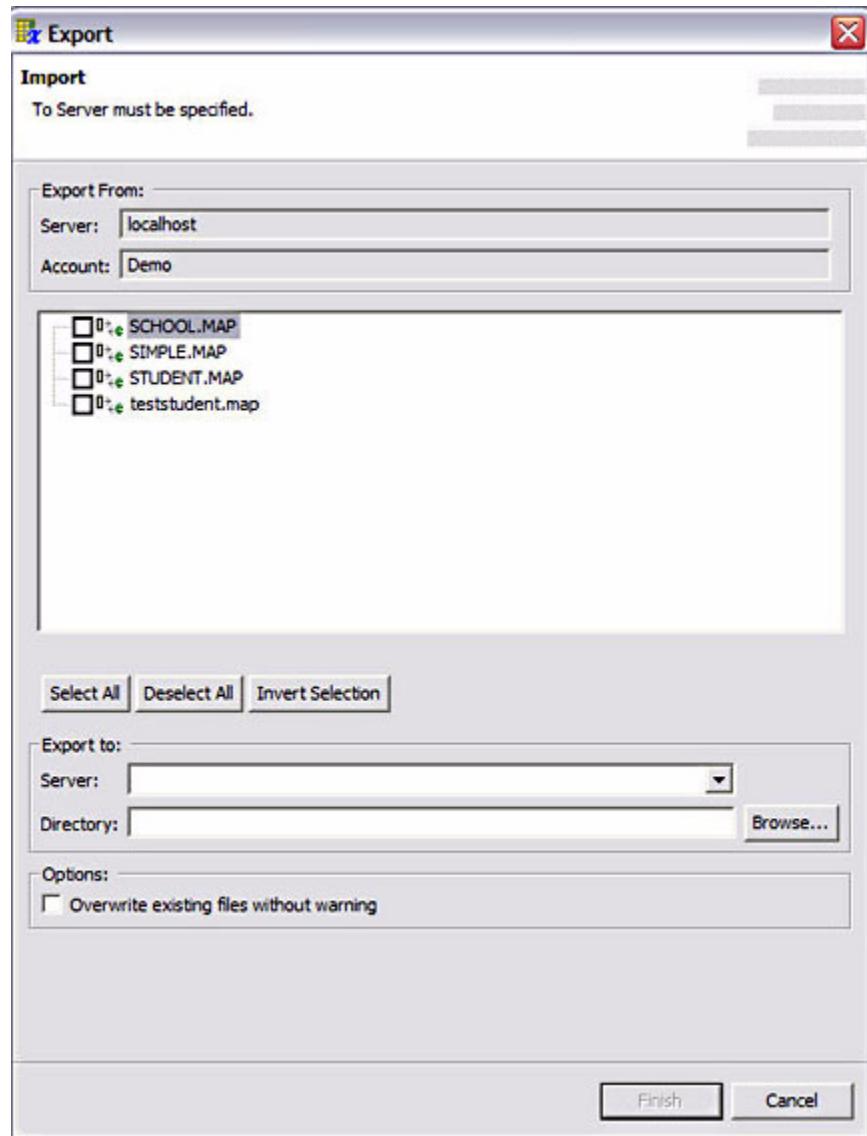
In the **Account** area, enter the name of the account where you want the mapping file to reside, or click **Browse** to search for the account.

Select the **Overwrite existing files without warning** check box if you want to overwrite a mapping file of the same name that may exist in the target account.

Click **Finish** to import the mapping file.

## Exporting a mapping file

To export a mapping file, right-click XMap Files, then click **Export**. The **Export** dialog box appears, as shown in the following example:



From the **Export From** area, in the **Server** box, where the mapping file resides.

In the **Account** box, enter the full path to the location of the mapping file, or click **Browse** to search for the location.

In the **Export To** area, in the **Server** box, enter the name of the server where you want to write the mapping file.

In the **Directory** area, enter the name of the account where you want the mapping file to reside at the operating-system level, or click **Browse** to search for the account.

Select the **Overwrite existing files without warning** check box if you want to overwrite a mapping file of the same name that may exist in the target account.

Click **Finish** to import the mapping file.

## **XML/DB tool logging**

To enable logging with the XML/DB Tool, create an empty record in the VOC file of the UV account you are using the XML/DB Tool called XTOOLDBG prior to initializing an XML/DB Tool session.

The log file captures output from the XML/DB Tool to the client, and output from the client to the XML/DB Tool, along with the time logging started, and any error codes sent to the client.

The log file, located in the /tmp directory on UNIX platforms and the C:\tmp directory on Windows platforms, has the format XTOOLSUB\_dddd\_tttt.log, where *dddd* is the internal date stamp and *tttt* is the internal time stamp. If the /tmp or C:\tmp directory does not exist, you must create it prior to initializing logging.

# Chapter 10: U2 Dynamic Object

# 10

U2 Dynamic Object . . . . .	10-2
XML Support in U2 Dynamic Object . . . . .	10-4
Converting from XML to JSON . . . . .	10-4
Converting from UDO to XML . . . . .	10-6
U2 Dynamic Object API . . . . .	10-7
UDO definitions . . . . .	10-7
UDO functions . . . . .	10-10
UDORead . . . . .	10-10
UDOWrite . . . . .	10-11
UDOSetOption . . . . .	10-11
UDOGetProperty . . . . .	10-12
UDOClone . . . . .	10-12
UDOFree . . . . .	10-13
UDOGetProperty . . . . .	10-14
UDO SetProperty . . . . .	10-14
UDODeleteProperty . . . . .	10-15
UDOGetPropertyNames . . . . .	10-16
UDOGetNextProperty . . . . .	10-16
UDOCREATE . . . . .	10-18
UDOGetType . . . . .	10-18
UDOIsTypeOf . . . . .	10-19
UDOArrayGetSize . . . . .	10-19
UDOArrayGetItem . . . . .	10-20
UDOArraySetItem . . . . .	10-21
UDOArrayInsertItem . . . . .	10-22
UDOArrayAppendItem . . . . .	10-23
UDOArrayDeleteItem . . . . .	10-24

UDOArrayGetNextItem . . . . .	10-24
UDOGetLastError . . . . .	10-25
XML2UDO options and UDO2XML options . . . . .	10-27
UDOOPTION_XML2UDO_INFERNUMBER. . . . .	10-27
UDOOPTION_XML2UDO_INFERBOOLEAN . . . . .	10-28
UDOOPTION_XML2UDO_INFERNULL . . . . .	10-28
UDOOPTION_XML2UDO_EMPTY2NULL . . . . .	10-29
UDOOPTION_XML2UDO_TRIMWHITESPACE . . . . .	10-30
UDOOPTION_XML2UDO_CASEINSENSITIVE . . . . .	10-31
UDOOPTION_XML2UDO_KEEPROOT . . . . .	10-32
UDOOPTION_UDO2XML_XMLSTYLE . . . . .	10-33
UDOOPTION_UDO2XML_NAMESPACE and UDOOPTION_UDO2XML_NAMESPACEPREFIX . . . . .	10-34
UDOOPTION_UDO2XML_ROOTNAME . . . . .	10-35
UDOOPTION_UDO2XML_NULL2EMPTY . . . . .	10-36

## U2 Dynamic Object

U2 Dynamic Object (UDO) provides an object structure in UniVerse.

JSON is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999 ([www.json.org](http://www.json.org)). It is a text format that is completely language independent, but uses conventions that are familiar to programmers of the C-family of languages.

These are the characteristics of U2 Dynamic Object:

- Strictly follows the JSON specification
- Is not persistent; in memory only
- Purely data object; no method support
- Dynamic; properties can be added or removed on the fly
- No class definition
- No inheritance support
- Unlimited nesting levels
- Can create objects from JSON and XML data. In the case of XML data, UniVerse converts the data into JSON based on a set of well understood rules which may not guarantee that the conversion is reversible.
- Can be serialized into JSON or XML.,,

JSON is built on the following two structures:

- A collection of name/value pairs. This is realized as an object, record, structure, dictionary, hash table, keyed list, or associated array.
- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

In JSON, these structures take on the following forms:

- An object is an unordered set of name/value pairs. An object begins with a left brace ("{" and ends with a right brace ("}"). Each name is followed by a colon ":" and the name/value pairs are separated by a comma (",").

- An array is an ordered collection of values. An array begins with a left bracket ("[") and ends with a right bracket ("]"). Values are separated by a comma (",").
- A value can be a string in double quotation marks, a number, a true, false, or null, an object or an array. These structures can be nested.
- A string is a sequence of zero or more Unicode characters wrapped in double quotation marks, using backslash escapes. A character is represented as a single character string. A string is like a C or Java string.
- A number is like a C or Java number, except that the octal and hexadecimal formats are not used.

## XML Support in U2 Dynamic Object

U2 Dynamic Object is based on the JSON specification, so JSON is its native serialization format. To support XML in U2 Dynamic Object, UniVerse performs a conversion between XML and JSON. A bidirectional conversion is not always possible because XML is more expressive and flexible than JSON. Because of this, once an XML document is converted to JSON format, it may not always be possible to convert the JSON data back to the original XML document.

### Converting from XML to JSON

This section describes how UniVerse converts XML data to JSON data.

- UniVerse ignores the XML declaration, namespaces, processing instructions, and XML comments.
- UniVerse ignores the XML schema reference, so no data type matching is performed. Since JSON only supports null, Boolean, string, and number data types, the only data type lost is number. UniVerse now has INFER type options which notify UDO to automatically infer the data type based on the text value of an element or attribute.
- UniVerse maps both XML elements and attributes to UDO properties.
- UniVerse changes the name of an attribute to `@name` to avoid potential naming conflicts, and to mark that it originated as an attribute and not an element. For a text node, UniVerse gives it a “`#text`” property name to indicate that it came from a text node.
- For an XML element, UniVerse groups all subelements with identical names into a JSON array, and preserves the order of these same elements. However, UniVerse may or may not preserve the order among different elements and text nodes, depending on if these elements and text nodes are mixed together in the original XML document.

- Since a UDO root object itself does not have a name, UniVerse ignores the root element by default. However a KEEPROOT option exists for keeping the root element in the UDO object. In this case, the UDO root object that UniVerse generates contains a single property with the same name as the XML root element.
- UniVerse skips text nodes that only have white space characters. For example, UniVerse skips the new line, tab and space characters between two XML tags, unless there are other non-white space characters mixed with them. In that case, you can enable the UDOOPTION\_XML2UDO\_TRIMWHITEPACE option to trim only the preceding and trailing white space in the text.

The following table lists all the patterns for converting an XML element to JSON.

XML	JSON	Reversible
<e/>	{"e": ""}	Y
<e>true</e> or <e>false</e>	{"e": "true"} or {"e": "false"}	Y
<e>text</e>	{"e": "text"}	Y
<e name="value" />	{"e": {"@name": "value"}}	Y
<e name="value"> text</e>	{"e": {"@name": "value", "#text": "text"}}	Y
<e><a>text</a><b>text</b></e>	{"e": { "a": "text", "b": "text" }}	Y
<e><a>text1</a><a>text2</a></e>	{"e": { "a": ["text1", "text2"] }}	Y
<e><a>text1</a><b>text</b><a>text2</a></e>	{"e": { "a": ["text1", "text2"], "b": "text" }}	N
<e> text <a>text</a></e>	{"e": {"#text": "text", "a": "text" }}	Y
<e>text1<a>text</a>text2</e>	{"e": { "#text": ["text1", "text2"], "a": "text" }}	N

Conversion Patterns

## Converting from UDO to XML

At this release, the UDOFORMAT\_XML flag has been added, in addition to the UDOFORMAT\_JSON flag. When you pass UDOFORMAT\_XML to the UDOWrite() function, it writes a UDO object in XML format instead of the native JSON format.

A number of new options have been added at this release to specify the XML document style, root element name, namespace, and so forth.

This section describes how UniVerse converts UDO data to XML data.

- If the name of a property on a UDO object is @xxxx, UniVerse writes the property as XML attribute “xxxx.” If the property is “text,” UniVerse writes it as an XML text node.

JSON	XML
{ "@name": "attribute", "#text": "more text here" }	<UDO name="attribute">more text here</UDO>

### JSON to XML

- The UDO to XML conversion guarantees the structure of the XML UniVerse generates by default. However, we do not guarantee that it is well-formed. Specifically, UniVerse does not check if a JSON property name is a valid XML name, and does not check for duplicate XML attribute names.
- The default root element name of the XML document is “UDO.”
- An empty UDO array does not produce any XML content. However, an empty UDO object result in an empty XML element.

JSON	XML
{ "myEmptyArray" : [] }	<UDO></UDO>
{ "myEmptyObj" : {} }	<UDO><myEmptyObj/></UDO>

### JSON to XML

---

## **U2 Dynamic Object API**

A U2 Dynamic Object (UDO) object contains a collection of properties, that is, name/value pairs. UDO uses the name in the pair to identify the property. The value in the pair can be a string, a number, true, false, null, an array, or another object. An array is an ordered list of values, and can be easily serialized into and deserialized from JSON data.

A UDO object can be viewed as a tree structure with each object and array a node in the tree. Each node can have a number of child nodes. A child node is an object or an array that is either a property value or an array item. The top-most object or array is the root node.

A stand-alone UDO object or array is not a property value on another UDO object and is not a UDO array element. A root node of a UDO tree is stand-alone.

### **UDO definitions**

A UDO type UniVerse BASIC variable can be passed to subroutines as a parameter. These definitions are defined in the UDO.H file.

#### ***UDO status codes:***

- UDO\_SUCCESS
- UDO\_ERROR
- UDO\_INVALIDHANDLE

#### ***UDO value types***

- UDO\_FALSE
- UDO\_TRUE
- UDO\_NULL
- UDO\_STRING
- UDO\_NUMBER
- UDO\_OBJECT

- UDO\_ARRAY

A UDO object property or a UDO array item can be any one of the above types.

#### *UDO error codes*

- UDOERROR\_OUTOFMEMORY
- UDOERROR\_INVALIDJSON
- UDOERROR\_NOTSTANDALONE
- UDOERROR\_INVALIDXML
- UDOERROR\_DOMAPIFAILURE
- UDOERROR\_NOTFOUND
- UDOERROR\_INVALIDNAME
- UDOERROR\_NOTANOBJECT
- UDOERROR\_NOTANARRAY
- UDOERROR\_INVALIDINDEX
- UDOERROR\_OUTOFBOUND
- UDOERROR\_INVALIDOPTION
- UDOERROR\_INVALIDOPTIONVALUE
- UDOERROR\_OPTIONNOTSET
- UDOERROR\_INVALIDFORMAT
- UDOERROR\_INVALIDVALUE
- UDOERROR\_INVALIDTYPE

#### *UDO flags*

- UDOFORMAT\_JSON
- UDOFORMAT\_XML

### ***UDO options***

- UDOOPTION\_XMLSTYLE
  - UDO\_XMLSYTLE\_ATTR
  - UDO\_XMLSTYLE\_ELEM
- UDOOPTION\_XMLNS
- UDOOPTION\_UDO2XML\_NAMESPACE
- UDOOPTION\_XML2UDO\_INFERNUMBER
- UDOOPTION\_XML2UDO\_INFERBOOLEAN
- UDOOPTION\_XML2UDO\_INFERNULL
- UDOOPTION\_XML2UDO\_EMPTY2NULL
- UDOOPTION\_XML2UDO\_TRIMWHITESPACE
- UDOOPTION\_XML2UDO\_CASEINSENTIVE
- UDOOPTION\_XML2UDO\_KEEPROOT
- UDOOPTION\_UDO2XML\_ROOTNAME
- UDOOPTION\_UDO2XML\_NULL2EMPTY
- UDOOPTION\_UDO2XML\_NAMESPACEPREFIX
- UDOOPTION\_OUTPUTMODE
  - UDO\_OUTPUT\_COMPACT
  - UDO\_OUTPUT\_FORMATTED

### ***UDO option values***

- UDO\_OPTION\_ON
- UDO\_OPTION\_OFF

## UDO functions

A UDO function always returns UDO\_SUCCESS when it completes with no errors. Otherwise, it returns UDO\_ERROR or UDO\_INVALIDHANDLE. If a UDO function fails, UDO leaves the output variables, if any, unchanged.

### UDORead

The UDORead function creates a UDO object from a JSON string or XMLstring.

#### Syntax

**UDORead**(*inputString*, *inputType*, *udoHandle*[out])

#### Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>inputString</i>	A JSON or XML string.
<i>inputtype</i>	UDOFORMAT_JSON or UDOFORMAT_XML.
<i>udoHandle</i> [out]	The UniVerse BASIC variable that holds a reference to the UDO object upon successful return of the function.

**UDORead Parameters**

## **UDOWrite**

Writes a UDO object in JSON or XML format.

### **Syntax**

**UDOWrite**(*udoHandle*, *outputType*, *outputString*[*out*])

### **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>udoHandle</i>	Must be a UDO type variable.
<i>outputType</i>	UDOFORMAT_JSON or UDOFORMAT_XML.
<i>outputString</i> [ <i>out</i> ]	The string that holds the serialized output.

**UDOWrite Parameters**

## **UDOSetOption**

Sets the options for the UDO API.

### **Syntax**

**UDOSetOption**(*option*, *value*)

### **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>option</i>	The UDOOPTION you want to set.
<i>value</i>	A string type option value. For a list of UDOOPTION_OUTPUTMODE values, see “ <a href="#">UDO options</a> ”.

**UDOSetOption Parameters**

## **UDOGetOption**

The UDOGetOption function gets the value of a UDO option.

### **Syntax**

**UDOGetOption**(*option*, *value*[out])

### **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>option</i>	The UDOOPTION you want to get.
<i>value</i> [out]	A string type option value. For a list of UDOOPTION_OUT-PUTMODE values, see “ <a href="#">UDO options</a> .”.

**UDOGetOption Parameters**

## **UDOCClone**

The UDOCClone function clones a UDO object or array so that changes to the new object or array will not affect the original object.

### **Syntax**

**UDOCClone**(*udoHandle*, *newUdoHandle*[out])

## **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>udoHandle</i>	Must be a UDO type variable.
<i>newUdoHandle</i>	When the UDOClone function returns successfully, newUDOHandle points to a stand-alone object or array that is the exact replication of the original object.

### **UDOCopy Parameters**

## **UDOFree**

The UDOFree function forcefully removes a UDO object or array from memory.

### **Syntax**

**UDOFree(udoHandle)**

The following table describes the parameter of the syntax.

Parameter	Description
<i>udoHandle</i>	Must be a stand-alone UDO object or array.

### **UDOFree Parameters**

UDO will clear all UniVerse BASIC variables that reference the object or array and its descendants. Any attempt to access these variables, other than assigning a new value, fails.

You should always call this function when a UDO object or array is no longer needed. This avoids a potential memory leak.

## **UDOGetProperty**

The UDOGetProperty function returns the value and type of property on the UDO object.

### **Syntax**

**UDOGetProperty(***udoHandle, name, value[out], value\_type[out]***)**

### **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>udoHandle</i>	Must be a UDO object.
<i>name</i>	The name of the property.
<i>value[out]</i>	If the property is a UDO_OBJECT or UDO_ARRAY type (it is either a UDO object or an array), the output value holds only a reference to the object or array. Further changes to the object or array through this reference, such as updating a property value on the object or removing an array item, affects the original object as well.  If the property is a UDO_STRING, UDO_NUMBER, UDO_TRUE, UDO_FALSE, or UDO_NULL type, the output variable value holds the actual value instead of a reference. Further changes to this variable do not affect the original property value.
<i>value_type[out]</i>	The type of the value returned by <i>value</i> . For a list of UDO types, see “ <a href="#">UDO value types</a> .”

### **UDOGetProperty Parameters**

## **UDOSetProperty**

The UDOSetProperty function creates or updates a property on a UDO object.

## Syntax

**UDOSetProperty**(*udoHandle*, *name*, *value*)

## Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>udoHandle</i>	Must be a UDO object.
<i>name</i>	The name of the property. If the property does not exist, UDO creates a new property for the object. If the property exists, the new value replaces the old value. If the old property is of UDO_OBJECT or UDO_ARRAY type, the old value, either a UDO object or an array, is marked as stand-alone and will be removed from memory if it is not referenced by any UniVerse BASIC variable. If the new value is of UDO_OBJECT or UDO_ARRAY type, it must be a stand-alone object or array, and it must not be the ancestor of the current UDO object.
<i>value</i>	The value of the property.

### UDOSetProperty Parameters

## UDODeleteProperty

The UDODeleteProperty function deletes a property from the UDO object.

## Syntax

**UDODeleteProperty**(*udoHandle*, *name*)

## **Parameter**

The following table describes each parameter of the syntax.

Parameter	Description
<i>udoHandle</i>	Must be a UDO object.
<i>name</i>	The name of the property. If the property is of UDO_OBJECT or UDO_ARRAY type, its value (either a UDO object or a UDO array) is marked as stand-alone and will be removed from memory if it is not referenced by any UniVerse BASIC variable.

### **UDOSetProperty Parameters**

## **UDOGetPropertyNames**

The UDOGetPropertyNames function returns a UDO array that holds the names of all the properties in the UDO object.

## **Syntax**

**UDOGetPropertyNames(***udoHandle*, *udoArray[out]***)**

## **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>udoHandle</i>	Must be a UDO object.
<i>udoArray[out]</i>	The UDO array to hold the names of all the properties in the UDO object.

### **UDOGetPropertyNames Parameters**

## **UDOGetNextProperty**

The UDOGetNextProperty function provides a convenient way to walk through all the properties in a UDO object, without needing to know the property names in advance.

When all properties on the UDO object are exhausted, the `UDOGetNextProperty()` function returns `UDO_ERROR`, then goes back to the first property.

We recommend that you avoid modifying the properties on a UDO object when calling the `UDOGetNextProperty()` to retrieve the properties.

## Syntax

`UDOGetNextProperty(udoHandle, name[out], value[out], value_type[out])`

## Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<code>udoHandle</code>	Must be a UDO type object.
<code>name[out]</code>	The name of the array that holds the names of all the properties in the UDO object.
<code>value[out]</code>	If the property is a <code>UDO_OBJECT</code> or <code>UDO_ARRAY</code> type (it is either a UDO object or an array), the output value holds only a reference to the object or array. Further changes to the object or array through this reference, such as updating a property value on the object or removing an array item, affects the original object as well.  If the property is a <code>UDO_STRING</code> , <code>UDO_NUMBER</code> , <code>UDO_TRUE</code> , <code>UDO_FALSE</code> , or <code>UDO_NULL</code> type, the output variable value holds the actual value instead of a reference. Further changes to this variable do not affect the original property value.
<code>value_type[out]</code>	The type of the value returned by <code>value</code> . For a list of UDO types, see “ <a href="#">UDO value types</a> .”

### `UDOGetNextProperty` Parameters

## **UDOCREATE**

The UDOCREATE function creates a UDO item of the type you specify.

### **Syntax**

**UDOCREATE(***udoType*, *udoHandle*[out])

### **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>udoType</i>	Must be one of UDO_OBJECT, UDO_ARRAY, UDO_TRUE, UDO_FALSE, or UDO_NULL.
<i>udoHandle</i>	If <i>udoType</i> is UDO_OBJECT, <i>udoHandle</i> holds an empty object. If <i>udoType</i> is UDO_ARRAY, <i>udoHandle</i> holds an empty array. If <i>udoType</i> is UDO_TRUE, UDO_FALSE, or UDO_NULL, <i>udoHandle</i> holds the specified value.

**UDOCREATE Parameters**

## **UDOGETTYPE**

The UDOGETTYPE() function gets the UDO value type of a UniVerse BASIC variable.

### **Syntax**

**UDOGETTYPE(***udoHandle*, *type*[out])

## **Parameters**

The following table describes each parameter of the syntax.

Parameters	Description
<i>udoHandle</i>	Can be a UDO handle, or a UniVerse BASIC string or number.
<i>type[out]</i>	The UDO value type. For a list of UDO types, see “ <a href="#">UDO value types</a> .”

UDOGetType Parameters

## **UDOIsTypeOf**

The UDOIsTypeOf() function tests the UDO value type of a UniVerse BASIC variable.

## **Syntax**

**UDOIsTypeOf(***udoHandle, type***)**

## **Parameters**

The following table describes each parameter of the syntax.

Parameters	Description
<i>udoHandle</i>	Can be a UDO handle, or a UniVerse BASIC string or number.
<i>type[in]</i>	The UDO value type. For a list of UDO types, see “ <a href="#">UDO value types</a> .”

UDOGetType Parameters

## **UDOArrayGetSize**

The UDOArrayGetSize() function gets the size of a UDO array.

## **Syntax**

**UDOArrayGetSize(*udoHandle*, *size*[out])**

## **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>udoHandle</i>	Must be a UDO array.
<i>size</i>	The size of the UDO array.
<b>UDOArrayGetSize Parameters</b>	

## **UDOArrayGetItem**

The UDOArrayGetItem() function returns a UDO array item by its index.

## **Syntax**

**UDOArrayGetItem(*udoHandle*, *index*, *value*[out], *value\_type*[out])**

## Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>udoHandle</i>	Must be a UDO array.
<i>index</i>	The position of the UDO array index returned. Must be a positive integer.
<i>value[out]</i>	<p>The UDO value type of the array item. If the array item is of UDO_OBJECT or UDO_ARRAY type, the output variable "item" holds only a reference to the object or array. Further changes to the object or array through this reference, such as updating a property value or removing an array item, affect the original item as well.</p> <p>If the array item is of UDO_STRING, UDO_NUMBER, UDO_TRUE, UDO_FALSE or UDO_NULL type, the output variable "item" holds the actual value instead of a reference. Further changes to this variable do not affect the original property value.</p>
<i>value_type[out]</i>	The type of the value returned by <i>value</i> . For a list of UDO types, see " <a href="#">UDO value types</a> ."

### UDOArrayGetItem Parameters

## UDOArraySetItem

The UDOArraySetItem() function sets or inserts a UDO array element at the position you specify.

### Syntax

**UDOArraySetItem(***udoHandle, index, value***)**

## **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>udoHandle</i>	Must be a UDO array.
<i>index</i>	The position what you want to set or insert the element. Must be a positive integer.
<i>value</i>	The value of the array item you are setting.

**UDOArraySetItem Parameters**

If the index is larger than the size of the array, UDO will pad the array with UDO\_NULL values before it inserts the array item into the array.

Otherwise, if the old array item is of UDO\_OBJECT or UDO\_ARRAY type, either an object or an array will be marked as stand-alone and removed from memory if it is not referenced by any UniVerse BASIC variable.

If the new array item is of UDO\_OBJECT or UDO\_ARRAY type, it must be a stand-alone object or array and it must not be the ancestor of the current UDO object.

## **UDOArrayInsertItem**

The UDOArrayInsertItem() function inserts a UDO array element at the position you specify by index.

### **Syntax**

**UDOArrayInsertItem(***udoHandle, index, value***)**

## **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>udoHandle</i>	Must be a UDO array.
<i>index</i>	The position what you want to insert the item. Must be a positive integer.
<i>value</i>	The value of the array item you are inserting.

**UDOArrayInsertItem Parameters**

If the index is larger than the size of the array, UDO will pad the array with UDO\_NULL values before it inserts the array item into the array.

## **UDOArrayAppendItem**

The UDOArrayAppendItem() function appends the item you specify to the UDO array.

### **Syntax**

**UDOArrayAppendItem(*udoHandle*, *value*)**

## **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>udoHandle</i>	Must be a UDO array.
<i>value</i>	The value of the array item you are appending.

**UDOArrayAppendItem Parameters**

If the new array item is of UDO\_OBJECT or UDO\_ARRAY type, it must be a stand-alone object or array, and it must not be the ancestor of the current UDO object.

## **UDOArrayDeleteItem**

The UDOArrayDeleteItem() function deletes the array item you specify by its index.

### **Syntax**

**UDOArrayDeleteItem**(*udoHandle*,*index*)

### **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>udoHandle</i>	Must be a UDO array.
<i>index</i>	The index of the item to be deleted. Must be a positive integer.

### **UDOArrayDeleteItem**

If the array item is of UDO\_ARRAY or UDO\_OBJECT type, UDO will make either the UDO object or a UDO array as stand-alone and will remove it from memory if it is not referenced by any UniVerse BASIC variable.

## **UDOArrayGetNextItem**

The UDOArrayGetNextItem() function returns the next UDO array item relative to the current position, which is the position of the array the last time it was accessed by this function. The initial position is 1.

### **Syntax**

**UDOArrayGetNextItem**(*udoHandle*, *value*[out], *type*[out])

## **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>udoHandle</i>	Must be a UDO array.
<i>value[out]</i>	The value of the item.
<i>type[out]</i>	The type of the value returned by <i>value</i> . For a list of UDO types, see <a href="#">“UDO value types.”</a> .

### **UDOArrayGetNextItem Parameters**

After exhausting the entire array, the UDOArrayGetNextItem() function returns UDO\_ERROR and the current position is reset to 1.

We recommend that you not modify the array when calling the UDOArrayGetNextItem() function. If you must modify the array, remember that UDOArrayGetNextItem() always returns the item at the current position +1.

## **UDOGetLastError**

If the previous UDO call returned UDO\_ERROR, use the UDOGetLastError() function to return the error code and error message.

### **Syntax**

**UDOGetLastError(errorCode[out], errorMessage[out])**

## **Parameters**

The following table describes each parameter of the syntax.

Parameter	Description
<i>errorCode</i>	The UDO error code. For a list of error code, see “ <a href="#">UDO error codes</a> .”
<i>errorMessage</i>	The UDO error message.

### **UDOGetLastError Parameters**

---

## **XML2UDO options and UDO2XML options**

This section describes the XML2UDO Options.

### **UDOOPTION\_XML2UDO\_INFERNUMBER**

You can enable the UDOOPTION\_XML2UDO\_INFERNUMBER option to instruct the UDORead() function to check if the text value, trimmed of white spaces, of an XML element or attribute is a number. If so, UniVerse creates a UDO property of a number type rather than a string type.

By default, this option is turned off.

If you change a XML2UDO option or a UDO2XML option, the changed value remains active until you exit the database shell.

#### ***Example***

The following example shows the UDOOPTION\_XML2UDO\_INFERNUMBER option.

```
UDOSetOption (UDOOPTION_XML2UDO_INFERNUMBER, UDO_OPTION_ON)  
or  
UDOSetOption (UDOOPTION_XML2UDO_INFERNUMBER, "ON")
```

#### ***Option Effect***

The following table illustrates the effect of this option:

---

	<b>XML</b>	<b>JSON</b>
Off	<e>1.732</e>	{"e":"1.732"}
On	<e>1.732</e>	{"e":1.732}
<hr/> <b>UDOOPTION_XML2UDO_INFERNUMBER Effect</b>		

---

## **UDOOPTION\_XML2UDO\_INFERBOOLEAN**

You can enable the UDOOPTION\_XML2UDO\_INFERBOOLEAN option to instruct the UDORead() function to check if the text value, trimmed of white spaces, of an XML element or attribute is a boolean value (true or false). If so, UniVerse creates a UDO property or true or false instead of a string type.

By default, this option is off.

### ***Example***

The following example shows the UDOOPTION\_XML2UDO\_INFERBOOLEAN option.

```
UDOSetOption(UDOOPTION_XML2UDO_INFERBOOLEAN, UDO_OPTION_ON)  
or  
UDOSetOption(UDOOPTION_XML2UDO_INFERBOOLEAN, "ON")
```

### ***Option Effect***

The following table illustrates the effect of this option:

	<b>XML</b>	<b>JSON</b>
Off	<e>true</e>	{"e": "true"}
On	<e>true</e>	{"e": true}
UDOOPTION_XML2UDO_INFERBOOLEAN Effect		

## **UDOOPTION\_XML2UDO\_INFERNULL**

You can enable the UDOOPTION\_XML2UDO\_INFERNULL option to instruct the UDORead() function to check if the text value, trimmed of white spaces, of an XML Element or attribute matches “null.” If so, UniVerse creates a UDO property of null type instead of string type.

By default, this option is off.

### ***Example***

The following example shows the UDOOPTION\_XML2UDO\_INFERNULL option.

```
UDOSetOption(UDOOPTION_XML2UDO_INFERNULL, UDO_OPTION_ON)
```

or

```
UDOSetOption(UDOOPTION_XML2UDO_INFERNULL, "ON")
```

### ***Option Effect***

The following table illustrates the effect of this option:

	<b>XML</b>	<b>JSON</b>
Off	<e>null</e>	{"e": "null"}
On	<e>null</e>	{"e": null}

UDOOPTION\_XML2UDO\_INFERNULL Effect

## **UDOOPTION\_XML2UDO\_EMPTY2NULL**

You can enable the UDOOPTION\_XML2UDO\_EMPTY2NULL option to instruct the UDORead() function to check if the text value, trimmed of white spaces, of an XML element or attribute is an empty string. If so, UniVerse creates a UDO property or null instead of a string type.

By default, this option is off.

### ***Example***

The following example shows the UDOOPTION\_XML2UDO\_EMPTY2NULL option.

```
UDOSetOption(UDOOPTION_XML2UDO_EMPTY2NULL, UDO_OPTION_ON)
```

or

```
UDOSetOption(UDOOPTION_XML2UDO_EMPTY2NULL, "ON")
```

### ***Option Effect***

The following table illustrates the effect of this option:

	<b>XML</b>	<b>JSON</b>
Off	<e/> or <e></e>	{"e": ""}
On	<e/> or <e></e>	{"e": null}
UDOOPTION_XML2UDO_EMPTY2NULL		

### **UDOOPTION\_XML2UDO\_TRIMWHITESPACE**

You can enable the UDOOPTION\_XML2UDO\_TRIMWHITESPACE option to instruct the UDORead() function to trim the preceding and trailing white space characters in the text value of an XML element or attribute.

By default, this option is off.

#### ***Example***

The following example shows the UDOOPTION\_XML2UDO\_TRIMWHITESPACE option.

```
UDOSetOption(UDOOPTION_XML2UDO_TRIMWHITESPACE,  UDO_OPTION_ON)  
or  
UDOSetOption(UDOOPTION_XML2UDO_TRIMWHITESPACE,  "ON")
```

## ***Option Effect***

The following table illustrates the effect of this option:

	<b>XML</b>	<b>JSON</b>
Off	<e attr=" attr value "> text </e>	{"@attr":" attr value ","#text":"text "}
On	<e attr=" attr value "> text </e>	{"@attr":"attr value","#text":"text"}

UDOOPTION\_XML2UDO\_TRIMWHITESPACE Effect

## **UDOOPTION\_XML2UDO\_CASEINSENSITIVE**

You can enable the UDOOPTION\_XML2UDO\_CASEINSENSITIVE option with the UDOOPTION\_XML2UDO\_INFERBOOLEAN and the UDOOPTION\_XML2UDO\_INFERNULL options to instruct the UDORead() function to ignore cases when checking if the text value of an XML element or attribute is true, false, or null.

By default, this option is off.

### ***Example***

The following example shows the UDOOPTION\_XML2UDO\_CASEINSENSITIVE option.

```
UDOSetOption(UDOOPTION_XML2UDO_CASEINSENSITIVE, UDO_OPTION_ON)
```

or

```
UDOSetOption(UDOOPTION_XML2UDO_CASEINSENSITIVE, "ON")
```

### ***Option Effect***

The following table illustrates the effect of this option:

	<b>XML</b>	<b>JSON</b>
Off	<e> True </e>	{"e": " True "}
	<e> Null </e>	{"e": " Null "}
On	<e> True </e>	{"e": true}
	<e> Null </e>	{"e": null}

UDOOPTION\_XML2UDO\_CASEINSENSITIVE Effect

### **UDOOPTION\_XML2UDO\_KEEPROOT**

You can enable the UDOOPTION\_XML2UDO\_KEEPROOT to instruct the UDORead() function to keep the XML root element in the UDO object.

By default, this option is off.

#### ***Example***

The following example shows the UDOOPTION\_XML2UDO\_KEEPROOT option.

```
UDOSetOption(UDOOPTION_XML2UDO_KEEPROOT, UDO_OPTION_ON)  
or  
UDOSetOption(UDOOPTION_XML2UDO_KEEPROOT, "ON")
```

## ***Option Effect***

The following table illustrates the effect of this option:

	<b>XML</b>	<b>JSON</b>
Off	<?xml version="1.0"?> <root> <e>text</e> </root>	{"e":"text"}
On	<?xml version="1.0"?> <root> <e>text</e> </root>	{ "root": {"e": "text"} }

UDOOPTION\_XML2UDO\_KEEPROOT Effect

## **UDOOPTION\_UDO2XML\_XMLSTYLE**

You can enable the UDOOPTION\_UDO2XML\_XMLSTYLE option to choose the XML style to use when using the UDOWrite() function to write a UDO object in XML.

The possible values are:

- UDO\_XMLSTYLE\_ATTR or “ATTRIBUTE”
- UDO\_XMLSTYLE\_ELEM or “ELEMENT”

The default value is UDO\_XMLSTYLE\_ATTR

### ***Example***

The following example shows the UDOOPTION\_UDO2XML\_XMLSTYLE option.

```
UDOSetOption(UDO_XMLSTYLE_ATTR)
```

or  
     UDOSetOption("ATTRIBUTE")  
 or  
     UDOSetOption(UDO\_XMLSTYLE\_ELEM)  
 or  
     UDOSetOption("ELEMENT")

### ***Option Effect***

The following table illustrates the effect of this option:

UDO: {"myKey": "myValue", "myKey1": "myValue1" }

<b>UDOOPTION_UDO2XML_XMLSTYLE</b>	<b>UDOWrite() Output</b>
UDO_XMLSTYLE_ATTR	<?xml version="1.0"?> <UDO myKey="myValue" myKey1="myValue1" />
UDO_XMLSTYLE_ELEM	<?xml version="1.0"?> <UDO> <myKey>myValue</myKey> <myKey1>myValue1</myKey1> </UDO>

#### **UDOOption\_UDO2XML\_STYLE Effect**

## **UDOOPTION\_UDO2XML\_NAMESPACE and UDOOPTION\_UDO2XML\_NAMESPACEPREFIX**

By default, the UDOWrite() does not include a namespace in the XML document when writing a UDO object in XML format. The NAMESPACEPREFIX is also blank by default. You can use the UDOOPTION\_UDO2XML\_NAMESPACE to specify an XML namespace. You can also use UDOOPTION\_UDO2XML\_NAMESPACEPREFIX to specify a prefix for that namespace.

## ***Example***

The following example shows the UDOOPTIONUDO2XML\_NAMESPACE and the UDOOPTIONUDO2XML\_NAMESPACEPREFIX options.

```
UDOSetOption(UDOOPTIONUDO2XML_NAMESPACE, "http://www.acme.com")  
UDOSetOption(UDOOPTIONUDO2XML_NAMESPACEPREFIX, "acme")
```

## ***Option Effect***

The following table illustrates the effect of this option:

	JSON	XML
No namespace specified	{“Key”: “Value”}	<?xml version="1.0"?> <UDO Key="Value" />
namespace="http://ww ww.acme.com"	{“Key”: “Value”}	<?xml version="1.0"?> <UDO xmlns="http://www.acme.com" Key="Value" />
namespace="http://w ww.acme.com" prefix="acme"	{“Key”: “Value”}	<?xml version="1.0"?> <UDO xmlns:acme="http://www.acme.c om" Key="Value" />
UDOOPTIONUDO2XML_NAMESPACE Effect		

## **UDOOPTIONUDO2XML\_ROOTNAME**

By default, the UDOWrite() function uses “UDO” as the root element name when writing a UDO object in XML. You can use the UDOOPTIONUDO2XML\_ROOTNAME option to change the root element name.

### **Example**

The following example shows the UDOOPTIONUDO2XML\_ROOTNAME option.

```
UDOSetOption(UDOOPTIONUDO2XML_ROOTNAME, "Order")
```

### ***Option Effect***

The following table illustrates the effect of this option:

	<b>JSON</b>	<b>XML</b>
No root name specified	{"ID":"1001", "PRICE": 300}	<?xml version="1.0"?> <UDO ID="1001" PRICE="300" />
root name="Order"	{"ID":"1001", "Price": 300}	<?xml version="1.0"?> <Order ID="1001" PRICE="300" />

UDOOPTIONUDO2XML\_ROOTNAME Effect

## **UDOOPTIONUDO2XML\_NULL2EMPTY**

You can enable the UDOOPTIONUDO2XML\_NULL2EMPTY option to output a UDO property of a null type as an empty XML element.

The default value for this option is Off.

### ***Example***

The following example shows the UDOOPTIONUDO2XML\_NULL2EMPTY option.

```
UDOSetOption(UDOOPTIONUDO2XML_NULL2EMPTY, UDO_OPTION_ON)  
UDOSetOption(UDOOPTIONUDO2XML_NULL2EMPTY, "ON")
```

## ***Option Effect***

The following table illustrates the effect of this option:

	<b>JSON</b>	<b>XML</b>
Off	{ "myKey" : null }	<?xml version="1.0"?> <UDO myKey="null" />
On	{ "myKey" : null }	<?xml version="1.0"?> <UDO myKey="" />

UDOOPTIONUDO2XMLNULL2EMPTY Effect

# Appendix A: MQSeries API for UniData and UniVerse reason codes

This appendix describes the MQSeries Application Program Interface Reason Codes.

A

---

Reason Code Number	Reason Code Name	Description
0	AMRC_NONE	The request was successful with no error or warning required.
1	AMRC_UNEXPECTED_ERR	An unexpected error occurred.
2	AMRC_INVALID_Q_NAME	The specified queue name was too long, or contained invalid characters.
3	AMRC_INVALID_SENDER_NAME	The specified sender service name was too long, contained invalid characters, or used the reserved prefix "SYSTEM."
4	AMRC_INVALID_RECEIVER_NAME	The specified receiver service name was too long, contained invalid characters, or used the reserved prefix "SYSTEM."
5	AMRC_INVALID_PUBLISHER_NAME	The specified publisher service name was too long, contained invalid characters, or used the reserved prefix "SYSTEM."
6	AMRC_INVALID_SUBSCRIBER_NAME	The specified subscriber name was too long, contained invalid characters, or used the reserved prefix "SYSTEM."
7	AMRC_INVALID_POLICY_NAME	The specified policy name was too long, contained invalid characters, or used the reserved prefix "SYSTEM."

---

#### MQSeries API Reason Codes

---

Reason Code Number	Reason Code Name	Description
8	AMRC_INVALID_MSG_NAME	The specified message name was too long, contained invalid characters, or used the reserved prefix "SYSTEM."
9	AMRC_INVALID_SESSION_NAME	The specified session name was too long, contained invalid characters, or used the reserved prefix "SYSTEM."
10	AMRC_INVALID_DIST_LIST_NAME	The specified distribution list name was too long, contained invalid characters, or used the reserved prefix "SYSTEM."
11	AMRC_POLICY_HANDLE_ERR	Reserved for future use.
12	AMRC_SERVICE_HANDLE_ERR	The service handle specified for a sender, receiver, distribution list, publisher, or subscriber was not valid.
13	AMRC_MSG_HANDLE_ERR	The specified message handle was not valid.
14	AMRC_SESSION_HANDLE_ERR	The specified session handle was not valid.
15	AMRC_BROWSE_OPTIONS_ERR	The specified browse options value was not valid or contained an invalid combination of options.
16	AMRC_INSUFFICIENT_MEMORY	There was not enough memory available to complete the requested operation.

---

**MQSeries API Reason Codes (Continued)**

Reason Code Number	Reason Code Name	Description
17	AMRC_WAIT_TIME_READ_ONLY	An attempt was made to set the wait time in a policy object for which the wait-time was read-only.
18	AMRC_SERVICE_NOT_FOUND	The specified (sender, receiver, distribution list, publisher, or subscriber) service was not found, so the request was not carried out.
19	AMRC_MSG_NOT_FOUND	The specified message was not found, so the request was not carried out.
20	AMRC_POLICY_NOT_FOUND	The specified policy was not found, so the request was not carried out.
21	AMRC_SENDER_NOT_UNIQUE	The specified name could not be resolved to a unique sender because more than one sender object with that name exists.
22	AMRC_RECEIVER_NOT_UNIQUE	The specified name could not be resolved to a unique receiver because more than one receiver object with that name exists.
23	AMRC_PUBLISHER_NOT_UNIQUE	The specified name could not be resolved to a unique publisher because more than one publisher object with that name exists.
24	AMRC_SUBSCRIBER_NOT_UNIQUE	The specified name could not be resolved to a unique subscriber because more than one subscriber object with that name exists.

---

**MQSeries API Reason Codes (Continued)**

---

Reason Code Number	Reason Code Name	Description
25	AMRC_MSG_NOT_UNIQUE	The specified name could not be resolved to a unique message because more than one message object with that name exists.
26	AMRC_POLICY_NOT_UNIQUE	The specified name could not be resolved to a unique policy because more than one policy with that name exists.
27	AMRC_DIST_LIST_NOT_UNIQUE	The specified name could not be resolved to a unique distribution list because more than one distribution list with that name exists.
28	AMRC_RECEIVE_BUFF_PTR_ERR	The buffer pointer specified for receiving data was not valid.
29	AMRC_RECEIVE_BUFF_LEN_ERR	The buffer length specified for receiving data was not valid.
30	AMRC_SEND_DATA_PTR_ERR	The buffer pointer specified for sending data was not valid.
31	AMRC_SEND_DATA_LEN_ERR	The data length specified for sending data was not valid.
32	AMRC_INVALID_IF SERVICE_OPEN	The requested operation could not be performed because the specified service (sender, receiver, publisher, or subscriber) was open.
33	AMRC_SERVICE_ALREADY_OPEN	The session was already closed (or terminated).

---

**MQSeries API Reason Codes (Continued)**

Reason Code Number	Reason Code Name	Description
34	AMRC_DATA_SOURCE_NOT_UNIQUE	Message data for a send operation was passed in an application data buffer or a file, and was also found in the specified message object. Data to be sent can be included in an application buffer or a message object, but not both. Similarly, data can be included in a file or a message object, but not both. If data is sent in an application buffer or file, the message object can be reset first to remove existing data.
35	AMRC_NO_MSG_AVAILABLE	No message was available for a receive request after the specified wait time.
36	AMRC_SESSION_ALREADY_OPEN	The session was already open (or initialized).
37	AMRC_SESSION_ALREADY_CLOSED	The session was already closed (or terminated).
38	AMRC_ELEM_NOT_FOUND	The specified element was not found.
39	AMRC_ELEM_COUNT_PTR_ERR	The specified element count pointer was not valid.
40	AMRC_ELEM_NAME_PTR_ERR	The specified element name pointer was not valid.
41	AMRC_ELEM_NAME_LEN_ERR	The specified element name length value was not valid.
42	AMRC_ELEM_INDEX_ERR	The specified element index value was not valid.

**MQSeries API Reason Codes (Continued)**

---

Reason Code Number	Reason Code Name	Description
43	AMRC_ELEM_PTR_ERR	The specified element pointer was not valid.
44	AMRC_ELEM_STRUC_ERR	The specified element structure was not valid. The structure id, version, or a reserved field contained an invalid value.
45	AMRC_ELEM_STRUC_NAME_ERR	At least one of the name (length and pointer) fields in the specified element structure was not valid. Ensure that the name length, pointer, and name string are valid.
46	AMRC_ELEM_STRUC_VALUE_ERR	At least one of the value (length and pointer) fields in the specified element structure was not valid. Ensure that the value length, pointer, and value string are valid.
47	AMRC_ELEM_STRUC_NAME_BUFF_ERR	At least one of the name buffer (length and pointer) fields in the specified element structure was not valid.
48	AMRC_ELEM_STRUC_VALUE_BUFF_ERR	At least one of the value buffer (length and pointer) fields in the specified structure was not valid.

---

**MQSeries API Reason Codes (Continued)**

---

Reason Code Number	Reason Code Name	Description
49	AMRC_TRANSPORT_ERR	An error was reported by the underlying (MQSeries) message transport layer. The message transport reason code can be obtained by the secondary reason code value returned from a "GetLastError" request for the AMI object concerned.
50	AMRC_TRANSPORT_WARNING	A warning was reported by the underlying (MQSeries) message transport layer. The message transport reason code can be obtained by the secondary reason code value returned from a "GetLastError" request for the AMI object concerned.
51	AMRC_ENCODING_INCOMPLETE	The message contains mixed values for integer, decimal, and floating point encodings, one or more of which are undefined. The encoding value returned to the application reflects only the encoding values that were defined.
52	AMRC_ENCODING_MIXED	The message contains mixed values for integer, decimal, and floating point encodings, one or more of which conflict. An encoding value of undefined was returned to the application.
53	AMRC_ENCODING_ERR	The specified encoding value was not valid.

---

**MQSeries API Reason Codes (Continued)**

---

Reason Code Number	Reason Code Name	Description
54	AMRC_BEGIN_INVALID	The begin request was not valid because there were no participating resource managers registered.
55	AMRC_NO_REPLY_TO_INFO	A response sender service specified when attempting to receive a request message was not updated with reply-to information because the request message contained no reply-to information. An attempt to send a reply message using the response send will fail.
56	AMRC_SERVICE_ALREADY_CLOSED	The specified (sender, receiver, distribution list, publisher, or subscriber) service was already closed).
57	AMRC_SESSION_NOT_OPEN	The request failed because the session was not open.
58	AMRC_DIST_LIST_INDEX_ERR	The specified distribution list index value was not valid.
59	AMRC_WAIT_TIME_ERR	The specified wait-time value was not valid.
60	AMRC_SERVICE_NOT_OPEN	The request failed because the specified (sender, receiver, distribution list, publisher, or subscriber) service was not open.
61	AMRC_HEADER_TRUNCATED	The RFH header of the message was truncated.
62	AMRC_HEADER_INVALID	The RFH header of the message was not valid.

---

**MQSeries API Reason Codes (Continued)**

---

Reason Code Number	Reason Code Name	Description
63	AMRC_DATA_LEN_ERR	The specified data length was not valid.
64	AMRC_BACKOUT_REQUEUE_ERR	The backout count of a received message was found to have exceeded its backout limit. The message was returned to the application. It could not be requeued to the backout requeue queue.
65	AMRC_BACKOUT_LIMIT_ERR	The backout count of a received message was found to have exceeded its backout limit. The message was returned to the application. It could not be requeued to the backout requeue queue.
66	AMRC_COMMAND_ALREADY_EXISTS	A publish, subscribe, or unsubscribe command could not be added to the message because the message already contained a command element. If this message is generated from the high-level interface, it may mean that you have tried to use the same message name for sending and receiving publish/subscribe messages. It can also occur if the same message object is reused to send a message without being reset.

---

**MQSeries API Reason Codes (Continued)**

---

Reason Code Number	Reason Code Name	Description
67	AMRC_UNEXPECTED_RECEIVE_ERR	An unexpected error occurred after a received message was removed from the underlying transport layer. The message was returned to the application.
68	AMRC_UNEXPECTED_SEND_ERR	An unexpected error occurred after a message was successfully sent. Output information updated as a result of the send request should never occur.
70	AMRC_SENDER_USAGE_ERR	The specified sender service definition type was not valid for sending responses. To be valid for sending a response, a sender service must not have a repository definition, must have been specified as a response service when receiving a previous request message, and must not have been used for any purpose other than sending responses.
71	AMRC_MSG_TRUNCATED	The received message that was returned to the application has been truncated.
72	AMRC_CLOSE_SESSION_ERR	An error occurred while closing the session. The session is closed.
73	AMRC_READ_OFFSET_ERR	The current data offset used for reading bytes from a message is not valid.

---

**MQSeries API Reason Codes (Continued)**

Reason Code Number	Reason Code Name	Description
74	AMRC_RFH_ALREADY_EXISTS	A publish, subscribe, or unsubscribe command could not be added to the message because the message already contained an RFH header. The message requires a reset first, to remove existing data.
75	AMRC_GROUP_STATUS_ERR	The specified group status value was not valid.
76	AMRC_MSG_ID_LEN_ERR	The specified message id length value was not valid.
77	AMRC_MSG_ID_PTR_ERR	The specified message id pointer was not valid.
78	AMRC_MSG_ID_BUFF_LEN_ERR	The specified message id buffer length value was not valid.
79	AMRC_MSG_ID_BUFF_PTR_ERR	The specified message id buffer pointer was not valid.
80	AMRC_MSG_ID_LEN_PTR_ERR	The specified message id length pointer was not valid.
81	AMRC_CORREL_ID_LEN_ERR	The specified correlation id length value was too long.
82	AMRC_CORREL_ID_PTR_ERR	The specified correlation id pointer was not valid.
83	AMRC_CORREL_ID_BUFF_LEN_ERR	The specified correlation id buffer length value was not valid.
84	AMRC_CORREL_ID_BUFF_PTR_ERR	The specified correlation id buffer pointer was not valid.

**MQSeries API Reason Codes (Continued)**

---

Reason Code Number	Reason Code Name	Description
85	AMRC_CORREL_ID_LEN_PTR_ERR	The specified correlation id pointer was not valid.
86	AMRC_FORMAT_LEN_ERR	The specified message format string was too long.
87	AMRC_FORMAT_PTR_ERR	The specified format pointer was not valid.
88	AMRC_FORMAT_BUFF_PTR_ERR	The specified format buffer pointer was not valid.
89	AMRC_FORMAT_LEN_PTR_ERR	The specified format length pointer was not valid.
90	AMRC_FORMAT_BUFF_LEN_ERR	The specified format buffer length value was not valid.
91	AMRC_NAME_BUFF_PTR_ERR	The specified name buffer pointer was not valid.
92	AMRC_NAME_LEN_PTR_ERR	The specified name length pointer was not valid.
93	AMRC_NAME_BUFF_LEN_ERR	The specified name buffer length value was not valid.
94	AMRC_Q_NAME_LEN_ERR	The specified queue name length value was not valid.
95	AMRC_Q_NAME_PTR_ERR	The specified queue name pointer was not valid.
96	AMRC_Q_NAME_BUFF_PTR_ERR	The specified queue name buffer pointer was not valid.
97	AMRC_Q_NAME_LEN_PTR_ERR	The specified queue name length pointer was not valid.
98	AMRC_Q_NAME_BUFF_LEN_ERR	The specified queue name buffer length value was not valid.

---

**MQSeries API Reason Codes (Continued)**

Reason Code Number	Reason Code Name	Description
99	AMRC_WAIT_TIME_PTR_ERR	The specified wait time pointer was not valid.
100	AMRC_CCSID_PTR_ERR	The specified coded character set id pointer was not valid.
101	AMRC_ENCODING_PTR_ERR	The specified encoding pointer was not valid.
102	AMRC_DEFN_TYPE_PTR_ERR	The specified definition type pointer was not valid.
103	AMRC_CCSID_ERR	The specified coded character value was not valid.
104	AMRC_DATA_LEN_PTR_ERR	The specified data length pointer was not valid.
105	AMRC_GROUP_STATUS_PTR_ERR	The specified group status pointer was not valid.
106	AMRC_DATA_OFFSET_PTR_ERR	The specified data offset pointer was not valid.
107	AMRC_RESP_SENDER_HANDLE_ERR	The response sender service handle specified when receiving a request message was not valid.
108	AMRC_RESP_RECEIVER_HANDLE_ERR	The response receiver service handle specified when sending a request message was not valid.
109	AMRC_NOT_AUTHORIZED	The user is not authorized by the underlying transport layer to perform the specified request.

**MQSeries API Reason Codes (Continued)**

---

Reason Code Number	Reason Code Name	Description
110	AMRC_TRANSPORT_NOT_AVAILABLE	The underlying transport layer is not available.
111	AMRC_BACKED_OUT	The unit of work has been backed out.
112	AMRC_INCOMPLETE_GROUP	The specified request failed because an attempt was made to send a message that was not in a group when the existing message group was incomplete.
113	AMRC_SEND_DISABLED	The specified request could not be performed because the service in the underlying transport layer is not enabled for send requests.
114	AMRC_SERVICE_FULL	The specified (sender, receiver, distribution list, publisher, or subscriber) service was already open.
115	AMRC_NOT_CONVERTED	Data conversion of the received message was unsuccessful. The message was removed from the underlying message transport layer with the message data unconverted.
116	AMRC_RECEIVE_DISABLED	The specified request could not be performed because the service in the underlying transport layer is not enabled for receive requests.

---

**MQSeries API Reason Codes (Continued)**

---

Reason Code Number	Reason Code Name	Description
117	AMRC_GROUP_BACKOUT_LIMIT_ERR	The backout count of a received message was found to have exceeded its backout limit. The message was returned to the application. It was not requeued to the backout requeue queue because it represented a single message within a group of more than one.
118	AMRC_SENDER_COUNT_PTR_ERR	The specified distribution list sender count pointer was not valid.
119	AMRC_MULTIPLE_REASONS	A distribution list open or send was only partially successful and returned multiple different reason codes in its underlying sender services.
120	AMRC_NO_RESP_SERVICE	The publish request was not successful because a response receiver service is required for registration and was not specified.
121	AMRC_DATA_PTR_ERR	The specified data pointer was not valid.
122	AMRC_DATA_BUFF_LEN_ERR	The specified data buffer length value was not valid.
123	AMRC_DATA_BUFF_PTR_ERR	The specified data buffer pointer was not valid.

---

**MQSeries API Reason Codes (Continued)**

---

Reason Code Number	Reason Code Name	Description
124	AMRC_DEFN_TYPE_ERR	The definition type defined for the service point in the repository was inconsistent with the definition type of the underlying message transport queue object when it was opened.
125	AMRC_BACKOUT_INVALID	The backout request was not valid. On OS/390 under CICS, IMS, or RRS, this can be due to calling the AMI backout functions rather than the transaction managers' own functions.
126	AMRC_COMMIT_INVALID	The commit request was not valid. On OS/390 under CICS, IMS, or RRS, this can be due to calling the AMI commit functions rather than the transaction managers' own functions.
127	AMRC_DATA_OFFSET_ERR	The specified data offset value was not valid.
128	AMRC_FILE_SYSTEM_ERR	A file system error occurred during a file transfer call. If this occurs, we recommend that the current unit of work be backed out. This will ensure that messages put to or received from the service are in a consistent state.

---

**MQSeries API Reason Codes (Continued)**

Reason Code Number	Reason Code Name	Description
129	AMRC_FILE_ALREADY_EXISTS	The AMI was unable to receive the file as the current file disposition is “new,” and a file with the same name already exists on your system. The first message of the file transfer is returned to the application. If this occurs, we recommend that the current unit of work is backed out. This will ensure that the messages received from the service are in a consistent state.
130	AMRC_REPORT_CODE_PTR_ERR	The specified report code pointer was not valid.
131	AMRC_MSG_TYPE_PTR_ERR	The specified message type pointer was not valid.
132	AMRC_FILE_FORMAT_CONVERTED	The AMI received a file successfully, but needed to convert between different file types. An example is from an OS/390 fixed-length data set to a UNIX file or between OS/390 datasets with different geometries.
133	AMRC_FILE_TRUNCATED	On a file send or receive operation, the entire file was not processed. We recommend that the current unit of work is backed out. This will ensure that the messages put to or received from the service are in a consistent state.

---

**MQSeries API Reason Codes (Continued)**

---

---

Reason Code Number	Reason Code Name	Description
134	AMRC_FILE_NOT_FOUND	The file supplied on a file send call could not be opened. Check that the file exists and that the application has read access to it.
135	AMRC_NOT_A_FILE	A message was received from the service, but it does not appear to have been sent as part of a (physical mode) file transfer operation. The message is returned to the application.
136	AMRC_FILE_NAME_LEN_ERR	The file name length passed in to a file transfer call was not valid.
137	AMRC_FILE_NAME_PTR_ERR	The file name pointer passed in to a file transfer call was not valid.
138	AMRC_RFH2_FORMAT_ERR	The format of an MQRFH2 rules and formatting header of a received message was not valid.

---

**MQSeries API Reason Codes (Continued)**

---

Reason Code Number	Reason Code Name	Description
139	AMRC_CCSID_NOT_SUPPORTED	<p><b>Warning:</b> OS/390 V2 R9 (or later) is required to enable AMI publish subscribe or message element support under CICS. Ensure that your Language Environment installation is set up to use Unicode character conversion. See the <i>OS/390 C/C++ Programming Guide</i> for a list of the coded character sets supported under OS/390.</p> <p><b>Failure:</b> The coded character set of name/value elements in the rules and formatting header of a received message, or that specified for passing elements between the application and the AMI, is not supported.</p>
140	AMRC_FILE_MSG_FORMAT_ERR	<p>When using physical mode file transfer, only two message formats are allowed: AMFMT_STRING (for text mode transfer), and AMFMT_NONE (for binary mode transfer). When using logical mode file transfer, any message format may be used for messages generated from OS/390 datasets. On other platforms, and for HFS files on OS/390, only AMFMT_STRING and AMFMT_NONE can be used.</p>

---

#### MQSeries API Reason Codes (Continued)

---

Reason Code Number	Reason Code Name	Description
141	AMRC_MSG_TYPE_NOT_REPORT	The message is not a report message.
142	AMRC_ELEM_STRUC_TYPE_ERR	At least one of the type (length and pointer) fields in the specified element structure was not valid.
143	AMRC_ELEM_STRUC_TYPE_BUFF_ERR	At least one of the type buffer (length and pointer) fields in the specified element structure was not valid. Ensure that the type length, pointer, and type string are valid.
144	AMRC_FILE_TRANSFER_INVALID	An application running under CICS on OS/390 tried to perform a file transfer operation, which is invalid in this environment.
145	AMRC_FILE_NOT_WRITTEN	The file used for a receive could not be opened. The first message of the file is returned to the application. If this occurs, we recommend that the current unit of work is backed out. This will ensure that the messages held on the service are in a consistent state.
146	AMRC_FILE_FORMAT_NOT_SUPPORTED	An attempt was made to send a file type that is not supported. Unsupported file types include OS/390 VSAM datasets, and OS/390 partitioned datasets (though an individual member of a PDS may be sent).

---

**MQSeries API Reason Codes (Continued)**

---

Reason Code Number	Reason Code Name	Description
147	AMRC_NEGATIVE_RECEIVE_BUFF_LEN	The value of the buffer length parameter that is specified on a receive message request was negative.
148	AMRC_LIBRARY_NOT_FOUND	A policy handler library file name specified in the repository was not found in the handler's directory.
149	AMRC_LIBRARY_FUNCTION_PTR_ERR	A policy handler library that is specified by the repository attempted to register a function with an invalid function pointer value (for example, NULL).
150	AMRC_LIBRARY_INV_POINT_ERR	A policy handler library that is specified by the repository attempted to register a function with an invocation point value that was not valid.
151	AMRC_LIBRARY_DUP_FUNCTION	A policy handler library that is specified by the repository attempted to register a function with an invocation point value that is already registered.

---

**MQSeries API Reason Codes (Continued)**

---

Reason Code Number	Reason Code Name	Description
152	AMRC_POLICY_HANDLER_ERR	An error was returned from a policy handler library invocation that occurred while processing the application function call. The policy handler reason code can be obtained by the secondary reason code value returned from a getLastError request for the AMI object concerned.
153	AMRC_POLICY_HANDLER_WARNING	A warning was returned from a policy handler library invocation that occurred while processing the application function call. The policy handler reason code can be obtained by the secondary reason code value returned from a getLastError request for the AMI object concerned.
154	AMRC_REPORT_CODE_ERR	The specified report (or feedback) code value was not valid.
201	AMRC_ACCEPT_DIRECT_ERR	The specified accept direct requests value was not valid.
202	AMRC_ACCEPT_DIRECT_PTR_ERR	The specified accept direct requests pointer was not valid.
203	AMRC_ACCEPT_TRUNCATED_ERR	The specified accept truncated value was not valid.

---

**MQSeries API Reason Codes (Continued)**

Reason Code Number	Reason Code Name	Description
204	AMRC_ACCEPT_TRUNCATED_PTR_ERR	The specified accept truncated pointer was not valid.
205	AMRC_ANON_ERR	The specified anonymous value was not valid.
206	AMRC_ANON_PTR_ERR	The specified anonymous pointer was not valid.
207	AMRC_APPL_GROUP_BUFF_LEN_ERR	The specified application group buffer length value was not valid.
208	AMRC_APPL_GROUP_BUFF_PTR_ERR	The specified application group buffer pointer was not valid.
209	AMRC_APPL_GROUP_LEN_ERR	The specified application group length value was not valid.
210	AMRC_APPL_GROUP_LEN_PTR_ERR	The specified application group length pointer was not valid.
211	AMRC_APPL_GROUP_PTR_ERR	The specified application group pointer was not valid.
212	AMRC_BIND_ON_OPEN_ERR	The specified bind on open value was not valid.
213	AMRC_BIND_ON_OPEN_PTR_ERR	The specified bind on open pointer was not valid.
214	AMRC_CHL_NAME_BUFF_LEN_ERR	The specified channel name buffer length value was not valid.
215	AMRC_CHL_NAME_BUFF_PTR_ERR	The specified channel name buffer pointer was not valid.

**MQSeries API Reason Codes (Continued)**

---

Reason Code Number	Reason Code Name	Description
216	AMRC_CHL_NAME_LEN_ERR	The specified channel name length value was not valid.
217	AMRC_CHL_NAME_LEN_PTR_ERR	The specified channel name length pointer was not valid.
218	AMRC_CHL_NAME_PTR_ERR	The specified channel name pointer was not valid.
219	AMRC_CLOSE_DELETE_ERR	The specified close delete value was not valid.
220	AMRC_CLOSE_DELETE_PTR_ERR	The specified close delete pointer was not valid.
221	AMRC_CONTEXT_ERR	The specified message context value was not valid.
222	AMRC_CONTEXT_PTR_ERR	The specified message context pointer was not valid.
223	AMRC_CONVERT_ERR	The specified convert message value was not valid.
224	AMRC_CONVERT_PTR_ERR	The specified convert message pointer was not valid.
225	AMRC_COUNT_ERR	The specified backout or retry count value was not valid.
226	AMRC_COUNT_PTR_ERR	The specified backout or retry count pointer was not valid.
227	AMRC_CUST_PARM_BUFF_LEN_ERR	The specified custom parameter buffer length value was not valid.

---

**MQSeries API Reason Codes (Continued)**

Reason Code Number	Reason Code Name	Description
228	AMRC_CUST_PARM_BUFF_PTR_ERR	The specified customer parameter buffer pointer was not valid.
229	AMRC_CUST_PARM_LEN_ERR	The specified custom parameter length value was not valid.
230	AMRC_CUST_PARM_LEN_PTR_ERR	The specified custom parameter length pointer was not valid.
231	AMRC_CUST_PARM_PTR_ERR	The specified custom parameter pointer was not valid.
232	AMRC_DLY_PERSISTENCE_ERR	The specified delivery persistence value was not valid.
233	AMRC_DLY_PERSISTENCE_PTR_ERR	The specified delivery persistence pointer was not valid.
234	AMRC_DST_SUPPORT_ERR	The specified distribution list support value was not valid.
235	AMRC_DST_SUPPORT_PTR_ERR	The specified distribution list support pointer was not valid.
236	AMRC_EXPIRY_ERR	The specified message expiry value was not valid.
237	AMRC_EXPIRY_PTR_ERR	The specified message expiry pointer was not valid.
238	AMRC_FILE_DISP_ERR	The specified file disposition value was not valid.

**MQSeries API Reason Codes (Continued)**

---

Reason Code Number	Reason Code Name	Description
239	AMRC_FILE_DISP_PTR_ERR	The specified file disposition pointer was not valid.
240	AMRC_FILE_RCD_LEN_ERR	The specified file record length value was not valid.
241	AMRC_FILE_RCD_LEN_PTR_ERR	The specified file record length pointer was not valid.
242	AMRC_GROUP_ID_BUFF_LEN_ERR	The specified group id group buffer length value was not valid.
243	AMRC_GROUP_ID_BUFF_PTR_ERR	The specified group id buffer pointer was not valid.
244	AMRC_GROUP_ID_LEN_ERR	The specified group id length value was not valid.
245	AMRC_GROUP_ID_LEN_PTR_ERR	The specified group id length pointer was not valid.
246	AMRC_GROUP_ID_PTR_ERR	The specified group id pointer was not valid.
247	AMRC_HANDLE_POISON_MSG_ERR	The specified handle poison message value was not valid.
248	AMRC_HANDLE_POISON_MSG_PTR_ERR	The specified handle poison message pointer was not valid.
249	AMRC_HANDLE_PTR_ERR	The specified handle pointer was not valid.
250	AMRC_IMPL_OPEN_ERR	The specified implicit open value was not valid.
251	AMRC_IMPL_OPEN_PTR_ERR	The specified implicit open pointer was not valid.

---

**MQSeries API Reason Codes (Continued)**

Reason Code Number	Reason Code Name	Description
252	AMRC_INFORM_IF_RET_ERR	The specified inform if retained value was not valid.
253	AMRC_INFORM_IF_RET_PTR_ERR	The specified inform if retained pointer was not valid.
254	AMRC_INTERVAL_ERR	The specified retry interval value was not valid.
255	AMRC_INTERVAL_PTR_ERR	The specified retry interval pointer was not valid.
256	AMRC_LEAVE_OPEN_ERR	The specified leave open value was not valid.
257	AMRC_LEAVE_OPEN_PTR_ERR	The specified leave open pointer was not valid.
258	AMRC_LOCAL_ERR	The specified publish or subscribe locally value was not valid.
259	AMRC_LOCAL_PTR_ERR	The specified publish or subscribe locally pointer was not valid.
260	AMRC_MCD_PARM_BUFF_LEN_ERR	The specified MCD parameter buffer length value was not valid.
261	AMRC_MCD_PARM_BUFF_PTR_ERR	The specified MCD parameter buffer pointer was not valid.
262	AMRC_MCD_PARM_LEN_ERR	The specified MCD parameter length value was not valid.
263	AMRC_MCD_PARM_LEN_PTR_ERR	The specified MCD parameter length pointer was not valid.

---

**MQSeries API Reason Codes (Continued)**

---

---

Reason Code Number	Reason Code Name	Description
264	AMRC_MCD_PARM_PTR_ERR	The specified MCD parameter pointer was not valid.
265	AMRC_MGR_NAME_BUFF_LEN_ERR	The specified queue manager name buffer length value was not valid.
266	AMRC_MGR_NAME_BUFF_PTR_ERR	The specified queue manager name buffer pointer was not valid.
267	AMRC_MGR_NAME_LEN_ERR	The specified queue manager name length value was not valid.
268	AMRC_MGR_NAME_LEN_PTR_ERR	The specified queue manager name length pointer was not valid.
269	AMRC_MGR_NAME_PTR_ERR	The specified queue manager name pointer was not valid.
270	AMRC_MSG_LEN_ERR	The specified message length value was not valid.
271	AMRC_MSG_LEN_PTR_ERR	The specified message length pointer was not valid.
272	AMRC_MSG_TYPE_ERR	The specified message type value was not valid.
273	AMRC_NEW_CORREL_ID_ERR	The specified new correlation id value was not valid.
274	AMRC_NEW_CORREL_ID_PTR_ERR	The specified new correlation id pointer was not valid.

---

**MQSeries API Reason Codes (Continued)**

Reason Code Number	Reason Code Name	Description
275	AMRC_NEW_PUBS_ONLY_ERR	The specified new publications only value was not valid.
276	AMRC_NEW_PUBS_ONLY_PTR_ERR	The specified new publications only pointer was not valid.
277	AMRC_PERSISTENCE_ERR	The specified persistence value was not valid.
278	AMRC_PERSISTENCE_PTR_ERR	The specified persistence pointer was not valid.
279	AMRC_PRIORITY_ERR	The specified priority value was not valid.
280	AMRC_PRIORITY_PTR_ERR	The specified priority pointer was not valid.
281	AMRC_PUB_ON_REQ_ERR	The specified publish on request value was not valid.
282	AMRC_PUB_ON_REQ_PTR_ERR	The specified publish on request pointer was not valid.
283	AMRC_PUB_OTHERS_ONLY_ERR	The specified publish to others only value was not valid.
284	AMRC_PUB_OTHERS_ONLY_PTR_ERR	The specified publish to others only pointer was not valid.
285	AMRC_READ_ONLY_ERR	The specified wait time read only value was not valid.
286	AMRC_READ_ONLY_PTR_ERR	The specified wait time read only pointer was not valid.

**MQSeries API Reason Codes (Continued)**

---

Reason Code Number	Reason Code Name	Description
287	AMRC_REMOVE_ALL_ERR	The specified remove all subscriptions value was not valid.
288	AMRC_REMOVE_ALL_PTR_ERR	The specified remove all subscriptions pointer was not valid.
289	AMRC_REPORT_OPTION_ERR	The specified report option value was not valid.
290	AMRC_REPORT_OPTION_PTR_ERR	The specified report option pointer was not valid.
291	AMRC_RETAIN_ERR	The specified retain publications value was not valid.
292	AMRC_RETAIN_PTR_ERR	The specified retain publications pointer was not valid.
293	AMRC_SEGMENT_ERR	The specified segment message value was not valid.
294	AMRC_SEGMENT_PTR_ERR	The specified segment message pointer was not valid.
295	AMRC_SEQ_NO_ERR	The specified sequence number value was not valid.
296	AMRC_SEQ_NO_PTR_ERR	The specified sequence number pointer was not valid.
297	AMRC_SET_NAME_INVALID	The specified name cannot be changed.
298	AMRC_SHARED_ERR	The specified open shared value was not valid.

---

**MQSeries API Reason Codes (Continued)**

Reason Code Number	Reason Code Name	Description
299	AMRC_SHARED_PTR_ERR	The specified open shared pointer was not valid.
300	AMRC SND_TYPE_ERR	The specified sender type value was not valid.
301	AMRC SND_TYPE_PTR_ERR	The specified sender type pointer was not valid.
302	AMRC SRV_TYPE_ERR	The specified service type value was not valid.
303	AMRC SRV_TYPE_PTR_ERR	The specified service type pointer was not valid.
304	AMRC SPLIT_LOGICAL_ERR	The specified split logical value was not valid.
305	AMRC SPLIT_LOGICAL_PTR_ERR	The specified split logical pointer was not valid.
306	AMRC SUBS_POINT_BUFF_LEN_ERR	The specified subscription point buffer length value was not valid.
307	AMRC SUBS_POINT_BUFF_PTR_ERR	The specified subscription point buffer pointer was not valid.
308	AMRC SUBS_POINT_LEN_ERR	The specified subscription point length value was not valid.
309	AMRC SUBS_POINT_LEN_PTR_ERR	The specified subscription point length pointer was not valid.
310	AMRC SUBS_POINT_PTR_ERR	The specified subscription point pointer was not valid.
311	AMRC SUPPRESS_REG_ERR	The specified suppress registration value was not valid.

---

**MQSeries API Reason Codes (Continued)**

---

Reason Code Number	Reason Code Name	Description
312	AMRC_SUPPRESS_REG_PTR_ERR	The specified suppress registration pointer was not valid.
313	AMRC_SYNCPOINT_ERR	The specified sync point value was not valid.
314	AMRC_SYNCPOINT_PTR_ERR	The specified sync point pointer was not valid.
315	AMRC_TCP_ADDR_BUFF_LEN_ERR	The specified TCP/IP address buffer length value was not valid.
316	AMRC_TCP_ADDR_BUFF_PTR_ERR	The specified TCP/IP address buffer pointer was not valid.
317	AMRC_TCP_ADDR_LEN_ERR	The specified TCP/IP address length value was not valid.
318	AMRC_TCP_ADDR_LEN_PTR_ERR	The specified TCP/IP address length pointer was not valid.
319	AMRC_TCP_ADDR_PTR_ERR	The specified TCP/IP address pointer was not valid.
320	AMRC_TRP_TYPE_ERR	The specified transport type value was not valid.
321	AMRC_TRP_TYPE_PTR_ERR	The specified transport type pointer was not valid.
322	AMRC_TRUSTED_ERR	The specified trusted value was not valid.
323	AMRC_TRUSTED_PTR_ERR	The specified trusted pointer was not valid.

---

**MQSeries API Reason Codes (Continued)**

---

Reason Code Number	Reason Code Name	Description
324	AMRC_USE_CORREL_ID_ERR	The specified use correlation id value was not valid.
325	AMRC_USE_CORREL_ID_PTR_ERR	The specified use correlation id pointer was not valid.
326	AMRC_WAIT_WHOLE_GROUP_ERR	The specified wait for whole group value was not valid.
327	AMRC_WAIT_WHOLE_GROUP_PTR_ERR	The specified wait for whole group pointer was not valid.
328	AMRC_CON_INT_PROP_ID_ERR	The specified connection integer property identifier was not valid.
329	AMRC_CON_STR_PROP_ID_ERR	The specified connection string property identifier was not valid.
330	AMRC_MSG_INT_PROP_ID_ERR	The specified message integer property identifier was not valid.
331	AMRC_MSG_STR_PROP_ID_ERR	The specified message string property identifier was not valid.
332	AMRC_POLICY_INT_PROP_ID_ERR	The specified policy integer property identifier was not valid.
333	AMRC_POLICY_STR_PROP_ID_ERR	The specified policy string property identifier was not valid.
334	AMRC_SRV_INT_PROP_ID_ERR	The specified service integer property identifier was not valid.

---

**MQSeries API Reason Codes (Continued)**

---

Reason Code Number	Reason Code Name	Description
335	AMRC_SRV_STR_PROP_ID_ERR	The specified service string property identifier was not valid.
336	AMRC_INVALID_IF_CON_OPEN	The requested operation could not be performed because the specified connection was open.
337	AMRC_CON_HANDLE_ERR	The specified connection handle was not valid.
400	AMRC_INVALID_TRACE_LEVEL	A specified trace level was not valid.
401	AMRC_CONN_NAME_NOT_FOUND	The connection name obtained from the repository was not found in the local host file.
402	AMRC_HOST_FILE_NOT_FOUND	The local host file with the specified name was not found.
403	AMRC_HOST_FILENAME_ERR	The local host file name was not valid. The value of the appropriate environment variable should be corrected.
404	AMRC_HOST_FILE_ERR	The contents of the local host file are not valid.
405	AMRC_POLICY_NOT_IN_REPOS	The definition name that was specified when creating a policy was not found in the repository. The policy was created using default values.

---

**MQSeries API Reason Codes (Continued)**

Reason Code Number	Reason Code Name	Description
406	AMRC_SENDER_NOT_IN_REPOS	The definition name that was specified when creating a sender was not found in the repository. The sender was created using default values.
407	AMRC_RECEIVER_NOT_IN_REPOS	The definition name that was specified when creating a receiver was not found in the repository. The receiver was created using default values.
408	AMRC_DIST_LIST_NOT_IN_REPOS	The definition name specified for creating a distribution list was not found in the repository. The object was not created.
409	AMRC_PUBLISHER_NOT_IN_REPOS	The definition name that was specified when creating a publisher was not found in the specified repository. The publisher was created using default values.
410	AMRC_SUBSCRIBER_NOT_IN_REPOS	The definition name that was specified when creating a subscriber was not found in the repository. The subscriber was created using default values.
411	AMRC_RESERVED_NAME_IN_REPOS	The name specified for creating an object was not found in the repository and is a reserved name that is not valid in a repository. The specified object was not created.

**MQSeries API Reason Codes (Continued)**

---

Reason Code Number	Reason Code Name	Description
414	AMRC_REPOS_FILENAME_ERR	The repository file name was not valid. The value of the appropriate environment variable should be corrected.
415	AMRC_REPOS_WARNING	A warning associated with the underlying repository data was reported.
416	AMRC_REPOS_ERR	An error was returned when initializing or accessing the repository. This can occur for any of the following reasons:  <ul style="list-style-type: none"> <li>■ The repository XML file (for instance, amt.xml) contains data that is not valid.</li> <li>■ The DTD file (amt.dtd) was not found or contains data that is not valid.</li> <li>■ The files needed to initialize the repository (located in directories intlFiles and locales) could not be located.</li> </ul> Check that the DTD and XML files are valid and correctly located, and that the path settings for the local host and repository files are correct.
418	AMRC_REPOS_NOT_FOUND	The repository file was not found. The value of the appropriate environment variable should be corrected.
419	AMRC_TRANSPORT_LIBRARY_ERR	An error occurred loading the transport library.

---

#### MQSeries API Reason Codes (Continued)

---

Reason Code Number	Reason Code Name	Description
420	AMRC_HOST_CACHE_ERR	A module was loaded for use as a repository file cache, but the module does not appear to be a valid repository cache.
421	AMRC_REPOS_CACHE_ERR	A module was loaded for use as a host file cache, but the module does not appear to be a valid host cache.
422	AMRC_PRIMARY_HANDLE_ERR	The primary handle (that is, the first parameter) passed on the API call was not valid. The most probable reason for failure is that the handle passed is a synonym handle, which is not valid as the <i>primary</i> handle on any call to the AMI.
423	AMRC_SESSION_EXPIRED	Under the IMS environment, the current session has been marked as expired. Delete the current session and create a new one for the duration of this transaction.
424	AMRC_DTD_NOT_FOUND	An AMI dtd file (amt.dtd) was not found with the xml repository file in the same directory.

---

**MQSeries API Reason Codes (Continued)**

---

Reason Code Number	Reason Code Name	Description
425	AMRC_LDAP_ERR	An error was encountered accessing the AMI repository information in the LDAP directory, or communicating with the LDAP server. The LDAP error code can be obtained from the secondary reason code value that is returned from a GetLastError request for the AMI object concerned.
500	AMRC_JAVA_FIELD_ERR	A field referenced in AMI Java code cannot be found in the AMI Java native library. This is probably due to an incompatibility between the AMI class files and the AMI Java library. (Not applicable to the C and C++ programming languages).
501	AMRC_JAVA_METHOD_ERR	A method referenced in AMI Java code cannot be found in the AMI Java native library. This is probably due to an incompatibility between the AMI class files and the AMI Java library. (Not applicable to the C and C++ programming languages).
502	AMRC_JAVA_CLASS_ERR	A class referenced in AMI Java code cannot be found in the AMI Java native library. This is probably due to an incompatibility between the AMI class files and the AMI Java library. (Not applicable to the C and C++ programming languages).

---

**MQSeries API Reason Codes (Continued)**

---

Reason Code Number	Reason Code Name	Description
503	AMRC_JAVA_JNI_ERR	An unexpected error occurred when calling the AMI Java native library. This is probably due to an incompatibility between the AMI class files and the AMI Java library. (Not applicable to the C and C++ programming languages).
504	AMRC_JAVA_CREATE_ERR	An unexpected error occurred when creating an AMI Java object. This is probably due to an incompatibility between the AMI class files and the AMI Java library. (Not applicable to the C and C++ programming languages).
505	AMRC_JAVA_NULL_PARM_ERR	The AMI Java code detected a null parameter that is not valid. (Not applicable to the C and C++ programming languages).

---

**MQSeries API Reason Codes (Continued)**

# Appendix B: The U2XMAP file

B

This Appendix describes the DTD for the U2XMAP file.

```
<!-- U2 XMAP version 1.0 -->
<!ELEMENT U2XMAP (OPTIONS?, DATASOURCE*,
                   TABLECLASSMAP+,
                   RelatedTable*) >
<!ATTLIST U2XMAP Version CDATA #FIXED "1.0">

<!ELEMENT OPTIONS (EmptyString?, DateFormat?, Cascade?,
                   ExistRecord?,
                   (IgnoreNameSpace | NameSpace*))>

<!ELEMENT EmptyString EMPTY>
<!ATTLIST EmptyString IsNULL (ON|OFF) #REQUIRED>

<!ELEMENT DateFormat EMPTY>
<!ATTLIST DateFormat Format CDATA #REQUIRED>

<!ELEMENT Cascade EMPTY>

<!ELEMENT ExistRecord EMPTY>
<!ATTLIST ExistRecord Action (Ignore | Replace | Append)
#REQUIRED>

<!ELEMENT IgnoreNameSpace EMPTY>

<!ELEMENT NameSpace EMPTY>
<!ATTLIST NameSpace
          Prefix NMOKEN #IMPLIED
          URI CDATA #REQUIRED>

<!ELEMENT DATASOURCE EMPTY>
<!ATTLIST DATASOURCE
          Name CDATA #REQUIRED
          ODBCDataSource CDATA #REQUIRED
          Username CDATA #REQUIRED
          Password CDATA #REQUIRED>

<!ELEMENT TABLECLASSMAP (GenerateID?, (ColumnMap | TableMap)*,
                        OrderColumn*)
<!ATTLIST TABLECLASSMAP
          MapName CDATA #REQUIRED
          StartNode CDATA #REQUIRED
          Action (Ignore | Replace | Append)
#IMPLIED
          DataSource CDATA #IMPLIED
```

```

    TableName CDATA #REQUIRED>

<!ELEMENT GenerateID EMPTY>
<!ATTLIST GenerateID
      Column CDATA #REQUIRED
      XFIELD CDATA #IMPLIED
      SUBR CDATA #IMPLIED>

<!ELEMENT ColumnMap EMPTY>
<!ATTLIST ColumnMap
      Node CDATA #REQUIRED
      Column CDATA #REQUIRED
      Action (Ignore | Replace | Append) #IMPLIED>

<!ELEMENT TableMap EMPTY>
<!ATTLIST TableMap
      Node CDATA #REQUIRED
      MapName CDATA #IMPLIED>

<!ELEMENT OrderColumn EMPTY>
<!ATTLIST OrderColumn
      Column CDATA #REQUIRED
      Generate (Yes | No) #REQUIRED>

<!ELEMENT RelatedTable (MapParentKey, ForeignKey)>

<!ELEMENT MapParentKey EMPTY>
<!ATTLIST MapParentKey
      Table CDATA #REQUIRED
      Column CDATA #REQUIRED
      Generate (Yes | No) #REQUIRED>

<!ELEMENT MapChildKey EMPTY>
<!ATTLIST MAPForeignKey
      Table CDATA #REQUIRED
      Column CDATA #REQUIRED>

```

## Mapping root

The U2XMAP element is the root element type of the mapping file document.

```

<!ELEMENT U2XMAP (OPTIONS?, TABLECLASSMAP+)
<!ATTLIST U2XMAP Version CDATA #FIXED "1.0">

```

The element has three child elements:

- OPTIONS
- TABLECLASSMAP
- RelatedTable

## ***Options***

```
<!ELEMENT OPTIONS (EmptyString?, DateFormat?, Cascade?, ExistRecord?,
(IgnoreNameSpace | NameSpace*))
```

Options are a container to hold the various options you can set.

### ***Option EmptyString***

```
<! ELEMENT EmptyString EMPTY>
<!ATTLIST EmptyString isNULL (ON|OFF) #REQUIRED>
```

If the attribute isNull in EmptyString is ON, when the value of an optional XML element is omitted, the corresponding database field value is set to NULL, otherwise it is considered missing. Similarly, when a database field value is NULL and the attribute isNULL in EmptyString is ON, the corresponding value of an optional XML element is omitted, otherwise it is set to an empty string.

### ***Option DateFormat***

```
<! ELEMENT DateFormat EMPTY>
<!ATTLIST DateFormat Format CDATA #REQUIRED>
```

*Format* is a conversion code UniVerse uses to perform the ICONV() / OCONV() function when getting the data from the XML document or creating the XML document. This code can be any conversion code understood by UniVerse. You can also use the value of “XMLFormat” which converts the data into a standard data format for the XML document, or takes the standard XML data format (yyyy-mm-dd).

### ***Option Cascade***

```
<! ELEMENT Cascade EMPTY>
```

The U2XMAP dataset can define whether to use the cascade mode or not.

This option only applies when an XML document is mapped to more than one UniVerse table. By default, the cascade mode is off, which means that the system takes care of the parent-child record relationship according to the RelateTable rules described in the U2XMAP file. Setting the cascade mode ON allows a user to control the parent-child record relationship. When the cascade mode is ON, once the current parent table record is established, all child table records that are either read from or written to the child table are associated with the current parent table record. This affords a user the freedom of associating parent and child table records the way he or she sees fit.

#### *Option ExistRecord*

```
<!ELEMENT ExistReocrd EMPTY>
<!ATTLIST ExistRecord Action (Ignore | Replace | Append) #REQUIRED>
```

Use the ExistRecord option to specify the action when a record ID from the XML document already exists in the UniVerse file. When you specify “Ignore” the XML.TODB command will not change the record with the same record ID. If you specify “Replace” XML.TODB replaces the existing record ID. If you specify “Append” XML.TODB appends the value from the XML document to the multivalued or multi-subvalued field in the UniVerse file if the record already exists in the database.

For example, assume STUDENT (ID=12345) exists in the UniVerse file with COURSES C1, C2, and C3. Further assume that the same STUDENT ID exists in the XML document with COURSES C4 and C5. If ExistRecord is Ignore, the existing student record remains unchanged. In ExistRecord is Replace, XML.TODB replaces the existing record with the record from the XML document, so the COURSES field becomes C4 and C5. If ExistRecord is Append, XML.TODB appends course C4 and C5 to the COURSES field, so the COURSES field contains COURSES C1, C2, C3, C4, and C5.



*Note: Append is only valid with multivalued fields. If specified with a singlevalued field, it is ignored.*

#### *Option Namespace*

```
<!ELEMENT NameSpace EMPTY>
<!ATTLIST NameSpace
    Prefix NMTOKEN #IMPLIED
    URI CDATA #REQUIRED>
<!ELEMENT IgnoreNameSpace EMPTY>
```

<IgnoreNameSpace> means UniVerse ignores all the Namespace information. This option applies when converting the XML document to the UniVerse database.

Namespace elements give URIs and their associated prefixes. These are used as follows:

- In the mapping document, prefixes identify to which namespace an element or attribute belongs. They can be used in the Name attribute of the ElementType and Attribute element types.
- When transferring data from an XML document to the UniVerse database, UniVerse uses namespace URIs to identify elements and attributes in that document. The XML document can use different prefixes than are used in the mapping document.
- When transferring data from the UniVerse database to an XML document, namespace URIs and prefixes are used to prefix element and attribute names in that document.

Namespace elements are not required. If they are used, the same URI or prefix cannot be used more than once. Zero-length prefixes ("") are not currently supported.

If you do not define a Prefix, UniVerse uses the Default Name space, used only for generating the XML document. In this case, UniVerse puts a default NameSpace in the output document.

### ***Table class maps***

```
<!ELEMENT TABLECLASSMAP (GenerateID?, (ColumnMap | TableMap)*,
OrderColumn*)
<!ATTLIST TABLECLASSMAP
  MapName CDATA #REQUIRED
  StartNode CDATA #REQUIRED
  Action (Ignore | Replace | Append) #IMPLIED
  DataSource CDATA #IMPLIED
  TableName CDATA #REQUIRED>
```

<TABLECLASSMAP> elements instruct the data transfer system to map an XML element to a database file. So, if you have more than one such element, the data transfer system puts records into more than one database file. Basically, the properties in this element map to the column in the database file. You cannot pull the properties in other elements to map to this database table.

<MapName> is the given name for this <TABLECLASSMAP>. It can be referenced when you define the relationship of the tables. <StartNode> is the start element in the XML document tree. If this map is under other <TABLECLASSMAP>, this will be a related path. <TableName> is the database file name.

<Action> can overwrite the global action for an existing record. You can specify own action for this table only.

### ***Element <GenerateID>***

```
<!ELEMENT GenerateID EMPTY>
<!ATTLIST GenerateID
    Column CDATA #REQUIRED
    XFIELD CDATA #IMPLIED
    SUBR CDATA #IMPLIED>
```

The <GenerateID> elements instruct the data transfer system to create an ID for the record if a unique ID cannot be found. If you specify XFIELD, UniVerse uses the value from the field in the dictionary for the file as the ID, adds 1, and writes the record back to the dictionary.

If you specify SUBR, UniVerse calls this subroutine to obtain an ID for the current record. The subroutine must be catalogued, and have one parameter defining the return value for the ID.

If you do not specify XFIELD or SUBR, XML.TODB automatically generates a unique ID for the record.

### ***Element <ColumnMap>***

```
<!ELEMENT ColumnMap EMPTY>
<!ATTLIST ColumnMap
    Node CDATA #REQUIRED
    Action (Ignore | Replace | Append) #IMPLIED
    Column CDATA #REQUIRED>
```

<ColumnMap> elements instruct the data transfer system to transfer the data between a certain XML document node and the database file field. The value of <Node> is a U2 xpath (subset of the 'xpath' with some extension). The value of <Column> is the field name in the database file.

<Action> can overwrite the global action for Existed Record". You can specify your own action for this field only.

## ***Element <TableMap>***

```
<! ELEMENT ColumnMap EMPTY>
<! ATTLIST TableMap
      Node CDATA #REQUIRED
      MapName CDATA #IMPLIED>
```

<TableMap> elements instruct the data transfer system where to put this subset of data in the XML document. It also instructs the system that this map is the child map of the current map. The value of <Node> is a U2 xpath (subset of the 'xpath' with some extension). The value of <MapName> is the MAP name in this map file.

## ***Element <OrderColumn>***

```
<! ELEMENT OrderColumn EMPTY>
<! ATTLIST OrderColumn
      Column CDATA #REQUIRED
      Generate (Yes | No) #REQUIRED>
```

UniVerse uses <OrderColumn> elements to store information about the order in which elements and PCDATA occur in their parent element.

Storing order information is optional. If you do not store it, there is no guarantee that the order will be preserved in a round trip from an XML document to the database and back again.

The Generate attribute of the OrderColumn element tells the system whether to generate order information or not. The presence or absence of the OrderColumn element tells the system whether to use order information. If you do not generate order information, you must map another column to the order column.

## ***Related table maps***

```
<! ELEMENT RelatedTable MapParentKey, MapChildKey>
<! ATTLIS RelatedTable
      MapName CDATA #REQUIRED
      Generate (Yes | No) #REQUIRED>

<! ELEMENT MapParentKey EMPTY>
<! ATTLIST Column CDATA #REQUIRED>

<! ELEMENT MapChildKey EMPTY>
<! ATTLIST Column CDATA #REQUIRED>
```

<RelatedTable> elements state the relationship between 2 maps or files. In class terms, think of this as a property added to the class being defined that points to the related class. In XML terms, this means that the element type for the related class is a child of the element type for the class being defined.

This element tells the data transfer system the relationship of these two tables. The keys are used to join these two tables.

# Appendix C: CallMQ API for UniData and UniVerse reason codes

This appendix describes the CallMQI reason codes.

C

## CallMQI reason codes

This table describes the CallMQI reason codes.

Reason Code Number	Reason Code Name	Description
0	MQRC_NONE	No reason was reported.
99	IPHANTOM_LICN_ERROR	No license was available for interactive phantom.
100	U2MQI_ERR_MQUNAVAILABLE	WebSphere MQ MQI libraries not available.
101	U2MQI_ERR_UNKNOWN	An unknown error occurred in CallMQI.
102	U2MQI_ERR_NOBINDIR	UVBIN environment variable not set.
103	U2MQI_ERR_FORK	Error during exec of MQI process.
104	U2MQI_ERR_PIPECREATE	Error creating pipes to MQI process.
105	U2MQI_ERR_PIPEWRITETOMQI	Error writing pipe to MQI process.
106	U2MQI_ERR_PIPEREADFROMMMQI	Error reading pipe from MQI process.
107	U2MQI_ERR_PIPEWRITETOu2	Error writing pipe to U2 process.
108	U2MQI_ERR_PIPEREADFROMU2	Error reading pipe from u2 process.
109	U2MQI_ERR_EXEC	Error during exec of MQI process.
110	U2MQI_ERR_INVALIDFORMAT	Variable does not match required format.

### U2 WebSphere MQ API Reason Codes

---

Reason Code Number	Reason Code Name	Description
111	U2MQI_ERR_NOT_HANDLE	Variable not of type MQShandle.
112	U2MQI_ERR_NULL_HANDLE	Uninitialized handle.
113	U2MQI_ERR_INVALID_HANDLE	The handle has been closed.
114	U2MQI_ERR_UNKNOWN_HANDLE	An unexpected handle value reported.
115	U2MQI_ERR_SESSION_IN_USE	An active CallMQI session already exists.
116	U2MQI_ERR_CREATE_HANDLE	An error occurred creating session handle.
117	U2MQI_ERR_DL_OPEN	An error occurred opening WebSphere MQ MQI library.
118	U2MQI_ERR_DL_FUNC	An error occurred calling function in WebSphere MQ MQI library.
119	U2MQI_ERR_RCVMSGOPTS	Invalid receive-message option.
2001	MQRC_ALIAS_BASE_Q_TYPE_ERROR	The alias base queue did not resolve to a valid queue type.
2002	MQRC_ALREADY_CONNECTED	Application already connected to the queue manager.
2003	MQRC_BACKED_OUT	The unit of work was backed out.
2004	MQRC_BUFFER_ERROR	Invalid buffer parameter.
2005	MQRC_BUFFER_LENGTH_ERROR	Invalid buffer length parameter.

---

**U2 WebSphere MQ API Reason Codes (Continued)**

Reason Code Number	Reason Code Name	Description
2006	MQRC_CHAR_ATTR_LENGTH_ERROR	The length of the character attributes are invalid.
2007	MQRC_CHAR_ATTRS_ERROR	The character attributes parameter is invalid.
2008	MQRC_CHAR_ATTRS_TOO_SHORT	Not all character attributes will fit in the space provided.
2009	MQRC_CONNECTION_BROKEN	The queue manager connection is no longer active.
2010	MQRC_DATA_LENGTH_ERROR	The data length parameter is invalid.
2011	MQRC_DYNAMIC_Q_NAME_ERROR	The dynamic queue name is invalid.
2012	MQRC_ENVIRONMENT_ERROR	Call invalid in current environment.
2013	MQRC_EXPIRY_ERROR	The specified expiry value is invalid.
2014	MQRC_FEEDBACK_ERROR	The specified feedback value is invalid.
2016	MQRC_GET_INHIBITED	MQGET calls are inhibited on this queue.
2017	MQRC_HANDLE_NOT_AVAILABLE	This request exceeded the maximum number of handles available.
2018	MQRC_HCONN_ERROR	The connection handle to the queue manager is invalid.
2019	MQRC_HOBJ_ERROR	The handle to the WebSphere MQ object is invalid.

---

**U2 WebSphere MQ API Reason Codes (Continued)**

---

Reason Code Number	Reason Code Name	Description
2021	MQRC_INT_ATTR_COUNT_ERROR	The integer attribute count is invalid.
2022	MQRC_INT_ATTR_COUNT_TOO_SMALL	Not all integer attributes will fit in the space provided.
2023	MQRC_INT_ATTRS_ARRAY_ERROR	The integer attribute parameter is invalid.
2024	MQRC_SYNCPOINT_LIMIT_REACHED	The current unit of work cannot handle any more messages.
2025	MQRC_MAX_CONNS_LIMIT_REACHED	The maximum number of connections has been reached.
2026	MQRC_MD_ERROR	The MQMD structure is not valid.
2027	MQRC_MISSING_REPLY_TO_Q	No reply to queue value is specified.
2029	MQRC_MSG_TYPE_ERROR	The specified message type is invalid.
2030	MQRC_MSG_TOO_BIG_FOR_Q	The message exceeded the maximum length allowed by the queue.
2031	MQRC_MSG_TOO_BIG_FOR_Q_MGR	The message exceeded the maximum length allowed by the queue manager.
2033	MQRC_NO_MSG_AVAILABLE	No message available on the queue.
2034	MQRC_NO_MSG_UNDER_CURSOR	The browse cursor is not positioned over the message.

---

**U2 WebSphere MQ API Reason Codes (Continued)**

Reason Code Number	Reason Code Name	Description
2035	MQRC_NOT_AUTHORIZED	Not authorized to perform this action.
2036	MQRC_NOT_OPEN_FOR_BROWSE	The queue was not opened for browse.
2037	MQRC_NOT_OPEN_FOR_INPUT	The queue was not opened for input.
2038	MQRC_NOT_OPEN_FOR_INQUIRE	The queue is not open for inquire calls.
2039	MQRC_NOT_OPEN_FOR_OUTPUT	The queue was not opened for output.
2041	MQRC_OBJECT_CHANGED	The object definition has changed since opening.
2042	MQRC_OBJECT_IN_USE	The object is already in use.
2043	MQRC_OBJECT_TYPE_ERROR	Invalid object type specified.
2044	MQRC_OD_ERROR	The MQOD object descriptor is invalid.
2045	MQRC_OPTION_NOT_VALID_-FOR_TYPE	The specified option is not valid for object type.
2046	MQRC_OPTIONS_ERROR	The specified options are not valid.
2047	MQRC_PERSISTENCE_ERROR	The specified persistence value is invalid.
2048	MQRC_PERSISTENT_NOT_ALLOWED	The specified queue does not support persistent messages.
2049	MQRC_PRIORITY_EXCEEDS_MAXIMUM	Message priority exceeds maximum value for queue manager.

---

**U2 WebSphere MQ API Reason Codes (Continued)**

---

---

Reason Code Number	Reason Code Name	Description
2050	MQRC_PRIORITY_ERROR	The specified priority is invalid.
2051	MQRC_PUT_INHIBITED	MQPUT calls are inhibited on this queue.
2052	MQRC_Q_DELETED	The dynamic queue has been deleted.
2053	MQRC_Q_FULL	Maximum number of messages reached for queue.
2055	MQRC_Q_NOT_EMPTY	The queue is not empty or is currently in use.
2056	MQRC_Q_SPACE_NOT_AVAILABLE	No storage available for queue.
2057	MQRC_Q_TYPE_ERROR	The specified queue was invalid.
2058	MQRC_Q_MGR_NAME_ERROR	The specified queue manager name cannot be found.
2059	MQRC_Q_MGR_NOT_AVAILABLE	The specified queue manager is unavailable.
2061	MQRC_REPORT_OPTIONS_ERROR	Report options specified in message descriptor were not recognized by queue manager.
2062	MQRC_SECOND_MARK_NOT_ALLOWED	This unit of work already contains a marked message.
2063	MQRC_SECURITY_ERROR	A security-related error occurred.
2065	MQRC_SELECTOR_COUNT_ERROR	The selector count is invalid.

---

**U2 WebSphere MQ API Reason Codes (Continued)**

Reason Code Number	Reason Code Name	Description
2066	MQRC_SELECTOR_LIMIT_EXCEEDED	The selector count is too big.
2067	MQRC_SELECTOR_ERROR	The selector is invalid.
2068	MQRC_SELECTOR_NOT_FOR_TYPE	The queue type does not support one or more of the selectors.
2069	MQRC_SIGNAL_OUTSTANDING	A signal is already outstanding for this queue handle.
2070	MQRC_SIGNAL_REQUEST_ACCEPTED	The signal request was accepted, though no message was available.
2071	MQRC_STORAGE_NOT_AVAILABLE	Storage resources are not sufficient for this request.
2072	MQRC_SYNCPOINT_NOT_AVAILABLE	Syncpoint is not supported.
2079	MQRC_TRUNCATED_MSG_ACCEPTED	Truncated message returned, and original message removed from the queue.
2080	MQRC_TRUNCATED_MSG FAILED.	Truncated message returned, but original message not removed from the queue
2082	MQRC_UNKNOWN_ALIAS_BASE_Q	Unrecognized base queue name in alias queue definition.
2085	MQRC_UNKNOWN_OBJECT_NAME	The specified object name is invalid.
2086	MQRC_UNKNOWN_OBJECT_Q_MGR.	The specified queue manager name is invalid.

---

**U2 WebSphere MQ API Reason Codes (Continued)**

---

---

Reason Code Number	Reason Code Name	Description
2087	MQRC_UNKNOWN_REMOTE_Q_MGR	The specified remote queue manager is invalid.
2090	MQRC_WAIT_INTERVAL_ERROR	The specified wait interval is invalid.
2091	MQRC_XMIT_Q_TYPE_ERROR	The transmission queue is not a local queue.
2092	MQRC_XMIT_Q_USAGE_ERROR	The transmission queue's usage attribute not set to MQUS_TRANSMISSION.
2093	MQRC_NOT_OPEN_FOR_PASS_ALL	The queue was not opened with the MQOO_- PASS_ALL_CONTEXT option.
2094	MQRC_NOT_OPEN_FOR_PASS_IDENT	The queue was not opened with the MQOO_- PASS_IDENTITY_CONTEXT option.
2095	MQRC_NOT_OPEN_FOR_SET_ALL	The queue was not opened with the MQOO_SET_ALL_- CONTEXT option.
2096	MQRC_NOT_OPEN_FOR_SET_IDENT	The queue was not opened with the MQOO_SET_IDENTITY_- CONTEXT option.
2097	MQRC_CONTEXT_HANDLE_ERROR	The queue handle in the MQPMO Context field was not opened for save-all-context.
2098	MQRC_CONTEXT_NOT_AVAILABLE	The queue handle in the MQPMO Context field does not have an associated context.

---

**U2 WebSphere MQ API Reason Codes (Continued)**

Reason Code Number	Reason Code Name	Description
2099	MQRC_SIGNAL1_ERROR	The signal field is invalid.
2100	MQRC_OBJECT_ALREADY_EXISTS	A queue by that name already exists.
2101	MQRC_OBJECT_DAMAGED	The object is damaged and unusable.
2102	MQRC_RESOURCE_PROBLEM	System resources are not sufficient for this request.
2103	MQRC_ANOTHER_Q_MGR_CONNECTED	Already connected to a different queue manager.
2104	MQRC_UNKNOWN_REPORT_OPTION	Report option in message not recognized by queue manager.
2105	MQRC_STORAGE_CLASS_ERROR	Storage class object for queue does not exist.
2106	MQRC_COD_NOT_VALID_FOR_XCF_Q	COD options are not accepted by XCF queues.
2109	MQRC_SUPPRESSED_BY_EXIT	The exit program suppressed the call.
2110	MQRC_FORMAT_ERROR	The message format is incorrect.
2111	MQRC_SOURCE_CCSID_ERROR	The message's coded character set identifier is invalid.
2112	MQRC_SOURCE_INTEGER_ENC_ERROR	The message's encoding specified an unrecognized source integer encoding.
2113	MQRC_SOURCE_DECIMAL_ENC_ERROR	The message's encoding specified an unrecognized decimal encoding.

---

**U2 WebSphere MQ API Reason Codes (Continued)**

---

---

Reason Code Number	Reason Code Name	Description
2114	MQRC_SOURCE_FLOAT_ENC_ERROR	The message's encoding specified an unrecognized floating-point encoding.
2115	MQRC_TARGET_CCSID_ERROR	The coded character set identifier in the message descriptor parameter is invalid.
2116	MQRC_TARGET_INTEGER_ENC_ERROR	The encoding in the message descriptor specified an unrecognized target integer encoding.
2117	MQRC_TARGET_DECIMAL_ENC_ERROR	The encoding in the message descriptor specified an unrecognized decimal encoding.
2118	MQRC_TARGET_FLOAT_ENC_ERROR	The encoding in the message descriptor specified an unrecognized floating-point encoding.
2119	MQRC_NOT_CONVERTED	The message data failed to convert.
2120	MQRC_CONVERTED_MSG_TOO_BIG	The converted message exceeded the size of the buffer.
2123	MQRC_OUTCOME_MIXED	A commit or roll back of a unit of work yielded different results across resource managers.
2124	MQRC_OUTCOME_PENDING	Outcome of the commit action is pending.
2127	MQRC_ADAPTER_STORAGE_SHORTAGE	Adapter unable to acquire sufficient storage.

---

**U2 WebSphere MQ API Reason Codes (Continued)**

Reason Code Number	Reason Code Name	Description
2129	MQRC_ADAPTER_CONN_- LOAD_ERROR	The adapter connection module failed to load.
2130	MQRC_ADAPTER_SERV_LOAD_ERROR	Adapter not able to load the CSQBSRV service module.
2131	MQRC_ADAPTER_DEFS_ERROR	Required identifier missing from adapter subsystem definition module.
2132	MQRC_ADAPTER_DEFS_LOAD_ERROR	The adapter subsystem definition module failed to load.
2133	MQRC_ADAPTER_CONV_- LOAD_ERROR	Adapter not able to load the conversion service modules.
2135	MQRC_DH_ERROR	The MQDH structure is invalid.
2136	MQRC_MULTIPLE_REASONS	The call resulted in a return of multiple reason codes.
2137	MQRC_OPEN_FAILED	Object failed to open.
2138	MQRC_ADAPTER_DISC_LOAD_ERROR	The disconnection module failed to load.
2140	MQRC_CICS_WAIT_FAILEDC	ICS rejected the Wait request.
2141	MQRC_DLH_ERROR	The MQDLH structure is invalid.
2142	MQRC_HEADER_ERROR	The WebSphere MQ header structure is invalid.
2143	MQRC_SOURCE_LENGTH_ERROR	The source length parameter is invalid.

**U2 WebSphere MQ API Reason Codes (Continued)**

---

Reason Code Number	Reason Code Name	Description
2145	MQRC_SOURCE_BUFFER_ERROR	Invalid source buffer parameter.
2146	MQRC_TARGET_BUFFER_ERROR	The target buffer parameter is invalid.
2148	MQRC_IIH_ERROR	Invalid MQIIH structure in message data.
2149	MQRC_PCF_ERROR	PCF structure lengths in message data not equal to message length.
2150	MQRC_DBCS_ERROR	The double-byte character set (DBCS) conversion failed.
2152	MQRC_OBJECT_NAME_ERROR	The object name is invalid.
2153	MQRC_OBJECT_Q_MGR_NAME_ERROR	The queue manager name in the MQOD structure is invalid.
2154	MQRC_RECS_PRESENT_ERROR	Invalid value specified for number of records.
2155	MQRC_OBJECT_RECORDS_ERROR	The MQOR object records are invalid.
2156	MQRC_RESPONSE_RECORDS_ERROR	Invalid response records.
2157	MQRC_ASID_MISMATCH	The primary and home ASIDs do not match.
2158	MQRC_PMO_RECORD_FLAGS_ERROR	Invalid put message record flags specified in MQPMO structure.

---

**U2 WebSphere MQ API Reason Codes (Continued)**

Reason Code Number	Reason Code Name	Description
2159	MQRC_PUT_MSG_RECORDS_ERROR	Invalid put message records.
2160	MQRC_CONN_ID_IN_USE	Connection identifier in use by another system.
2161	MQRC_Q_MGR QUIESCING	The specified queue manager is quiescing.
2162	MQRC_Q_MGR_STOPPING	The queue manager is stopping.
2163	MQRC_DUPLICATE_RECov_COORD	Recovery coordinator already in use for this connection.
2173	MQRC_PMO_ERROR	The structure of the MQPMO parameter is not valid.
2183	MQRC_API_EXIT_LOAD_ERROR	The API exit failed to load.
2184	MQRC_REMOTE_Q_NAME_ERROR	The name of the remote queue is invalid.
2185	MQRC_INCONSISTENT_PERSISTENCE	The persistence specifications are inconsistent.
2186	MQRC_GMO_ERROR	The structure of the MQGMO parameter is not valid.
2188	MQRC_STOPPED_BY_CLUSTER_EXIT	The cluster workload exit refused the call.
2189	MQRC_CLUSTER_RESOLUTION_ERROR	The cluster failed to resolve because the repository manager did not respond.

**U2 WebSphere MQ API Reason Codes (Continued)**

---

Reason Code Number	Reason Code Name	Description
2190	MQRC_CONVERTED_STRING_TOO_BIG	The converted string exceeded the size of the field.
2191	MQRC_TMC_ERROR	Invalid MQTMC2 structure in message data.
2192	MQRC_STORAGE_MEDIUM_FULL	External storage resources are full.
2192	MQRC_PAGESET_FULL	The page set is full.
2193	MQRC_PAGESET_ERROR	Cannot access the page-set data.
2194	MQRC_NAME_NOT_VALID_FOR_TYPE	The specified name is not valid for the object type.
2195	MQRC_UNEXPECTED_ERROR	An unexpected error occurred.
2196	MQRC_UNKNOWN_XMIT_Q	The specified transmission queue is invalid.
2197	MQRC_UNKNOWN_DEF_XMIT_Q	The default transmission queue is not a locally-defined queue.
2198	MQRC_DEF_XMIT_Q_TYPE_ERROR	The default transmission queue is not a local queue.
2199	MQRC_DEF_XMIT_Q_USAGE_ERROR	The default transmission queue's usage attribute not set to MQUS_TRANSMISSION.
2201	MQRC_NAME_IN_USE	The specified name is already in use.

---

**U2 WebSphere MQ API Reason Codes (Continued)**

Reason Code Number	Reason Code Name	Description
2202	MQRC_CONNECTION QUIESCING	Connection in a quiescing state.
2203	MQRC_CONNECTION_STOPPING	The connection to the queue manager is stopping.
2204	MQRC_ADAPTER_NOT_AVAILABLE	The CICS adapter is unavailable.
2206	MQRC_MSG_ID_ERROR	Message identifiers are not supported on this queue.
2207	MQRC_CORREL_ID_ERROR	This queue does not support correlation identifiers.
2209	MQRC_NO_MSG_LOCKED	No messages are locked.
2217	MQRC_CONNECTION_NOT_AUTHORIZED	Not authorized to connect with queue manager.
2219	MQRC_CALL_IN_PROGRESS	MQI call issued before previous call processed.
2220	MQRC_RMH_ERROR	Invalid MQRMH structure in message data.
2235	MQRC_CFH_ERROR	Invalid MQCFH structure in message data.
2236	MQRC_CFIL_ERROR	Invalid MQCFIL or MQRCFIL64 structure in message data.
2237	MQRC_CFIN_ERROR	Invalid MQCFIN or MQCFIN64 structure in message data.
2238	MQRC_CFSL_ERROR	Invalid MQCFSL structure in message data.

**U2 WebSphere MQ API Reason Codes (Continued)**

---

Reason Code Number	Reason Code Name	Description
2239	MQRC_CFST_ERROR	Invalid MQCFST structure in message data.
2241	MQRC_INCOMPLETE_GROUP	A message group on the queue handle was not complete.
2242	MQRC_INCOMPLETE_MSG	A logical message on the queue handle was not complete.
2243	MQRC_INCONSISTENT_CCSIDS	The message segments have inconsistent coded character set identifiers.
2244	MQRC_INCONSISTENT_ENCODINGS	The message segments have inconsistent encoding values.
2245	MQRC_INCONSISTENT_UOW	The unit of work specifications are inconsistent.
2246	MQRC_INVALID_MS-G_UNDER_CURSOR	The message under the cursor is invalid.
2247	MQRC_MATCH_OPTIONS_ERROR	The match options are invalid.
2248	MQRC_MDE_ERROR	The MQMDE structure is invalid.
2249	MQRC_MSG_FLAGS_ERROR	A message flag is not recognized by the queue manager.
2250	MQRC_MSG_SEQ_NUMBER_ERROR	The message sequence number is out of range.
2251	MQRC_OFFSET_ERROR	The message segment offset value is invalid.

---

**U2 WebSphere MQ API Reason Codes (Continued)**

Reason Code Number	Reason Code Name	Description
2252	MQRC_ORIGINAL_LENGTH_ERROR	The original length value is invalid.
2253	MQRC_SEGMENT_LENGTH_ZERO	Message segment data length was zero.
2255	MQRC_UOW_NOT_AVAILABLE	The queue manager cannot create temporary unit of work.
2256	MQRC_WRONG_GMO_VERSION	The supplied version of MQGMO is incorrect.
2257	MQRC_WRONG_MD_VERSION	The supplied version of MQMD is incorrect.
2258	MQRC_GROUP_ID_ERROR	The specified group identifier is invalid.
2259	MQRC_INCONSISTENT_BROWSE	The browse specifications are inconsistent across calls.
2260	MQRC_XQH_ERROR	Invalid MQXQH structure in message data.
2265	MQRC_TM_ERROR	Invalid MQTM structure in message data.
2266	MQRC_CLUSTER_EXIT_ERROR	The cluster workload exit failed.
2267	MQRC_CLUSTER_EXIT_LOAD_ERROR	The cluster workload exit failed to load.
2268	MQRC_CLUSTER_PUT_INHIBITED	All queues in the cluster are put-inhibited.
2269	MQRC_CLUSTER_RESOURCE_ERROR	A cluster resource error occurred.

**U2 WebSphere MQ API Reason Codes (Continued)**

---

Reason Code Number	Reason Code Name	Description
2270	MQRC_NO_DESTINATIONS_AVAILABLE	There were no destination queues available.
2272	MQRC_PARTIALLY_CONVERTED	The message data was only partially converted.
2273	MQRC_CONNECTION_ERROR.	The connection failed.
2331	MQRC_MSG_TOKEN_ERROR	The match message token option is not valid in this context.
2332	MQRC_MISSING_WIH	The MQWIH structure is missing from the message data.
2333	MQRC_WIH_ERROR	Invalid MQWIH structure in message data.
2334	MQRC_RFH_ERROR	Invalid MQRFH or MQRFH2 structure in message data.
2342	MQRC_DB2_NOT_AVAILABLE	The queue manager cannot access DB2.
2343	MQRC_OBJECT_NOT_UNIQUE	The queue name is not unique.
2344	MQRC_CONN_TAG_NOT_RELEASED	Connection tag currently unavailable for reuse.
2345	MQRC_CF_NOT_AVAILABLE	The coupling-facility is unavailable.
2346	MQRC_CF_STRUC_IN_USE	The coupling-facility for shared queue already in use.

---

**U2 WebSphere MQ API Reason Codes (Continued)**

Reason Code Number	Reason Code Name	Description
2347	MQRC_CF_STRUC_LIST_HDR_IN_USE	The coupling-facility list-header for shared queue already in use.
2348	MQRC_CF_STRUC_AUTH_FAILED	Access to the coupling-facility is not authorized.
2349	MQRC_CF_STRUC_ERROR	The coupling-facility is not defined.
2351	MQRC_GLOBAL_UOW_CONFLICT	There is a conflict between two global units of work.
2352	MQRC_LOCAL_UOW_CONFLICT	There is a conflict between the global unit of work and the local unit of work.
2353	MQRC_HANDLE_IN_USE_FOR_UOW	The connection handle is already being used in a global unit of work.
2354	MQRC_UOW_ENLISTMENT_ERROR	Failed to enlist in global unit of work.
2355	MQRC_UOW_MIX_NOT_SUPPORTED	Mixed unit of work calls are not supported.
2360	MQRC_OBJECT_LEVEL_INCOMPATIBLE	The object definition is not compatible with the queue manager.
2365	MQRC_SEGMENTS_NOT_SUPPORTED	Message segments not supported for specified queue.
2366	MQRC_WRONG_CF_LEVEL	Incompatible coupling-facility structure.
2373	MQRC_CF_STRUC_FAILED	The coupling-facility for shared queue failed.

**U2 WebSphere MQ API Reason Codes (Continued)**

---

Reason Code Number	Reason Code Name	Description
2374	MQRC_API_EXIT_ERROR	An API exit did not complete correctly.
2375	MQRC_API_EXIT_INIT_ERROR	The API exit failed to initialize.
2376	MQRC_API_EXIT_TERM_ERROR	The API exit failed to terminate.
2381	MQRC_KEY_REPOSITORY_ERROR	Location of key repository is incorrect.
2382	MQRC_CRYPTO_HARDWARE_ERROR	Invalid cryptographic hardware configuration.
2391	MQRC_SSL_ALREADY_INITIALIZED	SSL environment already initialized.
2393	MQRC_SSL_INITIALIZATION_ERROR	SSL environment failed to initialize.
2394	MQRC_Q_INDEX_TYPE_ERROR	Queue not indexed by a group identifier.
2414	MQRC_CFIF_ERROR	Invalid MQCFIF structure in message data.
2415	MQRC_CFSF_ERROR	Invalid MQCFSF structure in message data.
2416	MQRC_CFGR_ERROR	Invalid MQCFG structure in message data.
2420	MQRC_EPH_ERROR	The MQEPH structure is invalid.

---

**U2 WebSphere MQ API Reason Codes (Continued)**

# Appendix D: WebSphere MQ API for UniData and UniVerse error codes

D

This appendix describes the error codes that come specifically from UniVerse to the MQI interface layer. These are above and beyond the errors that WebSphere MQ itself can report. These errors appear in the ReasonCode value of the dynamic array that is returned by the CallMQI functions. As an example,

```
RES = MQCONN("saturn.queue.manager", HCONN)
IF RES<COMPCODE> # MQCC_OK THEN
    * Call failed, or had a warning
    * Check the ReasonCode
    IF RES<REASON> = U2MQI_ERR_MQUNAVAILABLE THEN
        PRINT "WebSphere MQ libraries not found"
    END
END
```

This table describes the CallMQI error codes.

Identifier	Description
100	U2MQI_ERR_MQUNAVAILABLE
101	U2MQI_ERR_UNKNOWN
102	U2MQI_ERR_NOBINDIR
103	U2MQI_ERR_FORK
104	U2MQI_ERR_PIPECREATE
105	U2MQI_ERR_PIPEWRITETOMQI
106	U2MQI_ERR_PIPEREADFROMMQI
107	U2MQI_ERR_PIPEWRITETO2
108	U2MQI_ERR_PIPEREADFROMU2
109	U2MQI_ERR_EXEC
110	U2MQI_ERR_INVALIDFORMAT
111	U2MQI_ERR_NOT_HANDLE
112	U2MQI_ERR_NULL_HANDLE
113	U2MQI_ERR_INVALID_HANDLE
114	U2MQI_ERR_UN- KNOWN_HANDLE
115	U2MQI_ERR_SESSION_IN_USE
116	U2MQI_ERR_CREATE_HANDLE
117	U2MQI_ERR_DL_OPEN
	U2MQI_ERR_DL_FUNC

CallMQI Error Codes