



UniVerse

UniVerse 11.1 New Features

Notices

Edition

Publication date: October 2010

Book number: UNV-111-NEWF-1

Product version: UniVerse 11.1

Copyright

© Rocket Software, Inc. 1985-2010. All Rights Reserved.

Trademarks

The following trademarks appear in this publication:

Trademark	Trademark Owner
Rocket Software™	Rocket Software, Inc.
Dynamic Connect®	Rocket Software, Inc.
RedBack®	Rocket Software, Inc.
SystemBuilder™	Rocket Software, Inc.
UniData®	Rocket Software, Inc.
UniVerse™	Rocket Software, Inc.
U2™	Rocket Software, Inc.
U2.NET™	Rocket Software, Inc.
U2 Web Development Environment™	Rocket Software, Inc.
wIntegrate®	Rocket Software, Inc.
Microsoft® .NET	Microsoft Corporation
Microsoft® Office Excel®, Outlook®, Word	Microsoft Corporation
Windows®	Microsoft Corporation
Windows® 7	Microsoft Corporation
Windows Vista®	Microsoft Corporation
Java™ and all Java-based trademarks and logos	Sun Microsystems, Inc.
UNIX®	X/Open Company Limited

The above trademarks are property of the specified companies in the United States, other countries, or both. All other products or services mentioned in this document may be covered by the trademarks, service marks, or product names as designated by the companies who own or market them.

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 and with the inclusion of the copyright notice. This software and any copies thereof may not be provided or otherwise made available to any other person. No title to or ownership of the software and associated documentation is hereby transferred. Any unauthorized use or reproduction of this software or documentation may be subject to civil or criminal liability. The information in the software and documentation is subject to change and should not be construed as a commitment by Rocket Software, Inc.

Restricted rights notice for license to the U.S. Government: Use, reproduction, or disclosure is subject to restrictions as stated in the "Rights in Technical Data-General" clause (alternate III), in FAR section 52.222-14. All title and ownership in this computer software remain with Rocket Software, Inc.

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.

Please be aware: Any images or indications reflecting ownership or branding of the product(s) documented herein may or may not reflect the current legal ownership of the intellectual property rights associated with such product(s). All right and title to the product(s) documented herein belong solely to Rocket Software, Inc. and its subsidiaries, notwithstanding any notices (including screen captures) or any other indications to the contrary.

Contact information

Rocket Software
275 Grove Street Suite 3-410
Newton, MA 02466-2272
USA
Tel: (617) 614-4321 Fax: (617) 630-7100
Web Site: www.rocketsoftware.com

Table of Contents

Chapter 1	Automatic Data Encryption Enhancements	
	Encrypting the @ID Field	1-3
	Encrypting Index Files	1-8
	@ID and Index Encryption Interaction	1-13
	PHANTOM Inheritance of Activated Fields and Disabled Fields	1-14
	Password Policies	1-17
	Protecting Master Keys with Multiple, Changeable Passwords	1-22
	One-pass Reencryption	1-25
	Password for TCL Command Not Echoed	1-26
Chapter 2	U2 Data Replication for UniVerse	
	U2 Data Replication for UniVerse	2-2
	Processes Added at UniVerse 11.1	2-3
	UniVerse Commands Supported in Replication	2-11
	Installing U2 Data Replication for UniVerse	2-15
	Administering U2 Data Replication	2-19
Chapter 3	External Database Access (EDA)	
	External Database Access	3-2
	External Database Access Drivers Provided with EDA	3-3
	EDA DB2 Driver	3-5
	EDA SQL Server Driver	3-7
	External Database Access Driver API	3-9
Chapter 4	U2 Websphere MQ API	
	In This Chapter	4-3
	Preface	4-4
	Overview of Messaging	4-5
	Overview of IBM WebSphere MQ	4-6
	U2 WebSphere MQ API	4-7
	Setup and Configuration for the WebSphere MQ API	4-8

Programming with the U2 WebSphere MQ API	4-11
U2MQI.H INCLUDE File	4-12
Error Handling in CallMQI	4-12
Encoding Message IDs and Correlation IDs	4-13
U2 WebSphere MQ API Programmatic Interfaces	4-14
2 WebSphere MQ API Programmatic Interfaces	4-15
Closing a Queue	4-17
Connecting to a Message	4-19
Disconnecting from a Queue Manager.	4-21
Retrieving a Message from a Queue	4-23
Returning an Error Message	4-25
Querying the Attributes	4-26
Opening a Queue	4-28
Putting a Message on a Queue	4-30
Opening a Queue	4-32
Programming Examples	4-35
CallMQI Dynamic Array Structures	4-40
MQOD Dynamic Array Structure	4-40
MQMD Dynamic Array Structure	4-42
MQGMO Dynamic Array Structure	4-46
MQPMO Dynamic Array Structure	4-48
CallMQI Error Codes	4-52
CallMQI Reason Codes	4-54
Additional Reading.	4-73

Chapter 5 Non-Root uvadm

Extended uvadm	5-3
Overview Of UniVerse Installation on UNIX	5-4
Login ID	5-4
The <i>uv.load</i> Script	5-4
The <i>uv_upgrade</i> Command	5-8
Initial Installation of UniVerse	5-9
Upgrading an Existing UniVerse System as uvadm.	5-15
Examining the Load and Installation Scripts	5-23
<i>uv.load</i> Options	5-23
System Administration Menus	5-24

Chapter 6 JPA

Installation Files	6-4
Prerequisites	6-4
Installation.	6-4

Introduction	6-5
Features	6-6
U2JPA and Object-Relational Mapping	6-7
Virtual Fields in U2JPA	6-9
Fetch Types	6-9
Restrictions	6-10
Relationship Mapping	6-10
Using U2JPA	6-15
Entity Managers	6-15
Persistence Units	6-15
Transaction Support	6-17
Connection Pooling	6-18
Connection Pool Size	6-18
Activating Connection Pooling	6-18
SSL Connection and NLS Support	6-19
Managing Entities	6-20
U2JPA Methods	6-22
U2JPA Query Support	6-27
Query Parameters	6-28
Query Result Paging	6-28
Subroutine Calls	6-31
Named Query	6-32
U2JPA Eclipse Plug-in	6-34
The U2JPA Wizard.	6-34
The Persistence.xml Configuration File	6-35
Step-By-Step Guide to Using U2JPA.	6-36
Create a New U2JPA Project in Eclipse	6-36
Create a New User Library	6-40
Use the U2 Resource View in Eclipse	6-42
Create a U2JPA package	6-44
Run the Program	6-50

Chapter 7 **Miscellaneous New Features**

Database Auditing Using Index-Based Subroutines	7-3
Programming Example	7-5
BUILD.INDEX CONCURRENT.	7-6
Syntax.	7-6
ODBC Enhancements	7-7
Scalability	7-7
ODBC 3.0 Supported Functions	7-7

API Conformance Levels	7-8
UniVerse ODBC-Supported API Functions	7-10

Automatic Data Encryption Enhancements

Encrypting the @ID Field	1-3
Encrypting Index Files	1-8
@ID and Index Encryption Interaction	1-13
PHANTOM Inheritance of Activated Fields and Disabled Fields.	1-14
Password Policies	1-17
Protecting Master Keys with Multiple, Changeable Passwords	1-22
One-pass Reencryption	1-25
Password for TCL Command Not Echoed	1-26

This chapter describes enhancements to Automatic Data Encryption. These enhancements include:

- Encryption of the @ID field
- Encryption of indexed fields
- Inheritance of activated keys and disabled fields by phantom processes
- User-configurable password policies for encryption keys and encryption wallets
- Encryption key and encryption wallet with changeable password
- No echo of input passwords for all command line encryption commands
- Master key protected by multiple passwords that you can change
- One pass data reencryption

Encrypting the @ID Field

When you encrypt the @ID field of a record, you must activate the encryption key in order to decrypt the @ID. If the encryption key is not active, the LIST, RESIZE, ENCRYPT.FILE and DECRYPT.FILE commands will fail with an error.

The maximum length for the @ID field is 256 characters. Note that encrypting the @ID increases the length by 33%. Make sure that the maximum length is not violated after encryption, or encryption will fail.

When you encrypt the @ID field, the default hash type of the file automatically changes to 18, unless the file is a type 30 (dynamic) file, which remains unchanged. This hash type is best for randomly distributed @IDs, which is true for cipher text @IDs.

When you encrypt the @ID, UniVerse disables all index files for the file and sets the status to “not yet built.” This is because the existing index file still contains the @ID values in clear text. You must use the ENCRYPT.INDEX command to build the encrypted index file. ENCRYPT.INDEX uses the same encryption algorithm and key you specified when encrypting the @ID. You do not need to provide this information unless you want to use a different algorithm or key for the index.



Note: Due to the many complications that can arise if the @ID is encrypted, we suggest that you only encrypt the @ID if it is absolutely necessary.

TCL Commands Affected by @ID Encryption

This section describes the TCL commands that are affected by the @ID encryption enhancement.

ENCRYPT.FILE

The syntax of the ENCRYPT.FILE command remains the same as previous releases, but you can now specify @ID as the field name.

Syntax

```
ENCRYPT.FILE filename ... { WHOLERECORD | fieldname },alg,key[,pass]  
[fieldname,alg,key[,pass]]
```

The following table describes each parameter of the syntax.

Parameter	Description
<i>filename</i>	The name of the file to be encrypted.
WHOLERECORD	Specifies to fully encrypt every record in the file.
<i>fieldname,alg,key,pass</i>	Specifies the field name to encrypt, and the algorithm, key, and password to use. You can use a different algorithm and key for each field. If you do not specify a password, but created the key using password protection, UniVerse prompts for the password. If several fields use the same password, you only have to specify it once, at the first field that uses that key.
<i>fieldname</i>	The name of the field to encrypt.
<i>alg</i>	The algorithm to use for encryption. See “UniVerse Encryption Algorithms” in <i>UniVerse Security</i> for a list of valid values.
<i>key</i>	The key ID to use for the field encryption.
<i>pass</i>	The password corresponding to the <i>key</i> .

ENCRYPT.FILE Parameters

If you encrypt the @ID, the index files for the file are automatically set to encrypt index mode. UniVerse sets a flag in the index files indicating that the indexes must be rebuilt, and makes the indexes unavailable. You must run ENCRYPT.INDEX for the indexes to become available.

DECRYPT.FILE

The syntax of the DECRYPT.FILE command remains the same as previous releases, but you can now specify @ID as the field name.

Syntax

```
DECRYPT.FILE filename ... { WHOLERECORD | fieldname },key[,pass]  
[fieldname,key[,pass]]...>
```

The following table describes each parameter of the syntax.

Parameter	Description
<i>filename</i>	The name of the file to decrypt.
WHOLERECORD	Specifies to fully decrypt every record in the file.
<i>fieldname,key,pass</i>	Specifies the field name to decrypt, and the key, and password to use. If you do not specify a password, but created the key using password protection, UniVerse prompts for the password. If several fields use the same password, you only have to specify it once, at the first field that uses that key.
<i>fieldname</i>	The name of the field to decrypt.
<i>key</i>	The key ID to use for the field decryption.
<i>pass</i>	The password corresponding to the <i>key</i> .

DECRYPT.FILE Parameters

The DECRYPT.FILE does not decrypt the index files. You must run the DECRYPT.INDEX command explicitly to decrypt the index files.

Other TCL Commands Affected by @ID Encryption

The following table describes TCL commands affected if you encrypt the @ID.

Command	Behavior
BUILD.INDEX	Prompts for password if the key for the encrypted index is not activated.
CREATE.INDEX	Prompts for password if the key for the encrypted index is not activated.
LIST.INDEX	Adds an “E” at the end of the “Type” field for encrypted index.
UPDATE.INDEX	Prompts for password if the key for the encrypted index is not activated.

TCL Commands Affected by @ID Encryption

When you execute the following commands, UniVerse tries to decrypt the data, but does not report an error if the decryption fails. If the encryption key is activated, clear text will be returned, otherwise encrypted data will be returned.

- COUNT
- CHECK.SUM
- STAT
- SUM
- ESEARCH/SEARCH

You must activate the encryption key password before executing the following commands:

- COPY
- FORM.LIST
- CONVERT.SQL
- EXCHANGE
- FILE.USAGE
- FILE.USAGE.CLEAR
- FILE.USAGE.OFF
- RECORD
- REFORMAT
- SREFORMAT
- REVISE
- SEARCH
- SELECT
- SORT
- SORT.ITEM
- SORT.LABEL
- SSELECT

The following commands use encrypted data:

- ACCOUNT.FILE.STATS
- ANALYZE.FILE

- FILE.STAT
- GROUP.STAT

Encrypting the @ID with U2 Data Replication

If you enable U2 Data Replication, UniVerse passes encrypted data between the publishing system and the subscribing system.

Encrypting the @ID with UniVerse Transaction Logging

If you are using UniVerse Transaction Logging and encrypting the @ID, UniVerse saves the @ID as encrypted data. You must specify the key password on the command line so Transaction Logging can decrypt the data when applying changes back to the data file.

If index files are involved and the indexes are encrypted, you must also specify the key password to the encrypted index on the command line to activate these keys.

Encrypting the @ID with uvbackup and uvrestore

If you use uvbackup and uvrestore with an encrypted @ID, you must specify the key password on the command line so uvrestore decrypt the @ID when you restore the data back to the original file.

If index files are involved and the indexes are encrypted, you must also specify the key password to the encrypted index on the command line to activate these keys.

Encrypting Index Files

Before this release, UniVerse did not allow an indexed field to be encrypted.

WHOLERECORD encryption is now supported on a file containing indexes. You can also create indexes on previously nonindexed fields on WHOLERECORD-encrypted files. Likewise, you can now encrypt a field that has an index, or you can create an index on an encrypted field.

The encryption key must be active at the time you are building the index.

When you encrypt the @ID field, all existing indexes on the file automatically become encrypted indexes. When you encrypt a file with the WHOLERECORD mode, all existing indexes become encrypted indexes. When you encrypt a field that has an index, the index becomes an encrypted index. For these “inexplicitly encrypted” indexes, you must execute the ENCRYPT.INDEX command to actually build and encrypt the data in the indexes. You can also use the BUILD.INDEX command to encrypt them, but BUILD.INDEX is much slower than ENCRYPT.INDEX.

If you want to encrypt an index on a nonencrypted field and the @ID is not encrypted, or you want to use different encryption parameters (key, password, or algorithm) than the defaults, you must execute the ENCRYPT.INDEX command and fully specify the encryption parameters. The command will rebuild the index according to your specifications.

You can encrypt a virtual field index, but you cannot encrypt the virtual field itself. If any data field in a file is encrypted, you should analyze all virtual field indexes and encrypt any index that involves the encrypted data field. Otherwise, you run the risk of exposing the encrypted data in clear text through an unencrypted virtual field index file.

Considerations When Encrypting an Index

Many complexities arise when encrypting an @ID and an index. Some of the points to consider are:

- Encrypting the @ID automatically causes all existing indexes to be encrypted. You can have a nonencrypted data field with an encrypted index, but you cannot have a nonencrypted index if the @ID is encrypted.

- If you decrypt the @ID, all encrypted indexes will remain encrypted. You can manually decrypt indexes on nonencrypted fields.
- A field and its index can be encrypted with different encryption parameters (encryption algorithm, key and password).
- Once you encrypt an index file, the index keeps the encryption parameters you specify unless you explicitly change them.
- You can encrypt a virtual field index, but not the virtual field itself.
- Replication may need to access encryption keys in order to update an index.

The encryption key must be active when you use or update an index, or all operations will fail.

Encrypting an index will not affect the way an index is used.

Commands Used for Index Encryption

ENCRYPT.INDEX Command

At this release, a new command, ENCRYPT.INDEX is introduced. This command encrypts the index file associated with a field. It does not rebuild the index.

Syntax

```
ENCRYPT.INDEX <filename> <field>[, <alg>, <key>[,<pass>]]
[<field>[, <alg>, <key>[,<pass>]] ...
```

Parameters

The following table describes each parameter of the syntax:

Parameter	Description
<i>filename</i>	The name of the file for you which you to encrypt the index.
<i>field</i>	The name of the field you want to encrypt.

ENCRYPT.INDEX Parameters

Parameter	Description
<i>alg</i>	A string containing the cipher name
<i>key</i>	The encryption key
<i>pass</i>	The password for the encryption key.

ENCRYPT.INDEX Parameters (Continued)

Any field you specify must already have an index created and built. If it is already encrypted as a result of the @ID, WHOLERECORD, or field encryption, you can omit the algorithm and key specifications.

DECRYPT.INDEX Command

The DECRYPT.INDEX command decrypts the index file associated with a field. It does not rebuild the index.

Syntax

```
DECRYPT.INDEX <filename> <field>[, <key>[,<pass>]]  
[<field>[,<key>[,<pass>]]]
```

Parameters

The following table describes each parameter of the syntax:

Parameter	Description
<i>filename</i>	The name of the file for you which you to decrypt the index.
<i>field</i>	The name of the field you want to decrypt.
<i>key</i>	The encryption key.
<i>pass</i>	The password for the encryption key.

DECRYPT.INDEX Parameters

If the file has @ID or WHOLERECORD encryption, you cannot decrypt any encrypted indexes. If the data field is still encrypted, you cannot decrypt the index associated with the field.

REENCRYPT.INDEX Command

The REENCRYPT.INDEX command encrypts and decrypts the index file associated with a field. This command only deals with the index file, it does not rebuild the index from the data file.

Syntax

```
REENCRYPT.INDEX <filename> -D <decrypt_specs> -E  
<encrypt_specs>
```

Parameters

The following table describes each parameter of the syntax:

Parameter	Description
<i>filename</i>	The name of the file for you which you to decrypt the index.
<i>decrypt_specs</i>	<i>field</i> The name of the field you want to decrypt. <i>key</i> The actual key or file name containing the key. <i>pass</i> The password for the encryption key.
<i>encrypt_specs</i>	<i>field</i> The name of the field you want to encrypt. <i>alg</i> A string containing the cipher name <i>key</i> The actual key or file name containing the key. <i>pass</i> The password for the encryption key.

REENCRYPT.INDEX Parameters

You must specify the -D option before the -E option. Normally, you specify a field in both -D and -E to reencrypt the field. You can also only specify a field in -D or -E provided that:

- For -D option – The data fields must be nonencrypted, and no @ID/WHOLERECORD encryption can exist in the file.
- For -E option – The index should already have been created for the field you specify, and it is not already encrypted.

FILEINFO() Function

The FILEINFO() function returns information about encrypted indexes on the file in a dynamic array. Information for each encrypted index is separated by value marks (@VM). For each index, UniVerse returns the following information, separated by subvalue marks (@SVM):

```
index_name<@SVM>key_id<@SVM>algorithm
```

Utilities Affected by Index Encryption

This section discusses utilities affected by index encryption.

uvbackup/uvrestore

When using uvbackup, UniVerse automatically activates encryption keys on an as-needed basis, decrypts the index data, encrypts the data, and backs the data up to the media. Encryption keys and passwords do not have to be provided.

When using uvrestore, as long as the target system has the exact automatic data encryption configuration as the source system, UniVerse can successfully restore the data without prompting for encryption keys and passwords. However, if files contain triggers or virtual field indexes, you may provide the encryption key and password. Otherwise, the data may not be correct. UniVerse will display a warning if the key is not available.

For encrypted indexes or an index on an encrypted field, you must active the encryption keys, otherwise uvrestore will skip these files.

A new parameter, -k, has been added to uvrestore at this release to enable you to provide encryption keys and passwords to use during the restore process. UniVerse then decrypts as much data as possible and passes the data to the triggers.

@ID and Index Encryption Interaction

The following table summarizes behaviors associated with @ID and index encryption.

@ID Encryption	Behavior
@ID encrypted	All index files will be encrypted. Cipher text is stores in the data and index files. You cannot decrypt the index while the @ID in encrypted, you can only reencrypt the index.
@ID not encrypted	If the @ID is not encrypted and you specified WHOLERECORD encryption, all index files will be encrypted. If the @ID is not encrypted and you specified fields to be encrypted, the associated indexes for the fields you specified will be encrypted.

@ID and Index Encryption Interaction

PHANTOM Inheritance of Activated Fields and Disabled Fields

Before this release, a PHANTOM process did not have access to encryption keys activated or disabled fields for the parent process. Beginning at this release, UniVerse automatically passes key activation and field disablement information to phantom processes.

In addition to PHANTOM processes, any child process of a UniVerse session also inherits key activation and field disablement information.



Changing Passwords for Encryption Keys

Before this release, you could not change a password for an encryption key without re-creating the encryption key itself and specifying a new password. If you had used that key to encrypt data, the data had to be decrypted with the old encryption key and reencrypted with the new encryption key.

***Note:** Changing the password for the encryption key does not change the key itself. UniVerse always encrypts data with the key information derived when you first created the encryption key. The consequence is that if you have to re-create the key due to key store corruption and no available backup, you must create the key using the same password as when it was first created.*

At this release, UniVerse allows you to change the encryption key password. Only the user who created the encryption key can change the password, and you must provide the original password.

You can also change the password to an encryption key wallet. By default, when you change the password for a key, the password for that key in all wallets that contain that key are also changed. You can choose not to change the password in the wallets by specifying the NOCASCADE option.

The CHANGE.ENCRYPTION.PASSWORD Command

Use the CHANGE.ENCRYPTION.PASSWORD command to change the password for an encryption key.

Syntax

```
CHANGE.ENCRYPTION.PASSWORD ID existing_password  
new_password [NOCASCADE]
```

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
ID	The name of the encryption key or the wallet ID.
<i>existing_password</i>	The current password for the encryption key or wallet.
<i>new_password</i>	The new password for the encryption key or wallet.
NOCASCADE	By default, when you change the password for a key, the password for that key in all wallets that contain that key is also changed. You can choose not to change the password in wallets by specifying the NOCASCADE option.

CHANGE.ENCRYPTION.PASSWORD Parameters

The new password should conform to password policies. To specify no password, enter a quoted empty string (“”) on the command line. If you do not specify the CHANGE.ENCRYPTION.PASSWORD parameters on the command line, UniVerse prompts you for the current password and the new password. To specify no password, press ENTER when prompted.

Password Policies

Prior to UniVerse 11.1, no rules existed for passwords for encryption keys.

Beginning at this release, UniVerse allows users to specify groups of password policies for encryption keys and wallets. UniVerse stores password policies in an encryption password policy file in UVHOME. The name of the file is “.uvspolicy.” The file has the following characteristics:

- You manage this password policy file from XAdmin or through the encman utility.
- UniVerse reuses the user’s policy file when performing an upgrade installation.
- UniVerse applies the password policy to all users, and to all keys and wallets.
- You can configure and enforce separate policies for the encryption key, the wallet key, and the master key.

You must have root or uvadm privilege to configure password policies.

UniVerse supports the following password policies:

Policy	Values
EnforcePolicy	0 = No policy enforced 1 = Defined policies enforced. 1 is the default.
MinimumLength	The minimum length of the password. The valid value range is 6 - 64. The default value is 7. As a best practice, the length of the password should never be less than 6.
MaximumLength	The maximum length of the password. The valid value range is 6 - 64. The default value is 32.
MinimumAge	The days between encryption key password changes. 0 indicates to allow immediate change. If MaximumAge is not 0, MinimumAge cannot exceed MaximumAge.
MaximumAge	The days after which the encryption key password expires. The default value is 90. Valid values are 0 - ($2^{31} - 1$). 0 indicates never expire.

Supported Password Policies

Policy	Values
RequiredCharSet	<p>The character set for the password. The default values are:</p> <ul style="list-style-type: none"> ■ ALPHA_UPPER (decimal value 1, binary value 0001) ■ ALPHA_LOWER (decimal value 2, binary value 0010) ■ NUMERIC (decimal value 4, binary value 0100) ■ SPECIAL (decimal value 8, binary value 1000) <p>SPECIAL characters are all printable nonalphanumeric characters except a single quotation mark (‘), double quotation marks (“”), and a backslash (\).</p> <p>Nonprintable characters are not permitted.</p> <p>You can use a bitwise OR to combine these values together to specify a combination of characters. For example, if you want to specify both ALPHA_UPPER and ALPHA_LOWER, enter a value of 3.</p> <p>You can specify a number of required characters by specifying that there should be at least 1 – 4 types of characters. For example, if you specify REQUIRE_3, any combination of 3 out of 4 types of characters are permitted. For example, the following passwords would be considered valid:</p> <ul style="list-style-type: none"> ■ Pass7834 ■ mypass12/23 ■ PMod89\$12 <p>The actual values are:</p> <ul style="list-style-type: none"> ■ REQUIRE_1 16 ■ REQUIRE_2 32 ■ REQUIRE_3 48 ■ REQUIRE_4 64

Supported Password Policies

Policy	Values
ExpirationWarning	The number of days before UniVerse starts to display a pre-expiration warning. UniVerse only displays the warning message when you are executing TCL commands. UniVerse suppresses the warning message when executing UniVerse BASIC programs. The default value is 14. The valid value range is 0 - 30.
Complexity	<p>0 – Do not perform various checks to ensure the strength of a password.</p> <p>1 - ACCOUNTKEY – Checks if the account name to which the session is logged in or the encryption key ID is part of the password. Validation is case-sensitive.</p> <p>2 - SUCCESSION – Checks if there are more than two of the same character in succession.</p> <p>You can use a bitwise OR to combine these values together to specify a combination of characters.</p>
MinimumHistory	The number of previous passwords that cannot be reused. 0 means no restriction. The default value is 8. The valid value range is 0 - 24.

Supported Password Policies

The password policy checking is only performed when you create new encryption keys or change a current password.

Beginning at UniVerse 11.1, the password policy file must be available when you create a new encryption key or change an existing encryption key. If the file does not exist, a message is displayed to the terminal and logged to an audit file. UniVerse then uses the system defaults to enforce password compliance.

Setting up Password Policies

Use the `encman` utility to set up password policies.

```
encman -passpolicy [ALL | KEY | WALLET | MASTERKEY [policy_name
policy_value]]
```

The following table describes each parameter of the syntax:

Parameter	Description
ALL	Manage the password policy to encryption keys, wallets, and master key.
KEY	Manage the password policy to only encryption keys.
WALLET	Manage the password policy to only wallet encryption keys.
MASTERKEY	Manage the password policy to only the master key.
<i>policy_name</i>	The password policy you want to change. Valid values are: EnforcePolicy MinimumLength MaximumLength MinimumAge MaximumAge RequiredCharSet ExpirationWarning Complexity MinimumHistory You can specify “default” as the value for <i>policy_name</i> . If you specify “default,” UniVerse sets all policy names to the system defaults.
<i>policy_value</i>	The value for the policy you are defining. You can specify “default” as the value for <i>policy_value</i> . If you specify “default,” UniVerse sets all policy values to the system defaults.

encman -passpolicy Parameters

If you do not enter the policy_name or policy_value, UniVerse prompts for them, as shown in the following example:

```
C:\IBM\UU\testeda>\ibm\uo\bin\enclan -passpolicy
Current directory: C:\IBM\UU\testeda.
Changed directory to C:\IBM\UU.
Configure policy for [A]ll, [K]ey, [W]allet, or [M]asterKey? :_
```

Enter the type of component for which you are defining the password policy. A screen similar to the following example appears:

```
Password policy values for ADEKey:
[1] EnforcePolicy.....: 1
[2] MinimumLength.....: 7
[3] MaximumLength.....: 64
[4] MinimumAge.....: 1
[5] MaximumAge.....: 90
[6] RequiredCharSet.....: 7
[7] ExpirationWarning...: 14
[8] Complexity.....: 3
[9] MinimumHistory.....: 4
Enter policy to change [1-9], or press <ENTER> to continue:
```

Enter the number of the policy you want to change, then enter the value for the policy.

You must be logged on as root or uvadm to change a password policy.

Protecting Master Keys with Multiple, Changeable Passwords

You must create a master key before you can perform any other automatic data encryption operation. Some operations, such as retagging the key store, exporting and importing a key store, and so forth, also require that you provide the master key.

Beginning at this release, you can specify up to two passwords to be associated with the master key. You can also change the number of passwords used to protect the master key. You can change the master key password as often as you wish, subject to the established password policy. For more information about password policies, see “[Password Policies](#)” on page 17. You must provide all current passwords in order to change any password for the master key.



***Note:** If a master key has two passwords, you must change them together, you cannot change just one of the passwords. When enforcing the password history rule, multiple passwords are checked in the same password pool. For example, if `MinimumHistory` is set to 4, you can change two passwords and recycle them after two iterations, as opposed to four iterations if using a single password.*

You do not have to protect a master key with a password.

Specifying Master Key Passwords

Use the `uvregen` command to change the master key:

```
uvregen -m <master_key> [CURRENT] [-P <password_A> [-P <password_B>]]
```

The following table defines each parameter of the syntax.

Parameter	Description
<code>-m <master_key></code>	The master key you want to define
<code>CURRENT</code>	Specify <code>CURRENT</code> if you are changing the passwords for the current master key.
<code>-P <password_A></code> <code>-P <password_B></code>	You can specify up to two passwords for the master key. If you do not enter <code>-P</code> on the command line, or only enter <code>-P</code> once, UniVerse prompts for the remaining new passwords. If you are resetting the password(s) for a master key, <code>uvregen</code> prompts to input passwords for the current master key, then the new passwords for the master key, if not fully specified on the command line. If you the password you specify begins with “@” it specifies that a file contains the master key.

uvregen Parameters

Retagging a Key Store

You can use the `encman` utility with the `-P` option to unlock the master key to use in a retag operation.

```
encman -retag -m [<master_key> | [CURRENT] [-P <password_A> [-P  
<password_B>]]
```

If the master key is protected by passwords, specify `CURRENT`. If you do not specify `-P`, the `encman` utility prompts for the passwords.

If the master key is not protected by passwords, you must specify the master key string. Do not specify the `-P` option.

Exporting and Importing a Key Store

You must provide the master key when you import or export a key store. If the master key is protected by passwords, provide the passwords on the command line using the `-P` option.

```
encman {-export | -import} [<master_key> | [CURRENT] [-P  
<password_A> [-P <password_B>]] [<password>] <filename> ] ...
```

Specify the master key as either `CURRENT` or the actual key string. If you do not enter the master key or master key passwords on the command line, UniVerse prompts for them. If the master key is protect by passwords, you should specify `CURRENT` on the command line.

One-pass Reencryption

Periodic file reencryption using different encryption keys is often needed due to internal policies or regulation. This process is often referred to as reeking a file.

In order to rekey a file before this release, you had to first decrypt the file, then reencrypt it. At this release decryption and encryption have been combined, reducing the time needed to rekey a file.

Use the REENCRYPT.FILE command to rekey a file.

Syntax

```
REENCRYPT.FILE <filename> <resize options> -D<enclist>  
-E<enclist>
```

The following table describes each parameter of the syntax.

Parameter	Description
<i>filename</i>	The name of the file you want to reencrypt.
<i>resize options</i>	Any options available with the RESIZE command.
-D <enclist>	The list of fields you want to decrypt. Each field you specify in the <i>enclist</i> has the following format: <i><field_name></i> WHOLERECORD, <key>[.pass]
-E <enclist>	The list of fields you want to encrypt. Each field you specify in the <i>enclist</i> has the following format: <i><field_name></i> WHOLERECORD, <alg>, <key>[.pass]

REENCRYPT.FILE Parameters

The -D and -E options are independent of each other, meaning that you can specify different fields or modes can be different from the current definition. For example, you may want to change from WHOLERECORD encryption to field encryption.

If you specify nonencrypted fields in the -D encryption list, or the -E encryption list contains fields already encrypted by not decrypted by the -D list, UniVerse generates an error and terminates the process.

Password for TCL Command Not Echoed

Beginning at UniVerse 11.1, UniVerse supports no-echo password mode for all Automatic Data Encryption commands that require passwords to be entered on the command line.

To use no-echo mode, you must enter the Automatic Data Encryption command without any parameters, then follow the prompts.

UniVerse will now prompt twice for a new password when executing the `CREATE.ENCRYPTION.KEY`, `CREATE.ENCRYPTION.WALLET`, and `CHANGE.ENCRYPTION.PASSWORD` commands.

U2 Data Replication for UniVerse

U2 Data Replication for UniVerse	2-2
Processes Added at UniVerse 11.1	2-3
UniVerse Commands Supported in Replication	2-11
Installing U2 Data Replication for UniVerse	2-15
Administering U2 Data Replication	2-19

U2 Data Replication for UniVerse

U2 Data Replication for UniVerse provides an automatic way to deliver read-only copies of UniVerse files to other UniVerse systems. You can use the replicated data as a standby system in case of system failure, or as a reporting system.

The system where the source data resides is called the **publisher**. A system requesting copies of file updates from the publisher is called a **subscriber**.

Processes Added at UniVerse 11.1

At UniVerse 11.1, daemons have been added for U2 Data Replication and cleaning up dead user processes.

A daemon is a background process that performs a specific task or set of tasks. Daemons wait in the background until they receive a request for their specific function. A number of standard UNIX daemons run on every UNIX platform to control system processes, schedule commands, handle print requests, and perform other similar functions. Refer to your host operating system documentation for detailed information about the UNIX daemons that run on your system.

Shared Memory Manager

The Shared Memory Manager daemon, `uvsmm`, records all operating system resources, such as shm segments and semaphores, identifying which UniVerse user holds each.

Replication Manager Daemon

The Replication Manager is a UniVerse daemon that must be running on the replication system. The publishing system and the subscribing system each have one replication manager. The replication manager performs the following tasks:

- Reads and loads replication configuration information when UniVerse starts
- Creates replication processes for each replication group
- Monitors and controls the replication processes
- Handles commands from Replication Admin and XAdmin
- Dynamically reconfigures the replication environment
- Handles failover replication, recovery, and synchronization of replication data

uvcleanupd Daemon

The clean up daemon, uvcleanupd, detects terminated user processes at check time intervals. If uvcleanupd detects a terminated process, internal flags are set. The uvsmm daemon periodically check to see if uvcleanupd has set internal flags. If it detects flags, uvsmm performs the necessary cleanup and resets its own flag to zero. The uvcleanupd daemon performs clean up that is not handled by uvsmm. When the uvsmm daemon has reset its flag to zero, the uvcleanupd daemon resets its flag to zero, makes the user process ID available, and frees the local control table.

Both the uvdlockd and uvcleanupd daemons are started when UniVerse starts. Beginning at this release, the uvdlockd daemon only handles deadlocks.

uvcleanupd Command

Use the uvcleanupd command forces a clean up of resources owned by a terminated process. You must be a UniVerse administrator or the user that has the same signature as the terminated process to execute the uvcleanupd command.

Syntax

uvcleanupd {*-p pid* | *-n sig*}

Parameters

The following table describes each parameter of the syntax:

Parameter	Description
<i>-p pid</i>	Instructs the uvcleanupd daemon to make the LCT entry for the pid you specify available, and allows the uvcleanupd daemon to clean up the entry.
<i>-n sig</i>	Instructs the uvcleanupd daemon to make the LCT entry for the signature you specify available, and allows the uvcleanupd daemon to clean up the entry.

uvcleanupd Parameters

Note: The *-p* and *-n* options should be used together. These two options are used to indentify the matching LCT entry.

Starting UniVerse Daemons

Use one of the following commands to start the UniVerse daemons:

- `uv -admin -start [-init]`
- `uv.rc start [init]`

If you specify the `-init` option, U2 Data Replication clears the pending replication logs saved to synchronize the subscriber database. You can use this option for initial startup, or after refreshing a subscriber database.

Stopping UniVerse Daemons

Use one of the following commands to stop the UniVerse daemons:

- `uv -admin -stop [-force]`
- `uv.rc stop [force]`

These commands check whether any users are still logged on to the system. If users are logged on, UniVerse does not stop the system. If you specify the `-force` option, UniVerse stops the system even if there are users logged on to the system.

Displaying UniVerse Daemons

Use the `showuv` command to display daemon processes currently running on the system. The following example illustrates output from the `showuv` command:

```
% ./showuv
      USER      PID      TIME COMMAND
      root    831708      0:00
/tarpan1/srcman/alpha/unishared/unirpc/unirpcd
      root    950360      0:01
/tarpan1/srcman/alpha/uv111_100505_5580/bin/uvcleanu
pd -t 15
      root    651270      0:00
/tarpan1/srcman/alpha/uv111_100505_5580/bin/uvdlockd
-config
      root    385206      0:00
/tarpan1/srcman/alpha/uv111_100505_5580/bin/uvrepman
ager
      root    962604      0:00
/tarpan1/srcman/alpha/uv111_100505_5580/bin/uvsmm -t
15
```

Displaying SMM Segment Contents

Use the `uvsms` command to display SMM segment contents.

Syntax

```
uvsms [options]
```

Options

The following table describes the options available for the `uvsms` command:

Options	Description
-h	Displays the SMM segment header.
-G <i>shm_ID</i>	Displays the specific shared memory segment you specify with <i>shm_ID</i> .
-g <i>shm_nono</i>	Displays the specific shared memory segment you specify with <i>shm_nono</i> .
-L <i>pid</i>	Displays the specific LCT entry you specify with <i>pid</i> .
-l <i>lct_no</i>	Displays the specific LCT entry you specify with <i>lct_no</i> .
-S <i>shm_ID</i>	Displays the LCT entry of the session where the shared memory segment is created.

uvsms Options

Global Control Table

The Global Control Table (GCT) manages shared memory segments systemwide. This table is located in the Shared Memory Management segment. Each GCT records the use of global pages in a shared memory segment. UniVerse determines the number of GCTs in the CTL by the configuration parameter `SHM_GNTBLS`. `SHM_GNTBLS` must not exceed the kernel parameter `shmmni`.

Local Control Tables

Each Local Control Table (LCT) records the shared memory activity of a UniVerse process group. UniVerse determines the number of LCTs in the CTL by the NUSERS configuration parameter. Each LCT comprises four subtables:

Subtable Name	Description
PI	Process Information table. Each PI table registers all processes within a process group.
CT	Counter table. Each CT records information about the behavior of the process group.
MI	Memory Information table. Each MI table records all global pages or self-created shared memory segments used by the process group.
CI	Control Information table. Each CI table records all blocks allocated from shared memory for temporary buffers.

LCT Subtables

A process group related to a process group leader, or a user executing a UniVerse system-level command from a UNIX prompt.

Default Account Level Replication File

The repacct.def file, located in the UVHOME directory, includes a default list of files to be included or excluded for an account-level replication group.

Following is an example of the accrep.def file.

```
# Definition of default excluded objects for account replication
EXCLUDED_FILE=DICT.DICT
EXCLUDED_FILE=DICT.VOC
EXCLUDED_FILE=SQL.HELP
EXCLUDED_FILE=BCI.HELP
EXCLUDED_FILE=BASIC.HELP
EXCLUDED_FILE=HELP.FILE
EXCLUDED_FILE=ERRMSG
EXCLUDED_FILE=SYS.HELP
EXCLUDED_FILE=SYS.MESSAGE
EXCLUDED_FILE=SYS.TERMINALS
EXCLUDED_FILE=REVISE.DISCUSSIONS
EXCLUDED_FILE=REVISE.PROCESSES
EXCLUDED_FILE=STAT.FILE
EXCLUDED_FILE=UNIVERSE.STAT.FILE
EXCLUDED_FILE=TESTQ

EXCLUDED_FILE=&SAVEDLISTS&
EXCLUDED_FILE=&PARTFILES&
EXCLUDED_FILE=&COMO&
EXCLUDED_FILE=&PH&
EXCLUDED_FILE=&UFD&
EXCLUDED_FILE=&DEVICE&

EXCLUDED_FILE=UV;ACCESS
EXCLUDED_FILE=UV.ACCOUNT
EXCLUDED_FILE=UV.FLAVOR
EXCLUDED_FILE=UV.LOGS
EXCLUDED_FILE=UV.TRANS

EXCLUDED_FILE=UV_UDRPUB
EXCLUDED_FILE=UV_UDRSUB
EXCLUDED_FILE=UDRSYS

EXCLUDED_FILE=UV_SCHEMA
EXCLUDED_FILE=UV_ASSOC
EXCLUDED_FILE=UV_COLUMNS
EXCLUDED_FILE=UV_TABLES
EXCLUDED_FILE=UV_VIEWS
EXCLUDED_FILE=UV_USERS

FILE=SUB_WRITEABLE DATA VOC
FILE=SUB_WRITEABLE AE_DOC
FILE=SUB_WRITEABLE &MAP&
FILE=SUB_WRITEABLE &KEYSTORE&
FILE=SUB_WRITEABLE &SAVEDLISTS&
FILE=SUB_WRITEABLE PTERM.FILE
FILE=SUB_WRITEABLE UNIVERSE.MENU.FILE
FILE=SUB_WRITEABLE UV.SAVEDLISTS
```

Terminology

The following table describes the UniVerse executable names for U2 Data Replication for UniVerse.

Replication Component	Executable Name
Shared Memory Manager Daemon	uvsmm
Stop Shared Memory Management Tool	stopuvsmm
Replication Manager Daemon	uvrepmanager
Show sms information	uvsms
Dead User Cleanup Daemon	uvcleanupd
Publisher Process	uvpub
Subscriber Process	uvsub
Replication Writer	uvrw
Publisher Listener Process	uvpublistener
Publisher Synching Tool	uvpubsyncer
Replication Manager RPC Tool	uvrmconn
Replication Diagnosis Tool	uvreptool
Stop Replication Manager Tool	stopuvrm
Replication Admin Tool	uv_repadmin
Show UNIX daemons	uv_repadmin
Show format processing	psfmt
Test Process ID	test_pid
Show host information	hostinfo

UniVerse Data Replication Executables

The next table describes the U2 Data Replication for UniVerse services names.

Replication Service	UniVerse Service Name
U2 Replication Service	uvrep
Replican Manager Connection Service	uvrmconn

UniVerse Data Replication Services

The next table describes the U2 Data Replication for UniVerse log files. These log files are located in the UVHOME directory.

U2 Replication Log File	UniVerse System Log File
SMM log file	uvssm.log
SMM Error log file	uvsmm.errlog
Clean Daemon log file	uvcleanupd.log
Cleanup Daemon Error log file	uvcleanupd.errlog
Replication Manager log file	uvrm.log
Replication Manager Error log file	uvrm.errlog
Publisher Error log file	uvpub<n>.errlog
Subscriber Error log file	uvsub<n>.errlog
Replication Writer Error log file	uvrw.errlog
UV Dead Lock Daemon log file	uvdlockd.log

Replication Log Files

For U2 Data Replication for UniVerse, the user session ID is referred to as UVSHNO.

UniVerse Commands Supported in Replication

The following UniVerse commands are replicated when using record-level in this release:

UniVerse BASIC Commands

- Any UniVerse BASIC statements that write to or delete a UniVerse file, except for those executed against sequential files and Type 25 files.

UniVerse SQL Commands

- INSERT
- UPDATE
- DELETE.SQL

UniVerse TCL Commands

- COPY
- ED and VI
- SETFILE
- DELETE.LIST
- COPY.LIST
- SAVE.STACK
- CREATE.ENCRYPTION.KEY
- CREATE.ENCRYPTION.WALLET
- GRAND.ENCRYPTION.KEY
- REVOKE.ENCRYPTION.KEY
- DELETE.ENCRYPTION.KEY
- DELETE.ENCRYPTION.WALLET
- WALLET.ADD.KEY
- WALLET.REMOVE.KEY



- SAVE.EDAMAP

Note: The SETFILE command creates a new VOC pointer, and also loads that pointer into the Replication File Table. If you delete the old VOC pointer using the DELETE.FILE command, the file may still replicate with the new VOC pointer. UniVerse replicates SETFILE as a file-level operation so the new pointer on the subscriber system is also loaded into the Replication File Table.

The SET.FILE command only creates a Q pointer in the VOC file. This operation is replicated only as a record-level update to the VOC file. UniVerse does not load the Q pointer into the Replication File Table. When an application opens the Q pointer and updates it, that update is replicated through the file that the Q pointer references.

Commands Supported for File-Level Replication

U2 Data Replication for UniVerse supports the following file-level commands.

- CLEAR.FILE
- CREATE.FILE
- DELETE.FILE
- CREATE TABLE
- DROP TABLE
- CNAME
- SET.FILE

If you execute the CLEAR.FILE command against a distributed file, UniVerse replicates each part file in one CLEAR.FILE command on the subscriber.

You cannot change the name of a multilevel file, only the names of its subfiles can be changed.

Commands Not Supported at this Release

The following commands are not supported at this release of UniVerse:

File-Level Commands

- DEFINE.DF
- DF.MODIFY

- REBUILD.DF
- CREATE TRIGGER
- DELETE TRIGGER
- RESIZE
- CREATE.INDEX
- DELETE.INDEX
- BUILD.INDEX
- DISABLE.INDEX
- ENABLE.INDEX
- UPDATE.INDEX
- ENCRYPT.INDEX
- DECRYPT.INDEX
- ENCRYPT.FILE
- DECRYPT.FILE
- EDA.CONVERT
- CONFIGURE.FILE
- UVFIXFILE
- RECOVER.FILE

SQL Commands

- ALTER TABLE
- CREATE INDEX
- CREATE SCHEMA
- CREATE TRIGGER
- CREATE VIEW
- DROP INDEX
- DROP SCHEMA
- DROP TRIGGER
- DROP VIEW
- CONVERT.SQL

- REVOKE
- SET.SQL

Account Environment Commands

- SAVE.LIST
- PREPARE.SML
- INITIALIZE.CATALOG
- CLEAN.ACCOUNT
- CONVERT.VOC
- D3RESTORE
- DATE.FORMAT
- MAKE.MAP.FILE
- PASSWD
- SET.CONVERT.TERM.EURO
- SET.SYSTEM.EURO
- SET.REMOTE.ID
- SET.TELNET
- UPDATE.ACCOUNT
- XMLSETOPTIONS

Installing U2 Data Replication for UniVerse

When you install U2 Data Replication for UniVerse, the installation process installs the following components:

- The default configuration parameters in the uvconfig file.
- The data replication log directory.
- A default Replication System Definition file (repsys) in the UVHOME directory.
- An empty Replication Configuration File (reconfig) in the UVHOME directory.
- A U2 Replication Service (uvsub) and Replication Manager Connection Service (uvrmconn) in the system RPC service table.

If you configure U2 Data Replication for UniVerse, UniVerse starts the following daemons:

Daemon	Function
uvsmm	Manages shared memory segments for UniVerse Data Replication.
uvrepmanager	The UniVerse Data Replication Manager.
uvcleanupd	The UniVerse deadlock daemon. UniVerse starts this daemon when you start the UniVerse system. New functions cleanup dead users for UniVerse Data Replication.

UniVerse Data Replication Daemons

New uvconfig Parameters for U2 Data Replication

The uvconfig file contains the following new parameters when you install UniVerse Data Replication:

Parameter	Description
REP_FLAG	Enables/disables UniVerse Data Replication. This parameter is exclusive to UDRMODE. If both REP_FLAG and UDRMODE are set to 1, UniVerse will not start.
MAX_REP_SHMSZ	Defines the maximum shared memory segment size for U2 Data Replication.
TCA_SIZE	The size of the Transaction Control Area.
MAX_LRF_FILESIZE	Defines the maximum size of the replication log file.
N_REP_OPEN_FILE	Defines the number of replication component files a UniVerse process can open.
REP_LOG_PATH	The path to replication log files. The default value is \$UVHOME/replug.
MAX_RW_INGRP	The maximum number of Replication Writer processes that can be configured a replication group. The default value is 8.

uvconfig Parameters for U2 Data Replication

Configuration Parameters for Shared Memory

The following table describes the new uvconfig parameters added for shared memory management at this release.

Parameter	Description
NUSERS	Number of user sessions. Recommended value is twice the number of licensed users.
SHM_MAX_SIZE	Maximum size of the shared memory segment.
SHM_ATT_ADD	Starting address for attaching shared memory segments.
SHM_LBA	Reserved for future use. Do not change.

Shared Memory Configuration Parameters

Parameter	Description
SHM_MIN_NATT	Number of shared memory segments to keep attached.
SHM_GNTBLS	Number of global memory tables for shared memory segments.
SHM_GNPAGES	Number of pages in each shared memory segment.
SHM_GNPAGESZ	Page size of each shared memory segment.
SHM_LPINENTS	Number of processes in each user session.
SHM_LMINENTS	Number of memory entries in each user session.
SHM_LCINENTS	Number of memory control entries in each user session.
SHM_LPAGESZ	Local page size.
SHM_FREEPCT	Percentage of free shared memory to keep.
SHM_NFREES	Number of free shared memory segments to keep.

Shared Memory Configuration Parameters (continued)

Following is an example of the repsys file:

```
# Define the local system
SYSTEM=System 1_uv111
HOSTNAME=Server1
ADDRESS=Server1.rs.com
VERSION=111
DHCP=0

# Define System2
SYSTEM=System2_uv111
HOSTNAME=Server2
VERSION=111
DHCP=0
AUTORESUME=1
TIMEOUT=600
EXCEPTION_ACTION=/test/uv103/repexception.sh
```

Replication Configuration File

Following is an example of the repconfig file:

```
# Definition of group SAMPLEGROUP
GROUP=SAMPLEGROUP
LEVEL=ACCOUNT
# Define publisher account
ACCOUNT=/test/uv111/HS.SALES
RESERVED_FILE_SPACE=500
#Definition of exclusive objects
EXCLUDED_FILE=DICT VOC
EXCLUDED_FILE=&XML&
EXCLUDED_FILE=VOCLIB
FILE=SUB_WRITEABLE DATA VOC
# Define the Publisher
DISTRIBUTION=P:System1_uv111
# Define the standby subscriber
DISTRIBUTION=IB:System2_uv111
# Definition of tunable parameters
N_LOGINFO=20400
REP_BUFSZ=1638400
N_REPWITER=2
RW_IGNORE_ERROR=1
RW_REEVALUATE_VF=0
RW_SKIP_TRIGGER=1
# End of definition of group SAMPLEGROUP
```

The UniVerse BASIC FILEINFO function can now report whether a file has been enabled for replication. The following key has been added to this function:

Key Symbolic Name	Key Value	Return Value Description
FINFO\$REPSTATUS	30	0: The file is not enabled for replication, or replication is not running. 1: The file is a publishing file. 2: The file is a subscribing file

FILEINFO Enhancement

Administering U2 Data Replication

Use the Extensible Administration Tool (XAdmin) to administer U2 Data Replication for UniVerse. For information about the administration screens, see the “*U2 Data Replication for UniVerse*” manual.

External Database Access (EDA)

External Database Access	3-2
External Database Access Drivers Provided with EDA	3-3
External Database Access Driver API	3-9

External Database Access

External Database Access (EDA) enables you to convert data stored in the UniVerse database to a 1NF database, such as IBM DB2, then access that data using existing UniVerse BASIC programs, Retrieve, or UniVerse SQL.



Note: *EDA was not designed to access data that already resides in a 1NF database. To access this type of data, use the UniVerse BASIC SQL Client Interface (BCI).*

For detailed information about EDA, see the “*External Database Access*” manual.

External Database Access Drivers Provided with EDA

This section describes the drivers UniVerse provides with EDA.

EDA Oracle Driver

UniVerse provides an EDA Oracle driver at this release. The EDA Oracle driver is a dynamic-loading library which the EDA engine uses to exchange data with an Oracle database.

The EDA Oracle driver supports Oracle Version 11g.

Set Up the EDA Environment

On UNIX platforms, execute the operating system-level command `edasetup.sh` to set up the EDA environment. This command prompts you for information and generates the `edaconfig` file in the `$UVHOME` account. The `edaconfig` file contains the following information:

```
DRIVER=ORACLE
ORACLEPATH=/test1/oracle/instantclient_11_1
LOGLEVEL=0
```

You can change the `LOGLEVEL` to 1 or 2. The higher the log level, the more information is captured.

On Windows platforms, manually edit the `edaconfig` file to add `LOGLEVEL`.

Set Up the Oracle Connection File

Set up the `tnsnames.ora` file, used to connect to the Oracle database. The following example illustrates the `tnsnames.ora` file:

```
ORDEVDB=
(DESCRIPTION =
  (ADDRESS = (PROTOCOL = TCF) (HOST=test.com) (PORT = 1521))
  (CONNECT_DATA =
    (SERVER = DEDICATED)
    (SERVICE_NAME = ORDEVDB)
  )
)
```

On UNIX platforms, the tnsnames.ora file must reside in the directory you specified as ORACLEPATH in the edaconfig file.

On Windows platforms, the tnsnames.ora file must reside in \$ORACLE_HOME\NETWORK\ADMIN.

Set Up Dynamic-Loading Library

To load the Oracle OCI libraries, you must set up a dynamic-loading library path. The following table specifies where to add the Oracle library path based on the platform you are using:

Platform	Oracle Path Location
AIX	Add Oracle library path to LIBPATH
Windows	Add Oracle library path to PATH
Other	Add Oracle library path to environment variable for loading dynamic library.

Oracle Path Locations by Platform

After you add the Oracle library path, restart unirpdc.

Create the EDA Data Source

When you create the EDA data source in the EDA Schema Manager, the external DB Name should be the connection name you specified in the tnsnames.ora file, such as ORDEVDEB.

Data Type Mapping

The next table describes how EDA maps UniVerse data to Oracle data types:

UniVerse Data	Oracle Data Type
Characters	VARCHAR
Date	DATE
Time	TIMESTAMP

Mapping UniVerse Data to Oracle Data Type



UniVerse Data	Oracle Data Type
Number (integer)	NUMBER(38)
Number (noninteger)	NUMBER
Unmapped fields	CLOB

Mapping UniVerse Data to Oracle Data Type (continued)

Note: When mapping UniVerse time data to SQL Server DATETIME, the date portion is filled with 01/01/2000.

EDA DB2 Driver

UniVerse provides an EDA DB2 driver at this release. The EDA DB2 driver is a dynamic-loading library which the EDA engine uses to exchange data with a DB2 database.

The EDA DB2 driver supports DB2 8.0 and greater.

The DB2 driver is available on AIX and HP Itanium platforms.

Set Up the EDA Environment

On UNIX platforms, execute the operating system-level command `edasetup.sh` to set up the EDA environment. This command prompts you for information and generates the `edaconfig` file in the `$UVHOME` account. The `edaconfig` file contains the following information:

```
DRIVER=DB2
DB2PATH=/home/db2inst1/sqllib
LOGLEVEL=0
```

You can change the `LOGLEVEL` to 1 or 2. The higher the log level, the more information is captured.

On Windows platforms, manually edit the `edaconfig` file to add `LOGLEVEL`.

Install DB2 or the DB2 Client

Install the DB2 client on the machine where UniVerse is installed. Create a database on the DB2 server to which UniVerse can connect. If you install the DB2 client after you install UniVerse, you must restart unirpcd.

Set up Connection to the DB2 Database

After you install the DB2 client, create a cataloged database on the DB2 client to use to connect to the database on the DB2 server.

Create the EDA Data Source

Create an EDA data source in the EDA Schema Manager.

Make sure you use the cataloged database name as the External DB Name if you installed the DB2 client. Otherwise, use the database name directly as the External DB Name.

Data Type Mapping

The following table describes how EDA maps UniVerse data to DB2 data types:

UniVerse Data	Oracle Data Type
Characters	VARCHAR
Date	DATE
Time	TIME
Number (integer)	INTEGER
Number (noninteger)	FLOAT
Unmapped fields	CLOB

Mapping UniVerse Data to DB2 Data Type

EDA SQL Server Driver

UniVerse provides an EDA SQL Server driver at this release. The EDA SQL Server driver is a dynamic-loading library which the EDA engine uses to exchange data with a SQL Server database.

The EDA SQL Server driver support SQL Server 2005 and greater.

Note: The EDA SQLServer driver is available on Windows platforms only.



Install SQL Server and Create ODBC Data Source

Install SQL Server, or at least the SQL Server Native Client, on the same machine where UniVerse is installed.

Create a database on SQL Server to which UniVerse can connect.

Create an ODBC data source to use to connect to the SQL Server database. Use Control Panel -> Administrative Tools -> Data Sources (ODBC) to open the ODBC Data Source Administrator dialog box. Add a system DSN and select SQL Native Client as the driver.

Create the EDA Data Source

Create an EDA Data Source in the EDA Schema Manager. Make sure you use the ODBC data source created in the previous step as the External DB Name.

Set Up the EDA Configuration File

Create an edaconfig file in *uvhome* for the EDA SQL Server driver log. Add the following line to the edaconfig file:

```
LOGLEVEL=0
```

You can change the LOGLEVEL to 1 or 2. The higher the log level, the more information is captured.

SQL Server Data Types

The following table describes how EDA maps UniVerse data to SQL Server data types.

UniVerse Data	SQL Server Data Type
Characters	VARCHAR
Date	DATETIME
Time	DATETIME
Number (integer)	INT
Number (noninteger)	REAL
Unmapped fields	VARCHAR(MAX)

Mapping UniVerse Data to SQL Server Data Types



***Note:** When mapping UniVerse date data to SQL Server DATETIME, the time portion is filled with 00:00:00.*

When mapping UniVerse time data to SQL Server DATETIME, the date portion is filled with 01/01/1753.

External Database Access Driver API

External Database Access (EDA) enables you to convert data stored in the UniVerse database to a 1NF database, such as IBM DB2, Oracle, and SQL Server, then access that data using existing UniVerse BASIC programs, Retrieve, or UniVerse SQL.



Note: EDA was not designed to access data that already resides in a 1NF database. To access this type of data, use the UniBasic SQL Client Interface (BCI).

The EDA Driver API enables you to write your own driver to access data in any relational database, such as Informix Dynamic Server. The EDA Driver API is a set of sixteen functions which EDA calls to communicate with an external database.

U2 Websphere MQ API

In This Chapter	4-2
Preface	4-3
Overview of Messaging	4-4
Overview of IBM WebSphere MQ	4-5
U2 WebSphere MQ API	4-6
Setup and Configuration for the WebSphere MQ API	4-7
Programming with the U2 WebSphere MQ API	4-10
U2MQI.H INCLUDE File	4-11
Error Handling in CallMQI	4-11
Encoding Message IDs and Correlation IDs	4-12
U2 WebSphere MQ API Programmatic Interfaces	4-13
2 WebSphere MQ API Programmatic Interfaces	4-14
Closing a Queue	4-16
Connecting to a Message	4-18
Disconnecting from a Queue Manager	4-20
Retrieving a Message from a Queue	4-22
Returning an Error Message	4-24
Querying the Attributes	4-25
Opening a Queue	4-27
Putting a Message on a Queue	4-29
Opening a Queue	4-31
Programming Examples	4-34
CallMQI Dynamic Array Structures	4-39
MQOD Dynamic Array Structure	4-39
MQMD Dynamic Array Structure	4-41
MQGMO Dynamic Array Structure	4-45
MQPMO Dynamic Array Structure	4-47

CallMQI Error Codes	4-51
CallMQI Reason Codes	4-53
Additional Reading.	4-72

In This Chapter

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

This chapter consists of the following sections:

- “Preface”
- “Overview of Messaging”
- “Overview of IBM WebSphere MQ”
- “U2 WebSphere MQ API”
- “Setup and Configuration for the WebSphere MQ API”
- “Programming with the U2 WebSphere MQ API”
- “U2 WebSphere MQ API Programmatic Interfaces”
- “Programming Examples”
- “CallMQI Dynamic Array Structures”
- “CallMQI Error Codes”
- “CallMQI Reason Codes”
- “Additional Reading”

Preface

The U2 WebSphere MQ API (CallMQI) makes use of the WebSphere MQ Message Queue API (MQI).

CallMQI is available for download from <https://u2tc.rocketsoftware.com/>.

This manual assumes you have a general understanding of WebSphere MQ. Numerous documents for WebSphere MQ, including those referenced in this chapter, 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 message delivery. This frees applications from having to take on this burden themselves with low-level communication details.

Message-based applications also have the advantage 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 (renamed from IBM MQSeries) 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 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 queue manager, open the destination queue, and retrieve the message.

U2 WebSphere MQ API

The U2 WebSphere MQ AMI (CallMQI) interface 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 is the BASIC equivalent of the WebSphere MQ Message Queue API. Like the WebSphere MQI, CallMQI consists of functions that let your application interact with the messaging service.

Examples of the capabilities now provided by CallMQI include the ability to:

- Explicitly set message IDs and correlation fields
- Set the user ID to perform a message action

Setup and Configuration for the WebSphere MQ API

To connect 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 remote queue manager via the WebSphere MQ "client libraries."

Complete the following steps to connect to a remote 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 must verify the WebSphere MQ Client installation before you can 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.

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. Next, the program connects the CallMQI API to a queue manager named saturn.queue.manager. It then opens a new queue, QUEUE1, and places a test message on the queue. The test message is retrieved from the queue and prints it out. CallMQI then closes the queue and disconnects from the queue manager.

To run the sample program, compile and run the program to verify that CallMQI is functioning correctly. Note that you will need to replace the names of both the queue manager and the queue with the respective names used when you verified the client installation. Also, ensure that you set up a client channel appropriately 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
    STOP
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
    STOP
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
    STOP
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

    STOP
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
    STOP
END
*
RET = MQDISC(HCONN)
IF RET # MQCC_OK THEN
    MQERR.RET = MQGETERROR(REASONCODE, REASONMSG)
    PRINT "MQDISC: " : REASONCODE " : " : REASONMSG
    STOP
END

```

Programming with the U2 WebSphere MQ API

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 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 40, 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.

Alternatively, when you generate 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).

U2 WebSphere MQ API Programmatic Interfaces

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



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

2 WebSphere MQ API Programmatic Interfaces

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



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.

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*)

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, hObj 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 on *page 54, CallMQI Reason Codes*.

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.

Connecting to a Message

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 on *page 54, CallMQI Reason Codes*.

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.

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 on *page 54, CallMQI Reason Codes*.

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.

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 msg 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 msgBufferLen 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 on *page 54, CallMQI Reason Codes*.

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.

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 return code.

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

Return Code Status

Usage Notes

A complete list of the reason codes can be found on *page 54, CallMQI Reason Codes*.

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 <code>MQGETERROR</code> function can be called for further details about the warning.
2 - MQCC_FAILED	The function call failed. The <code>MQGETERROR</code> 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 on *page 54, CallMQI Reason Codes*.

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.

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>	<p>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.</p> <p>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]</p>
<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.

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 on *page 54, CallMQI Reason Codes*.

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.

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]
<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. [IN/OUT]
<i>putMsgOpts</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]
<i>msg</i>	The message to be put on the queue. [IN]

MQPUT 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 on *page 54, CallMQI Reason Codes*.

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

Opening 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

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. For convenience, the CallMQI interface provides a default MQMD dynamic array, MQMD.DFLT, for initializing new MQMD dynamic arrays. [IN/OUT]
<i>putMsgOpts</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]
<i>msg</i>	The message to be put on from the queue. [IN]

MQPUT1 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 on *page 54, CallMQI Reason Codes*.

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.

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)
  * ...
ENDD
```

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)
  * ...
END
```

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.
9	MQOD.KnownDestCount	Number of local queues opened successfully.

MQOD Dynamic Array Structure

Attribute	Identifier	Description
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
4	MQOD.DFLT<MQOD.ObjectName>	""
5	MQOD.DFLT<MQOD.ObjectQMgrName>	""

MQOD.DFLT Dynamic Array Structure

Attribute	Identifier	Description
6	MQOD.DFLT<MQOD.DynamicQName>	"AMQ.*"
7	MQOD.DFLT<MQOD.AlternateUserId>	""
8	MQOD.DFLT<MQOD.RecsPresent>	0
9	MQOD.DFLT<MQOD.KnownDestCount>	0
10	MQOD.DFLT<MQOD.UnknownDestCount>	0
11	MQOD.DFLT<MQOD.InvalidDestCount>	0
12	MQOD.DFLT<MQOD.ObjectRecOffset>	0
13	MQOD.DFLT<MQOD.ResponseRecOffset>	0
14	MQOD.DFLT<MQOD.ObjectRecPtr>	MQPTR
15	MQOD.DFLT<MQOD.ResponseRecPtr>	MQPTR
16	MQOD.DFLT<MQOD.AlternateSecurityId>	MQBYTE40
17	MQOD.DFLT<MQOD.ResolvedQName>	""
18	MQOD.DFLT<MQOD.ResolvedQMgrName>	""

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.UserIdentifier	User identifier.
18	MQMD.AccountingToken	Accounting token.
19	MQMD.ApplIdentityData	Application data relating to identity.
20	MQMD.PutApplType	Type of application that put the message.

MQMD Dynamic Array Structure

Attribute	Identifier	Description
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

MQMD.DFLT Dynamic Array Structure

Attribute	Identifier	Description
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.UserIdentifier>	""
18	MQMD.DFLT<MQMD.AccountingToken>	MQBYTE32
19	MQMD.DFLT<MQMD.ApplIdentityData>	""
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

MQMD.DFLT Dynamic Array Structure (Continued)

Attribute	Identifier	Description
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.
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.

MQGMO Dynamic Array Structure

Attribute	Identifier	Description
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.ResolvedQName >	""

MQGMO.DFLT Dynamic Array Structure

Attribute	Identifier	Description
8	MQGMO.DFLT<MQGMO.MatchOptions>	MQMO_MATCH_MSG_ID + MQMO_MATCH_CORREL_ID
9	MQGMO.DFLT<MQGMO.GroupStatus>	MQGS_NOT_IN_GROUP
10	MQGMO.DFLT<MQGMO.SegmentStatus>	MQSS_NOT_A_SEGMENT
11	MQGMO.DFLT<MQGMO.Segmentation>	MQSEG_INHIBITED
12	MQGMO.DFLT<MQGMO.Reserved1>	""
13	MQGMO.DFLT<MQGMO.MsgToken>	MQBYTE16
14	MQGMO.DFLT<MQGMO.Returned-Length>	MQRL_UNDEFINED

MQGMO.DFLT Dynamic Array Structure (Continued)

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.

MQPMO Dynamic Array Structure

Attribute	Identifier	Description
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.
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.UnknownDest-Count>	0
8	MQPMO.DFLT<MQPMO.InvalidDest-Count>	0
9	MQPMO.DFLT<MQPMO.ResolvedQName>	""
10	MQPMO.DFLT<MQPMO.ResolvedQMgrName>	""
11	MQPMO.DFLT<MQPMO.RecsPresent>	0
12	MQPMO.DFLT<MQPMO.PutMsgRecFields>	MQPMRF_NONE
13	MQPMO.DFLT<MQPMO.PutMsgRecOffset>	0

MQGPO.DFLT Dynamic Array Structure

Attribute	Identifier	Description
14	MQPMO.DFLT<MQPMO.ResponseRecOffset>	0
15	MQPMO.DFLT<MQPMO.PutMsgRecPtr>	MQPTR
16	MQPMO.DFLT<MQPMO.ResponseRecPtr>	MQPTR

MQGPO.DFLT Dynamic Array Structure (Continued)

CallMQI Error Codes

The following error codes represent errors 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
END
```

This table describes the CallMQI error codes.

	Identifier	Description
100	U2MQI_ERR_MQUNAVAILABLE	MQI libraries not available
101	U2MQI_ERR_UNKNOWN	Unknown error
102	U2MQI_ERR_NOBINDIR	UVBIN/UDTBIN env variable not found
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_PIPEREADFROMMQI	Error reading pipe from MQI process
107	U2MQI_ERR_PIPEWRITETOU2	Error writing pipe to UV process
108	U2MQI_ERR_PIPEREADFROMU2	Error reading pipe from UV process
109	U2MQI_ERR_EXEC	Error during exec of MQI process
110	U2MQI_ERR_INVALIDFORMAT	Variable does not match required format
111	U2MQI_ERR_NOT_HANDLE	Variable not of correct type

CallMQI Error Codes

	Identifier	Description
112	U2MQI_ERR_NULL_HANDLE	Uninitialized handle
113	U2MQI_ERR_INVALID_HANDLE	Handle has been closed
114	U2MQI_ERR_UNKNOWN_HANDLE	Unexpected handle value reported
115	U2MQI_ERR_SESSION_IN_USE	An active MQI connection already exists
116	U2MQI_ERR_CREATE_HANDLE	Error creating handle
117	U2MQI_ERR_DL_OPEN	Error opening MQI library
	U2MQI_ERR_DL_FUNC	Error calling function in MQI library

CallMQI Error Codes (Continued)

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_PIPEREADFROMMQI	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 req'd 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.
2006	MQRC_CHAR_ATTR_LENGTH_ERROR	The length of the character attributes are invalid.

U2 WebSphere MQ API Reason Codes (Continued)

Reason Code Number	Reason Code Name	Description
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.
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.

U2 WebSphere MQ API Reason Codes (Continued)

Reason Code Number	Reason Code Name	Description
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.
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.

U2 WebSphere MQ API Reason Codes (Continued)

Reason Code Number	Reason Code Name	Description
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.
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.

U2 WebSphere MQ API Reason Codes (Continued)

Reason Code Number	Reason Code Name	Description
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.
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.

U2 WebSphere MQ API Reason Codes (Continued)

Reason Code Number	Reason Code Name	Description
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.
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_EXT option.

U2 WebSphere MQ API Reason Codes (Continued)

Reason Code Number	Reason Code Name	Description
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.
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.

U2 WebSphere MQ API Reason Codes (Continued)

Reason Code Number	Reason Code Name	Description
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_CCSDID_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.
2114	MQRC_SOURCE_FLOAT_ENC_ERROR	The message's encoding specified an unrecognized floating-point encoding.
2115	MQRC_TARGET_CCSDID_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.

U2 WebSphere MQ API Reason Codes (Continued)

Reason Code Number	Reason Code Name	Description
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.
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.

U2 WebSphere MQ API Reason Codes (Continued)

Reason Code Number	Reason Code Name	Description
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_FAILED	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.
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.

U2 WebSphere MQ API Reason Codes (Continued)

Reason Code Number	Reason Code Name	Description
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.
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.

U2 WebSphere MQ API Reason Codes (Continued)

Reason Code Number	Reason Code Name	Description
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.
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.

U2 WebSphere MQ API Reason Codes (Continued)

Reason Code Number	Reason Code Name	Description
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.
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.

U2 WebSphere MQ API Reason Codes (Continued)

Reason Code Number	Reason Code Name	Description
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.
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_CCSDS	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_MSG_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.

U2 WebSphere MQ API Reason Codes (Continued)

Reason Code Number	Reason Code Name	Description
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.
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.

U2 WebSphere MQ API Reason Codes (Continued)

Reason Code Number	Reason Code Name	Description
2269	MQRC_CLUSTER_RESOURCE_ERROR	A cluster resource error occurred.
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_CRYPTTO_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_CFGFR_ERROR	Invalid MQCFGR structure in message data.
2420	MQRC_EPH_ERROR	The MQEPH structure is invalid.

U2 WebSphere MQ API Reason Codes (Continued)

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>.

MQSeries Primer

MQSeries Application Programming Guide

MQSeries Application Programming Reference

MQSeries Clients

Non-Root uvadm

Extended uvadm	5-2
Overview Of UniVerse Installation on UNIX	5-3
Login ID	5-3
The <i>uv.load</i> Script	5-3
The <i>uv_upgrade</i> Command	5-7
Initial Installation of UniVerse	5-8
Upgrading an Existing UniVerse System as uvadm	5-14
Examining the Load and Installation Scripts	5-22
<i>uv.load</i> Options	5-22
System Administration Menus	5-23

Extended uvadm

This section describes the steps you must complete to install UniVerse as *uvadm*, either as an initial installation or an upgrade installation. Beginning at this release, *uvadm* supports the following UniVerse commands:

- cleanup_lock
- convchar
- convencfile
- encman
- ld_nls_map
- list_readu (LIST.READU)
- load_shm_cat
- lock_maint
- master
- message
- modify_shm
- resize
- udpaccount
- unlock (UNLOCK)
- uvbackup/uvrestore
- uvdrlogd
- uvdrrepdm
- uvfixfile
- uvlictool
- uvregen
- uvtic
- uvtime

Overview Of UniVerse Installation on UNIX

UniVerse is installed on an existing UNIX system in directories you specify during the installation process. These directories are:

- *uv* – The UniVerse home directory
- *unishared* – The UniVerse/UniData shared directory

The examples in this manual use the default directories of */usr/ibm/uv* and */usr/ibm/unishared*. During the installation process, you can substitute the paths you choose for these directories.



Warning: Do not use symbolic links when specifying your *uvhome* and *unishared* directories. If you use symbolic links, the UniVerse license authorization routines fail.

Login ID

To install UniVerse, you must log on to your system as either *root* or *uvadm*.

If you log on as *root*, all UniVerse home account files and directories are owned and administered by *root*.

If you log on as *uvadm*, or as a *uvadm* group user, all UniVerse home account files and directories are owned and administered by *uvadm*. If you log on as *uvadm* to install UniVerse, you must have write permissions on the root directory (*/*) and the directories where you install the UniVerse home directory and the UniVerse/UniData shared directory.

The *uv.load* Script

Execute the UniVerse installation while you are logged on as *root*, *uvadm*, or a *uvadm* group user. Use the *cpio* or *tar* command to load a short installation script called *uv.load*, and then execute the *uv.load* script to load the files from the installation media.

To ensure a valid installation, the *uv.load* script performs the following actions:

- Stops any active UniVerse spooler
- Removes the shared memory segments used by UniVerse

The spooler must be stopped before you install UniVerse because the restoration cannot overwrite an active file. *uv.load* then loads the files from the installation media into the directories you specify. Depending on the type of installation you are performing, the script then performs the following actions:

- If you log on as *root* and are installing UniVerse for the first time, *uv.load* automatically executes the *uv.install* script.
- If you log on as *uvadm* and are installing UniVerse for the first time, you exit the *uv.load* script, then manually execute the *uv.install* script.
- If you log on as *uvadm*, or as a *uvadm* group user, and are upgrading an existing UniVerse system, you must use *uv_upgrade* to run *uv.load*, which then automatically executes the *uv.install* script.

UniVerse checks the files loaded from the installation media to make sure they have a checksum that matches the checksum on the installation media. If the checksums do not match, UniVerse indicates that the files were not loaded correctly from the installation media, and instructs you to reload the groups that were in error.



Note: *If you have not been able to reload the groups that are in error, contact your maintenance vendor for assistance.*

The *uv.load* script performs the following actions:

- Creates the UniVerse directory structure and grants proper permissions. The following table describes the default UniVerse directory structure.

Path	Description
<i>/usr/uv</i>	The UniVerse home directory.
<i>/usr/uv/bin</i>	Contains the executables that make up the UniVerse product.
<i>/usr/uv/sample</i>	Contains various prototype files.
<i>/usr/uv/NEWACC</i>	Contains the templates for the supported flavors of the VOC file.
<i>/usr/uv/terminfo</i>	Contains the UniVerse terminal characteristics database.
<i>/usr/uv/catdir</i>	The system catalog.
<i>/usr/unishared</i>	Contains subdirectories and files shared by UniVerse and UniData systems.

UniVerse Directory Structure

In addition, the installation procedure loads several UniVerse type 1 files or multiple data files, which are implemented as UNIX directories in the UniVerse home directory (for example, BP, B.P.O, APP.PROGS, and APP.PROGS.O.)

- Initializes the shared memory tables used by UniVerse, and modifies the UNIX initialization script to execute the uv.rc file, which performs UniVerse initialization.
- Installs the UniVerse spooler and executes the spooler daemon. See *Administering UniVerse* for a more information about the UniVerse spooler.



- Compiles the UniVerse terminal definitions. You should always install the new definitions unless you have modified the supplied definitions and want to preserve your changes.

Installing the UniVerse terminal definitions updates only the UniVerse-specific characteristics kept in `/usr/ibm/uv/terminfo`. If you want to update the non-UniVerse-specific characteristics in `/usr/lib/terminfo`, you must manually invoke the `uvtic` command with the `-a` option.

Warning: *If this is an initial installation, the UniVerse terminal definitions are always installed. You should make sure the UNIX terminal definitions exist. If the UNIX version does not exist, use the `-a` option with the `uvtic` command to create it.*

- Creates the SQL catalog if it does not already exist.
- Initializes the UniVerse catalog space (for initial installation only) and catalogs a number of subroutines. For more information about catalog space, see *Administering UniVerse*.
- Copies the sample shell initialization file, `.profile`, from the UniVerse sample directory to the UniVerse home directory. The `.profile` file contains paths for system commands, default protection for created files, and characteristics of the login terminal.

You must activate the UniVerse license or upgrade by entering the authorization code supplied by your vendor. To activate the license, you must log on as `root`, `uvadm`, or a `uvadm group user`, enter the UV account, then enter the authorization information in the License Activation screen.

After you authorize UniVerse, you can then add other accounts to be used in a UniVerse or UNIX environment, or perform administration of peripherals, such as spooler devices. To perform these tasks, use the UniAdmin client application or the **System Administration** menus. For detailed information, refer to *Administering UniVerse*.



Note: *When you log on to the UV account, you are in the System Administration menu system. If you exit the menus, you can reenter them by using the `LOGIN` command. To exit the **System Administration** menus, press `ESC`.*

The *uv_upgrade* Command

Execute the *uv_upgrade* command while you are logged in as *uvadm*, or as a member of the *uvadm* group, to upgrade UniVerse installations. The *uv_upgrade* command is used to ensure that *uvadm* users have the appropriate administrative privileges necessary to upgrade existing UniVerse installations.

The *./uv_upgrade ./uv.load* command invokes the *uv.load* script, which loads the files from the installation media. Users who are logged on as *uvadm* or as a member of the *uvadm* group must use the *uv_upgrade* command to upgrade existing UniVerse installations.



Note: *If UniVerse 11.1, or later, is installed, root, uvadm, or uvadm group users can run uv_upgrade. If UniVerse 10.3 or earlier is installed, only the root or uvadm user can run uv_upgrade.*



Initial Installation of UniVerse

Complete the following steps to install UniVerse for the first time on your system.

Note: Before you start the installation, make sure your kernel parameters are adequate to meet your UniVerse environment requirements. Also make sure you have write permissions on the root directory (/) and on the directories where you plan to install the UniVerse home directory and the UniVerse/UniData shared directory.

1. Log in as *uvadm*, and change directories to any directory to which you have write permissions, as shown in the following example:

```
$cd /tmp
```

2. Make sure there are no users logged in to the system before you begin the installation.

Insert the release media into the appropriate device. If you are unfamiliar with the loading procedure, refer to your hardware instruction manual.

You can install UniVerse on the following platforms:

- ? AIX IBM RISC System/6000
- ? HP-UX Itanium/Integrity Server
- ? Sun SOLARIS Ultra-Sparc
- ? Sun SOLARIS x86
- ? Redhat LINUX
- ? SuSE LINUX

3. Use the following *uv.load* command to install from CD-ROM:

```
cpio -ivcBdum uv.load < /cdrom/STARTUP
```

4. Execute the *uv.load* script while the installation media is still in the device, as shown in the following example:

```
$ ./uv.load
```

uv.load displays the current installation settings, as shown in the following example:

The current settings of the available options are:

```
UniVerse installer      : uvadm
UniVerse administrator : uvadm  uid=214 gid=200
```

- 1) UniVerse home directory: /usr/ibm/uv
(currently: Not Installed.)
- 2) UniVerse-UniData shared directory: /usr/ibm/unishared
(currently: Not Installed.)

```
3) Compile terminfo definitions:      true
4) Rewind tape name                  /dev/rmt/0m
5) No-rewind tape name                /dev/rmt/0mn
6) Long File Names                   OFF
```

Enter a field number to change, q to abort installation, or press <Return> to begin installation of UniVerse:

5. Enter the number of the value you want to change. If you choose 1, you are prompted for the UniVerse home directory. If you choose 2, you are prompted for the shared directory name. You cannot change option 3. If you choose 4 or 5, you are prompted for a new device pathname. If you choose 6, its value is toggled.

Option 6 sets the default file creation characteristics (see *LONGNAMES* in *Administering UniVerse*). You can toggle option 6 between *LONGNAMES OFF*, *LONGNAMES ON*, and *LONGNAMES ON NEWACC*. The default is *LONGNAMES OFF*, meaning *LONGNAMES* is not active.

You can also enter q at this point to quit the installation procedure.

If the values displayed are acceptable, press ENTER to start installing UniVerse.

6. If an SQL catalog does not exist and the *uvsql* user name does not exist in the */etc/passwd* file, the following message will appear:

```
The user 'uvsql' does not exist. This user is the default owner of the SQL
catalog. Would you like to:
```

- 1). Continue with the installation, making 'uvadm' the default owner of the SQL catalog.
- 2). Suspend the installation so that you can create the 'uvsql' user.
- 3). Stop the installation.

Your choice (Default action is 1):

Choose 1 to make *uvadm* the owner of the SQL catalog. Choose 2 to shell out of the installation procedure and create the *uvsql* user in the */etc/passwd* file. When you exit from doing this, you return to the installation procedure.

7. If you are installing UniVerse from media other than CD-ROM, the following screen appears:

The following UniVerse groups are available for installation:

MAIN OBJ DEVELOP DOC FILESIZE PORTING UCI IC UOJ UVODBC

Which groups do you wish to install (press <Return> for all groups)?

The following table describes the groups available for installation on your system:

Group	Description
MAIN	This group is the standard UniVerse environment. It contains the files and programs necessary for running UniVerse. The MAIN group must be installed on your system for UniVerse to run.
OBJ	The OBJ group contains object files and archive files that are used to install and link the UniVerse GCI product.
DEVELOP	The DEVELOP group contains tools for application development, including the BASIC compiler, catalog manipulation commands, and debugging tools. It is not necessary for this group to be loaded to run BASIC programs or cataloged programs.
DOC	This group contains the online help facility for UniVerse and BASIC. Also included is the demonstration account. This group is not necessary for UniVerse to run.
FILESIZE	This group contains tools for verifying and adjusting the type, modulo, and separation of UniVerse files. This group is not necessary for UniVerse to run.
PORTING	This group contains tools for porting applications from Pick, Reality, and Prime INFORMATION systems to your UniVerse system. This group is not necessary for UniVerse to run.
UCI	This group contains the files and programs needed to run UCI. This group is not necessary for UniVerse to run.
IC	This group contains the files and programs needed to run InterCall. This group is not necessary for UniVerse to run.
UOJ	This group contains the files and programs needed to run UniObjects for Java. This group is not necessary for UniVerse to run.
UVODBC	This group contains the files and programs needed to run UniVerse ODBC. This group is not necessary for UniVerse to run.

UniVerse Groups

If you want to install all the groups, press ENTER. If you do not want to install all the groups, enter the names of the groups you want to install, separated by spaces. You must enter the groups exactly as they appear at the prompt, and in the same order. When you have entered all the groups, press ENTER to begin the installation of the groups.

Note: If a group fails to load correctly, UniVerse displays a message indicating that the checksums do not verify for the listed groups. You should repeat the installation procedure for those groups that do not verify. When you repeat the installation procedure, you should verify that the UniVerse home directory is correct. You can repeat the installation procedure later to add groups that you do not install now.

8. After the installation process install of the all selected groups, the following message appears:

```
This initial installation of UniVerse must now be completed by logging in as 'root' and executing the script '/usr/ibm/uv/uv.install'.
```

```
UniVerse will remain in an inoperable state until this script has been executed.
```

Perform the following steps:

- Log in as root:

```
$ su
$ password:
```

- Change directory to the UniVerse home directory:

```
# cd /usr/uv
```

- Execute the *uv.install* script:

```
# ./uv.install
```

9. After the *uv.install* completes successfully, the **Upgrade UniVerse License** screen appears. Enter the license activation information as requested by the prompts. The information you must enter is:

- Serial number
- Maximum number of local users
- Expiration date, or press ENTER for the default
- Package list
- Number of Device Licenses for which you are authorized

Press ENTER. The licensing process displays a configuration code.

10. Remember the Configuration Code the licensing process displays. The configuration code is of the following format:
CCTM5-ZZ3UZ-QFZ7Z-ZZZW6-Z3ZZ4-XBKLC-UZTI9
The licensing process prompts for **Local Authorization Code**.
11. To obtain your authorization code, go to:
<https://u2tc.rocketsoftware.com>
Click **Authorize Products**. Follow the instructions on the website to obtain your authorization code.
12. Once you have your authorization code, go back to the **Upgrade UniVerse License** window and enter the authorization code in the **Local Authorization Code** field.
13. When the authorization completes successfully, the following message appears:

```
UniVerse licensing is complete. Please shut down and
restart uniVerse.
Use the uniVerse system administration menu
to create additional uniVerse accounts.
```

Shut down and restart UniVerse:
 - Enter the following command to shut down UniVerse:

```
$ bin/uv -admin -stop
```
 - Enter the following command to restart UniVerse:

```
$ bin/uv -admin -start
```
14. Log in to the UV account. The **System Administration** menu appears. This menu system allows you to perform tasks such as adding users or setting up your spooler. See *Administering UniVerse* for complete instructions. To exit the menus, press **ESC** until you get to the UniVerse prompt:

```
>
```

To reenter the **System Administration** menu system, use the command:

```
>LOGIN
```
15. To exit the UniVerse environment, enter **Q** at the UniVerse prompt. A standard shell prompt appears, as shown in the following example.

```
>Q
$
```
16. If you installed UniVerse from CD-ROM, remove the CD-ROM from the device by entering the following command:

```
$ umount /mnt
```



17. When you install UniVerse on your system for the first time, you must add the UniRPC daemon's port to the /etc/services file.

Add the following line to the /etc/services file:

uvrpc 31438/tcp # uvrpc port

Note: You can check to see if the the entry already exists in the /etc/services file by executing the following command:

'cat /etc/services |grep 31438'

Once the port information is added to the /etc/services file, you can start the RPC Service from the UniVerse System Administration Menu in the UniVerse account.

To do this, select **Package > RPC Administration > Start the rpc daemon**. A dialog box opens and asks for filename information. You can either add a new filename or accept the default. Add the correct filename information and then click **Enter**. A pop-up window opens and asks if you want to start the daemon. Click **Yes**.

The UniRPC daemon will now start automatically when UniVerse restarts.

Upgrading an Existing UniVerse System as *uvadm*

Complete the following steps to upgrade an existing UniVerse system from the system console. Make the entries shown in bold type, and press **ENTER** after each entry.



Note: Before you start the installation, make sure your kernel parameters are adequate to meet your UniVerse environment requirements. Also make sure you have write permissions on the root directory (/) and on the directories where you plan to install the UniVerse home directory and the UniVerse/UniData shared directory.

1. If the existing UniVerse system is owned by *uvadm*, enter the following command at the login prompt:

```
$ login: uvadm
uvadm's password:
```

2. Change directories to any directory to which you have write permissions, as shown in the following example:

```
$ cd /tmp
```

3. Make sure there are no users logged in to the system before you begin the installation.

Insert the release media into the appropriate device. If you are unfamiliar with the loading procedure, refer to your hardware instruction manual.

You can install UniVerse on the following platforms:

- ? AIX IBM RISC System/6000
- ? HP-UX Itanium/Integrity Server
- ? Sun SOLARIS Ultra-Sparc
- ? Sun SOLARIS x86
- ? Redhat LINUX
- ? SuSE LINUX

4. Use the following command to install from CD-ROM:

```
cpio -ivcBdum uv.load uv_upgrade < /cdrom/
```

5. Execute the *./uv_upgrade* command to invoke the *uv.load* script, with the release media still in the device:

```
$ ./uv_upgrade
```

6. A screen similar to the following appears:

```
UniVerse Upgrade Procedure
=====
```

```
The current upgrade is being done as 'uvadm'. The
existing installed uniVerse (at /usr/ibm/uv)
is being administered by the user 'root'.
```

```
Choose one of the following below:
```

- 1) Keep 'root' as the owner and administrator of uniVerse.
The current installation continues uninterrupted.
- 2) Make 'uvadm' the new owner and administrator of uniVerse.
The current installation continues uninterrupted.
- 3) Stop the installation.

```
Your choice (Default action is 1):
```

Choose 2 to install UniVerse and make *uvadm* the owner and administrator of UniVerse.

7. *uv.load* displays the current installation settings:

```
The current settings of the available options are:
```

```
UniVerse installer      : uvadm
UniVerse administrator : uvadm uid=214 gid=200
```

- 1) UniVerse home directory: /usr/ibm/uv
- 2) UniVerse-UniData shared directory: /usr/ibm/unishared
- 3) Compile terminfo definitions: true

```

4) Rewind tape name           /dev/rmt/0m
5) No-rewind tape name       /dev/rmt/0mn
6) Long File Names           OFF

```

Enter a field number to change, q to abort installation, or press <Return> to begin installation of UniVerse:

The UniVerse installer will appear either as root or uvadm, depending on how you logged in at step 1.

Enter the number of the value you want to change. If you choose 1, you are prompted for the UniVerse home directory. If you choose 2, you are prompted for the shared directory name. If you choose 4 or 5, you are prompted for a new device path. You cannot change 6.

You can also enter **q** at this point to quit the installation procedure.

If the values displayed are acceptable, press **ENTER** to start installing UniVerse.

If you choose 3, a list similar to the following is displayed:

Compiling the uniVerse terminfo definitions will overlay the descriptions for terminals listed below:

```

4410          at386          pt200          viewpoint
AT386         att3b1        qt102          viewpoint60
M=           att4410        regent20       vp
Mu           av             regent25       vp60
a210         dialup        regent40       vp60:regent40
a210:adm5    dumb          regent60       vt100
a210:hzl410  fenix         s4             vt100-am
a210:hzl500  gt             sun            vwpt
a210:qt102   hft             sun-cmd        vwpt60
a210:regent25 hzl410         sun-w          wy200
a210:tvi910  hzl500         tvi910        wy200-w
a210:tvi910+ ibm5151        tvi910+       wy50
a210:tvi920  icl6404        tvi920        wy50:hzl500
a210:tvi925  network        tvi925        wy50:tvi910
a210:vp      pc7300         tvi955        wy50:tvi920
adm5         performer    unixpc        wy50:tvi925
at           pt             unknown       wy50:vp

```

Enter YES to compile and install terminfo.src:

Enter **Y** to compile and install *terminfo.src*. Enter **N** to change option 3 from true to false.

- 8.** If you do not already have an SQL catalog and there is no user name *uvsql* in the */etc/passwd* file, you see the following message:

The user 'uvsql' does not exist. This user is the default owner of the SQL catalog. Would you like to:

- 1). Continue with the installation, making 'uvadm' the default owner of the SQL catalog.
- 2). Suspend the installation so that you can create the 'uvsql' user.
- 3). Stop the installation.

Your choice (Default action is 1):

Choose 1 to make *uvadm* the owner of the SQL catalog. Choose 2 to shell out of the installation procedure and create the *uvsql* user in the */etc/passwd* file. When you exit from doing this, you return to the installation procedure.

9. If you are installing UniVerse from any media other than CD-ROM, the following prompt appears:

The following uniVerse groups are available for installation:

MAIN OBJ DEVELOP DOC FILESIZE PORTING UCI IC UOJ UVODBC

Which groups do you wish to install (press <Return> for all groups) ?

Choose which groups you want to install on your system. The following groups are available for installation on your system:

Group	Description
MAIN	This group is the standard UniVerse environment. It contains the files and programs necessary for running UniVerse. The MAIN group must be installed on your system for UniVerse to run.
OBJ	This group contains object files and archive files that are used to install and link the UniVerse optional product GCI.
DEVELOP	This group contains tools for application development including the BASIC compiler, catalog manipulation commands, and debugging tools. It is not necessary for this group to be loaded to run BASIC programs or cataloged programs.
DOC	This group contains the online help facility for UniVerse and BASIC. Also included is the demonstration account. This group is not necessary for UniVerse to run.
FILESIZE	This group contains tools for verifying and adjusting the type, modulo, and separation of UniVerse files. This group is not necessary for UniVerse to run.
PORTING	This group contains tools for porting applications from Pick, Reality, and Prime INFORMATION systems to your UniVerse system. This group is not necessary for UniVerse to run.
UCI	This group contains the files and programs needed to run UCI. This group is not necessary for UniVerse to run.

UniVerse Groups

Group	Description
IC	This group contains the files and programs needed to run InterCall. This group is not necessary for UniVerse to run.
UOJ	This group contains the files and programs needed to run UniObjects for Java. This group is not necessary for UniVerse to run.
UVODBC	This group contains the files and programs needed to run UniVerse ODBC. This group is not necessary for UniVerse to run.

UniVerse Groups (Continued)

If you want to install all the groups, press ENTER at the prompt, otherwise enter the names of the groups you want to install, separated by spaces. You must enter the groups exactly as they appear at the prompt and in the same order. When you have entered all the groups, press Return to begin the installation of the groups.

Note: If a group fails to load correctly, a message is displayed indicating that the checksums do not verify for the listed groups. You should repeat the installation procedure for those groups that do not verify. When you repeat the installation procedure, you should verify that the UniVerse home directory is correct. You can repeat the installation procedure later to add groups that you do not install now.

10. You may see the following message:


```
Unable to get disk shared memory segment: Invalid argument
```

 If this happens, shut down and restart UniVerse. Then proceed to step 15.
11. After the installation process installs all of the groups, the **Upgrade UniVerse License** screen appears. Enter the license activation information as requested by the prompts. The information you must enter is:
 - Serial number
 - Maximum number of local users
 - Expiration date, or press ENTER for the default
 - Package list
 - Number of Device Licenses for which you are authorized
 Press ENTER. The licensing process displays a configuration code.

12. Remember the Configuration Code the licensing process displays. The configuration code is of the following format:
CCTM5-ZZ3UZ-QFZ7Z-ZZZW6-Z3ZZ4-XBKLC-UZTI9
The licensing process prompts for **Local Authorization Code**.
13. To obtain your authorization code, go to:
<https://u2tc.rocketsoftware.com>
Click **Authorize Products**. Follow the instructions on the website to obtain your authorization code.
14. Once you have your authorization code, go back to the **Upgrade UniVerse License** window and enter the authorization code in the **Local Authorization Code** field.
15. When the authorization completes successfully, the following message appears:

```
UniVerse licensing is complete. Please shut down and
restart    uniVerse.
```

```
Use the uniVerse system administration menu
to create additional uniVerse accounts.
```

Shut down and restart UniVerse using the following steps:
Change to the UniVerse home directory, as shown in the following example:

```
$ cd /usr/ibm/uv
```

Enter the following command to shut down UniVerse:

```
$ bin/uv -admin -stop
```

Enter the following command to restart UniVerse:

```
$ bin/uv -admin -start
```
16. Log in to the UV account. The **UniVerse System Administration** menu appears. This menu system lets you perform tasks such as adding users or setting up your spooler. See *Administering UniVerse* for complete instructions. To exit the menus, press ESC until you get to the UniVerse prompt:

```
>
```

To reenter the **System Administration** menu system, use the command:

```
>LOGIN
```
17. To exit the UniVerse environment, enter Q at the UniVerse prompt. A standard shell prompt appears, as shown in the following example.

```
>Q
$
```



18. Each user account VOC file must be updated to the current UniVerse release level. Do this by invoking UniVerse in each user account. When you do this, the following prompt appears:

```
Your VOC is out of date. Update to current release (Y/N)?
```

Entering Y at the prompt updates the VOC to the current release level.

Note: The UniVerse LOGTO command does not check the release level of the VOC file when used to enter a UniVerse account. The check is done only when directly invoking UniVerse in a user account. If your application uses LOGTO, you must verify that all user accounts are updated to the current release level before running the application.

19. If you installed UniVerse from CD-ROM, remove the CD-ROM from the drive by entering the following command:

```
$ umount /mnt
```

20. When you install UniVerse on your system for the first time, you must add the UniRPC daemon's port to the /etc/services file.

Add the following line to the /etc/services file:

```
uvrpc 31438/tcp # uvrpc port
```

Note: You can check to see if the entry already exists in the /etc/services file by executing the following command:

```
'cat /etc/services |grep 31438'
```

Once the port information is added to the /etc/services file, you can start the RPC Service from the UniVerse System Administration Menu in the UniVerse account.

To do this, select **Package > RPC Administration > Start the rpc daemon**. A dialog box opens and asks for filename information. You can either add a new filename or accept the default. Add the correct filename information and then click **Enter**. A pop-up window opens and asks if you want to start the daemon. Click **Yes**.

The UniRPC daemon will now start automatically when UniVerse restarts.



Examining the Load and Installation Scripts

You can examine the UniVerse load or installation script using the *vi* editor. These scripts are in the UniVerse home directory, by default */usr/ibm/uv*, under the names *uv.load* and *uv.install*. You must have root privileges to view these files.

uv.load Options

You can use the options to the *uv.load* command to change the installation process. The syntax of the *uv.load* command is:

```
uv.load {-defaults} {-longnames} {-nochecksum} {-nocpio}  
        {uvhome}
```

The following table describes each parameter of the syntax:

Option	Description
<code>-defaults</code>	Specifies that all defaults are to be used—no prompting by the script. This allows for automatic installation.
<code>-longnames</code>	Forces LONGNAMES to ON NEWACC.
<code>-nochecksum</code>	Specifies to skip the checksum step. This should be done only after consulting with U2 support. If files are really damaged, the installed UniVerse will not execute correctly, with possible damage to files.
<code>-nocpio</code>	Specifies to skip the physical read of the installation media. This option can be used to restart the installation after files have been loaded.
<i>uvhome</i>	Specifies the UniVerse home directory if it is different from either the installed UniVerse or <i>/usr/ibm/uv</i> .

uv.load Options

System Administration Menus

The System Administration menus and data entry screens look and work the same way as Motif menus. For a complete description of these menus, see Appendix A of *Administering UniVerse*.

JPA

Installation Files	6-5
Prerequisites	6-5
Installation	6-5
Introduction	6-6
Features	6-7
U2JPA and Object-Relational Mapping.	6-8
Virtual Fields in U2JPA	6-10
Fetch Types	6-10
Restrictions	6-11
Relationship Mapping	6-11
Using U2JPA	6-16
Entity Managers	6-16
Persistence Units	6-16
Transaction Support	6-18
Connection Pooling	6-19
Connection Pool Size	6-19
Activating Connection Pooling	6-19
SSL Connection and NLS Support	6-20
Managing Entities	6-21
U2JPA Methods	6-23
U2JPA Query Support	6-28
Query Parameters	6-29
Query Result Paging	6-29
Subroutine Calls	6-32
Named Query	6-33
U2JPA Eclipse Plug-in	6-35
The U2JPA Wizard.	6-35

The Persistence.xml Configuration File	6-36
Step-By-Step Guide to Using U2JPA	6-37
Create a New U2JPA Project in Eclipse	6-37
Create a New User Library	6-41
Use the U2 Resource View in Eclipse	6-43
Create a U2JPA package	6-45
Run the Program	6-51

This chapter describes the U2 Java Persistence API.

Installation Files

The U2 Java Persistence API (U2JPA) is one of several APIs in the UniDK (Uni Development Kit). The UniDK is installed using the standard Microsoft Windows installation procedure. The following UniDK files are used for U2JPA development:

File	Description
u2jpa-1.0.0.jar	The main U2JPA class files for development and run time
u2jpa.plugin-1.0.0.zip	The files used in the U2JPA Eclipse plug-in.

UniDK Files for U2 J PA Development

Prerequisites

U2JPA requires that you have the following tools installed:

- Eclipse Ganymede for Java EE, or later.
- Apache OpenJPA 1.3.0 SNAPSHOT package.

Installation

The U2JPA package is installed, by default, in the following directory:

C:\U2\UniDK\U2JPA

You need to install the u2jpa plug-in directly into your Eclipse directory:

To install the u2jpa plug-in:

1. Navigate to the directory where U2JPA is installed.
2. Open the **u2jpa** folder, and then select the **u2jpa.plugin-1.0.0.zip** file.
3. Extract the u2jpa.plugin-1.0.0.zip file into your Eclipse directory.
The file contains two folders: features and plugins.
4. You can now begin working with U2JPA in Eclipse. Refer to “[Step-By-Step Guide to Using U2JPA](#),” for step-by-step instructions.

Introduction

The Java Persistence API is a Java language framework used to bridge the gap between the object model in which most modern applications are written, and the relational model in which most of the data applications reside. The goal of JPA is to provide a simple, powerful set of standards that can be used to map a Java object to records in a relational database. This object-relational mapping (ORM) allows the developer to work directly with an entity, not the database structure.

The U2 Java Persistence API (U2JPA) is a JPA provider that handles U2 nested data natively, and thus does not require normalization of U2 multivalued data. It is based on OpenJPA, which is the Apache open source implementation of JPA.

Unlike other JPA implementations that are based on JDBC, U2JPA is built upon UniObjects for Java as its data access layer. This allows developers to utilize all of the functionality available in UOJ, combined with the additional benefits of the JPA framework. U2JPA users can, for example, take advantage of device licensing and connection pooling, and can use SSL connections. U2JPA can also handle NLS for UniVerse and I18N for UniData.

JPA defines a set of metadata in the form of Java annotations or XML schema which can be used to designate Plain Old Java Objects (POJOs) as entities, and describe the relationships between them. Persistence is achieved through a Java persistence provider, such as U2JPA, which allows the developer more freedom.

Features

U2JPA features the following:

- The U2 Resource View plug-in connects to U2 servers so you can easily see U2 accounts, files, and cataloged programs.
- The U2JPA Eclipse plug-in provides a U2 Resource view in Eclipse that allows users to explore database files and accounts.
- In the U2 Resource view, you can use Eclipse's drag and drop functionality to create entity classes.
- U2JPA is standards-based, which frees the programmer to work directly with entities
- Based on OpenJPA
- Extends OpenJPA to provide JPA support for U2, using UniObjects for Java (UOJ)
- Compatible with many Java EE application servers, such as WebSphere, JBoss, Apache, and Geronimo
- Handles I18N for UniData and NLS for UniVerse
- Can use SSL connections

This chapter provides an overview of the Object-Relational Mapping of U2 data using U2JPA.

U2JPA and Object-Relational Mapping

JPA defines a set of persistence metadata in the form of Java annotations and XML schemas to map Java objects (entities) to relational data. U2JPA makes use of this mapping metadata to map Java objects to U2's multivalued data.



Note: At this release, the U2JPA Eclipse plug-in only supports the generation of entity classes using Java annotations. It does not support the generation of XML mapping files – orm.xml.

In U2JPA, each U2 data file is mapped to an entity class, with each of the single valued fields mapped to a data member of the entity class. Partial mapping of a data file is supported, which means that not all the fields in a data file must be mapped. This allows users to selectively map certain fields but leave other fields out of the entity class.

JPA requires that an entity class have an identifier member, which can either be in the form of a single field or of multiple fields. Because U2 always takes the field defined at location 0 as the primary key field, the @ID field, or a field with a LOC value = 0 in the dictionary, must always be included in an entity class and be annotated by @Id.

In the example below, a Java entity class named **Customer** is generated by the U2JPA Eclipse plug-in. The Customer class is mapped to the CUSTOMER data file using the JPA annotations @Entity, @Table, @Id, and @Column.

For information about JPA annotations, please refer to OpenJPA documentation at <http://openjpa.apache.org/documentation.html>.

```
@Entity
@Table(name="CUSTOMER")
public class Customer implements Serializable {
    @Id
    @Column(name="@ID")
    private String _id;

    @Column(name="NAME")
    private String name;

    @Column(name="ADDRESS")
    @PersistentCollection(fetch=FetchType.EAGER)
    private String[] address;

    public Customer(){};

    public String get_id() {return _id;}

    public void set_id(String _id) {this._id = _id;}

    public String getName() {return name;}

    public void setName(String name) {this.name = name;}

    public String[] getAddress() {
        return address;
    }

    public void setAddress(String[] address) {
        this.address = address;
    }
}
```

Note here that the CUSTOMER data file may have more fields in its dictionary than @ID, NAME, and ADDRESS. In this example, however, we choose to map only these three fields in the Customer entity class.

In this example, the ADDRESS column in the CUSTOMER data file is a non-associated multivalued field. It is mapped to a String[] type Java field, and the OpenJPA extended annotation @PersistentCollection is applied because the standard JPA does not support non-associated collections.

JPA defines an `@Basic` annotation that can be used on column mappings to specify the fetch type on a column. For example:

```
@Basic (fetch=FetchType.LAZY)
@Column(name="DetailedDescription")
String detailedDescription;
```

Virtual Fields in U2JPA

The U2 databases support virtual fields, which contain virtual information that is calculated or otherwise derived from other attributes, other files, or other information in the database. The result of a virtual attribute is information that does not literally exist in the data portion of a file.

U2JPA supports the mapping of virtual fields. Because virtual fields are not updatable, the U2JPA Eclipse plug-in does not create ‘set’ methods for them in the generated entity class, and the U2JPA provider simply ignores changes to virtual fields even if the users supply setters for virtual-field mapped members in the generate entity class.

Fetch Types

In JPA, metadata annotations often have a fetch property, which can take on one of two predefined values, known as fetch types: `LAZY` and `EAGER`. A `LAZY` fetch type indicates to the persistence provider that the annotated column may be loaded only when it is accessed. This tends to save the overhead of loading everything referenced in an entity all at once, thus boosting the performance. However, a `LAZY` fetch type is only an advisory type, meaning that persistence providers such as U2JPA can ignore it and always eager-load the column, which is what U2JPA does for column mappings.

In general, mapping a U2 data file to multiple entity classes in the same persistence context (the same persistence unit in the persistence.xml file) is not recommended. However, when a data file has a very large number of fields, this may be necessary to better organize the entity model and to improve performance. As long as these entity classes do not overlap each other, they should be safe to use without causing concurrency issues.

In U2JPA, unidirectional One-to-Many relationships are always eager-loaded, regardless of the fetch type specified for it. In Many-to-One and One-to-One relationships, U2JPA honors the fetchtype specified, although lazy-loading is preferred. For the same reasons, the U2JPA Eclipse plug-in always assigns FetchType.EAGER to unidirectional One-to-Many relationships and FetchType.LAZY to unidirectional Many-to-One relationships.

Restrictions

At this release, U2JPA:

- Does not support automatic identifier generation, and users must manually assign a unique ID to a new entity to be persisted to the database.
- Ignores lazy fetch type specified on columns.
- Supports only List and Array collection types. In the example above, instead of mapping ADDRESS to String [], we may choose to map it to List<String>.



***Note:** At this time, you must use the OpenJPA 1.3.0 SNAPSHOT version with U2JPA. Other versions of OpenJPA do not support mapping columns whose names contain a period '.', such as "FIRST.NAME".*

Relationship Mapping

Most entities need to be able to reference, or have relationships with, other entities. JPA supports mapping of the four types of relationships between entities:

- One-to-Many
- One-to-One
- Many-to-One
- Many-to-Many

Relationships can also be bidirectional or unidirectional. Unidirectional relationships require that only one entity in the relationship contain a reference to another entity. Bidirectional relationships require both entities to have references with each other.

Bidirectional relationships tend to make navigation between related entities easier, which in turn makes coding simpler. Unidirectional relationships, on the other hand, are easier to implement and simpler to manage for the persistence provider.



For example, if we build a bidirectional relationship between the Customer entity and the Order entity, we can directly navigate from a Customer entity to all of its Order entities, or from an Order entity to its Customer entity. In contrast, if there is a unidirectional relationship from the Customer entity to the Order entity, you cannot navigate directly from an Order entity to its Customer entity.

Note: U2JPA does not support Many-to-Many relationships or bidirectional relationships.

Unidirectional One-To-Many Relationships

The U2 multivalued databases have built-in, unidirectional One-to-Many relationship support. For example, the CUSTOMER record may contain a multivalued ORDER association, forming a natural unidirectional One-to-Many relationship between the CUSTOMER record and the ORDER association. U2JPA maps associations and sub-associations as distinct entities from the record-level entity.

In the example below, the CUSTOMER file is mapped to the Customer entity, while its ORDER association is mapped to the Order entity. They are associated by a unidirectional One-to-Many relationship mapping.

```
@Entity
@Table(name="CUSTOMER")
public class Customer implements Serializable {
    @Id
    @Column(name="@ID")
    private String _id;

    @OneToMany(fetch=FetchType.EAGER,cascade=CascadeType.ALL)
    private List<Order> orders;
    ...
}
@Entity
public class Order implements Serializable {
    @Id private String u2id;

    @Column(name="BUY_DATE")
    private Date buy_date;

    @Column(name="DISCOUNT")
    private java.math.BigDecimal discount;

    ... ..
}
```

Both the Customer entity class and the Order entity class in this example are generated by the U2 Eclipse JPA plug-in, which recognizes associations and sub-associations in the same data file, and then automatically generates a separate entity class for each of them and links them together through the One-to-Many relationships between a higher level entity and its sub-entities. For example, from a record-level entity to its association-level sub-entities, or from an association-level entity to its sub-association-level sub-entities. To guarantee data integrity at the MV or SMV level, if any of the D-type fields in an association or sub association are selected, the U2JPA Eclipse plug-in automatically selects all of them.

Cascade Types

A cascade type can be specified on a relationship. Cascade types are related to the operations that can be performed on an entity. A cascade type can be one or a combination of the following four types: PERSIST, REMOVE, REFRESH, and MERGE.

If a cascade type is put on a relationship, whenever the specified operation is performed on the entity, the persistence provider automatically cascades the operation to the entities linked by the relationship.

In the example shown below, the One-to-Many relationship between the Customer entity and the Order entity has a cascade type of CascadeType.ALL:

```
@OneToMany(fetch=FetchType.EAGER, cascade=CascadeType.ALL)
private List<Order> orders;
```

The CascadeType.All property combines the four cascade types. Using this property ensures that any of the four operations performed on a Customer entity are automatically carried out on the Order entities referenced in “List<Order> orders”. For example, if a Customer is removed, then all the Order entities that belong to it are removed as well.

U2JPA supports Collection, List, and Array types on the One-to-Many attribute.

To delete an entity on the many-side of a relationship, you must remove the entity from the one-side’s collection. For example, the remove statement below deletes the third order for the customer in the database:

```
customer.getOrders.remove (2)
```

Simply removing the Order entity itself by calling the EntityManager’s remove () method will not work.

Similarly, to add a new entity on the many-side of a relationship, you must add it to the one-side's collection. For example, you can add a new order to the customer record in the database, as shown:

```
customer.getOrders.add (newOrder)
```

Persisting the new Order entity itself by calling the EntityManager's persist() method will not work.



Note: U2JPA does not support the use of join-tables to build unidirectional One-to-Many relationships.

Unidirectional Many-to-One Relationship

U2JPA supports unidirectional Many-to-One relationships through the foreign key mechanism, in which one data file stores the primary key values of another data file in a foreign key field.

If the foreign key is multivalued or multi-subvalued, it has to belong to an association or a sub-association. In this case, the Many-to-One relationship is built at the association or sub-association level with the foreign data file.

In the example below, a Many-to-One relationship is mapped, in which the Order entity has a Many-to-One relationship with the Product entity. The foreign key column is “PRODID” as specified in `@JoinColumn(name="PRODID")`. The relationship field ‘product’ is loaded only when it is accessed, as defined by the “fetch=FetchType.LAZY” attribute in the `@ManyToOne` annotation.

```
@Entity
public class Order implements Serializable {
    @Id private String u2id;

    @Column(name="BUY_DATE")
    private Date buy_date;

    @Column(name="DISCOUNT")
    private java.math.BigDecimal discount;

    @ManyToOne(fetch=FetchType.LAZY)
    @JoinColumn(name="PRODID")
    private Product product;

    ... ..
}

@Entity
@Table(name="PRODUCTS")
public class Product implements Serializable {

    @Id
    @Column(name="@ID")
    private String _id;

    @Column(name="DESCRIPTION")
    private String description;

    @Column(name="LIST")
    private int list;

    public Product(){};

    ... ..
}
```

The U2JPA Eclipse plug-in is used to generate both entities and the relationship mapping between them. See “[U2JPA Eclipse Plug-in.](#)”

Using U2JPA

In JPA, a persistence entity is a lightweight Java class that is mapped to a data table. Each entity is generally mapped to a specific row in the data table, and entities can be connected to each other in a one-to-many relationship through object-relational metadata, which is defined in an xml file. These entities are managed by an entity manager.

Entity Managers

An entity manager implements the JPA interface `EntityManager`, which provides methods to find, persist, update, remove, and query entities.

The `EntityManager` instance is always associated with a persistence unit. A persistence unit tells the `EntityManager` which entity classes are to be managed in an application.

Persistence Units

A persistence unit defines a set of entity classes representing the data contained within a single data store. In the U2JPA world, a persistence unit corresponds to a U2 database account on the server. The persistence unit is defined by creating a `persistence.xml` configuration file and providing the needed properties in the metadata. The `persistence.xml` file must reside under the `META-INF` directory in the Java source directory, or in the JAR file.

A persistence unit may also have a set of properties to specify vendor-specific settings. The following persistence.xml example defines a persistence unit that includes entities mapped to data files in the UniVerse HS.SALES account:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0"
  xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
  <persistence-unit name="com.rs.u2.u2jpa.demo">
    <provider>
      org.apache.openjpa.persistence.PersistenceProviderImpl
    </provider>

    <class>u2u.u2jpa.demo2.Product</class>
    <class>u2u.u2jpa.demo2.Customer</class>
    <class>u2u.u2jpa.demo2.Order</class>

    <properties>
      <property name="openjpa.ConnectionURL"
value="rsu2:localhost:uvcs:HS.SALES;pooling=false" />
      <property name="openjpa.ConnectionUserName" value="upix" />
      <property name="openjpa.ConnectionPassword"
value="upi456789" />
      <property name="openjpa.abstractstore.AbstractStoreManager"
value="com.rs.u2.u2jpa.U2StoreManager" />
      <property name="openjpa.BrokerFactory" value="abstractstore" />
      <property name="openjpa.Optimistic" value="true" />
      <property name="openjpa.Sequence"
value="org.apache.openjpa.kernel.TimeSeededSeq" />
      <property name="openjpa.Compatibility"
value="storeMapCollectionInEntityAsBlob=true" />
      <property name="openjpa.Log"
value="File=c:/temp/u2jpa.log, Runtime=TRACE" />
    </properties>
  </persistence-unit>
</persistence>
```

In U2JPA, persistence units must contain the following:

- A unique name within the persistence.xml file.
- A provider element with the value set to:
“org.apache.openjpa.persistence.PersistenceProviderImpl”
- A number of <class> elements that specify the entity classes included in the persistence unit.

- The <properties> element, which includes a number of <property> elements that specify the OpenJPA/U2JPA specific properties defined for the persistence unit.

Some of the properties can be set manually, and passed to the EntityManager interface when it is initialized. Some examples of these properties are `openjpa.ConnectionUserName` and `openjpa.ConnectionPassword`.

Transaction Support

U2JPA supports both optimistic and pessimistic transactions.

Optimistic Transactions

Transactions in JPA are optimistic by default. Optimistic transactions do not lock the data until the transaction commit time, which improves performance and scalability, while minimizing potential deadlocks. In order to guarantee data integrity, concurrent changes to the same data are checked when a transaction commits, and the transaction aborts if concurrent changes are found.

Pessimistic Transactions

Pessimistic transactions behave differently from optimistic transactions, in that they exclusively lock anything read from the database. This eliminates the possibility of concurrent changes to the same data, but also increases the potential for dead locking among different transactions. Since no concurrent read is possible, performance and scalability can suffer in comparison with optimistic transactions.

U2JPA Optimistic Lock Check

U2JPA compares the current values of all the mapped fields of an entity and all its sub-entities to their original values when they are first loaded from the database. It does this to check whether a record-level entity is modified concurrently.



Since sub-entities do not have identifiers of themselves, there is no sure way to look for the original values of a sub-entity in the database. Therefore, when an optimistic lock check is needed on a sub-entity, U2JPA checks the top-level entity to which this sub-entity belongs. This means that even if this particular sub-entity isn't modified concurrently, if anything else changes in the entity tree rooted from the top-level entity, U2JPA throws an optimistic locking exception.

Note: Changes to entities must occur inside a transaction for them to be persisted to the database, as shown in the following code example:

```
...
em.getTransaction ().begin ();

Customer cust = em.find(Customer.class, "9");

cust.setName ("New Name");

em.getTransaction ().commit ();
...
```

Connection Pooling

U2JPA supports connection pooling. The term connection pooling refers to the technology that pools permanent connections to data sources for multiple threads to share. It improves application performance by saving the overhead of making a fresh connection each time one is required. Instead of physically terminating a connection when it is no longer needed, connections are returned to the pool and an available connection is given to the next thread with the same credentials.

Connection Pool Size

You can set the minimum and maximum size of the connection pool. If you do not define these sizes, the minimum size defaults to 1 and the maximum size defaults to 10. The minimum size determines the initial size of the connection pool.

Activating Connection Pooling

To activate connection pooling, use the `UniObjects.UOPooling` statement in your program, as shown in the following example:

```
UniObjects.UOPooling = true;
```

Specifying the Size of the Connection Pool

To specify the size of the connection pool, use **minpoolsize** to define the minimum number of connections, and **maxpoolsize** to define the maximum number of connections, as shown in the following example:

```
minpoolsize = 1;  
maxpoolsize = 10;
```

If you do not specify the minimum and maximum number of connections, UniVerse defaults to 1 for the minimum and 10 for the maximum.

Refer to the *UniObjects for Java Developer's Guide* for more information on UniObjects for Java connection pooling.

SSL Connection and NLS Support

U2JPA users can choose to use SSL when connecting to U2 servers, and can also specify the encoding that is used when NLS support is needed.

SSL Connection

U2JPA supports SSL connections through UniObjects for Java SSL support. U2JPA requires that trusted certificates be present in Java runtime's default trust store.

To activate SSL Connections, use the **SSL** statement in your program, as shown in the following example:

```
;ssl = true;
```

Refer to the *UniObjects for Java Developer's Guide* for more information on UniObjects for Java SSL support.

NLS Support

For UniVerse:

- If NLS is turned on for the server, U2JPA automatically uses the UTF8 encoding regardless whether the user specifies an encoding or whether the user specified encoding is UTF8.
- If NLS is turned off for the server side, U2JPA either uses the user specified encoding, or the system default one if the user does not specify one

For UniData:

- U2JPA either uses the user specified encoding, or the system default encoding if the user does not specify one.

Managing Entities

Persistence Context

A persistence context is a managed set of entity instances. Every persistence context is associated with a persistence unit, restricting the classes of the managed instances to the set defined by the persistence unit. Within a persistence context, entities are managed and uniquely identified. For example, there must not be more than one entity of the same class with the same identity.

The EntityManager controls entity life cycles, and can access datastore resources. When a persistence context ends, previously managed entities become detached. A detached entity is no longer under the control of the EntityManager and no longer has access to datastore resources. This means that changes to the detached entity do not get persisted to the database, nor do the unloaded (lazy-loading) fields get resolved.

There are two types of entity managers: Container Managed and Application Managed.

Container (Java EE application server) Managed EntityManagers

In the Java EE environment, the most common way to acquire an EntityManager is by using the @PersistenceContext annotation. This is also referred to as dependency injection. In a dependency injection, the application server creates the EntityManager during runtime, and the developer declares a dependency on an EntityManager. An EntityManager obtained in this way is container managed, because the container manages the life cycle of the entity manager.

```
...
@PersistenceContext (unitName="EmployeeService")
EntityManager em;
...
```

There are two styles of container-managed EntityManagers: transaction-scoped and extended. The transaction-scoped container always creates a new persistence context when a transaction starts and closes the persistence context when the transaction ends. Such persistence contexts are also called transaction persistence contexts.

The extended container style keeps the persistence context until the EntityManager itself closes, regardless of the transactions performed by the EntityManager. Such persistence contexts are also called extended persistence contexts. Transaction-scoped EntityManagers typically are used with stateless session beans, while extended style ones are typically associated with stateful session beans.

Application Managed EntityManagers

An EntityManager can be created via the createEntityManager() call of an EntityManagerFactory instance. An EntityManager created in such a way is an application managed entity manager. In application-managed EntityManagers, the application manages the life cycle of the entity manger, instead of the container. Application-managed EntityManagers manage persistence contexts in the same way as the extended style container-managed EntityManagers.

U2JPA Methods

In JPA, all operations on entities are done through EntityManagers. Some of these operations affect the life cycle of the entities, some are responsible for managing the identities of the managed objects in the persistence context, some control the cache, and some manage queries.

A persistence unit defines the set of entities managed by an EntityManager. This persistence unit defines the classes in each entity set related to an application. In this section, we discuss the methods available in U2JPA and their unique behaviors with respect to the U2 databases.

The U2JPA provider is based on the Apache OpenJPA persistence project. For more information about using OpenJPA methods, refer to the OpenJPA online manual at <http://openjpa.apache.org/documentation.htm>.

These are the methods you can use with U2JPA entities:

- clear
- find
- flush
- getReference
- lock
- merge
- refresh
- remove
- persist

public void clear()

The clear() method clears the EntityManager and effectively ends the persistence context. All entities managed by the EntityManager become detached. This method differs from EntityManager's close() method, in that the EntityManager is still active and U2JPA still keeps its internal data cache. To free all the resources held by U2JPA in the memory, the close () method must be called.

public <T> T find(Class<T> class, Object id)

The find() method returns the persistent instance of the given entity type with the given persistent identity. If the instance is already present in the current persistence context, the cached version is returned. Otherwise, a new instance is constructed and loaded with state from the datastore. If no entity with the given type and identity exists in the datastore, this method returns null.

If the passed-in parameter class is a top-level entity class, U2JPA returns an instance of the entity class. In addition, by default, U2JPA also loads its sub entities linked through One-to-Many relationships, as they are annotated to eagerly load. If Many-to-One or One-to-One relationships are present in the entity graph, they will not load until the application needs to access them. A user can modify the default fetch type of these relationships. For example, a user can annotate a One-to-Many relationship to lazy load, so the sub entities are not generated until they are accessed by the application. By the same token, Many-to-One and One-to-One relationships can be eagerly loaded if they are annotated to do so.

If the passed-in parameter class is a sub-entity class, U2JPA returns a null value.

public void flush()

The flush() method writes any changes that have been made in the current transaction to the datastore. U2JPA doesn't support flush().

public <T> T getReference(Class<T> class, Object id)

This method is similar to the find() method, but it does not necessarily have to go to the database when the entity is not found in the cache. Instead, the implementation may construct a hollow entity and then return it to you. Hollow entities do not have any state loaded. The states only get loaded when you attempt to access a persistent field. At that time, the implementation may throw an EntityNotFoundException if it discovers that the entity does not exist in the datastore. Unlike find(), the getReference() method does not return a null value.

public void lock (Object entity, LockModeType mode)

The `lock()` method locks the given entity using the named mode. You can use the `lock()` in an optimistic transaction to provide extra locking control, in addition to the default optimistic locking. In a pessimistic transaction, everything read from the database is exclusively locked (READU), and so using the `lock()` method does not produce any changes.

The `javax.persistence.LockModeType` enumeration defines two modes: `READ` and `WRITE`.

READ

In `READ` mode, different transactions can concurrently read the object, but cannot concurrently update it. The read lock provides a repeatable read isolation level. U2JPA implements read locks in an optimistic way. This means that if you read lock an entity, it results in an optimistic check of the entity at the transaction commit time. If the entity has been changed concurrently, an optimistic locking exception is thrown.

WRITE

In `WRITE` mode, different transactions cannot concurrently read or write the object. When a transaction that holds `WRITE` locks on any entity is committed, the entity version increments even if the entity itself does not change in the transaction. U2JPA always attempts to write the entity with the `write lock()` back to the database, and if the entity is changed concurrently, an optimistic locking exception is thrown.

The `lock()` method does not cascade along relationships. If the passed-in parameter entity is a sub-entity, U2JPA locks the top-level entity of which the passed in entity is a descendant. If the top level does not exist in the persistence context, an exception is thrown.

public Object merge(Object entity)

An entity becomes detached when the persistence context in which it belongs closes, or when it is serialized to be transferred to another tier for more processing.

The `merge()` method puts a detached entity in a persistence context, making it managed again. It then returns a managed copy of the detached entity. Changes made to the persistent state of the detached entity are applied to this managed instance.

If the passed-in parameter entity is a top level entity, all its sub-entities (entities from its associations and sub-associations) are merged, regardless of the cascade type specified on the relationships.

If the passed-in parameter entity is a sub-entity, only the sub-entity and its sub-entities are merged. However, the parent entities of this sub-entity are not automatically merged, which causes an exception to be thrown when the transaction commits if the parent entities are not in the persistence context.

public void refresh(Object entity)

The refresh() method is used to refresh the state of an entity in order to keep it up-to-date with the database. The side effect of this method call is that any changes made to the entity are lost.

If the passed-in parameter entity is a top level entity, all its sub-entities (entities from its associations and sub-associations) are refreshed, regardless of the cascade type specified on the relationships.

If the passed-in parameter entity is a sub-entity, only the sub-entity and its sub-entities are refreshed. However, the parent entities of this sub-entity are not refreshed, which can lead to potential data integrity problems.

public void remove(Object entity)

The remove() method is used to remove an entity by deleting the corresponding data record in the database when the transaction commits.

If the passed-in parameter entity is a top level entity, all its sub-entities (entities from its associations and sub-associations) will be removed regardless of the cascade type specified on the relationships.

If the passed-in parameter entity is a sub-entity and its parent entity is not removed at the same time, or it is not removed from its parent entity's collection, an exception is thrown when the transaction commits. To remove a sub-entity, the user must delete it from the parent entity's collection.

public void persist(Object entity)

The persist() method is used to put an entity into the persistence context managed by the EntityManager.

If the passed-in parameter entity is a top level entity, all its sub-entities (entities from its associations and sub-associations) are persisted regardless of the cascade type specified on the relationships.

If the passed-in parameter entity is a sub entity and its parent entity is not persisted at the same, an exception is thrown when the transaction commits. To add a sub entity, the user must add it to the parent entity's collection.

U2JPA Query Support

The primary query language used in JPA is the Java Persistence Query Language, or JPQL. JPQL is syntactically very similar to SQL, but is object-oriented rather than table-oriented. In most, if not all implementations, queries written in JPQL are translated into SQL at run time. For this reason, U2JPA does not support JPQL. Instead, U2JPA supports U2 native queries with restrictions via the native query interface of JPA.

U2JPA also supports positional query parameters (not named parameters), query results, query paging, projection-type queries, and non-typed subroutine calls. A simple native query to retrieve all PRODUCT entities from the database is shown below:

```
...
Query q = em.createNativeQuery("LIST PRODUCT", Product.class);
List<Product> pl = q.getResultList();
..."
```

Note that in the above example, in addition to the query command, we also pass in an entity class to the createNativeQuery() method call. This tells the persistence provider that the query is expected to return one or more instances of this entity class. Such queries are called typed queries because the type of the query result is known. In typed queries, the entity class returned must be a top-level entity.

Restrictions on accepted LIST, SORT query commands for a typed query

U2JPA supports the LIST and SORT query commands that conform to the following format pattern:

```
{LIST|SORT} filename [selection_criteria] [sorting_criteria]
```

where

`selection_criteria` works in the same manner specified in the U2 query manuals except that it must begin with the WITH keyword (not the WHEN).

`sorting_criteria` works in the same manner specified in the U2 query manuals except that it must not start with keywords BY.EXP or BY.EXP.DSND.

Note: Field names and record IDs can not be used in the commands for typed queries.

These restrictions are in place because U2JPA must load all the sub-entities in order to guarantee data integrity within the same database record. Examples of valid query commands are shown below:

```
LIST PRODUCTS WITH NAME LIKE "...5"

LIST PRODUCTS WITH UNIT_PRICE > 10 BY UNIT_PRICE

SORT PRODUCTS BY.DSND PRICE
```

Query Parameters

U2JPA only supports positional query parameters in the select conditions. To specify a parameter, use ‘?n’ in place of the values in the conditions, where n is a 1-based integer. To set the parameters, use the `setParameter` (position, value) method of the `Query` class.

Note: U2JPA does not support named parameters.

The example below shows how to correctly use a query parameter:

```
Query q = em.createNativeQuery("LIST PRODUCTS WITH NAME LIKE ?1
and
MADEIN = ?2", Product.class);
List<Product> pl =q.setParameter(1,"DVD").
setParameter(2,"USA").getResultList();
```

Query Result Paging

U2JPA supports paging. Users can use the `setFirstResult` (start) and `setMaxResults` (pagesize) methods to select a particular page of the result list. The following example shows how to page through a query result:

```
Query q = em.createNativeQuery("LIST PRODUCTS", Product.class);

int pagesize = 30; int start = 0;
do {
List<Product> pl=.setFirstResult(start)
.setMaxResults(pagesize)
.getResultList();
start += pl.size();
} while (pl.size() == pagesize);
```

Projection Query

Projection queries are queries that select just a few fields in a table or entity, instead of all the fields. An example of when you might want to use such a query might occur if you want to quickly generate a list of information on an entity, present it to the user, and then retrieve the full entity only when requested by the user. Projection queries are also be used to return the results of aggregate queries.

Because the information retrieved by a projection query is incomplete, it cannot construct an entity and so is not typed. The result of a projection query is either a list of generic Objects [] or a single Object []. Each Object [] contains the values of all the fields in the selection list. In U2JPA, the generic Object is actually a UOJ UniDynArray, as shown below:

```
Query q = em.createQuery("LIST PRODUCTS TITLE GENRE");

List<Object[]> tgl = q.getResultList();

for (Object[] tg : tgl) {
    UniDynArray title = (UniDynArray) tg[0]
    UniDynArray genre = (UniDynArray) tg[1];
    System.out.println(tg[0]);
    System.out.println(tg[1]);
}
```

In this example, we used the createQuery() method.

Unlike the typed query for which an entity class has to be passed into the createNativeQuery() method call, projection queries do not have a target entity class specified so instead the command must include at least one field name in the selection list, such as TITLE and GENRE, shown in the example. The syntax for the projection query command is:

```
{LIST|SORT} filename field_name_list [selection_criteria] [sorting_criteria]
```

where

```
field_name_list = field_name [ field_name]...
```

field_name must be present in the dictionary of the filename file, field names are separated by white space characters (space, tab, new line characters)

selection_criteria works in the same manner specified in the U2 query manuals except that it must start with the WITH keyword (not the WHEN).

sorting_criteria works in the same manner specified in the U2 query manuals except that it must not start with keywords BY.EXP or BY.EXP.DSND. The following examples show how to use a createNativeQuery() projection query:

```
Query q = em.createNativeQuery("LIST PRODUCTS TITLE GENRE");

List<UniDynArray[]> tgl = q.getResultList();

for (UniDynArray[] tg : tgl) {
    UniDynArray title = tg[0]
    UniDynArray genre = tg[1];
    System.out.println(tg[0]);
    System.out.println(tg[1]);
}
```

Projection Query: Constructor Expression

JPA supports constructor expressions, which allow developers to map an array of Object result types to custom objects. A constructor expression is defined using the NEW operator in the selection list. The argument to the NEW operator is the fully qualified name of the class that will be instantiated to hold the results for each row of data returned. The only requirement on this class is that it has a constructor with arguments matching the exact type and order that will be specified in the query.

U2JPA requires that the parameters for a constructor should either all be of UniDynArray type or of String type.

The syntax for projection query with constructor express is:

```
{LIST|SORT} filename NEW classname (field_name_list) [selection_criteria]
[sorting_criteria]
```

where

classname is the fully qualified name of the target class.

field_name_list = field_name [, field_name]...

field_name must be present in the dictionary of the filename file. Field names are separated by the comma in the same way that Java separates method parameters.

selection_criteria works in the same manner specified in the U2 query manual except that it must start with the WITH keyword (not the WHEN).

sorting_criteria works in the same manner specified in the U2 query manual except that it must not start with keywords BY.EXP or BY.EXP.DSND.

The following example shows how to use a projection query with a constructor expression:

```
package com.rs.u2.u2jpa.junit.model;
public class TitleGenre {
    private String title;
    private String genre;

    public TitleGenre(String title, String genre)
    {
        this.title = title;
        this.genre = genre;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    ...
}

Query q = em.createNativeQuery("LIST PRODUCTS "
    + "NEW com.rs.u2.u2jpa.junit.model.TitleGenre(TITLE,GENRE)");

List<TitleGenre> tgl = q.getResultList();
```

Subroutine Calls

U2JPA supports subroutine calls through the JPA native query interface. The following example shows how to call a subroutine:

```
...
Query q = em.createNativeQuery ("CALL U2JPASUB");
q.setHint("openjpa.hint.u2sub.numberofpara", "3");
q.setHint("openjpa.hint.u2sub.output.para", "2,3");

q.setParameter(1, "9");

UniDynArray[] result =(UniDynArray[]) q. getSingleResult();
System.out.println("parameter 2's value is " + result[0]);
System.out.println("parameter 3's value is " + result[1]);
```

As shown above, the syntax of the query command for calling a subroutine is:

CALL subroutinename

where

`subroutineName` is the name of cataloged subroutine, either globally or local to the U2 account of the persistence unit.

In addition, if the subroutine has parameters, users must use the `setHint()` method to tell the provider how many parameters are defined on the subroutine, and which ones are the output parameters. U2JPA defines two hint names for this purpose: `openjpa.hint.u2sub.numberofpara` and `openjpa.hint.u2sub.output.para`.

The value of `openjpa.hint.u2sub.numberofpara` must be set to the string representation of non-negative integer.

The value of `openjpa.hint.u2sub.output.para` must follow the pattern:

`[n [,n]...]`

where `n` is a 1-based integer, representing a parameter's position in the parameters for the subroutine.

If the subroutine has input parameters, the `setParameter (input_para_position, value)` method can be used to set the input values.

The result of a subroutine call, if it has output parameters, is an array of `UniDynArrays`, each holding the value of an output parameter defined by `openjpa.hint.u2sub.output.para`, and in the same order.

Named Query

Named queries are used primarily for organizing query definitions. To use a named query, a user defines the queries, assigns them names, and then references them by name at runtime. U2JPA supports named native queries. The example below describes how to define and use a named native query. For more information, please refer to OpenJPA user manual.

To define a named query:

```
@NamedNativeQuery(name="ListProducts",
    query="LIST PRODUCTS WITH AWARD_TYPE = ?1 OR @ID > ?3",
    resultClass=Product.class)

@Entity
@Table(name="PRODUCTS")
public class Product implements Serializable {
    .....
}
```

To use a named query:

```
Query q = em.createNamedQuery("ListProducts");

List<Object> tgl = q.setParameter(3, "3333333333")
    .setParameter(1, "Oscar")
    .getResultList();
Listing 15: An example of named query for subroutine call
@NamedQuery(name="GetMemberInfo",
    query="CALL MEMBERSUB",
    hints= {
@QueryHint(name="openjpa.hint.u2sub.numberofpara",
value="2"),
@QueryHint(name="openjpa.hint.u2sub.output.para",
value ="2")
```

You can also use a named query in a subroutine call. To do this:

```
@NamedQuery(name="GetMemberInfo",
    query="CALL MEMBERSUB",
    hints= {
@QueryHint(name="openjpa.hint.u2sub.numberofpara",
value="2"),
@QueryHint(name="openjpa.hint.u2sub.output.para",
value ="2")
    }
)
```

U2JPA Eclipse Plug-in

The U2JPA Eclipse plug-in is an authoring tool for the Eclipse environment that is used to generate Java entity classes for U2 data files.

The U2JPA plug-in provides a U2 Resource view and lets the user explore all the database accounts and the data files under these accounts. Users can use standard drag and drop functionality to create an entity class for a data file by selecting a file in the U2 Resource view and dragging it on to a Java package in Eclipse's Package Explorer or Project Explorer.

The U2JPA Eclipse plug-in includes a wizard to guide users through the process of generating entity classes for a file.

Currently the plug-in always generates a Many-to-One relationship between two data files, regardless of whether it is actually One-to-One because there is no sure way to know/guess what the cardinality of the relationship is and since U2 data is multi-level, Many-to-One just seems to be a better bet than One-to-One. Note that as far as U2JPA implementation is concerned, it doesn't make any difference whether the relationship is Many-to-One or One-to-One.

In addition, the plug-in has left the cascade type of the relationship unspecified (as opposed to Listing 2 where the One-to-Many relationship has a cascade type of ALL), meaning no operation shall be cascaded down along the relationship, which may not be the desired type for the user. That said, U2JPA user can always manually modify the generated entity classes later to specify the real cardinality of the relationship and its desired cascade type.

The U2JPA Wizard

The wizard is used to help users generate entity classes on a file. It does this by first presenting a dictionary view of the data file showing only the D-type and I-type fields.

In the first wizard page, a user can:

- Select the fields for the entity classes to be generated
- Change the SM attribute of a field, useful for Universe dictionaries which do not distinguish between MV and SV

- Designate a field to be a ‘foreign key’ to another data file so that a Many-to-One relationship is to be built with another entity class from the linked data file.

The next wizard page gives a tree view of the entity classes to be generated, and allows users to change Java names, as well as the data types of the Java fields (this is not recommended). When preparing this wizard page, the U2JPA:

1. Generates the class names and member names for the entity classes from the data file name and field names, based on Java naming convention;
2. Generates entity classes for associations in addition to the one for the data file;
3. Make informed guess of the data types of the class member fields from the conv code of the corresponding data field.
4. Generate Many-to-One relationships for the foreign keys if present by analyzing the TRANS () expression in the virtual fields. The user still needs to generate the entity classes for the foreign data files separately though.

After the user confirms the information on the screen by clicking the Finish button, the plug-in then generates the entity classes (.java files) and places them in the package.

Note that the generated entity classes do not go into the persistence.xml automatically for now. Eclipse provides a menu item to synchronize the entity classes with persistence.xml in the project.

The Persistence.xml Configuration File

The U2JPA persistence.xml file is managed by the U2JPA Eclipse plug-in, although you can also edit the file manually. Persistence units are maintained automatically, and generated entity classes are automatically added to the persistence.xml file, and synchronized with the persistence.xml file.

Step-By-Step Guide to Using U2JPA

This chapter gives you step-by-step instructions on how to map a U2 data set into a Java object using U2JPA. To do this, complete the following steps:

- Create a new U2JPA project in Eclipse
- Create a new U2JPA user library
- Use the U2 Resource view in Eclipse
- Create a U2JPA package
- Run the Program

The U2JPA package is installed, by default, in the following directory:

`C:\U2\UniDK\U2JPA`

You need to install the `u2jpa` plug-in directly into your Eclipse directory:

To install the `u2jpa` plug-in:

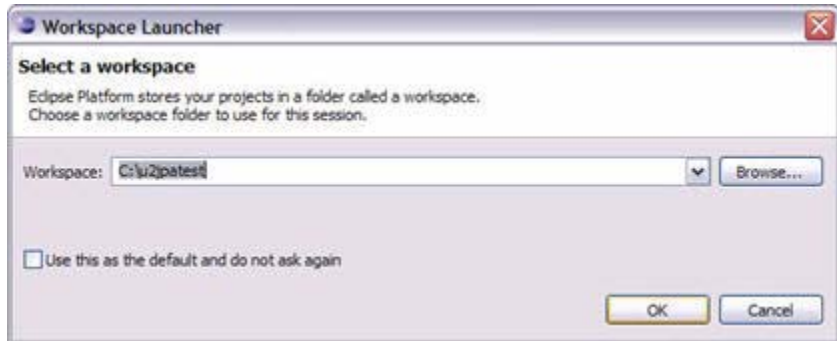
1. Navigate to the directory where U2JPA is installed.
2. Open the **u2jpa** folder, and then select the **u2jpa.plugin-1.0.0.zip** file.
3. Extract the `u2jpa.plugin-1.0.0.zip` file into your Eclipse directory.
 - The file contains two folders: features and plugins.

Create a New U2JPA Project in Eclipse

To open a new U2JPA project:

1. Create a `u2jpatest` folder on the `C:\` drive.

2. Open Eclipse. The Workspace Launcher opens.



Give your workspace a descriptive name and click **OK**. In our example, we name the workspace `c:\u2jptest`. The Eclipse editor opens.

3. Select **File > New > JPA Project** from the Eclipse menu. The New JPA Project dialog box opens.

4. Enter a creative name for your project in the Project Name field. In our example, we name the project **u2jpatest**. Select **<custom>** from the Configuration menu, as shown below:.



You must click the **<custom>** configuration option so that you can configure a custom U2JPA user library in the next steps.

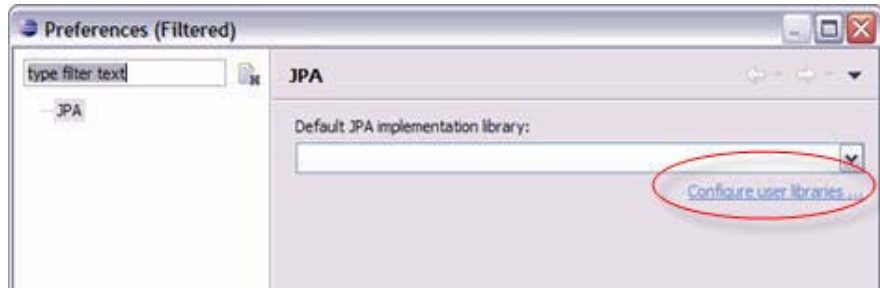
5. Click **Next**. The JPA Facet dialog box opens.

- The JPA Facet dialog box gathers details about the new JPA project. In this example, we accept the default Platform and Connection property settings, but need to add a new U2JPA configuration library. To do this, click the **Configure default JPA implementation library...** option from the JPA implementation properties menu:.



The Preferences dialog box opens.

7. Click the **Configure user libraries...** link.



The User Libraries dialog box opens.

Create a New User Library

We are going to create a new user library to bundle several external .jar files, which will be added to the class path when we run the project.

To create a new user library:

1. Click the **New** button, found on the right side of the User Library dialog box.



The New User Library dialog box opens.

2. Give the new user library a descriptive name and click **OK**. In our example, we name the library **u2jpa-1.0.0**. Focus returns to the User Libraries page.
3. Single-click the new library you just created to highlight the library. Click **Add JARs...**. The JAR selection pane opens.

4. Navigate to the appropriate JAR and click **Open**. You must repeat this step for each of the following JARs:
 - C:\u2\UniDK\u2jpa\u2jpa-1.0.0.jar
 - C:\apache-openjpa-all-1.3.0-SNAPSHOT.jar
 - C:\apache-openjpa-1.3.0-SNAPSHOT\lib (Note: Include all jars included in this directory).
5. Once you have added all the JAR files to the user library, click **OK**. Focus returns to the JPA Facet dialog box.
6. Click **Discover annotated classes automatically** from the Persistent class management menu, as shown below:



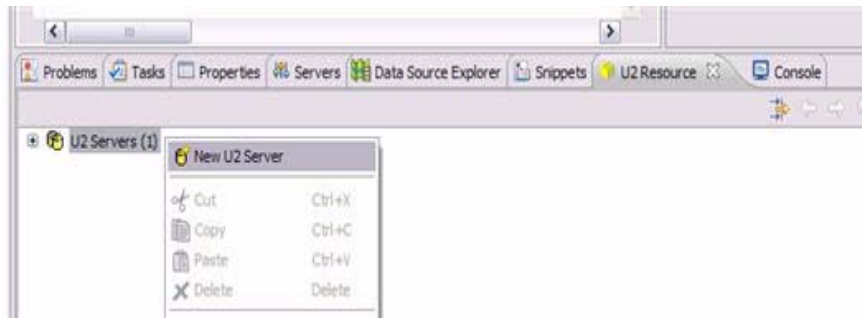
7. Select the new library from the JPA implementation menu. Select **Discover annotated classes automatically**, as shown, and then click **Finish**. Focus returns to the Eclipse project window.
8. A dialog box opens and asks if you want to open the project in with the JPA perspective. In this example, we click **No**.

Use the U2 Resource View in Eclipse

After you create the U2JPA user libraries, you can begin working with your U2 files in Eclipse using the U2 Resource view.

To use the U2 Resource view:

1. To open the U2 Resource view, click **Window > Show View > Other** to open the Show View window. Navigate the **U2 Views** folder. Open the folder and select **U2 Resource** from the menu options. Click **OK**. Focus returns to the U2 Resource view in the Eclipse project window.
2. Create a U2 server entry by right-clicking the **U2 Servers** option and selecting **New U2 Server**:



The Create a New U2 Server dialog box opens.

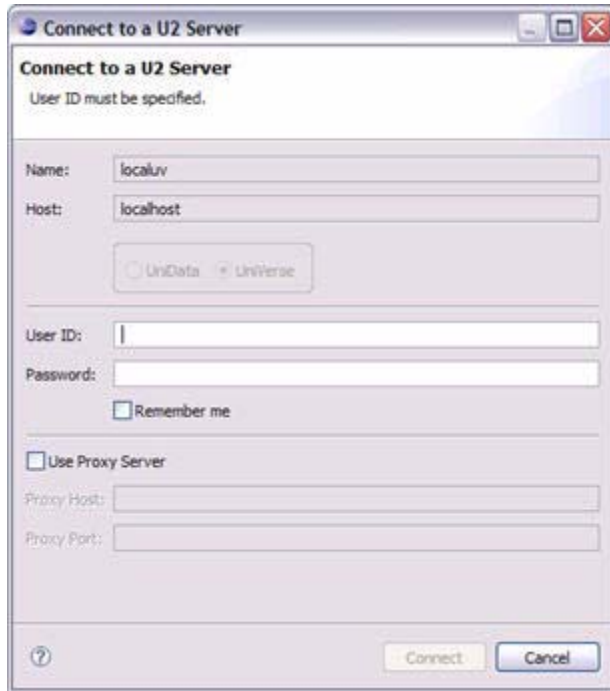
3. Provide a name for the new server in the Name field. Enter the IP address or Network name in the Name field. Define your server as either UniData or UniVerse.



You can also click the Advanced button if you want to alter any of the default UOJ connection information.

4. Click **Finish**. Focus returns to the Eclipse project window.

5. Right-click on the U2 Server you just created and then click **Connect** to establish a connection to the server. The Connect to a U2 Server dialog box opens:



6. Provide the valid login criteria for the server to which you are connecting. Click **Connect**. Your U2 data files are now available in Eclipse.

Create a U2JPA package

The next step is to create a new U2JPA package, and a new Java class.

To create a new U2JPA package:

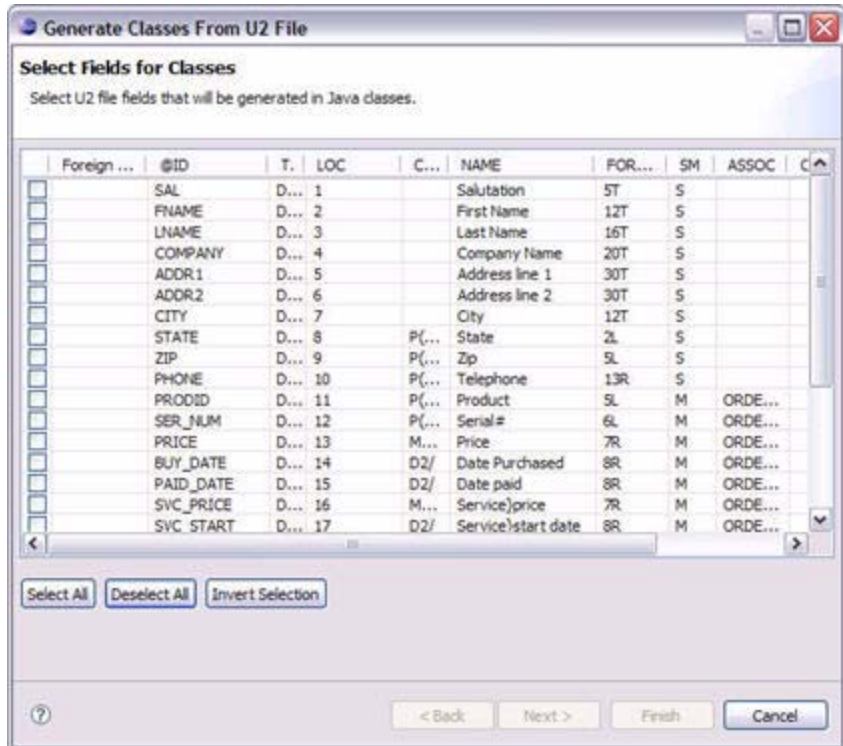
1. In the Project Explorer view, locate the u2jptest node in the tree view. Click the “+” symbol to expand the node. Right-click the **src** option, and then select **New > Package** from the menu. The new JPA package dialog box opens.

2. Enter a descriptive name for the JPA package. In this example, we name the package **com.rs.u2.u2jpatest**, as shown:



3. Click **Finish**. Focus returns to the Eclipse project window.

- Navigate to the **localuv > Accounts > HS.SALES > Database Files > CUSTOMER** file in the U2 Resource view. Drag the **CUSTOMER** file onto the **com.rs.u2.u2jptest** package you created earlier. The Generate Classes from U2 Files dialog box opens.

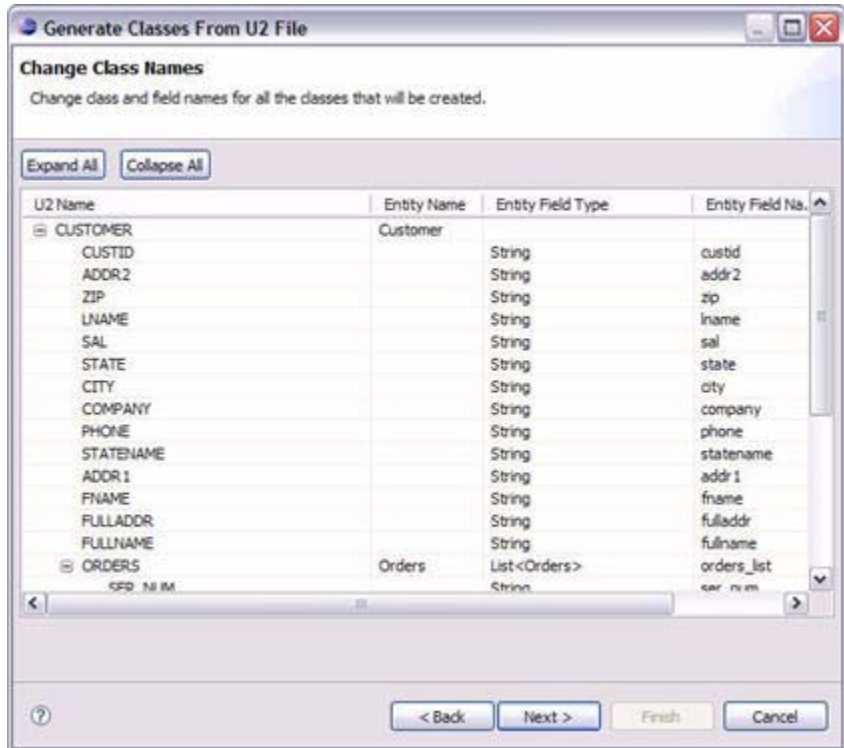


Select the U2 file fields you want to generate in Java classes. You can also:

- Select the fields for the entity classes to be generated
- Change the SM attribute of a field
- Designate a field to be a ‘foreign key’ to another data file so that a Many-to-One relationship can be built with another entity class from the linked data file.

In this example, we choose the **Select All** option and accept all defaults. After selecting the appropriate fields, click **Next**. The Change Class Names

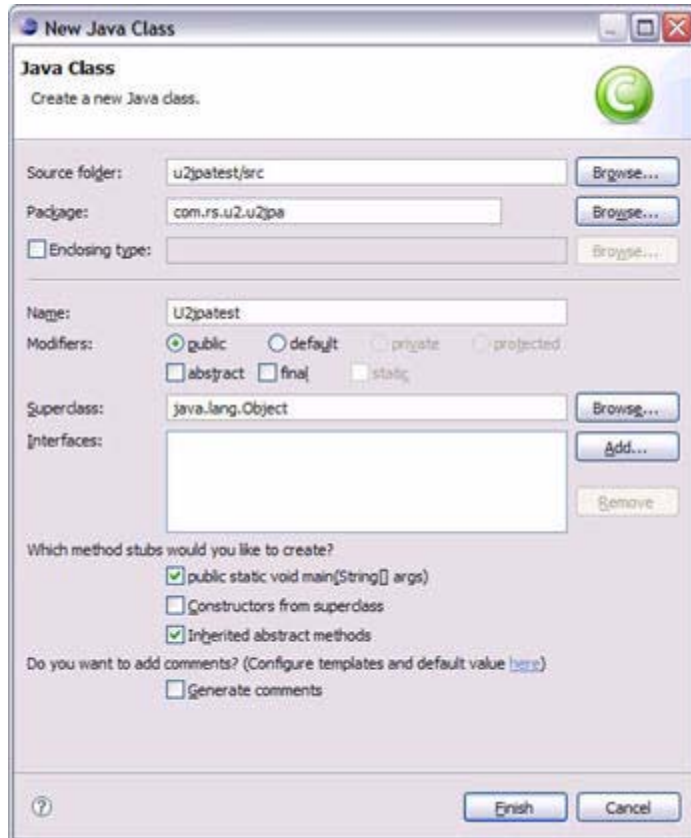
window opens.



In this window you can:

- Change the Java names
 - Change the data type of the java fields (not recommended)
5. Make any changes you feel are appropriate to meet your needs. In this example, we accept the defaults. When you are done, click **Next**. The New Persistence Unit window opens.
 6. Give the new persistence unit a descriptive name. In this example, we name the persistence unit **u2jptest**. Click **Finish**. Focus returns to the Eclipse project window.
 7. Right-click on new package you just created. Select **New > Class** from the menu. The New Java Class window opens

8. In the Name field, enter a descriptive name for the class. In this example, we name the class **U2jpatest**. Select the **public static void main(string[] args)** option from the method stubs menu, as shown:.



9. Click **Finish**. Focus returns to the Eclipse project window.
10. In the Solution Explorer, double-click the U2jpatest class that you just created. The Java code designer opens.

11. Add the following code to the editor:

```
public static void main(String[] args) {
    EntityManagerFactory emf =
        Persistence.createEntityManagerFactory("u2jptest");

    // Properties props = new Properties();
    // props.put("openjpa.ConnectionUserName", "upix");
    // props.put("openjpa.ConnectionPassword", "upi456789111");
    // EntityManager em = emf.createEntityManager(props);
    EntityManager em = emf.createEntityManager();
    Customer c = em.find(Customer.class, "2");
    U2JPAUtil.printEntity(c, 0);
}
```

Note: If you did not select the Remember Me option on the Connect to U2 Server page, use the following code snippet and manually add your login information:

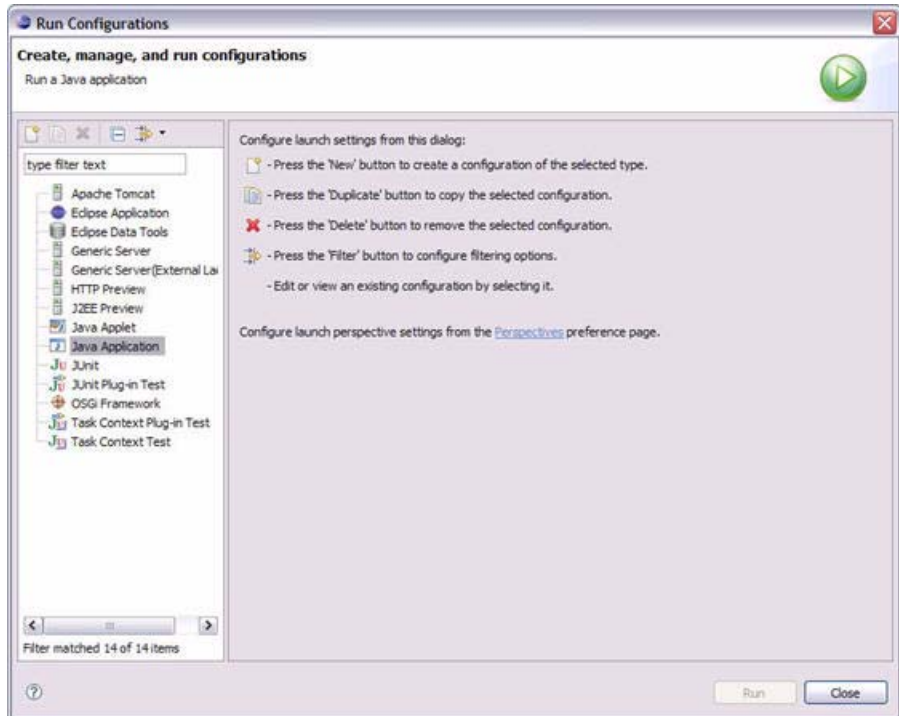
```
public static void main(String[] args) {
    EntityManagerFactory emf =
        Persistence.createEntityManagerFactory("com.rs.u2.u2jptest");

    Properties props = new Properties();
    props.put("openjpa.ConnectionUserName", "upix");
    props.put("openjpa.ConnectionPassword", "upi456789111");
    EntityManager em = emf.createEntityManager(props);
    //EntityManager em = emf.createEntityManager();
    Customer c = em.find(Customer.class, "2");
    U2JPAUtil.printEntity(c, 0);
}
```

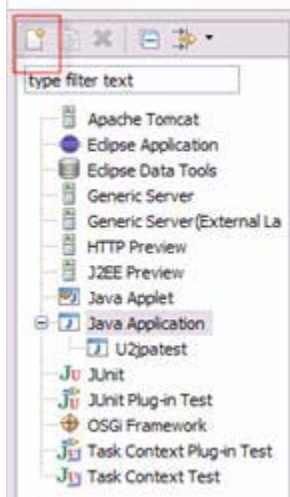
Run the Program

To run the program:

1. In the Project Explorer, right-click the **U2jptest.java** package that you created earlier, and then click **Run As > Run Configurations**. The Run Configurations window opens.

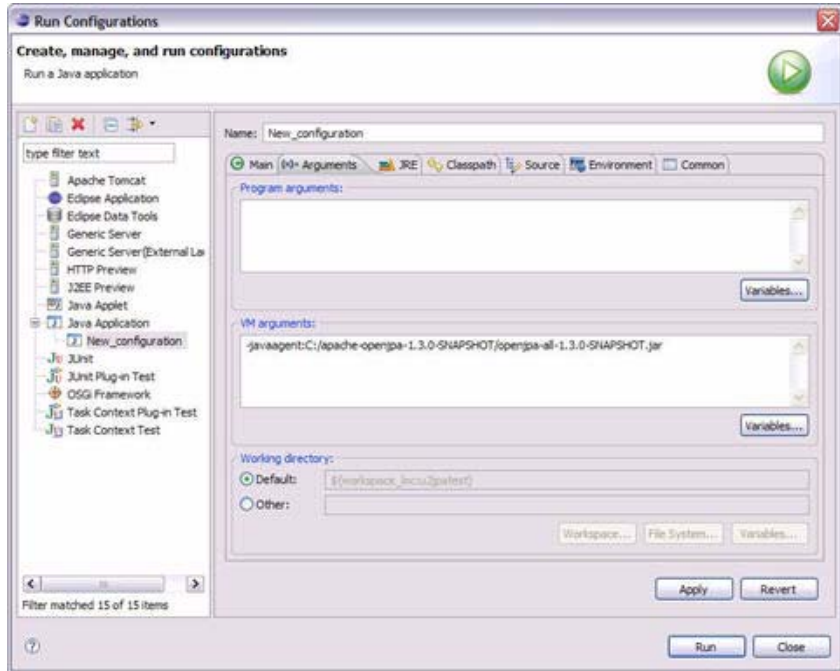


2. Select **Java Application** from the tree view options, and then click the **New launch configuration** icon:



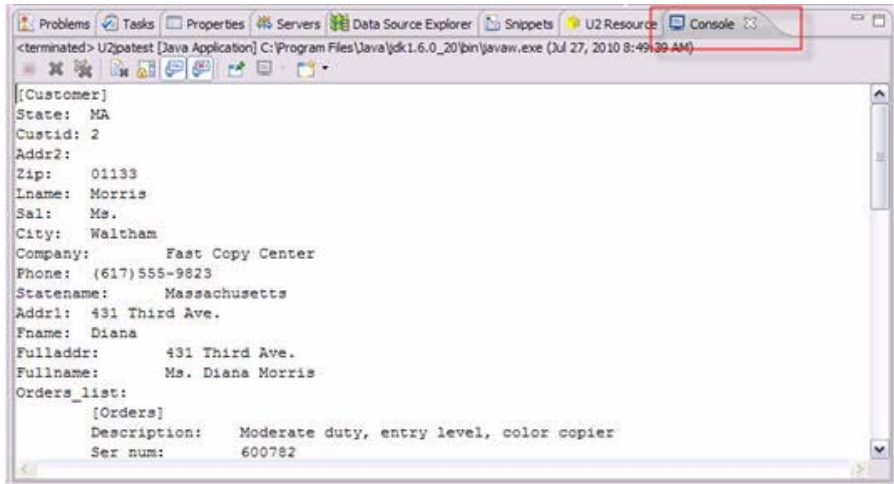
3. The Create Configuration screen opens, and displays a screen with several tabs. Click the **Arguments** tab.
4. Paste the following line of code in the **VM arguments** pane:
`-javaagent:C:/apache-openjpa-1.3.0-SNAPSHOT/openjpa-all-1.3.0-SNAPSHOT.jar`

5. Your screen should look similar to the following example:



6. Click **Apply** to apply the new VM argument and then click **Run**. Focus returns to the Eclipse project screen.

7. The results of your query appear in the Console tab, as shown below:



The screenshot shows an IDE window with the 'Console' tab selected. The console output displays the results of a query, including customer details and a list of orders. The 'Console' tab title is highlighted with a red box.

```
<terminated> U2jptest [Java Application] C:\Program Files\Java\jdk1.6.0_20\bin\javaw.exe (Jul 27, 2010 8:49:39 AM)

[[Customer]
State: MA
Custid: 2
Addr2:
Zip: 01133
Lname: Morris
Sal: Ms.
City: Waltham
Company: Fast Copy Center
Phone: (617)555-9823
Statename: Massachusetts
Addr1: 431 Third Ave.
Fname: Diana
Fulladdr: 431 Third Ave.
Fullname: Ms. Diana Morris
Orders_list:
  [Orders]
  Description: Moderate duty, entry level, color copier
  Ser num: 600782
```

Miscellaneous New Features

Database Auditing Using Index-Based Subroutines	7-3
Programming Example	7-5
BUILD.INDEX CONCURRENT	7-6
Syntax.	7-6
ODBC Enhancements	7-7
Scalability	7-7
ODBC 3.0 Supported Functions	7-7
API Conformance Levels.	7-8
UniVerse ODBC-Supported API Functions	7-10

Database Auditing Using Index-Based Subroutines

Triggers are used to enforce certain business rules when users or programs make changes to a database. In UniVerse, triggers are compiled and cataloged UniVerse BASIC subroutines. These triggers can be defined to fire for each row that is inserted, updated, or deleted.

UniVerse supports conventional triggers on both SQL tables and nested files. These triggers are created through the `CREATE TRIGGER` command, and they support the usual features common to triggers. They are stored formally as part of the database, and they must adhere to certain tenets. These triggers support many capabilities, some of which span SQL-specific areas.

In some circumstances, an application may not require all of the capabilities available to conventional triggers. In these cases, index-based subroutines may serve the purpose better because they have reduced overhead, which may translate to better performance.

Index-based subroutines tend to be more lightweight and flexible than conventional triggers. They are useful if you need to audit events that change the database, but you don't need the overhead associated with explicitly creating a trigger through a `CREATE.TRIGGER` command. Instead, you follow a set of guidelines to produce trigger behaviour from an index-based subroutine.

The trigger behavior is produced by creating a UniVerse BASIC subroutine, which is then called within an I-descriptor field for a file. You then create an index on that I-descriptor using the `NO.NULLS` keyword. The index updates when a record on the file is modified via a `DELETE`, `INSERT`, or `UPDATE` command, which causes the subroutine in the I-descriptor to fire.

The following example creates the INDEX and adds the I-descriptor to the file.

```
>CREATE.FILE TEST.IDX 2 1 1
>ED DICT TEST.IDX INDEX.IOTYPE
New record.
----: I
0001= I
0002= SUBR (" INDEX.SUB" , @ID, @RECORD)
0003=
0004=
0005= 10L
0006= S
0007=
Bottom at line 6.
----: FI
>CREATE.INDEX TEST.IDX INDEX.IOTYPE NO.NULLS
```

The index on the file is evaluated every time the file is modified, which invokes the INDEX.SUB subroutine. Therefore, whenever a record in the file is modified via an DELETE, INSERT, or UPDATE operation, the indexed subroutine can be used to track database file modifications.

The @IDX.IOTYPE variable is a new @variable that can be integrated in the indexed subroutine to determine the type of database operation that caused the indexed subroutine to fire. The value of the @IDX.IOTYPE variable specifies the type of operation being performed.

The following table describes values associated with the @IDX.IOTYPE.

Value	Description
0	The value returned when @IDX.IOTYPE is used outside the context of an indexed subroutine.
1	The value returned when the SUBR is called because an INSERT operation is performed.
2	The value returned when the SUBR is called because a DELETE operation is performed.
3	The value returned when a SUBR is called because an UPDATE operation is used to evaluate the original value operation.
4	The value returned when a SUBR is called because an UPDATE operation is used to evaluate the new value operation.

@IDX.IOTYPE Return Values

The @IDX.IOTYPE value is stacked, so that if the SUBR variable is not called in a nested format, the return value will be the current caller's status.

Programming Example

The following example demonstrates how to use the @IDX.IOTYPE variable with the supported INSERT, DELETE, UPDOLD, and UPDNEW values.

```
SUBROUTINE INDEX.SUB (RTNVAL)
  COMMON /INDEX.SUB/ OPENFLAG, F.AUDIT, OLDRECORD
  RTNVAL = ""
  IF NOT (OPENFLAG) THEN
    OPEN "AUDIT.FILE" TO F.AUDIT ELSE STOP "CANNOT OPEN
AUDIT.FILE"
    OPENFLAG = 1
  END
  *
  * The following case statement can be used to execute any specific
  * operations related to the type of operation being performed.
  *
  AUDIT.REC = ''
  BEGIN CASE
    CASE @IDX.IOTYPE = "1"
    CASE @IDX.IOTYPE = "2"
    CASE @IDX.IOTYPE = "3"
      OLDRECORD = LOWER(@RECORD)
    CASE @IDX.IOTYPE = "4"
      AUDIT.REC<2> = OLDRECORD
    CASE 1
      RETURN
  END CASE
  IF @IDX.IOTYPE # "3" THEN
    RECID = @DATE:"*":SYSTEM(12):"*":@ID
    AUDIT.REC<1> = @IDX.IOTYPE
    WRITE AUDIT.REC ON F.AUDIT, RECID
  END
  RETURN
END
```

BUILD.INDEX CONCURRENT

Prior to UniVerse 11.1, UniVerse locked the data file when processing a BUILD.INDEX statement. Because the file was locked, no updates were allowed to the file until the operation was complete.

Beginning at this release, you can build the index online. Note that building the index online is slower than building it offline, but users can make updates to the file while UniVerse is rebuilding the index.

Syntax

BUILD.INDEX *filename* { *attribute* [*attribute*...] | ALL } [CONCURRENT]

ODBC Enhancements

At this release, UniVerse ODBC is compliant with ODBC 3.0 standards. You can now develop thread safe Web applications using UniVerse ODBC.

Scalability

The UniVerse ODBC driver is now compatible with ODBC 3.0, creating a scalable environment with which to create your Web applications. Scalability allows multiple threads to share the same ODBC handle, creating a thread-safe environment for Web-based application development.

ODBC 3.0 Supported Functions

ODBC is an open-standard API that allows applications to access heterogeneous SQL data. Using ODBC, applications are insulated from underlying network and database implementations.

Many new features have been added to ODBC 3.0, including a faster and more flexible way to bind to multiple sets of data and improved capability for better error diagnostics.

The following table lists the new functions available in ODBC 3.0 that are supported by UniVerse ODBC:

ODBC 3.0 Functions

SQLAllocHandle

SQLCloseCursor

SQLEndTran

SQLFetchScroll

SQLFreeHandle

SQLGetConnectAttr

SQLGetDescField

Supported ODBC 3.0 Functions

ODBC 3.0 Functions

SQLGetDiagField

SQLGetDiagRec

SQLGetEnvAttr

SQLGetStmtAttr

SQLBindParam

SQLSetConnectAttr

SQLSetDescField

SQLSetDescRec

SQLSetEnvAttr

SQLSetStmtAttr

Supported ODBC 3.0 Functions

API Conformance Levels

ODBC specifies three levels of API conformance:

- Core API
- Level 1 API
- Level 2 API

The following are the definitions of the ODBC API Conformance Levels as specified in the *Microsoft ODBC 2.0 Programmer's Reference and SDK Guide*.

Core API

- Allocate and free environment, connection, and statement handles.
- Connect to data sources. Use multiple statements on a connection.
- Prepare and execute SQL statements. Execute SQL statements immediately.
- Assign storage for parameters in an SQL statement and result columns.
- Retrieve data from a result set. Retrieve information about a result set.

- Commit or roll back transactions.
- Retrieve error information.

Level 1 API/Q

- Core API functionality.
- Connect to data sources with driver-specific dialog boxes.
- Set and inquire values of statement and connection options.
- Send part or all of a parameter value.
- Retrieve part or all of a result column value.
- Retrieve catalog information (tables, columns, primary keys, foreign keys, special columns, statistics, procedures, and procedure columns).
- Retrieve information about driver and data source capabilities (API and SQL levels, data types, scalar functions).

Level 2 API

- Core and Level 1 functionality.
- Browse available connections and list available data sources.
- Send arrays of parameter values. Retrieve arrays of result column values.
- Retrieve the number of parameters and describe individual parameters.
- Use a scrollable cursor.
- Retrieve the native form of an SQL statement.
- Retrieve enhanced catalog information (privileges, keys, procedures).
- Call a translation DLL.

UniVerse ODBC Conformance Levels

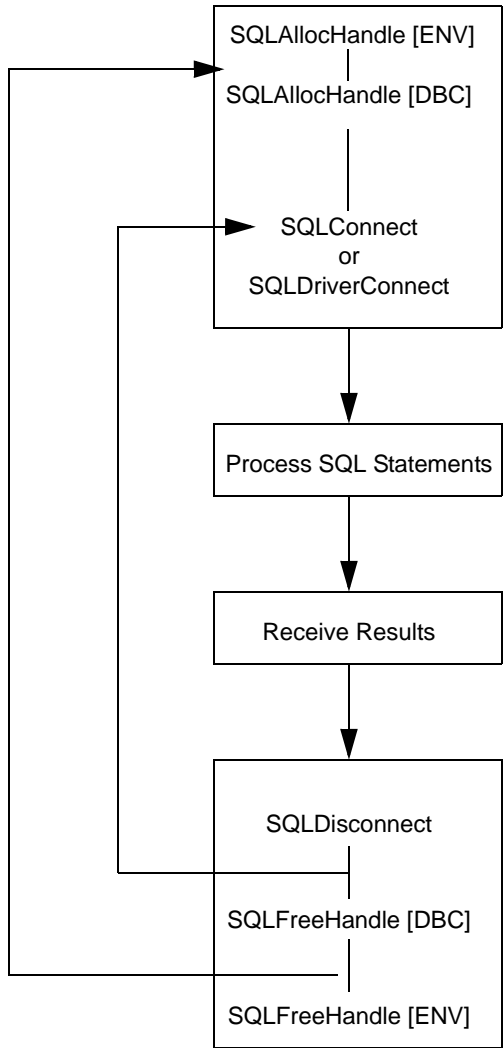
The UniVerse ODBC driver is a read/write release supporting the Level 2 API conformance level.

UniVerse ODBC-Supported API Functions

UniVerse ODBC supports the following API functions which are separated into groups according to the tasks they perform. For more information on the functions in this section, see *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

Connecting to a Data Source

The connection information for each data source is stored in the registry. When an application calls SQLConnect, the ODBC Driver Manager uses the information from the registry to load the appropriate driver and passes the SQLConnect arguments to it. The following figure shows the proper sequence of function calls for connecting to a data source and then disconnecting from the data source.



UniVerse ODBC supports the following functions in this category:

- SQLAllocHandle

- SQLBrowseConnect
- SQLConnect
- SQLConnectAttr
- SQLDriverConnect
- SQLGetConnectAttr
- SQLGetConnectionOption
- SQLDisconnect
- SQLGetEnvAttr
- SQLFreeConnect
- SQLFreeEnv
- SQLSetConnectAttr
- SQLSetConnectionOption
- SQLSetEnvAttr

Processing an SQL Statement

An ODBC application can execute any SQL statement supported by a DBMS. ODBC defines a standard syntax for SQL statements but does not require you to use this syntax. UniVerse SQL-specific statements can be sent through the same interface, although an ODBC application using UniVerse SQL's proprietary syntax extensions will not be able to access any other data source.

There are two ways to execute a statement:

- Prepared Execution
- Direct Execution

Prepared Execution

An application uses prepared execution if it needs to execute the same SQL statement more than once, or if it needs information about the result set prior to execution.

Direct Execution

An application uses direct execution if it needs to execute an SQL statement once and does not need information about the result set prior to execution.

Supporting Parameters

An SQL statement can contain parameter markers to indicate values that are supplied at execution time. The following three examples show how an application might use parameter markers in three types of statements: SELECT, non-SELECT, and procedure calls.

Example 1 — SELECT statements with a parameter marker:

```
SELECT CUST, TAPES-RENTED FROM CUSTOMER_TAPES_MV_SUB1 WHERE CUST =  
?
```

Example 2 — INSERT statement:

```
INSERT INTO CUSTOMER_TAPES_MV_SUB1 VALUES (?,?)
```

Example 3 — call a procedure in place of an SQL statement:

```
{CALL MYPROC (?, ?) }
```

Diagrams of Processing SQL Statements

The following figure shows a simple sequence of ODBC function calls to execute SQL statements. UniVerse ODBC supports the following functions in this category:

- SQLAllocStmt
- SQLAllocHandle
- SQLSetStmtOption
- SQLSetStmtAttr
- SQLGetStmtOption
- SQLSetStmtAttr
- SQLPrepare
- SQLBindParameter
- SQLBindParam
- SQLNumParams
- SQLParamData
- SQLParamOptions
- SQLPutData
- SQLExecute

- SQLExecDirect
- SQLTransact
- SQLEndTran
- SQLCancel
- SQLRowCount
- SQLFreeStmt
- SQLFreeHandle
- SQLGetCursorName
- SQLSetCursorName
- SQLCloseCursor
- SQLSetDescRec
- SQLSetDescField
- SQLGetDescField

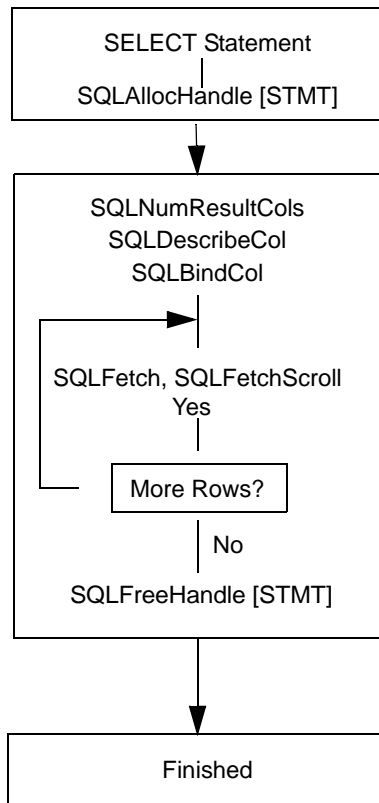


***Note:** The capability of `SQLParamOptions` to specify multiple values for parameters is not yet supported. The `SQLDescribeParam`, `SQLBulkOperations`, `SQLCopyDesc`, and `SQLDescRec` functions are not supported.*

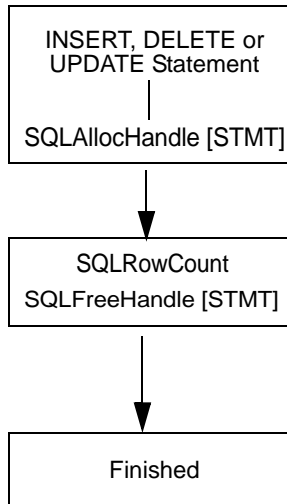
You can process multiple statements simultaneously for a given connection handle.

Retrieving Results and Information about Results

The data you retrieve as a result of executing an SQL SELECT statement is called a result set. It can contain zero or more rows. Your application can assign storage for results before or after it executes an SQL statement. It then calls `SQLFetch` to move to the next row of the result set and retrieve the data. The following figure shows a typical data retrieval operation.



SQL UPDATE, INSERT, and DELETE statements do not return result sets. Instead, your application can query the number of rows affected by an INSERT, DELETE, or UPDATE statement. The following figure shows a typical INSERT, DELETE, or UPDATE operation:



UniVerse ODBC supports the following functions in this category:

- SQLNumResultCols
- SQLDescribeCol
- SQLColAttributes
- SQLBindCol
- SQLGetData
- SQLFetch
- SQLFetchScroll
- SQLRowCount



***Note:** UniVerse ODBC supports only forward-scrolling capabilities. SQLColAttribute is not supported.*

UniVerse ODBC does not support rowset cursors and positioning within a rowset.

Retrieving Data Source and Driver Information

UniVerse ODBC supports the following functions in this category:

- SQLGetInfo
- SQLGetTypeInfo
- SQLGetFunctions

Error Handling

UniVerse ODBC supports the following error handling functions in this category:

- SQLGetDiagField
- SQLGetDiagRec