Rocket UniVerse

Security Features

Version 11.2.3

April 2014

UNV-1123-SECU-1
Notices

Edition

Publication date: April 2014
Book number: UNV-1123-SECU-1
Product version: Rocket UniVerse V11.2.3

Copyright

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

Trademarks

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

Examples

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

License agreement

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

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

Contact information

Website: www.rocketsoftware.com
Rocket Software, Inc. Headquarters
77 4th Avenue, Suite 100
Waltham, MA 02451-1468
USA
Tel: +1 781 577 4321
Fax: +1 617 630 7100
Contacting Global Technical Support

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

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

www.rocketsoftware.com/support

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

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

Email: u2support@rocketsoftware.com

Telephone:

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

### Chapter 1: Using SSL with the CallHTTP and socket interfaces
- Overview of SSL technology ........................................ 1-3
- SSL security programmatic interfaces for UniData and UniVerse 1-4
- Creating a security context ........................................ 1-5
- Saving a security context ......................................... 1-7
- Loading a security context ....................................... 1-9
- Showing a security context ....................................... 1-11
- Adding a certificate ................................................ 1-12
  - U2 Root Certificate Store ..................................... 1-15
- Adding an authentication rule ................................... 1-21
  - Certificate path rule .......................................... 1-21
- Setting a cipher suite ............................................ 1-24
- Getting a cipher suite .......................................... 1-33
- Setting a private key ............................................ 1-35
- Setting client authentication mode ............................ 1-38
- Setting the authentication depth ............................... 1-39
- Generating a key pair ............................................ 1-41
- Creating a certificate request ................................. 1-44
- Creating a certificate ............................................ 1-48
- Setting a random seed .......................................... 1-50
- Analyzing a certificate ......................................... 1-52
- Encoding and cryptographic functions ....................... 1-54
- Encoding data ..................................................... 1-55
- Encrypting data .................................................. 1-57
- Generating a message digest ................................... 1-65
- Generating a digital signature ................................. 1-67
- Additional reading ............................................... 1-70

### Chapter 2: Configuring SSL through the U2 Extensible Administration Tool
- About SSL .......................................................... 2-2
Chapter 3  Chapter 3: Using SSL with UniObjects for Java
Overview of SSL technology ........................................ 3-3
Software requirements ............................................ 3-4
Configuring the database server for SSL ....................... 3-5
Creating a secure connection ................................... 3-7
  Direct connection. ............................................. 3-8
  Establishing the connection .................................. 3-10
  Proxy tunneling ............................................... 3-11
  Externally secure ............................................. 3-13
Managing keys and certificates for a UOJ Client and a proxy server . 3-18
  Importing CA certificates Into UOJ client trustfile ........ 3-18
  Generating client certificates ............................. 3-19
  Managing key file and trust file for the proxy server .... 3-20

Chapter 4  Chapter 4: SSL support in UniVerse clients
Support for secure connections in UniVerse client ........... 4-3
  In this chapter ............................................... 4-3
UCI configuration parameters for ODBC and UniOLEDB .... 4-4
  Secure connection parameters ............................ 4-4
  Changing UCI configuration parameters ................. 4-6
UniObjects ConnectionString parameters .................. 4-9
About SSL property lists ....................................... 4-11
  Creating and maintaining SSL property lists ............ 4-11
  List encryption .......................................... 4-11
  Loading and decrypting an SSL property list ............ 4-12
  SSL properties ....................................... 4-12
Using the U2 SSL configuration editor ....................... 4-20
  Working with SSL property lists ......................... 4-20
  Starting the U2 SSL configuration editor ............... 4-20
Creating a new SSL property list ........................... 4-24
  Task 1: Assign name, password, SSL version, and store type to property list ........................................ 4-25
  Task 2: Specify certificates, private key and password, certificate revocation list, and cipher suites .......... 4-29
  Task 3: Specify authentication properties ................ 4-33
Chapter 5: Automatic Data Encryption

Overview .................................................. 5-4
Features ................................................. 5-4
Automatic Data Encryption components .............. 5-6
Master key .................................................. 5-6
The key store .............................................. 5-7
The metadata store ........................................ 5-7
Encryption keys and wallets ............................ 5-8
Encryption engine .......................................... 5-9
Password policies ........................................ 5-9
Management utilities and tools ......................... 5-10
Setting up Automatic Data Encryption ............... 5-11
New installation of UniVerse ......................... 5-11
Upgrade installation from UniVerse 10.3 and later releases 5-11
Upgrade installation from UniVerse 10.2 ............ 5-12
Setting up the master key ............................... 5-14
Creating the key store ................................... 5-15
Configuring password policies ......................... 5-16
Performing Automatic Data Encryption tasks ....... 5-18
No-echo Automatic Data Encryption command mode 5-18
TCL commands vs. the Extensible Administration Tool 5-18
Creating keys and wallets ............................. 5-18
Deleting encryption keys and wallets ................ 5-19
Granting and revoking access .......................... 5-20
Encrypting files ......................................... 5-21
Encryption modes ....................................... 5-21
UniVerse Security Features

- Encryption parameters ........................................ 5-23
- Encryption algorithms ......................................... 5-23
- Encrypting a file ................................................ 5-24
- Decrypting a file ................................................. 5-25
- Reencrypting a file .............................................. 5-26
- Encrypting the @ID field ....................................... 5-27
- Encrypting indexes .............................................. 5-28
- Finding information about encrypted files .................. 5-31
- Accessing data in encrypted files ............................ 5-33
- Passing key activation and decryption disablement to a PHANTOM process ........................................ 5-39
- Changing key passwords ....................................... 5-39
- Managing Automatic Data Encryption ........................ 5-41
  - Adding or changing the master key password ............... 5-41
  - Changing the master key .................................... 5-44
  - Backing up the key store and metadata store .............. 5-45
  - Retagging a key store ....................................... 5-48
- Managing password policies .................................. 5-49
- Automatic Data Encryption and UniVerse high availability.. 5-52
  - A step-by-step guide for high availability ................. 5-52
  - Methods for encrypting files marked or intended to be marked for UniVerse high availability .......................... 5-54
- UniVerse encryption algorithms ................................ 5-58
- Encryption commands ........................................... 5-59
  - CREATE.ENCRYPTION.KEY .................................. 5-59
  - CREATE.ENCRYPTION.WALLET ............................... 5-60
  - DELETE.ENCRYPTION.WALLET ............................... 5-60
  - WALLET.ADD.KEY ........................................... 5-61
  - WALLET.REMOVE.KEY ....................................... 5-62
  - LIST.ENCRYPTION.WALLET .................................. 5-62
  - DELETE.ENCRYPTION.KEY .................................. 5-63
  - LIST.ENCRYPTION.KEY ....................................... 5-64
  - GRANT.ENCRYPTION.KEY ..................................... 5-64
  - REVOKE.ENCRYPTION.KEY .................................... 5-65
  - ENCRYPT.FILE ............................................... 5-66
  - REENCRYPT.FILE ............................................. 5-68
  - ENCRYPT.INDEX ............................................... 5-69
  - DECRYPT.FILE ............................................... 5-70
  - REENCRYPT.FILE ............................................. 5-72
  - DECRYPT.INDEX ............................................... 5-73
Deleting an encryption wallet ........................................ 6-22
Adding a key to an encryption wallet ............................. 6-23
Deleting a key from an encryption wallet ...................... 6-24
Managing encryption ............................................. 6-25
  Encrypting a file ................................................ 6-25
  Decrypting a file ............................................... 6-30
  Reencrypting a file ............................................ 6-32
  Encrypting an index .......................................... 6-37
  Decrypting an index .......................................... 6-39
  Reencrypting an index ..................................... 6-41
Managing the key store ......................................... 6-45
  Exporting a key store ....................................... 6-48
  Importing a key store ...................................... 6-50
Managing master keys .......................................... 6-52
  Changing the master key ................................... 6-54
Managing password policies .................................. 6-57

Chapter 7: Database Audit Logging

Overview ............................................................ 7-3
The Audit Log file ............................................... 7-4
  The Audit Log file contents .................................. 7-6
Resources .......................................................... 7-10
Audit events ....................................................... 7-11
  System events ................................................ 7-11
  Data events .................................................. 7-12
  User event .................................................... 7-13
  DAT.QUERY.RESULTSET event record .................... 7-14
  SYS.LOGON event ............................................ 7-14
Configuring UniVerse for Audit Logging ....................... 7-15
  uvconfig file parameters .................................. 7-15
  AUDIT_LOG_LOC ............................................ 7-15
  AUDIT_LOG_MAX ........................................... 7-15
  AUDIT_LOG_ENC ............................................ 7-16
  The audit configuration file ............................... 7-16
Audit policies .................................................. 7-18
  Syntax terms ................................................ 7-18
  Global policies ............................................ 7-19
  Policy rules ................................................ 7-23
Examples .......................................................... 7-25
  Audit configuration examples ............................. 7-25
| Event group policy examples                      | 7-26 |
| Account and file policy                        | 7-26 |
| Account and file policy using environment variable | 7-26 |
| Process policy                                | 7-27 |
| Program policy                                | 7-27 |
| Specific policy without predefined event group policy | 7-28 |
| RESULTSET event                               | 7-28 |
| Excluding system audit files from Audit Logging | 7-28 |
| UniVerse BASIC changes                        | 7-29 |
| RetriVe and UniVerse SQL implications          | 7-31 |
| The audman utility                            | 7-32 |
| Options                                       | 7-32 |
Chapter 1: Using SSL with the CallHTTP and socket interfaces

Overview of SSL technology ........................................... 1-3
SSL security programmatic interfaces for UniData and UniVerse ... 1-4
Creating a security context ........................................... 1-5
Saving a security context ............................................. 1-7
Loading a security context ........................................... 1-9
Showing a security context .......................................... 1-11
Adding a certificate .................................................. 1-12
  U2 Root Certificate Store ....................................... 1-15
Adding an authentication rule ...................................... 1-21
  Certificate path rule ............................................. 1-21
Setting a cipher suite .............................................. 1-24
Getting a cipher suite .............................................. 1-33
Setting a private key ............................................... 1-35
Setting client authentication mode ................................ 1-38
Setting the authentication depth .................................. 1-39
Generating a key pair ............................................... 1-41
Creating a certificate request ..................................... 1-44
Creating a certificate ............................................... 1-48
Setting a random seed ............................................. 1-50
Analyzing a certificate ............................................. 1-52
Encoding and cryptographic functions ......................... 1-54
  Encoding data .................................................... 1-55
  Encrypting data .................................................. 1-57
  Generating a message digest .................................. 1-65
  Generating a digital signature ................................ 1-67
Additional reading .................................................. 1-70
This chapter describes how to set up and configure SSL for use with the CallHTTP and Socket interfaces.

This chapter consists of the following sections:

- “Overview of SSL technology”
- “SSL security programmatic interfaces for UniData and UniVerse”
- “Encoding and cryptographic functions”

For more information about CallHTTP and the SOAP API, see UniVerse BASIC Extensions.
Overview of SSL technology

Secure Sockets Layer (SSL) is a transport layer protocol that provides a secure channel between two communicating programs over which arbitrary application data can be sent securely. It is by far the most widely deployed security protocol used on the World Wide Web.

Although it is most widely used in applications to secure web traffic, SSL actually is a general protocol suitable for securing a wide variety of other network traffic that is based on TCP, such as FTP and Telnet.

SSL provides server authentication, encryption and message integrity. It optionally also supports client authentication.

UniData and UniVerse currently support HTTP and sockets API. SSL support is important for both protocols in order to deploy commercial applications to be able to securely process sensitive data, such as credit card transactions.

Throughout this chapter we talk about SSL exclusively, but in fact we also support the more recent development of TLS (Transport Layer Security) protocol, which basically is the adoption of SSL by the standard body IETF and contains support for more public key algorithm and cipher suites.

This document assumes that users who want to use this facility have some basic knowledge of public key cryptography.
SSL security programmatic interfaces for UniData and UniVerse

This section provides information on the SSL functions and properties for UniData and UniVerse.

Many of the functions described in this chapter require as input a pass phrase for various operations, such as encrypting a generated private key and saving a security context. To ensure a higher level of security, these functions require that pass phrase is assigned a value. General guidelines for passwords should be followed. Particularly, since English text usually has a very low entropy, that is, given part of a word or phrase, the rest is not completely unpredictable. Thus, we recommend that you choose a relatively long phrase, instead of a single word, when calling these functions.
Creating a security context

The `createSecurityContext()` function creates a security context and returns a handle to the context.

A security context is a data structure that holds all aspects of security characteristics that the application intends to associate with a secured connection. Specifically, the following information may be held for each context:

- Protocol version
- Sender’s certificate to be sent to the peer
- Sender’s private key for signature and key exchange
- Issuer’s certificate or certificate chain to be used to verify incoming certificate
- Certificate verification depth, strength and other rules
- Certificate Revocation List
- Flag to perform client authentication (useful for server socket only)
- Context ID and time stamp

Syntax

```java
createSecurityContext(context, version)
```

For any given connection, not all of the information is required.

A version (SSL version 2 or 3 or TLS version 1) can be associated with a security context. If no version is provided (for example, a null string is sent), the default value will be SSL version 3.

When you specify a version, you tell UniVerse the “minimal” version you want to allow. For example, if you specify SSLv3, UniVerse will allow both SSLv3 and TLSv1 to be processed, but not SSLv2. If you specify TLSv1, only TLSv1 is allowed.

For secure socket connections, socket APIs, `openSecureSocket()` or `initSecureServerSocket()` must be called to associate a security context with a connection, by a client or a server, respectively.

For secure HTTP connection (https), you must supply a valid context handle with the `createSecureRequest()` function.
All aspects of a context can be changed by the API’s described below.

Parameters

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>context</td>
<td>The Security context handle.</td>
</tr>
<tr>
<td>version</td>
<td>A string with the following values: SSLv2, SSLv3 or TLSv1.</td>
</tr>
</tbody>
</table>

createSecurityContext Parameters

The following table describes the status of each return code.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success.</td>
</tr>
<tr>
<td>1</td>
<td>Security context could not be created.</td>
</tr>
<tr>
<td>2</td>
<td>Invalid version.</td>
</tr>
</tbody>
</table>

Return Code Status
Saving a security context

The `saveSecurityContext()` function encrypts and saves a security context to a system security file. The file is maintained on a per account basis for UniData and UniVerse. The name is used as the record ID to access the saved security information. Since the information is encrypted, you should not attempt to directly manipulate it.

You may want your application to save a security context to be used later. Multiple contexts may be created to suit different needs. For example, you may want to use different protocols to talk to different servers. These contexts can be saved and reused.

When creating a saved context, you must provide both a `name` and a `passPhrase` to be used to encrypt the contents of the context. The same `name` and `passPhrase` must be provided to load the saved context back. To ensure a high level of security, we recommend that the `passPhrase` be relatively long, yet easy to remember.

Syntax

```
saveSecurityContext(context, name, passPhrase)
```

Parameters

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>context</code></td>
<td>The Security context handle.</td>
</tr>
<tr>
<td><code>name</code></td>
<td>String containing the name of the saved context.</td>
</tr>
<tr>
<td><code>passPhrase</code></td>
<td>String containing the password to encrypt the context contents.</td>
</tr>
</tbody>
</table>

saveSecurityContext Parameters
The following table describes the status of each return code.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success.</td>
</tr>
<tr>
<td>1</td>
<td>Invalid security context handle.</td>
</tr>
<tr>
<td>2</td>
<td>Invalid parameters (empty name or passPhrase).</td>
</tr>
<tr>
<td>3</td>
<td>Context could not be saved.</td>
</tr>
</tbody>
</table>

Return Code Status
Loading a security context

The loadSecurityContext() function loads a saved security context record into the current session.

The name and passPhrase parameters are needed to retrieve and decrypt the saved context. An internal data structure is created and its handle is returned in the context parameter.

Syntax

loadSecurityContext(context, name, passPhrase)

Parameters

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>context</td>
<td>The handle to be returned.</td>
</tr>
<tr>
<td>name</td>
<td>String containing the name of the saved the security contexts.</td>
</tr>
<tr>
<td>PassPhrase</td>
<td>String containing the passPhrase needed to decrypt the saved data.</td>
</tr>
</tbody>
</table>

loadSecurityContext Parameters

The following table describes the status of each return code.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success.</td>
</tr>
<tr>
<td>1</td>
<td>Context record does not exist.</td>
</tr>
</tbody>
</table>

Return Code Status
<table>
<thead>
<tr>
<th>Return Code</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Context record could not be accessed (e.g. wrong password).</td>
</tr>
<tr>
<td>3</td>
<td>Invalid content (file was not saved by the <code>saveSecurityContext()</code> function).</td>
</tr>
<tr>
<td>4</td>
<td>Other problems that caused context load failure. Refer to the log file for more information.</td>
</tr>
</tbody>
</table>

Return Code Status (Continued)
Showing a security context

The `showSecurityContext()` function dumps the SSL configuration parameters of a security context into a readable format.

The security context handle must have been returned by a successful execution of `createSecurityContext()` or `loadSecurityContext()`.

The configuration information includes: `protocol`, `version`, `certificate`, `cipher suite` used by this connection and other properties.

Warning: For security reasons, the `privateKey` installed into the context is not displayed. Once installed, there is no way for you to extract it.

Syntax

```
showSecurityContext(context,config)
```

Parameters

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>context</td>
<td>The Security Context handle.</td>
</tr>
<tr>
<td>config</td>
<td>A dynamic array containing the security context data.</td>
</tr>
</tbody>
</table>

saveSecurityContext Parameters

The following table describes the status of each return code.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success.</td>
</tr>
<tr>
<td>1</td>
<td>Invalid Security Context handle.</td>
</tr>
<tr>
<td>2</td>
<td>Configuration data could not be obtained.</td>
</tr>
</tbody>
</table>

Return Code Status
Adding a certificate

The `addCertificate()` function stores a certificate (or multiple certificates) into a security context to be used as a UniData or UniVerse server or client certificate. Alternatively, it can specify a certificate or a directory which contains the certificates that are either used as CA (Certificate Authority) certificates to verify incoming certificates or act as a Revocation list to check against expired or revoked certificates.

There are three kinds of certificates:

- Self-signed root certificate, or root CA certificate – these certificates are used to sign other certificates as a means to vouch for the authenticity of holders of those certificates.
- Intermediate CA certificates – these certificates are signed by a root CA certificate or another intermediate CA certificate and are used to sign other certificates.
- Server/Client certificates – these certificates are signed by root CA or intermediate CA certificates, and are used by a server or client to provide its identity.

Root CA or Intermediate certificates are sometimes also called Issuer certificates.

For a server/client certificate, a complete certificate chain contains all the certificates starting from the server/client certificate to it’s immediate intermediate CA certificate (and the intermediate CA certificate’s immediate intermediate CA certificates, if any), up to the root CA certificate. To verify a server/client certificate, the complete certificate chain needs to be established. For UniVerse, this means that all intermediate root CA certificates must be specified in the security context record. Note that sometimes the intermediate CA certificates can be sent from a server or client, along with the server client certificate. In this case, you only need to add the root CA certificate to the security context record.
A certificate’s purpose is to bind an entity’s name with its public key. It is a means of distributing public keys. A certificate always contains three pieces of information: a name that identifies the owner of this certificate, a public key of this owner, and a digital signature signed by a trusted third party called a Certificate Authority (CA) with its private key. If you have the CA’s public key, you can verify that the certificate is authentic, that is, whether or not the public key contained in the certificate is indeed associated with the entity specified with the name in the certificate. In practice, a certificate can and often does contain more information, for example, the period of time the certificate is valid.

SSL protocol specifies that when two parties start an SSL handshake, the server must always send its certificate to the client for authentication. It may optionally require the client to send its certificate to the server for authentication as well. Therefore, UniData and UniVerse applications that act as HTTPS clients are not required to maintain a client certificate. The application should work with web servers that do not require client authentication, while UniData and UniVerse applications that do act as SSL servers must install a server certificate.

Regardless of which role the application is going to assume, it needs to install a CA certificate or a CA certificate chain to be able to verify an incoming certificate.

**Syntax**

```
addCertificate(certPath, usedAs, format, algorithm, context, p12pass)
```
Parameters

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>certPath</td>
<td>A String containing the name of the OS level file that holds the certificate, or the directory containing certificates.</td>
</tr>
</tbody>
</table>
| usedAs    | Flag - 1: Used as a Client/Server certificate (SSL_CERT_SELF)  
2: Used as an issuer certificate (SSL_CERT_CA)  
3: Used as a Certificate Revocation List (SSL_CERT_CRL) |
| format    | Flag - 1: PEM format (SSL_FMT_PEM)  
2: DER format (SSL_FMT_DER)  
3: PKCS #12 format (SSL_FMT_P12) |
| algorithm | Flag - 1: RSA key (SSL_KEY_RSA)  
2: DSA key (SSL_KEY_DSA) |
| context   | The Security context handle. |
| p12pass   | Optional. Sets a password on the PKCS #12 file. This parameter should only be included if using a PKCS #12 certificate that has a password. Otherwise the parameter should be omitted. This feature was added at UniVerse 11.2.3. |

addCertificate Parameters

Note: To use the predefined constants, you must include SSL.H in your program. The value for PKCS #12 file format is 3. If you include the SSL.H shipped with UniVerse in your BASIC program, you can also use the pre-defined format constant SSL_FMT_P12.

The following table describes the status of each return code.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success.</td>
</tr>
<tr>
<td>1</td>
<td>Invalid Security Context handle.</td>
</tr>
<tr>
<td>2</td>
<td>Certificate file could not be opened or directory does not exist.</td>
</tr>
</tbody>
</table>
U2 Root Certificate Store

CA certificates are used in SSL handshaking phase to authenticate the communication peer. Before UniVerse 11.2, for U2 servers and C-based clients, the only way to supply the CA certificates was by specifying them in an Security Context Record (SCR) and providing the certificates in operating system-level files. Currently, the certificates supported are in conformance with X.509 standards and should be in either DER (Distinguished Encoding Rules, a special case of Abstract Syntax Notation 1, ASN.1) format, PEM (Privacy Enhanced Mail, an IETF standard) format, or PKCS #12 format.

Now, the U2 Root Certificate Store has been added, which resides in the $UVHOME directory. This file contains a metadata section that stores data such as version, creation and modification times, and so forth, followed by PEM-format certificates. Each certificate is either a root CA certificate or an intermediate CA certificate, delimited by:

```
-----BEGIN CERTIFICATE-----
```

and
```
-----END CERTIFICATE-----
```

When UniVerse ships, a preloaded U2 Root Certificate Store that contains root certificates exported from the current Java 7 CA store, such as cacerts, is provided. You can add to this file with your own certificates. You can do this by extracting root certificates from the PKCS #12 file you received from the CA vendor, or by exporting root certificate files from a Windows certificate store or Java’s cacerts file. You can also add or delete a specific root certificate from the U2 Root Certificate Store.

UniVerse loads CA certificates from the U2 Root Certificate Store when you do not specify a CA file. If you specify a CA certificate in the SCR or SPL for

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Unrecognized format.</td>
</tr>
<tr>
<td>4</td>
<td>Corrupted or unrecognized certificate contents.</td>
</tr>
<tr>
<td>5</td>
<td>Invalid parameter value(s).</td>
</tr>
</tbody>
</table>

Return Code Status (Continued)
the connection, UniVerse uses it directly and does not load a CA Certificate from the U2 Root Certificate Store.

**The rcsman utility**

Use the rcsman utility to manage the U2 Root Certificate Store (u2rcs). The following table describes the rcsman options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-list</td>
<td>Displays the contents of the U2 Root Certificate Store. If you specify more than one of the following options, all options must be met for a certificate to be listed.</td>
</tr>
<tr>
<td>-v</td>
<td>Verbose mode. If you do not specify -v, rcsman only displays alias and fingerprints.</td>
</tr>
<tr>
<td>-alias v</td>
<td>Displays certificates with matching alias only.</td>
</tr>
<tr>
<td>-subject v</td>
<td>Displays certificates with matching subject only. v can be any substring in the Subject value of a certificate.</td>
</tr>
<tr>
<td>-issuer v</td>
<td>Displays certificates with matching issuer only. v can be any substring in the Issuer value of the certificate.</td>
</tr>
<tr>
<td>-serial v</td>
<td>Displays certificates with matching serial number only.</td>
</tr>
<tr>
<td>-fingerprint v</td>
<td>Displays certificates with matching fingerprint only.</td>
</tr>
<tr>
<td>-import</td>
<td>Import one or more certificate to the U2 Root Certificate Store.</td>
</tr>
<tr>
<td>file -f</td>
<td>The file that contains the certificate to import.</td>
</tr>
</tbody>
</table>

rcsman utility options
UniVerse currently supports the following formats:

- **PEM** - PEM format, with matching `---BEGIN CERTIFICATE---`. You can add multiple certificates in this manner, with each certificate having an optional “Alias=alias-name” line in between. If you do not provide an alias, UniVerse will use the common name (CN) as the alias. This is the default format.

- **DER** - ASN1 binary format. If you specify this format, you can only import one certificate each time. You can only provide the alias on the command line with the -alias option.

- **PKCS #12** - PKCS #12 format is a security standard for packaging certificates and private keys into a binary archival file. When importing certificates from a PKCS #12 file, rcsman uses the common name (CN) in the certificate as the alias by default unless one is specified on command line.

When importing PKCS#12 file you must specify a password if the file is protected by one. To specify a PKCS #12 password on command line:

```
PKCS12,<password>
```

Or

```
PKCS12,"<password>"
```

if the password contains non-alphanumeric characters.

This feature was added at 11.2.3.

- **-alias v** Import the certificate with a matching alias value.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-format fmt</td>
<td>File format.</td>
</tr>
<tr>
<td></td>
<td>UniVerse currently supports the following formats:</td>
</tr>
<tr>
<td></td>
<td>- <strong>PEM</strong> - PEM format, with matching <code>---BEGIN CERTIFICATE---</code>. You can</td>
</tr>
<tr>
<td></td>
<td>add multiple certificates in this manner, with each certificate having an</td>
</tr>
<tr>
<td></td>
<td>optional “Alias=alias-name” line in between. If you do not provide an</td>
</tr>
<tr>
<td></td>
<td>alias, UniVerse will use the common name (CN) as the alias. This is the</td>
</tr>
<tr>
<td></td>
<td>default format.</td>
</tr>
<tr>
<td></td>
<td>- <strong>DER</strong> - ASN1 binary format. If you specify this format, you can only</td>
</tr>
<tr>
<td></td>
<td>import one certificate each time. You can only provide the alias on the</td>
</tr>
<tr>
<td></td>
<td>command line with the -alias option.</td>
</tr>
<tr>
<td></td>
<td>- <strong>PKCS #12</strong> - PKCS #12 format is a security standard for packaging</td>
</tr>
<tr>
<td></td>
<td>certificates and private keys into a binary archival file. When importing</td>
</tr>
<tr>
<td></td>
<td>certificates from a PKCS #12 file, rcsman uses the common name (CN) in</td>
</tr>
<tr>
<td></td>
<td>the certificate as the alias by default unless one is specified on</td>
</tr>
<tr>
<td></td>
<td>command line.</td>
</tr>
<tr>
<td></td>
<td>When importing PKCS#12 file you must specify a password if the file is</td>
</tr>
<tr>
<td></td>
<td>protected by one. To specify a PKCS #12 password on command line:</td>
</tr>
<tr>
<td></td>
<td>```</td>
</tr>
<tr>
<td></td>
<td>PKCS12,&lt;password&gt;</td>
</tr>
<tr>
<td></td>
<td>Or</td>
</tr>
<tr>
<td></td>
<td>PKCS12,&quot;&lt;password&gt;&quot;</td>
</tr>
<tr>
<td></td>
<td>if the password contains non-alphanumeric characters.</td>
</tr>
<tr>
<td></td>
<td>This feature was added at 11.2.3.</td>
</tr>
</tbody>
</table>

rcsman utility options (Continued)
The password to enable the import operation. The password is only required when the U2 Root Certificate Store is password protected.

- **export** Exports certificates to the U2 Root Certificate Store. If you specify more than one of the following options, all options must be met for a certificate to be exported.

  - **-file f** The file that contains the certificate to export.
  - **-alias v** Export the certificate with a matching alias value.
  - **-subject v** Export the certificate with a matching subject value. This value can be any substring in the Subject value of the certificate.
  - **-issuer v** Export the certificate with a matching issuer value. The value can be any substring in the Issuer value of the certificate.
  - **-fingerprint v** Export the certificate with a matching fingerprint value.
  - **-serial v** Export the certificate with the matching serial number only.

- **delete** Deletes a certificate from the U2 Root Certificate Store. If you specify more than one of the following options, all options must be met for a certificate to be deleted.

  - **-alias v** Deletes a certificate that matches the alias value you specify.
  - **-subject v** Deletes a certificate that matches the subject value you specify. The value can be any substring in the Subject value of the certificate.
  - **-issuer v** Deletes a certificate that matches the issuer value you specify. The value can be any substring in the Issuer value of the certificate.
  - **-fingerprint v** Deletes a certificate that matches the fingerprint value you specify.
Installing or upgrading the U2 Root Certificate Store

For a new installation, UniVerse installs the U2 Root Certificate Store, .u2rcs, in the $UVHOME directory. UniVerse creates a copy of the file, .u2rcs.bak, in the same location.

For an upgrade installation, UniVerse detects if the existing .u2rcs file has been modified in any way. If no modification is detected, the new .u2rcs file replaces the current .u2rcs file. If the file was modified, UniVerse keeps the current .u2rcs file, and displays a message advising you to replace the file manually, if desired. You can update the .u2rcs file using the updatercs.sh or updatercs.bat scripts that are shipped with UniVerse.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-serial v</td>
<td>Deletes a certificate that matches the serial value you specify.</td>
</tr>
<tr>
<td>-force</td>
<td>Deletes the certificate without further confirmation.</td>
</tr>
<tr>
<td>-password p</td>
<td>The password to enable the delete operation. The password is only required when the U2 Root Certificate Store is password protected.</td>
</tr>
<tr>
<td>-changepass</td>
<td>Changes the password to the U2 Root Certificate Store.</td>
</tr>
<tr>
<td>-oldpass p1</td>
<td>The old password. If you do not specify the old password, rcsman prompts for it.</td>
</tr>
<tr>
<td>-newpass p2</td>
<td>The new password. If you do not specify the new password, rcsman prompts for it. To specify an empty password, enter a quoted string (“”).</td>
</tr>
</tbody>
</table>
There can be only one server/client certificate per specific security context. Thus, adding a new server/client certificate will automatically replace an existing certificate. For issuer certificates, however, a new one will be chained with existing certificates so UniData and UniVerse applications can perform chained authentication. The new certificate will be added to the end of the chain, meaning that it will be used as the issuer certificate to authenticate the one before it. If the issuer certificate file is in PEM format, it can contain multiple certificates generated by simply concatenating certificates together. The order in which the certificates are stored does make a difference. Note that all certificates that form an issuer chain must be of the same type. That is, they must be either all RSA type or all DSA type. However, you can add both an RSA type and DSA type certificate to a context as specified by the algorithm parameter.

If the certPath parameter is a directory then all certificates under the directory will be used as issuer certificates when verifying an incoming certificate.
Adding an authentication rule

The **addAuthenticationRule()** function adds an authentication rule to a security context. The rules are used during SSL negotiation to determine whether or not the peer is to be trusted.

Currently, the following rules are supported:

**Verification Strength** rule - This rule governs the SSL negotiation and determines whether or not an authentication process is considered successful. There are two levels of security, *generous* and *strict*. If you specify *generous*, the certificate need only contain the subject name (common name) that matches one specified by “PeerName”, to be considered valid. There is no need to have its complete certificate chain established. If you specify *strict*, the incoming certificate must pass a number of checks, including signature check, expiry check, purpose check and issuer check. A complete certificate chain must be established.

*Note: Setting the rule to generous is recommended only for development or testing purposes.*

**PeerName** rule - By specifying the **PeerName** rule and attribute mark separated common names in ruleString, trusted server/client names will be stored into the context.

During the SSL handshake negotiation, the server will send its certificate to the client. By specifying trusted server names, the client can control which server or servers it should communicate with. During the handshake, once the server certificate has been verified by way of the establishment of the complete certificate chain, the subject name contained in the certificate will be compared against the trusted server names set in the context. If the server subject name matches one of the trusted names, communication will continue, otherwise the connection will not be established.

If no trusted peername is set, then any peer is considered legitimate.

**Certificate path rule**

The certificate path rule enables you to specify locations in which to search for certificates. From the list of options, choose a certificate path rule to specify the search path:
- **Default** – When you add a certificate to a security context record, the full path for that certificate is registered in the security context record. This path is derived from the current directory in which UniVerse is running. When the certificate is loaded into memory to establish the SSL connection, UniVerse by default uses this registered full path to retrieve the certificate.

- **Relative** – With this option, UniVerse looks for the certificate in the current directory in which UniVerse is running.

- **Note**: Some of the UniVerse processes, such as the Telnet server processes, run from the system directory.

- **Path** – With this option, UniVerse uses the path you specify for loading the certificate added to this security context record. You can specify either an absolute path or a relative path.

- **Env** – If you select this option, enter an environment variable name in the Env text box. With this option, the UniVerse process first obtains the value of the environment variable you specify, and then uses that value as the path to load the certificates.

  **Note**: UniVerse evaluates the environment variable only when the first SSL connection is made. The value is cached for later reference.

**Syntax**

```plaintext
addAuthenticationRule(context,serverOrClient, rule, ruleString)
```
Parameters

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>context</td>
<td>The Security Context handle.</td>
</tr>
</tbody>
</table>
| ServerOr-Client| Flag 1 - Server (SSL_SERVER)旗
|                | Flag 2 - Client (SSL_CLIENT)旗
|                | Any other value is treated as a value of 1.                  |
| Rule           | The rule name string. Valid settings are SSL_RULE_PEER_NAME, SSL_RULE_STRENGTH, or SSL_RULE_CERTPATH. |
| RuleString     | Rule content string. May be attribute mark separated.        |

addAuthenticationRule Parameters

The following table describes the status of each return code.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success.</td>
</tr>
<tr>
<td>1</td>
<td>Invalid Security Context handle.</td>
</tr>
<tr>
<td>2</td>
<td>Invalid rule name.</td>
</tr>
<tr>
<td>3</td>
<td>Invalid rule content.</td>
</tr>
</tbody>
</table>

Return Code Status
Setting a cipher suite

The `setCipherSuite()` function allows you to identify which cipher suites should be supported for the specified context. It affects the cipher suites and public key algorithms supported during the SSL/TLS handshake and subsequent data exchanges.

When a context is created, its cipher suites will all be set to SSLv3 suites by default.

The `CipherSpecs` parameter is a string containing `cipher-spec` separated by colons. An SSL cipher specification in `cipher-spec` is composed of 4 major attributes as well as several, less significant attributes. These are defined below.

Some of this information on ciphers is excerpted from the `mod_ssl` open source package of the Apache web server.

- **Key Exchange Algorithm** - RSA or Diffie-Hellman variants.
- **Authentication Algorithm** - RSA, Diffie-Hellman, DSS or none.
- **Cipher/Encryption Algorithm** - AES, DES, Triple-DES, RC4, RC2 or none.
- **MAC Digest Algorithm** - MD5, SHA or SHA1.

An SSL cipher can also be an export cipher and is either an SSLv2 or SSLv3/TLSv1 cipher (here TLSv1 is equivalent to SSLv3). To specify which ciphers to use, one can either specify all the ciphers, one at a time, or use aliases to specify the preference and order for the ciphers.
The following table describes each tag for the **Key Exchange Algorithm**.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KRSA</td>
<td>RSA key exchange</td>
</tr>
<tr>
<td>kDHr</td>
<td>Diffie-Hellman key exchange with RSA key</td>
</tr>
<tr>
<td>kDHd</td>
<td>Diffie-Hellman key exchange with DSA key</td>
</tr>
<tr>
<td>kEDH</td>
<td>Ephemeral (temp.key) Diffie-Hellman key exchange (no cert)</td>
</tr>
</tbody>
</table>

**Key Exchange Algorithm Cipher Tags**

The following table describes each tag for the **Authentication Algorithm**.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aNULL</td>
<td>No authentication</td>
</tr>
<tr>
<td>aRSA</td>
<td>RSA authentication</td>
</tr>
<tr>
<td>aDSS</td>
<td>DSS authentication</td>
</tr>
<tr>
<td>aDH</td>
<td>Diffie-Hellman authentication</td>
</tr>
</tbody>
</table>

**Authentication Algorithm Cipher Tags**
The following table describes each tag for the **Cipher Encoding Algorithm**.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eNULL</td>
<td>No encoding</td>
</tr>
<tr>
<td>DES</td>
<td>DES encoding</td>
</tr>
<tr>
<td>3DES</td>
<td>Triple-DES encoding</td>
</tr>
<tr>
<td>RC4</td>
<td>RC4 encoding</td>
</tr>
<tr>
<td>RC2</td>
<td>RC2 encoding</td>
</tr>
<tr>
<td>AES</td>
<td>AES encoding</td>
</tr>
</tbody>
</table>

**Cipher Encoding Algorithm Cipher Tags**

The following table describes each tag for the **MAC Digest Algorithm**.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5</td>
<td>MD5 hash function</td>
</tr>
<tr>
<td>SHA1</td>
<td>SHA1 hash function</td>
</tr>
<tr>
<td>SHA</td>
<td>SHA hash function</td>
</tr>
</tbody>
</table>

**MAC Digest Algorithm Cipher Tags**
The following table describes each of the **Aliases**.

<table>
<thead>
<tr>
<th>Alias</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSLv2</td>
<td>all SSL version 2.0 ciphers</td>
</tr>
<tr>
<td>SSLv3</td>
<td>all SSL version 3.0 ciphers</td>
</tr>
<tr>
<td>TLSv1</td>
<td>all TLS version 1.0 ciphers</td>
</tr>
<tr>
<td>EXP</td>
<td>all export ciphers</td>
</tr>
<tr>
<td>EXPORT40</td>
<td>all 40-bit export ciphers only</td>
</tr>
<tr>
<td>EXPORT56</td>
<td>all 56-bit export ciphers only</td>
</tr>
<tr>
<td>LOW</td>
<td>all low strength ciphers (no export, single DES)</td>
</tr>
<tr>
<td>MEDIUM</td>
<td>all ciphers with 128 bit encryption</td>
</tr>
<tr>
<td>HIGH</td>
<td>all ciphers using Triple-DES</td>
</tr>
<tr>
<td>RSA</td>
<td>all ciphers using RSA key exchange</td>
</tr>
<tr>
<td>DH</td>
<td>all ciphers using Diffie-Hellman key exchange</td>
</tr>
<tr>
<td>EDH</td>
<td>all ciphers using Ephemeral Diffie-Hellman key exchange</td>
</tr>
<tr>
<td>ADH</td>
<td>all ciphers using Anonymous Diffie-Hellman key exchange</td>
</tr>
<tr>
<td>DSS</td>
<td>all ciphers using DSS authentication</td>
</tr>
<tr>
<td>NULL</td>
<td>all cipher using no encryption</td>
</tr>
</tbody>
</table>

These can be put together to specify the order and ciphers you wish to use. To speed this up there are also aliases (SSLv2, SSLv3, TLSv1, EXP, LOW, MEDIUM, HIGH) for certain groups of ciphers. These tags can be joined together with prefixes to form the **cipher-spec**.
The following table describes the available prefixes.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>Add cipher to the list.</td>
</tr>
<tr>
<td>+</td>
<td>Add ciphers to the list and pull them to the current location in the list.</td>
</tr>
<tr>
<td>-</td>
<td>Remove the cipher from the list (it can be added again later).</td>
</tr>
<tr>
<td>!</td>
<td>Kill the cipher from the list completely (cannot be added again later).</td>
</tr>
</tbody>
</table>

Available Prefixes

A more practical way of looking at all of this is to use the `getCipherSuite()` function which provides a nice way to successively create the correct `cipher-spec` string. The default setup for a `cipher-spec` string is shown in the following example:

"ALL:!ADH=RC4+RSA:+HIGH:+MEDIUM:+LOW:SSLV2:+EXP"

As is shown in the example, you must first remove from consideration any ciphers that do not authenticate, for example, for SSL only the Anonymous Diffie-Hellman ciphers. Next, use ciphers using RC4 and RSA. Next include the high, medium and then the low security ciphers. Finally pull all SSLv2 and export the ciphers to the end of the list.

The complete list of particular RSA ciphers for SSL is given in the following table.
<table>
<thead>
<tr>
<th>Cipher Tag</th>
<th>Protocol</th>
<th>Key Ex.</th>
<th>Auth.</th>
<th>Enc.</th>
<th>MAC</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES256-SHA</td>
<td>SSLv3</td>
<td>RSA</td>
<td>RSA</td>
<td>AES(256)</td>
<td>SHA1</td>
<td></td>
</tr>
<tr>
<td>DES-CBC3-SHA</td>
<td>SSLv3</td>
<td>RSA</td>
<td>RSA</td>
<td>3DES(168)</td>
<td>SHA1</td>
<td></td>
</tr>
<tr>
<td>DES-CBC3-MD5</td>
<td>SSLv2</td>
<td>RSA</td>
<td>RSA</td>
<td>3DES(168)</td>
<td>MD5</td>
<td></td>
</tr>
<tr>
<td>AES-128-SHA</td>
<td>SSLv3</td>
<td>RSA</td>
<td>RSA</td>
<td>AES(1280)</td>
<td>SHA1</td>
<td></td>
</tr>
<tr>
<td>RC2-CBC-MD5</td>
<td>SSLv2</td>
<td>RSA</td>
<td>RSA</td>
<td>RC2(128)</td>
<td>MD5</td>
<td></td>
</tr>
<tr>
<td>RC4-SHA</td>
<td>SSLv3</td>
<td>RSA</td>
<td>RSA</td>
<td>RC4(128)</td>
<td>SHA1</td>
<td></td>
</tr>
<tr>
<td>RC4-MD5</td>
<td>SSLv3</td>
<td>RSA</td>
<td>RSA</td>
<td>RC4(128)</td>
<td>MD5</td>
<td></td>
</tr>
<tr>
<td>RC4-MD5</td>
<td>SSLv2</td>
<td>RSA</td>
<td>RSA</td>
<td>RC4(128)</td>
<td>MD5</td>
<td></td>
</tr>
<tr>
<td>RC4-64-MD5</td>
<td>SSLv2</td>
<td>RSA</td>
<td>RSA</td>
<td>RC4(64)</td>
<td>MD5</td>
<td></td>
</tr>
<tr>
<td>EXP1024-DES-CBC-SHA</td>
<td>SSLv3</td>
<td>RSA (1024)</td>
<td>RSA</td>
<td>DES(56)</td>
<td>SHA1</td>
<td>export</td>
</tr>
<tr>
<td>EXP1024-RC2-CBC-MD5</td>
<td>SSLv3</td>
<td>RSA (1024)</td>
<td>RSA</td>
<td>RC2(56)</td>
<td>MD5</td>
<td>export</td>
</tr>
<tr>
<td>DES-CBC-SHA</td>
<td>SSLv3</td>
<td>RSA</td>
<td>RSA</td>
<td>DES(56)</td>
<td>SHA1</td>
<td></td>
</tr>
<tr>
<td>DES-CBC-MD5</td>
<td>SSLv2</td>
<td>RSA</td>
<td>RSA</td>
<td>DES(56)</td>
<td>MD5</td>
<td></td>
</tr>
<tr>
<td>EXP1024-RC4-SHA</td>
<td>SSLv3</td>
<td>RSA (1024)</td>
<td>RSA</td>
<td>RC4(56)</td>
<td>SHA1</td>
<td>export</td>
</tr>
<tr>
<td>EXP1024-RC4-MD5</td>
<td>SSLv3</td>
<td>RSA (1024)</td>
<td>RSA</td>
<td>RC4(56)</td>
<td>MD5</td>
<td>export</td>
</tr>
<tr>
<td>EXP-DES-CBC-SHA</td>
<td>SSLv3</td>
<td>RSA (512)</td>
<td>RSA</td>
<td>DES(40)</td>
<td>SHA1</td>
<td>export</td>
</tr>
<tr>
<td>EXP-RC2-CBC-MD5</td>
<td>SSLv3</td>
<td>RSA (512)</td>
<td>RSA</td>
<td>RC2(40)</td>
<td>MD5</td>
<td>export</td>
</tr>
</tbody>
</table>

RSA Ciphers
<table>
<thead>
<tr>
<th>Cipher Tag</th>
<th>Protocol</th>
<th>Key Ex.</th>
<th>Auth.</th>
<th>Enc.</th>
<th>MAC</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXP-RC2-CBC-MD5</td>
<td>SSLv3</td>
<td>RSA (512)</td>
<td>RSA</td>
<td>RC2(40)</td>
<td>MD5</td>
<td>export</td>
</tr>
<tr>
<td>EXP-RC4-MD</td>
<td>SSLv3</td>
<td>RSA (512)</td>
<td>RSA</td>
<td>RC4(40)</td>
<td>MD5</td>
<td>export</td>
</tr>
<tr>
<td>EXP-RC4-MD5</td>
<td>SSLv2</td>
<td>RSA (512)</td>
<td>RSA</td>
<td>RC4(40)</td>
<td>MD5</td>
<td>export</td>
</tr>
</tbody>
</table>

**RSA Ciphers**
The complete list of particular DH ciphers for SSL is given in the following table.

<table>
<thead>
<tr>
<th>Cipher Tag</th>
<th>Protocol</th>
<th>Key Ex.</th>
<th>Auth.</th>
<th>Enc.</th>
<th>MAC</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>DHE-RSA-AES256-SHA</td>
<td>SSLv3</td>
<td>DH</td>
<td>RSA</td>
<td>AES(256)</td>
<td>SHA1</td>
<td></td>
</tr>
<tr>
<td>DHE-DSS-AES256-SHA</td>
<td>SSLv3</td>
<td>DH</td>
<td>DSS</td>
<td>AES(256)</td>
<td>SHA1</td>
<td></td>
</tr>
<tr>
<td>EDH-RSA-DES-CBC3-SHA</td>
<td>SSLv3</td>
<td>DH</td>
<td>RSA</td>
<td>3DES(168)</td>
<td>SHA1</td>
<td></td>
</tr>
<tr>
<td>EDH-DSS-DES-CBC3-SHA</td>
<td>SSLv3</td>
<td>DH</td>
<td>DSS</td>
<td>3DES(168)</td>
<td>SHA1</td>
<td></td>
</tr>
<tr>
<td>DHE-RSA-AES128-SHA</td>
<td>SSLv3</td>
<td>DH</td>
<td>RSA</td>
<td>AES(128)</td>
<td>SHA1</td>
<td></td>
</tr>
<tr>
<td>DHE-DSS-AES128-SHA</td>
<td>SSLv3</td>
<td>DH</td>
<td>DSS</td>
<td>AES(128)</td>
<td>SHA1</td>
<td></td>
</tr>
<tr>
<td>DHE-DSS-RC4-SHA</td>
<td>SSLv3</td>
<td>DH</td>
<td>DSS</td>
<td>RC4(128)</td>
<td>SHA1</td>
<td></td>
</tr>
<tr>
<td>EDH-RSA-DES-CBC-SHA</td>
<td>SSLv3</td>
<td>DH</td>
<td>RSA</td>
<td>DES(56)</td>
<td>SHA1</td>
<td></td>
</tr>
<tr>
<td>EDH-DSS-DES-CBC-SHA</td>
<td>SSLv3</td>
<td>DH</td>
<td>DSS</td>
<td>DES(56)</td>
<td>SHA1</td>
<td></td>
</tr>
<tr>
<td>EXP1024-DHE-DSS-DES-CBC-SHA</td>
<td>SSLv3</td>
<td>DH (1024)</td>
<td>DSS</td>
<td>DES(56)</td>
<td>SHA1</td>
<td>export</td>
</tr>
<tr>
<td>EXP-1024-DHE-DSS-RC4-SHA</td>
<td>SSLv3</td>
<td>DH (1024)</td>
<td>DSS</td>
<td>RC4(56)</td>
<td>SHA1</td>
<td>export</td>
</tr>
<tr>
<td>EXP-EDH-RSA-DES-CBC-SHA</td>
<td>SSLv3</td>
<td>DH (512)</td>
<td>RSA</td>
<td>DES(40)</td>
<td>SHA1</td>
<td>export</td>
</tr>
<tr>
<td>EXP-EDH-DSS-DES-CBC-SHA</td>
<td>SSLv3</td>
<td>DH (512)</td>
<td>DSS</td>
<td>DES(40)</td>
<td>SHA1</td>
<td>export</td>
</tr>
</tbody>
</table>

Diffie-Hellman Ciphers

Example:

```
SetCipherSuite(ctxHandle,"RSA:!EXP:!NULL:+HIGH:+MEDIUM:-LOW")
```
SetCipherSuite(ctxHandle,“SSLv3”)

Syntax

\texttt{setCipherSuite(context,cipherSpecs)}

Parameters

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>context</td>
<td>The Security Context handle.</td>
</tr>
<tr>
<td>CipherSpecs</td>
<td>String containing cipher suite specification described above.</td>
</tr>
</tbody>
</table>

\textbf{setCipherSuite Parameters}

The following table describes the status of each return code.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success.</td>
</tr>
<tr>
<td>1</td>
<td>Invalid Security Context handle.</td>
</tr>
<tr>
<td>2</td>
<td>Invalid cipher suite specification.</td>
</tr>
</tbody>
</table>
Getting a cipher suite

The `getCipherSuite()` function obtains information about supported cipher suites, their version, usage, strength and type for the specified security context. The result is put into the dynamic array `ciphers`, with one line for each cipher suite, separated by a field mark (@FM). The format of the string for one cipher suite is as follows.

```
Suite, version, key-exchange, authentication, encryption, digest, export
```

Refer to the cipher tables under the “Setting a cipher suite” for definitions of all suites. The following is an example of a typical Suite.

```
EXP-DES-CBC-SHA SSLv3 Kx=RSA(512) Au=RSA Enc=DES(40) Mac=SHA1 export
```

The suite is broken down as follows. The suite name is EXP-DES-CBC-SHA. It is specified by SSLv3. The Key-exchange algorithm is RSA with 512-bit key. The authentication is also done by RSA algorithm. The Data encryption uses DES (Data Encryption Standard, an NIST standard) with CBC mode. MAC (Message Authentication Code, a hash method to calculate message digest) will be done with SHA-1 (Secure Hash Algorithm 1, also an NIST standard) algorithm. The suite is exportable.

Only those methods that are active for the protocol will be retrieved.

Syntax

```
getCipherSuite(context,ciphers)
```

Parameters

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>context</code></td>
<td>The Security Context handle.</td>
</tr>
<tr>
<td><code>ciphers</code></td>
<td>A Dynamic array containing the cipher strings delimited by @FM.</td>
</tr>
</tbody>
</table>

`getCipherSuite Parameters`
The following table describes the status of each return code.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success.</td>
</tr>
<tr>
<td>1</td>
<td>Invalid Security Context handle.</td>
</tr>
<tr>
<td>2</td>
<td>Unable to obtain information.</td>
</tr>
</tbody>
</table>

| Return Code Status |
Setting a private key

The `setPrivateKey()` function loads the private key into a security context so that it can be used by SSL functions. If the context already had a set private key, it will be replaced.

SSL depends on public key crypto algorithms to perform its functions. A pair of keys is needed for each communicating party to transfer data over SSL. The public key is usually contained in a certificate, signed by a CA, while the private key is kept secretly by the user.

A private key is used to digitally sign a message or encrypt a symmetric secret key to be used for data encryption.

The `Key` parameter contains either the key string itself or a path that specifies a file that contains the key. UniData and UniVerse only accept PKCS #8 style private keys.

The `Format` parameter specifies if the key is in binary format or Base64 encoded format. If the key is in a file, Base64 format also means that it must be in PEM format.

The `KeyLoc` parameter specifies if the key is provided in a file or in a dynamic array string.

If the key was previously encrypted, a correct passPhrase must be given to decrypt the key first. We recommend that the private key be always in encrypted form. Note that if the private key is generated by the `generateKey()` function described under the “Generating a key pair”, it is always in PEM format and always encrypted by a pass phrase.

If the `validate` parameter is set, the private key is verified with the public key contained in the certificate specified for either the server or client. They must match for SSL to work. In some cases there is no need or it is impossible to check against a certificate. For example, the certificate may already be distributed to the other end and there is no need for a user application to authenticate itself. In that case, `validate` can be set to 0 (SSL_NO_VALIDATE).

If `validate` is required, the corresponding certificate should be added first by calling the `addCertificate()` function which is described under “Adding a certificate”.

The direct form of this function may be preferred by some applications where a hard coded private key can be incorporated into the application, eliminating the need to access an external key file, which may be considered a security hazard.

The private key is the single most important piece of secret information for a public-key-based crypto system. You must take every precaution to keep it secure. If the private key is compromised, there will be no data security. This is especially true for server private keys.

Syntax

```
setPrivateKey(key, format, keyLoc, passPhrase, validate, context, p12pass)
```

Parameters

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>A string containing either the key or path for a key file.</td>
</tr>
</tbody>
</table>
| Format    | 1 - PEM (Base64 encoded) format (SSL_FMT_PEM)  
            2 - DER (ASN.1 binary) format (SSL_FMT_DER)  
            3 - PKCS #12 format (SSL_FMT_P12) |
| KeyLoc    | 1 - key contained in key string (SSL_LOC_STRING)  
            2 - key is in a file specified by key (SSL_LOC_FILE) |
| passPhrase| String containing the path phrase required for gaining access to the key. It can be empty if the key is not pass phrase protected. THIS IS NOT RECOMMENDED! |
The following table describes the status of each return code.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>1</td>
<td>Invalid Security handle</td>
</tr>
<tr>
<td>2</td>
<td>Invalid format</td>
</tr>
<tr>
<td>3</td>
<td>Invalid key type</td>
</tr>
<tr>
<td>4</td>
<td>Key file cannot be accessed (non-existent or wrong pass phrase)</td>
</tr>
<tr>
<td>5</td>
<td>Certificate cannot be accessed</td>
</tr>
<tr>
<td>6</td>
<td>Private key does not match public key in certificate</td>
</tr>
<tr>
<td>7</td>
<td>Private key cannot be interpreted</td>
</tr>
<tr>
<td>99</td>
<td>Other errors that prevent private key from being accepted by UniData or UniVerse.</td>
</tr>
</tbody>
</table>

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validate</td>
<td>1 - Validate against matching public key (SSL_VALIDATE)</td>
</tr>
<tr>
<td></td>
<td>0 - Won’t bother to validate (SSL_NO_VALIDATE)</td>
</tr>
<tr>
<td>Context</td>
<td>The security context handle.</td>
</tr>
<tr>
<td>p12pass</td>
<td>Optional. Sets a password on the PKCS #12 file. This parameter should only be included if using a PKCS #12 certificate that has a password. Otherwise the parameter should be omitted. This feature was added at UniVerse 11.2.3.</td>
</tr>
</tbody>
</table>
Setting client authentication mode

The `setClientAuthentication()` function turns client authentication for a server socket on or off.

When `option` is set to on, during the initial SSL handshake, the server sends client authentication request to the client. It also receives the client certificate and performs authentication according to the issuer’s certificate (or certificate chain) set in the security context.

Syntax

```
setClientAuthentication(context, option)
```

Parameters

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>context</code></td>
<td>The Security Context handle.</td>
</tr>
<tr>
<td><code>option</code></td>
<td>1 - ON (SSL_CLIENT_AUTH_  2 - OFF (SSL_NO_CLIENT_AUTH)</td>
</tr>
</tbody>
</table>

The following table describes the status of each return code.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success.</td>
</tr>
<tr>
<td>1</td>
<td>Invalid Security Context handle.</td>
</tr>
</tbody>
</table>

Return Code Status
Setting the authentication depth

The `setAuthenticationDepth()` function sets how deeply UniData and UniVerse should verify before deciding that a certificate is not valid.

This function can be used to set both server authentication and client certification, determined by the value in parameter `serverOrClient`. The default depth for both is 1.

The `depth` is the maximum number of intermediate issuer certificate, or CA certificates which must be examined while verifying an incoming certificate. Specifically, a depth of 0 means that the certificate must be self-signed. A default depth of 1 means that the incoming certificate can be either self-signed, or signed by a CA which is known to the `context`.

You should set this value according to your organization’s Public Key Infrastructure setup. Usually it should not be more than 5, but it should be large enough to allow the whole certificate chain to be examined.

Syntax

`setAuthenticationDepth(context, depth, serverOrClient)`

Parameters

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>context</code></td>
<td>The Security Context handle.</td>
</tr>
<tr>
<td><code>depth</code></td>
<td>Numeric value for verification depth.</td>
</tr>
<tr>
<td><code>serverOrClient</code></td>
<td>1 - Server</td>
</tr>
<tr>
<td></td>
<td>2 - Client</td>
</tr>
</tbody>
</table>

`setAuthenticationDepth Parameters`
The following table describes the status of each return code.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success.</td>
</tr>
<tr>
<td>1</td>
<td>Invalid Security Context handle.</td>
</tr>
<tr>
<td>2</td>
<td>Invalid depth (must be greater than or equal to 0).</td>
</tr>
<tr>
<td>3</td>
<td>Invalid value for serverOrClient (must be 1 or 2)</td>
</tr>
</tbody>
</table>
Generating a key pair

The `generateKey()` function generates a public key cryptography key pair and encrypts the private key. You should then put it into an external key file protected by the provided pass phrase. The protected private key can later be used by UniData and UniVerse SSL sessions (through `setPrivateKey()` to secure communication. The public key will not be encrypted.

The generated private key will be in PKCS #8 form and is encoded in either PEM or DER format specified by `format`. The generated public key is in standard form. If `keyLoc` is 1 (SSL_LOC_STRING), the resulting key is put into a dynamic array in `privKey` and `pubKey`. Otherwise they are put into OS level files specified by `privKey` and `pubKey`.

This function can generate two types of keys, RSA and DSA, specified by `algorithm`. The key length is determined by `keyLength` and must be in the range of 512 to 16384.

For DSA key generation, `paramFile` must be specified. If a parameter file is provided through `paramFile` and it contains valid parameters, the parameters are used to generate a new key pair. If the specified file does not exist or does not contain valid parameters, a new group of parameters will be generated and subsequently used to generate a DSA key pair. The generated parameters are then written to the specified parameter file. Since DSA parameter generation is time consuming, we recommend that a parameter file be used to generate multiple DSA key pairs.

To make sure the private key is protected, a pass phrase MUST be provided. A one-way hash function will be used to derive a symmetric key from the pass phrase to encrypt the generated key. When installing the private key into a security context with the `setPrivateKey()` function, or generating a certificate request with the `generateCertRequest()` function, this pass phrase must be supplied to gain access to the private key.

**Syntax**

```
generateKey(privKey, pubKey, format, keyLoc, algorithm, keyLength, passPhrase, paramFile)
```
Parameters

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>privKey</td>
<td>A string storing the generated private key or name of the file storing the generated private key.</td>
</tr>
<tr>
<td>pubKey</td>
<td>A string storing the generated public key or name of the file storing the generated public key.</td>
</tr>
<tr>
<td>format</td>
<td>1 - PEM (SSL_FMT_PEM)  2 - DER (SSL_FMT_DER)</td>
</tr>
<tr>
<td>keyLoc</td>
<td>1 - Put the key into string privKey/pubKey. (SSL_LOC_STRING)  2 - Put the key into a file. (SSL_LOC_FILE)</td>
</tr>
<tr>
<td>algorithm</td>
<td>1 - RSA (SSL_KEY_RSA)  2 - DSA (SSL_KEY_DSA)</td>
</tr>
<tr>
<td>keyLength</td>
<td>Number of bits for the generated key. Between 512 and 16384.</td>
</tr>
<tr>
<td>passPhrase</td>
<td>A string storing the pass phrase to protect the private key.</td>
</tr>
<tr>
<td>paramFile</td>
<td>A parameter file needed by DSA key generation.</td>
</tr>
</tbody>
</table>

generateKey Parameters

The following table describes the status of each return code.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success.</td>
</tr>
<tr>
<td>1</td>
<td>Key pair cannot be generated.</td>
</tr>
<tr>
<td>2</td>
<td>Unrecognized key file format.</td>
</tr>
<tr>
<td>3</td>
<td>Unrecognized encryption algorithm.</td>
</tr>
<tr>
<td>4</td>
<td>Unrecognized key type or invalid key length (must be between 512 and 16384).</td>
</tr>
<tr>
<td>5</td>
<td>Empty pass phrase.</td>
</tr>
<tr>
<td>Return Code</td>
<td>Status</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>6</td>
<td>Invalid DSA parameter file.</td>
</tr>
<tr>
<td>7</td>
<td>Random number generator cannot be seeded properly.</td>
</tr>
<tr>
<td>8</td>
<td>Private key cannot be written.</td>
</tr>
</tbody>
</table>

Return Code Status (Continued)
Creating a certificate request

The `createCertRequest()` function generates a PKCS #10 certificate request from a private key in PKCS #8 form and its corresponding public key, and a set of user specified data. The request can be sent to a CA or used as a parameter to `createCertificate()`, as described in “Creating a certificate” to obtain an X.509 public key certificate.

The private key and its format, type, algorithm and pass phrase are specified the same as described in the “Generating a key pair” section.

The certificate request typically contains the information described in the following table.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>Defaults to 0.</td>
</tr>
<tr>
<td>Subject</td>
<td>The certificate holder’s identification data. This includes country, state/province, locality (city), organization, unit, common name, email address, etc.</td>
</tr>
<tr>
<td>Public key</td>
<td>The key’s algorithm (RSA or DSA) and value.</td>
</tr>
<tr>
<td>Signature</td>
<td>The requester’s signature, (signed by the private key).</td>
</tr>
</tbody>
</table>

The subject data must be provided by the requester through the dynamic array, `subjectData`. It contains @FM separated attributes in the form of “attri=value”.
The commonly used *subjectData* attributes are described in the following table.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Country</td>
<td>C=US</td>
</tr>
<tr>
<td>ST</td>
<td>State</td>
<td>ST=Colorado</td>
</tr>
<tr>
<td>L</td>
<td>Locality</td>
<td>L=Denver</td>
</tr>
<tr>
<td>O</td>
<td>Organization</td>
<td>O=MyCompany</td>
</tr>
<tr>
<td>OU</td>
<td>Organization Unit</td>
<td>OU=Sales</td>
</tr>
<tr>
<td>CN</td>
<td>Common Name</td>
<td>CN=<a href="mailto:service@mycompany.com">service@mycompany.com</a></td>
</tr>
<tr>
<td>Email</td>
<td>Email Address</td>
<td>Email=<a href="mailto:john.doe@xyz.com">john.doe@xyz.com</a></td>
</tr>
</tbody>
</table>

**subjectData Attributes**

Be aware that since the purpose of a certificate is to associate the certificate’s bearer with his or her identity, in order for the outside party to verify the identity of the certificate’s holder, some recognizable characteristics should be built between the holder and verifier. For example, it is a general practice that a server’s certificate uses its DNS name (such as myServer.com) as its common name (CN).

*Digest* specifies what algorithm is going to be used to generate a Message Authentication Code (MAC) which will then be signed with the provided private key as a digital signature as part of the request. Currently only two algorithms, MD5 and SHA1, are supported. SHA1 is recommended.

*Note:* For a DSA request, SHA1 will always be used.

For more information on certificates, see the references for X.509 and PKCS #10, and PKCS #12.

**Syntax**

```plaintext
createCertRequest(key, inFormat, keyLoc, algorithm, digest, passPhrase, subjectData, outFile, outFormat)
```
**Parameters**

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>key</code></td>
<td>A string containing the key or name of the file storing the key.</td>
</tr>
</tbody>
</table>
| `inFormat` | The key format.  
1 - PEM (SSL_FMT_PEM)  
2 - DER (SSL_FMT_DER) |
| `keyLoc`  | 1 - Put the key into string privKey/pubKey (SSL_LOC_STRING).  
2 - Put the key into a file (SSL_LOC_FILE). |
| `algorithm` | 1 - RSA (SSL_KEY_RSA)  
2 - DSA (SSL_KEY_DSA) |
| `digest`  | 1 - MD5(SSL_DIGEST_MD5)  
2 - SHA1 (SSL_DIGEST_SHA1) |
| `passPhrase` | A string storing the pass phrase to protect the private key. |
| `subjectData` | The Requester’s identification information. |
| `outFile` | A string containing the path name of the file where the certificate request is stored. By convention, this file should have a .req as its extension. |
| `outFormat` | The generated certificate format.  
1 - PEM (SSL_FMT_PEM)  
2 - DER (SSL_FMT_DER) |

**createCertRequest Parameters**

The following table describes the status of each return code.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success.</td>
</tr>
<tr>
<td>1</td>
<td>Private key file cannot be opened.</td>
</tr>
<tr>
<td>2</td>
<td>Unrecognized key or certificate format.</td>
</tr>
<tr>
<td>3</td>
<td>Unrecognized key type.</td>
</tr>
</tbody>
</table>

**Return Code Status**
<table>
<thead>
<tr>
<th>Return Code</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Unrecognized encryption algorithm.</td>
</tr>
<tr>
<td>5</td>
<td>Unrecognized key (corrupted key or algorithm mismatch).</td>
</tr>
<tr>
<td>6</td>
<td>Invalid pass phrase.</td>
</tr>
<tr>
<td>7</td>
<td>Invalid subject data (illegal format or unrecognized attribute, and so forth.).</td>
</tr>
<tr>
<td>8</td>
<td>Invalid digest algorithm.</td>
</tr>
<tr>
<td>9</td>
<td>Output file cannot be created.</td>
</tr>
<tr>
<td>99</td>
<td>Cert Request cannot be generated.</td>
</tr>
</tbody>
</table>
Creating a certificate

The `createCertificate()` function generates a certificate. The certificate can either be a self-signed certificate as a root CA that can then be used later to sign other certificates, or it can be a CA signed certificate. The generated certificate conforms to X509V3 standard.

As input, a certificate request file must be specified by `req`. Three actions can be chosen:

- Creating a self-signed root certificate
- Creating an intermediate CA certificate
- Creating a server/client certificate

For self-signed root certificates, a key file must be specified by `signKey`. For the other two actions, a CA certificate file must be specified by `CAcert`, along with the CA private key specified by `signKey`. The output certificate file is specified by `certOut`. The format for these files should all be in PEM format.

The difference between intermediate CA certificates and server/client certificates is that the intermediate CA certificate can be used to sign other certificates, while a server/client cannot be used to sign other certificates.

The `days` parameter specifies the number of days the generated certificate is valid. The certificate is valid starting from the current date until the number of days specified expires. If an invalid `days` value is provided (0 or negative) the default value of 365 (one year) will be used.

This function is provided mainly for the purpose of enabling application development and testing. As such, the certificate generated contains only a minimum amount of information and does not allow any extensions specified by the X509 standard and that are supported by many other vendors. You can use the U2 Extensible Administration tool We recommend that you implement a complete PKI solution partnered with a reputed PKI solution vendor.

Syntax

```
createCertificate(action, req, signKey, keyPass, CAcert, days, extensions, certOut)
```
Parameters

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| action    | 1 - Creating a self-signed root certificate (SSL_CERT_SELF_SIGN)  
2 - Creating an intermediate CA certificate (SSL_CERT_CA_SIGN)  
3 - Creating a server/client certificate (SSL_CERT_LEAF_SIGN) |
| req       | A string containing the certificate request file name. |
| signKey   | A String containing the private key file name. |
| keyPass   | A string containing the pass phrase to protect the private key. |
| CAcert    | A string containing the CA certificate. |
| days      | The number of days the certificate is valid for. The default is 365 days. |
| extensions| A string containing extension specifications. |
| certOut   | A string containing the generated certificate file. |

createCertificate Parameters

The following table describes the status of each return code.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success.</td>
</tr>
<tr>
<td>1</td>
<td>Cannot read certificate request file.</td>
</tr>
<tr>
<td>2</td>
<td>Cannot read the key file.</td>
</tr>
<tr>
<td>3</td>
<td>Cannot read the CA certificate file.</td>
</tr>
<tr>
<td>4</td>
<td>Cannot generate the certificate.</td>
</tr>
</tbody>
</table>

Return Code Status
Setting a random seed

The `setRandomSeed()` function generates a random seed file from a series of source files and sets that file as the default seed file for the supplied security context.

The strength of cryptographic functions depends on the true randomness of the keys. This function generates and sets the random seed file used by many of the UniData and UniVerse cryptographic functions. By default, UniData and UniVerse use the .rnd file in your UniData or UniVerse application’s current UDTHOME or UVHOME directory. You can override the default by calling this function.

**Note:** Your application or a U2 server may be running under a system directory such as C:\WINDOWS\system32 or /usr/uv112, which may not allow the file to be created. To avoid this situation, you should always specify a location that allows random files to be created.

The random seed file is specified by `outFile`, which is generated based on source files specified in `inFiles`. For Windows platforms, multiple files must be separated by “;” (a semi-colon). For Unix platforms, multiple files must be separated by “:” (a colon).

The `length` parameter specifies how many bytes of seed data should be generated.

If no source is specified in the `inFiles` parameter, then the `outFile` parameter must already exist.

If context is not specified, the seed file will be used as a global seed file that applies to all cryptographic functions. However, a seed file setting in a particular security context will always override the global setting.

**Syntax**

```plaintext
setRandomSeed(inFiles, outFile, length, context)
```
Parameters

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inFiles</td>
<td>A string containing source file names.</td>
</tr>
<tr>
<td>outFiles</td>
<td>A string containing the generated seed file.</td>
</tr>
<tr>
<td>length</td>
<td>The number of bytes that should be generated (the default is 1024 if less than 1024 is specified).</td>
</tr>
<tr>
<td>context</td>
<td>The Security Context handle.</td>
</tr>
</tbody>
</table>

setRandomSeed Parameters

The following table describes the status of each return code.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success.</td>
</tr>
<tr>
<td>1</td>
<td>Invalid parameter(s).</td>
</tr>
<tr>
<td>2</td>
<td>Random file generation error.</td>
</tr>
<tr>
<td>3</td>
<td>Random file set error.</td>
</tr>
</tbody>
</table>

Return Code Status
Analyzing a certificate

The `analyzeCertificate()` function decodes a certificate and puts plain text into the `result` parameter. The `result` parameter will then contain such information as the subject name, location, institute, issuer, public key, other extensions and the issuer’s signature.

Syntax

```pascal
analyzeCertificate(cert, format, result, p12pass)
```

Parameters

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cert</code></td>
<td>A string containing the certificate file name.</td>
</tr>
</tbody>
</table>
| `format`  | 1 - PEM (SSL_FMT_PEM)  
           | 2 - DER (SSL_FMT_DER)  
           | 3 - PKCS #12 (SSL_FMT_P12) |
| `result`  | A dynamic array containing parsed cert data, in ASCII format. |
| `p12pass` | Optional. Sets a password on the PKCS#12 file. This parameter should only be included if using a PKCS #12 certificate that has a password. Otherwise the parameter should be omitted. This feature was added at UniVerse 11.2.3. |
The following table describes the status of each return code.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success.</td>
</tr>
<tr>
<td>1</td>
<td>Failed to open cert file.</td>
</tr>
<tr>
<td>2</td>
<td>Invalid format.</td>
</tr>
<tr>
<td>3</td>
<td>Unrecognized cert.</td>
</tr>
<tr>
<td>4</td>
<td>Other errors.</td>
</tr>
</tbody>
</table>

| Return Code Status |
Encoding and cryptographic functions

This section describes the available encoding and cryptographic functions available through UniVerse BASIC.
Encoding data

The main purpose of data encoding is to allow the use of non-ASCII characters in a body of data such that the data can be transferred undisturbed by underlying protocols or displayed without causing problems.

The ENCODE() function performs data encoding on input data. Base 64 encoding is designed to represent arbitrary sequences of octets that do not need to be humanly readable. A 64-character subset of US-ASCII is used, enabling 6-bits to be represented per printable character. The subset has the important property that it is represented identically in all versions of ISO646, including US-ASCII, and all characters in the subset are also represented identically in all versions of EBCDIC. The encoding process represents 24-bit groups of input bits as output strings of 4 encoded characters.

There are two BASE64 encoding modes, default and one-line. In default mode, the encoded output stream must be represented in lines of no more than 76 characters each. All line breaks must be ignored by the decoding process. All other characters not found in the 64-character subset should trigger a warning by the decoding process. In one-line mode, the data is a continuous stream of the allowed ASCII characters without any line breaks.

URL encoding performs encoding or decoding on the data passed to the function according to the RFC 3986 standard. This algorithm changes all characters that need to be encoded to the “percent-escaped” form, such as changing “=” to “%3D” when encoding the data, then back to ASCII characters when decoding.

Syntax

\[
\text{ENCODE(algorithm, action, data, dataLoc, result, resultLoc)}
\]
Parameters

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>algorithm</td>
<td>A string containing the encode method name. The three valid values are:</td>
</tr>
<tr>
<td></td>
<td>• SSL_BASE64</td>
</tr>
<tr>
<td></td>
<td>• SSL_BASE64_ONELINE</td>
</tr>
<tr>
<td></td>
<td>• URLENCODE</td>
</tr>
<tr>
<td>action</td>
<td>1 - Encode (SSL_ENCODE)</td>
</tr>
<tr>
<td></td>
<td>2 - Decode (SSL_DECODE)</td>
</tr>
<tr>
<td>data</td>
<td>Data or the name of the file containing the data to be encoded or decoded.</td>
</tr>
<tr>
<td>dataLoc</td>
<td>1 - Data in a string (SSL_LOC_STRING)</td>
</tr>
<tr>
<td></td>
<td>2 - Data in a file (SSL_LOC_FILE)</td>
</tr>
<tr>
<td>result</td>
<td>Encoded or decoded data or the name of the file storing the processed data.</td>
</tr>
<tr>
<td>resultLoc</td>
<td>1 - Result in a string (SSL_LOC_STRING)</td>
</tr>
<tr>
<td></td>
<td>2 - Result in a file (SSL_LOC_FILE)</td>
</tr>
</tbody>
</table>

ENCODE Parameters

The following table describes the status of each return code.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success.</td>
</tr>
<tr>
<td>1</td>
<td>Unsupported algorithm.</td>
</tr>
<tr>
<td>2</td>
<td>Invalid parameters (invalid data or result location type, and so forth.).</td>
</tr>
<tr>
<td>3</td>
<td>The data cannot be read.</td>
</tr>
<tr>
<td>4</td>
<td>The data cannot be encoded or decoded.</td>
</tr>
</tbody>
</table>
Encrypting data

The ENCRYPT() function performs symmetric encryption operations. Various block and stream symmetric ciphers can be called through this function. The supported ciphers are listed below.

Ciphers are specified by algorithm and are not case sensitive. Base64 encoding and decoding can be specified with the action parameter. If encoding is specified, the encrypted data is Base64 encoded before being entered into result. If decoding is specified, the data is Base64 decoded before being encrypted. The data and its location are specified by data and dataLoc, respectively. Key can be explicitly specified or read from a file, or, alternatively, derived on the fly, specified by keyAction, in which case the key string is used as a pass phrase to derive the actual key. The encrypted or decrypted data is put into the dynamic array result, or a file, as specified by resultLoc.

Salt is used to provide more security against certain kinds of cryptanalysis attacks, such as dictionary attacks. If an empty salt is supplied, an internally generated salt will be used in deriving the key. Salt is ignored when action is set to decrypt. IV (Initialization Vector) is used to provide additional security to some block ciphers. It does not need to be secret but should be fresh, meaning different for each encrypted data. If an actual key is supplied, IV is generally needed. However if the encryption key is to be derived from a pass phrase, IV is generated automatically. Both salt and IV must be provided in hexadecimal format.

You have two ways to supply key and IV to the ENCRYPT() function. You can supply the actual key and IV, or you can supply a seed (also called a password) and optionally a salt, then let UniVerse derive the actual key and IV. When you do the latter, you have multiple options to tell UniVerse how to derive the key and IV, some of which will allow you to exchange encrypted data between UniVerse and third-party products.

Note: Some ciphers are more secure than others. For more details, please refer to the publications listed under “Additional reading”.

The following ciphers are supported. All cipher names are not case sensitive.

Note: Due to export restrictions, all ciphers may not be available for a specific distribution.
### 56-bit key DES algorithms:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>des-cbc</td>
<td>DES in CBC mode</td>
</tr>
<tr>
<td>des</td>
<td>Alias for des-cbc</td>
</tr>
<tr>
<td>des-cfb</td>
<td>DES in CFB mode</td>
</tr>
<tr>
<td>des-ofb</td>
<td>DES in OFB mode</td>
</tr>
<tr>
<td>des-ecb</td>
<td>DES in ECB mode</td>
</tr>
</tbody>
</table>

### 112-bit key DES algorithms:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>des-ede-cbc</td>
<td>Two key triple DES EDE in CBC mode</td>
</tr>
<tr>
<td>des-ede</td>
<td>Alias for des-ede-cbc</td>
</tr>
<tr>
<td>des-ede-cfb</td>
<td>Two key triple DES EDE in CFB mode</td>
</tr>
<tr>
<td>des-ede-ofb</td>
<td>Two key triple DES EDE in OFB mode</td>
</tr>
</tbody>
</table>

### 128-bit key AES algorithms:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aes-128-cbc</td>
<td>Alias for aes-128</td>
</tr>
<tr>
<td>aes-128-ecb</td>
<td>Alias for aes-128</td>
</tr>
</tbody>
</table>
### 168-bit key DES algorithms:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>des-ede3-cbc</td>
<td>Three key triple DES EDE in CBC mode</td>
</tr>
<tr>
<td>des-ede3</td>
<td>Alias for des-ede3-cbc</td>
</tr>
<tr>
<td>des3</td>
<td>Alias for des-ede3-cbc</td>
</tr>
<tr>
<td>des-ede3-cfb</td>
<td>Three key triple DES EDE in CFB mode</td>
</tr>
<tr>
<td>des-ede3-ofb</td>
<td>Three key triple DES EDE in OFB mode</td>
</tr>
</tbody>
</table>

### 192-bit AES algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aes-192-cbc</td>
<td>Alias for aes-192</td>
</tr>
<tr>
<td>aes-192-ecb</td>
<td>Alias for aes-192</td>
</tr>
</tbody>
</table>

### 256-bit AES algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aes-256-cbc</td>
<td>Alias for aes-256</td>
</tr>
<tr>
<td>aes-256-ecb</td>
<td>Alias for aes-256</td>
</tr>
</tbody>
</table>
### RC2 algorithms:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rc2-cbc</td>
<td>128-bit RC2 in CBC mode</td>
</tr>
<tr>
<td>rc2</td>
<td>Alias for rc2-cbc</td>
</tr>
<tr>
<td>rc2-cfb</td>
<td>128-bit RC2 in CBC mode</td>
</tr>
<tr>
<td>rc2-ecb</td>
<td>128-bit RC2 in ECB mode</td>
</tr>
<tr>
<td>rc2-ofb</td>
<td>128-bit RC2 in OFB mode</td>
</tr>
<tr>
<td>rc2-64-cbc</td>
<td>64-bit RC2 in CBC mode</td>
</tr>
<tr>
<td>rc2-40-cbc</td>
<td>40-bit RC2 in CBC mode</td>
</tr>
</tbody>
</table>

### RC4 algorithms:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rc4</td>
<td>128-bit RC4</td>
</tr>
<tr>
<td>rc4-40</td>
<td>40-bit RC4</td>
</tr>
</tbody>
</table>

### Blowfish algorithms (variable key size, typically 128 bits)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bf</td>
<td>BF</td>
</tr>
<tr>
<td>bf-cbc</td>
<td>BF in CBC mode</td>
</tr>
<tr>
<td>bf-cfb</td>
<td>BF in CFB mode</td>
</tr>
<tr>
<td>bf-ecb</td>
<td>BF in ECB mode</td>
</tr>
<tr>
<td>bf-ofb</td>
<td>BF in OFB mode</td>
</tr>
</tbody>
</table>
CAST algorithms (variable key size, typically 128 bits)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cast</td>
<td>CAST</td>
</tr>
<tr>
<td>cast-cbc</td>
<td>CAST in CBC mode</td>
</tr>
<tr>
<td>cast5-cbc</td>
<td>CAST5 in CBC mode</td>
</tr>
<tr>
<td>cast5-cfb</td>
<td>CAST5 in CFB mode</td>
</tr>
<tr>
<td>cast5-ecb</td>
<td>CAST5 in ECB mode</td>
</tr>
<tr>
<td>cast5-ofb</td>
<td>CAST5 in OFB mode</td>
</tr>
</tbody>
</table>

CAST algorithms

Syntax

ENCRIPT(algorithm, action, data, dataLoc, key, keyLoc, keyAction, salt, IV, result, resultLoc)
Parameters

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>algorithm</strong></td>
<td>A string containing the cipher name described in the above tables.</td>
</tr>
</tbody>
</table>
| **action** | 1 - Encrypt (SSL_ENCRYPT)  
2 - Base64 encode after encryption (SSL_ENCRYPT_ENCODE)  
3 - Decrypt (SSL_DECRYPT)  
4 - Base64 decode before encryption (SSL_DECODE_DECRYPT)  
5 - One-line Base64 encode after encryption (SSL_ENCRYPT_ENCODE_H)  
6 - One-line Base64 decode before decryption (SSL_DECODE_DECRYPT_H) |
| **data** | Data or the name of the file containing the data to be processed. |
| **dataLoc** | 1 - Data in a string (SSL_LOC_STRING)  
2 - Data in a file (SSL_LOC_FILE) |
| **key** | The actual key (password) or file name containing the key. |
| **keyLoc** | 1 - Key in a string (SSL_LOC_STRING)  
2 - Key in file (SSL_LOC_FILE) |
The following table describes the status of each return code.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success.</td>
</tr>
<tr>
<td>1</td>
<td>Invalid cipher.</td>
</tr>
<tr>
<td>2</td>
<td>Invalid parameters (location/action value is out of range, etc.).</td>
</tr>
<tr>
<td>3</td>
<td>The data cannot be read.</td>
</tr>
</tbody>
</table>
If you specify the KeyAction value as 3 (SSL_KEY_ACTUAL_OPENSSL), the key string and IV string must be in hexadecimal format with correct length for the algorithm you specify. You can exchange encrypted data with third-party products.

If you specify the KeyAction value as 2 (SSL_KEY_ACTUAL), a specific salt and algorithm will be used to derive the actual key and IV. The result cannot be exchanged with third-party products.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>The key cannot be derived.</td>
</tr>
<tr>
<td>5</td>
<td>Base 64 encoding/decoding error.</td>
</tr>
<tr>
<td>6</td>
<td>Encryption/decryption error.</td>
</tr>
</tbody>
</table>

Return Code Status (Continued)
Generating a message digest

The **DIGEST()** function generates a message digest of supplied data. A message digest is the result of a one-way hash function (digest algorithm) performed on the message. Message digest has the unique properties that a slight change in the input will result in a significant difference in the resulting digest. Therefore, the probability of two different messages resulting in the same digest (collision) is very unlikely. It is also virtually impossible to reverse to the original message from a digest. Message digest is widely used for digital signatures and other purposes.

The desired digest algorithm is specified in `algorithm`. The two supported digest algorithms are **MD5** (Message Digest 5, 128-bit) and **SHA1** (Secure Hash Algorithm 1, 160-bit). Data and its location are specified by `data` and `dataLoc`, respectively. The arrived digest will be put into a dynamic array in `result`. Since digest is short and has a fixed length, it is always put into a string and no file option is provided. The result is always in binary format.

**Note:** `DIGEST` data is arbitrary binary data and may contain UniVerse delimiters. If you do not want the data to contain delimiters, use the `ENCODE()` function to perform BASE64 encoding.

### Syntax

```
DIGEST(algorithm, data, dataLoc, result)
```

### Parameters

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>algorithm</code></td>
<td>A string containing the digest algorithm name (either “MD5” or “SHA1”).</td>
</tr>
</tbody>
</table>

**DIGEST Parameters**
The following table describes the status of each return code.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success.</td>
</tr>
<tr>
<td>1</td>
<td>Unsupported digest algorithm.</td>
</tr>
<tr>
<td>2</td>
<td>The data file cannot be read.</td>
</tr>
<tr>
<td>3</td>
<td>Message digest cannot be obtained.</td>
</tr>
<tr>
<td>4</td>
<td>Invalid parameters.</td>
</tr>
</tbody>
</table>

The following table describes the status of each return code.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Data or the name of the file containing the data to be digested.</td>
</tr>
<tr>
<td>dataLoc</td>
<td>1 - Data in a string (SSL_LOC_STRING)</td>
</tr>
<tr>
<td></td>
<td>2 - Data in a file (SSL_LOC_FILE)</td>
</tr>
<tr>
<td>result</td>
<td>A string to store the digest result.</td>
</tr>
</tbody>
</table>
Generating a digital signature

A digital signature is generally created over a piece of data or document by some cryptographic algorithm and used to prove the authenticity and integrity of the data or document, for example, the recipient of the data with a valid digital signature has reason to believe that the data is from a trusted sender and its contents are not modified.

The SIGNATURE() function generates a digital signature or verifies a signature using the supplied key.

The algorithm parameter specifies the digest algorithm used to construct the signature. The supported algorithms are MD5 and SHA1. There are four actions that can be specified: RSA-Sign, RSA-Verify, DSA-Sign, and DSA-Verify. Note that if DSA is chosen, only SHA1 can be specified in algorithm.

The data to be signed or verified against a signature can be supplied either directly in data, or read from a file whose names is in data.

For signing action, a private key should be specified. For verification, a public key is usually expected. However, a private key is also accepted for verification purposes. Key can be either in PEM or DER format. If a private key is password protected, the password must be supplied with pass.

For verification, key can also contain a certificate or name of a certificate file. A signature is expected in sigIn.

For signing action, the generated signature is put into result.

Syntax

```
SIGNATURE(algorithm, action, data, dataLoc, key, keyLoc, keyFmt, pass, sigIn, result, p12pass)
```
## Parameters

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>algorithm</strong></td>
<td>The digest algorithm used for signing or verification (must be either &quot;MD5&quot; or &quot;SHA1&quot;).</td>
</tr>
</tbody>
</table>
| **action** | 1 - RSA-Sign (SSL_RSA_SIGN)  
2 - RSA-Verify (SSL_RSA_VERIFY)  
3 - DSA-Sign (SSL_DSA_SIGN)  
4 - DSA-Verify (SSL_DSA_VERIFY) |
| **data** | Data or the name of the file containing the data to be signed or verified. |
| **dataLoc** | 1 - Data in a string (SSL_LOC_STRING)  
2 - Data in a file (SSL_LOC_FILE) |
| **key** | The key or the name of the file containing the key to be used to sign or verify. In the case of verification, key can be a certificate string or a file. |
| **keyLoc** | 1 - Key is in a string (SSL_LOC_STRING)  
2 - Key is in a file (SSL_LOC_FILE)  
3 - Key is in a certificate for verification. (Currently, no constant is defined) |
| **keyFmt** | 1 - PEM (SSL_FMT_PEM)  
2 - DER (SSL_FMT_DER)  
3 - PKCS #12 (SSL_FMT_P12) |
| **pass** | A string containing the pass phrase for the private key. |
| **sigIn** | A string containing a digital signature. |
| **result** | A generated signature or a file to store the signature. |
| **p12pass** | Optional. Sets a password on the PKCS#12 file. This parameter should only be included if using a PKCS #12 certificate that has a password. Otherwise the parameter should be omitted. This feature was added at UniVerse 11.2.3. |

### SIGNATURE Parameters
The following table describes the status of each return code.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success.</td>
</tr>
<tr>
<td>1</td>
<td>Unsupported digest algorithm.</td>
</tr>
<tr>
<td>2</td>
<td>The data cannot be read.</td>
</tr>
<tr>
<td>3</td>
<td>Message digest cannot be obtained.</td>
</tr>
<tr>
<td>4</td>
<td>Invalid parameters.</td>
</tr>
<tr>
<td>5</td>
<td>Key cannot be read or is in the wrong format / algorithm.</td>
</tr>
<tr>
<td>6</td>
<td>Incorrect Password.</td>
</tr>
<tr>
<td>7</td>
<td>Signature cannot be generated.</td>
</tr>
<tr>
<td>8</td>
<td>Signature cannot be verified.</td>
</tr>
</tbody>
</table>

Return Code Status
Additional reading

Due to the amount of terminology regarding cryptography in general and SSL in particular, interested readers may refer to the following publications.

“Applied Cryptography”, by Bruce Schneier

“Internet Cryptography”, by Richard E. Smith

“SSL and TLS: Designing and Building Secure Systems”, by Eric Rescorla
Chapter 2: Configuring SSL through the U2 Extensible Administration Tool

About SSL .................................................. 2-2
Accessing SSL configuration tasks .................. 2-3
Creating a certificate signing request .............. 2-5
Creating a certificate .................................. 2-12
Creating a security context record ................ 2-24
Configuring SSL for UniVerse servers .......... 2-46
About SSL

Secure Sockets Layer (SSL) is a transport layer protocol that provides a secure channel between two communicating programs over which you can send arbitrary application data securely. It is by far the most widely deployed security protocol used on the World Wide Web.

SSL provides server authentication, encryption, and message integrity. It can also support client authentication.

UniVerse currently supports CallHTTP and the Sockets API. SSL support is important for both of these protocols in order to deploy commercial applications and securely process sensitive data, such as credit card transactions.

This chapter assumes that users who want to use SSL have a basic knowledge of public key cryptography.
Accessing SSL configuration tasks

Use the U2 Extensible Administration Tool to configure SSL.

To access SSL configuration tasks:

1. Start the U2 Extensible Administration Tool as follows:
   If you installed this component to the default program group, choose Start > All Programs > Rocket U2 > Extensible Administration Tool.

2. In the Admin Tasks view, double-click SSL Configuration.
The **Configure SSL for Servers** editor opens in the upper center pane by default.

The **Configure SSL for Servers** editor provides a tab for each of the following tasks:
- Creating a certificate signing request
- Creating a certificate
- Creating a security context record
- Configuring SSL for UniVerse servers
Creating a certificate signing request

Complete the following steps to generate an X.509 certificate signing request (CSR). You can send the CSR to a third-party certificate authority (CA) to obtain a certificate, or use it as input to the Generate SSL Certificate wizard.

To create a certificate signing request:

1. The Certificate Signing Request tab is displayed by default when you initially access the Configure SSL for Servers editor. If the Certificate Signing Request tab is not currently selected, click the tab to open it now.

2. In the Certificate Signing Request tab, click the Generate a Certificate Request button. This starts the Generate Certificate Signing Request Wizard.
3. The **Generate Certificate Signing Request** dialog box contains an introduction to this task. Click **Next** to continue.

4. Specify the file name and algorithm in the **Certificate Signing Request File** dialog box, as shown in the following example:

   ![Certificate Signing Request Dialog Box](image)

   In the **Certificate Signing Request File** text box, enter the full path of the operating system-level file to hold the certificate signing request, or click **Browse** to search for the file location.

   From the **Digest Algorithm** options, select the type of algorithm to be used for the certificate signing request:

   - **SHA1** – SHA1 hash function
   - **MD5** – MD5 hash function

   Click **Next** to continue.
5. Specify the properties for the certificate signing request in the Request Properties dialog box, as shown in the following example:

The available properties are:
- C – Country code
- ST – State or province
- L – Locality (city)
- O – Organization name
- OU – Organization unit
- CN – Common name
- Email – Email address

You must define values in the C (Country Code), O (Organization), and CN (Common Name) boxes.
Click Next to continue.
6. Select an option to use an existing key pair or generate a new key pair for the certificate signing request in the **Key Pair Selection** dialog box, as shown in the following example:

Select the appropriate key pair option, as follows:
- If you are using a previously generated key pair for the certificate signing request, select the **Use existing key pair** option.
- If you are creating a new key pair for the certificate signing request, select the **Generate new key pair** option.

Click **Next** to continue.
7. Select the key algorithm and key files in **Key Pair Info** dialog box, as shown in the following example:

From the **Key Algorithm** options, select the type of algorithm to be used in generating the key pair:

- **RSA** – RSA key algorithm
- **DSA** – digital signature algorithm

From the **Key Length** list, select the length of the key in bits. This is the primary measure of the cryptographic strength of the key. Valid values are multiples of 64, ranging from 512 to 16384.

From the **Key File Format** options, select the type of format to be used for private and public key files:

- **PEM** – Privacy Enhanced Mail format
- **DER** – Distinguished Encoding Rules format

If you chose DSA as the key algorithm, enter the full path of an existing parameter file in the **Parameter File** text box, or click **Browse** to search for the file location. UniVerse uses the
parameter file to generate a new key pair. If you leave this box empty, UniVerse uses its default parameter table to generate the key pair.

In the **Private Key File** text box, enter the name of the file in which you want to store the generated private key, or click **Browse** to search for the existing key if you selected **Use Existing Key Pair**.

In the **Public Key File** text box, enter the name of the file in which you want to store the generated public key, or click **Browse** to search for the existing key if you selected **Use Existing Key Pair**.

Click **Next** to continue.

8. Define a password for the private key file in the **Password** dialog box, as shown in the following example:

   ![Password dialog box]

   In the **Password for Private Key** text box, enter a password for the private key file. Reenter the password in the **Confirm Password** text box.

   Click **Next** to generate the certificate signing request file.
9. Check the message indicating status of generating the certificate signing request in the **Review Status and Finish** dialog box, as shown in the following example:

![Review Status and Finish dialog box](image)

- If the certificate signing request was created successfully, the dialog box contains the message “Certificate signing request was generated successfully,” or “Key pair and certificate signing request were generated successfully.”
- Otherwise, the message “Failed to create certificate signing request” is displayed. You can click the **Back** button to return to previous dialog boxes and correct the error.

Click **Finish** to close the **Generate Certificate Signing Request Wizard** and return to the **Configure SSL for Servers** editor.
Creating a certificate

You can create three types of certificates using the U2 Extensible Administration Tool:

- Self-signed certificates as a root certificate authority (CA) that can be used later to sign other certificates.
- Certificates either signed by a root CA certificate or by an intermediate CA certificate.
- Server or client certificates.

To create a certificate:

1. In the Configure SSL for Servers editor, select the Certificate tab, as shown in the following example:
2. In the Certificate tab, click the Generate a Certificate button to begin creating an X.509 certificate or to view details of an existing certificate. This starts the Generate SSL Certificate Wizard.

3. The Generate SSL Certificate dialog box contains an introduction to this task,

![Generate SSL Certificate Wizard](image)

This wizard guides you through the process of generating an X.509 certificate.
It can also show you the details of existing certificates.
To continue, click Next.

To continue, click the Next button.
4. Specify the name for a new or existing certificate file in the **Certificate File** dialog box, as shown in the following example:

In the **Certificate File** text box, enter a unique name or full path for a new certificate file you are creating, or click **Browse** to search for the location of an existing certificate file.

If you select an existing certificate file, you can click the **Show** button to view the details of the certificate.

Click **Next** to continue.
5. Specify the name of the certificate signing request file in the Certificate Signing Request File dialog box, as shown in the following example:

In the Certificate Signing Request File text box, enter the full path of the existing certificate signing request file to use in generating the certificate, or click Browse to search for the file location.

Click Next to continue.
6. Set the period for which the certificate is valid in the **Validity Period** dialog box, as shown in the following example:

From the **Validity Period** list, select the number of days for which certificate is valid. The certificate is valid starting from the current date until the specified number of days elapses. The default value is 365 days.

Click **Next** to continue.
7. Specify the type of certificate in **Certificate Type** dialog box, as shown in the following example:

![Certificate Type dialog box](image)

From the list of options, select the type of certificate to be created:
- Self-sign root certificate
- Intermediate CA certificate
- Server or client certificate

Click **Next** to continue.
8. If the certificate type is Intermediate CA certificate or Server/Client certificate, you can optionally define certificate extensions in the X.509 v3 Certificate Extensions dialog box, as shown in the following example:

Select the check box for any certificate extension you want to define:

- **Subject Alt Name** – The subject alternative name extension allows additional identities to be bound to the subject of the certificate.
- **Key Usage** – This extension defines the purpose of the key contained in the certificate and can be used to put certain restrictions on key usage.
- **Basic Constraints** – This extension indicates whether the subject of the certificate is a certificate authority (CA).
- **Subject Key Identifier** – This extension provides a means of identifying certificates that contain a particular public key.
- **Authority Key Identifier** – This extension identifies the public key corresponding to the private key used to sign the certificate.

When you select an extension, help text for that extension is displayed in the lower half of the dialog box along with the relevant options for defining the extension.

Click **Next** to continue.
9. Your next action depends on the certificate type.
   - If the certificate type is **Self-sign root certificate**, specify the private key file in **Private Key File** dialog box, as shown in the following example:

   ![Private Key File Dialog Box](image)

   In the **Private Key File** text box, enter the full path of the private key file used to generate the certificate signing request file, or click **Browse** to search for the file location.

   Click **Next** to continue.
If the certificate type is **Intermediate CA certificate** or **Server/Client certificate**, specify the signing certificate file and its associated private key file in the **Signing Certificate and Private Key File** dialog box, as shown in the following example:

In the **Signing Certificate File** text box, enter the full path of the signing certificate file to be used to sign the certificate you are generating, or click **Browse** to search for the file location.

In the **Private Key File** text box, enter the full path of the private key file used to generate the signing certificate file, or click **Browse** to search for the file location.

Click **Next** to continue.
10. Enter the password for the private key file in the **Password** dialog box, as shown in the following example:

   ![Password dialog box](image)

   In the **Password for Private Key** text box, enter the password for the private key file selected in the previous step, as follows:
   
   - If the certificate type is **Self-sign root certificate**, enter the password for the private key file used to generate the certificate signing request file.
   
   - If the certificate type is **Intermediate CA certificate** or **Server/Client certificate**, enter the password for the private key file used to generate the signing CA certificate file.

   Reenter the password in the **Confirm Password** text box.
Click **Next** to generate the certificate.

11. Check the message indicating the status of generating the certificate in the **Review Status and Finish** dialog box, as shown in the following example:

If the certificate was created successfully, the dialog box contains the message “Certificate was generated successfully.”

Otherwise, the message “Failed to create certificate” is displayed. You can click the **Back** button to return to previous dialog boxes and correct the error.

Click **Finish** to close the **Generate SSL Certificate Wizard** and return to the **Configure SSL for Servers** editor.
Creating a security context record

A security context record (SCR) is a data structure that holds all aspects of security characteristics that the application intends to associate with a secured connection.

To create a security context record:

1. In the Configure SSL for Servers editor, select the Security Context Record tab, as shown in the following example:

   ![Configure SSL for Servers editor example]

   In the Security Context Record tab, from the SCR Database list, select the database in which to create or view the security context record.

   The full path of the selected database is populated in the Path box.

   To add a new security context record, click the Add button. This starts the Security Context Record Wizard.

2. In the Security Context Record tab, from the SCR Database list, select the database in which to create or view the security context record.

   The full path of the selected database is populated in the Path box.

   To add a new security context record, click the Add button. This starts the Security Context Record Wizard.
3. The **Security Context Record (SCR)** dialog box contains an introduction to this task. Make sure you have generated the necessary keys and certificates before proceeding, then click Next to continue.

4. Specify an ID and security protocol version for the security context record in the **Security Context Record ID** dialog box, as shown in the following example:

   ![Security Context Record ID dialog box](image)

   In the **Security Context Record ID** box, enter a unique ID for the security context record.

   From the **SSL/TLS Version** list, select the appropriate security protocol version for the security context record. Valid versions are:
   - SSLv2
   - SSLv3
   - TLSv1

   **Note:** We recommend that you use SSLv3 or TLSv1.

   Click Next to continue.
5. Specify how the security context record is intended to be used in the **SCR Usage Type** dialog box, as shown in the following example:

From the SCR usage type options, select an option indicating whether the security context record is to be used by a server or by a client:

- **SCR for server** – Security context record is to be used by a server
- **SCR for client** – Security context record is to be used by a client

Click **Next** to continue.
6. Your next action depends on the usage type you selected for the security context record in the previous step.

- If you selected the **SCR for server** option, set authentication parameters for the server in the **Server Authentication Properties** dialog box, as shown in the following example:

![Server Authentication Properties dialog box](image)

From the **Authentication Depth** list, select a value to indicate how deeply UniVerse verifies before determining that a certificate is not valid. Depth is the maximum number of intermediate issuer certificates, or CA certificates, UniVerse must examine while verifying an incoming certificate. A depth of 0 indicates that the certificate must be self-signed. A depth of 1 means that the incoming certificate can be either self-signed or signed by a CA known to the security context record. The default value is 3.

In the **Trusted Peer Names** box, you can add one or more trusted peer names, as detailed below. UniVerse uses this list of peer names to determine whether to trust a peer. Trusted server/client names are stored in the context.
During the SSL handshake negotiations, the server sends its certificate to the client. By specifying trusted server names for the client, the client can control with which server(s) it should communicate. During the handshake, after the server certificate has been authenticated by way of the issuer (CA) certificate(s), UniVerse compares the subject name contained in the certificate against the trusted server names set in the context. If the server subject name matches one of the trusted peer names, communication continues; otherwise UniVerse does not establish the connection.

If the Client Authentication option is selected, a similar process is performed on the server to determine if a client is to be trusted. If no trusted peer name is set, any peer is considered legitimate.

To add trusted peer names, click the Add button. The Trusted Peer Name dialog box appears, as shown in the following example:

Enter one or more trusted peer names in a comma-separated list, and then click OK to return to the Server Authentication Properties dialog box.

From the Authentication Strength options, choose the level of security to be used in the authentication process:
■ **Generous** – The certificate need only contain the subject name (common name) that matches one specified by “PeerName” to be considered valid.

■ **Strict** – The incoming certificate must pass a number of checks, including signature check, expiry check, purpose check, and issuer check.

*Note: We recommend setting the Generous option only for development or testing purposes.*

If you want to set the certificate signing request to use client authentication, select the **Client Authentication** check box. With this check box selected, during the initial SSL handshake, the server sends the client authentication request to the client. It also receives the client certificate and performs authentication according to the issuer’s certificate (or certificate chain) set in the security context.

Click **Next** to continue.
If you selected the **SCR for client** option, set authentication parameters for the client in the **Client Authentication Properties** dialog box, as shown in the following example:

Enter authentication properties for the client in a similar manner to the instructions for entering server authentication properties above. However, note that the **Client Authentication** check box is unavailable.

Click **Next** to continue.
7. Select a certificate path rule option in the **Certificate Path Rule** dialog box, as shown in the following example:

The certificate path rule enables you to specify locations in which to search for certificates. From the list of options, choose a certificate path rule to specify the search path:

- **Default** – When you add a certificate to a security context record, the full path for that certificate is registered in the security context record. This path is derived from the current directory in which UniVerse is running. When the certificate is loaded into memory to establish the SSL connection, UniVerse by default uses this registered full path to retrieve the certificate.

- **Relative** – With this option, UniVerse looks for the certificate in the current directory in which UniVerse is running.
Note: Some of the UniVerse processes, such as the Telnet server processes, run from the system directory.

- **Path** – With this option, UniVerse uses the path you specify for loading the certificate added to this security context record. You can specify either an absolute path or a relative path.

- **Env** – If you select this option, enter an environment variable name in the **Env** text box. With this option, the UniVerse process first obtains the value of the environment variable you specify, and then uses that value as the path to load the certificates.

  Note: UniVerse evaluates the environment variable only when the first SSL connection is made. The value is cached for later reference.
8. You can load a certificate, or multiple certificates, into a security context for use as a UniVerse server certificate or client certificate. Alternatively, you can specify a directory that contains the certificates to use as a certificate authority (CA) certificate to verify incoming certificates or act as a revocation list, checking for expired or revoked certificates.

The purpose of a certificate is to bind the name of an entity with its public key. It is basically a means of distributing public keys. A certificate always contains the following three pieces of information:

- Name
- Public Key
- Digital signature signed by a trusted third party called a certificate authority (CA) with its private key.

If you have the public key of the CA (contained in the CA certificate), you can verify that the certificate is authentic.

SSL protocol specifies that when two parties start a handshake, the server must always send its certificate to the client for authentication. It may also require the client to send its certificate to the server for authentication. UniVerse servers that act as HTTP clients are not required to maintain a client certificate. UniVerse applications that act as SSL socket servers must install a server certificate. UniObjects for Java servers and Telnet servers also require server certificates.

There can be only one server/client certificate per specific security context record. Adding a new certificate automatically replaces an existing certificate. However, for issuer certificates, UniVerse chains a new one with existing certificates so UniVerse applications can perform chained authentication.

If the issuer certificate is in PEM format, it can contain multiple certificates by concatenating certificates together.

Note: All certificates that form an issuer chain must be of the same type.

Your next action depends on whether you are creating a security context record to be used by a server or by a client, as indicated by the SCR usage type you selected in a previous step.
If you selected the **SCR for server** option, specify the server certificate file in the **Server Certificate** dialog box, as shown in the following example:

In the **Server Certificate File** text box, enter the full path of the file containing the server certificate, or click **Browse** to search for the file location.

From the **Certificate File Format** options, select the file format for the server certificate:

- **PEM** – Base64 encoded format
- **DER** – ASN.1 binary format
- **PKCS #12** - PKCS #12 format

9. Click **Next** to continue.
If you selected the **SCR for client** option, specify the client certificate file in the **Client Certificate** dialog box, as shown in the following example:

In the **Client Certificate File** text box, enter the full path of the file containing the client certificate, or click **Browse** to search for the file location.

From the **Certificate File Format** options, select the file format of the client certificate:

- **PEM** – Base64 encoded format
- **DER** – ASN.1 binary format
- **PKCS #12** - PKCS #12 format

10. Click **Next** to continue.
11. Specify the private key file associated with server or client certificate in the **Private Key File** dialog box, as shown in the following example:

![Private Key File dialog box](image)

In the **Private Key File** text box, enter the full path of the file that contains the private key associated with the server or client certificate, or click **Browse** to search for the file location.

In the **Password for Private Key** text box, enter the password for the private key file. Reenter the password in the **Confirm Password** text box.

From the **Private Key Format** options, select the format of the private key file:

- **PEM** – Base64 encoded format
- **DER** – ASN.1 binary format
- **PKCS #12** - PKCS #12 format

12. Click **Next** to continue.
13. Optionally, you can add one or more CA certificates to the security context record in the **CA Certificates** dialog box, as shown in the following example.

![CA Certificates dialog box](image)

To add a CA certificate, click the **Add** button.
The **Add CA Certificate** dialog box appears, as shown in the following example:

In the **Certificate File** box, enter the full path of the file containing the CA certificate, or click **Browse** to search for the file location. From the **Format** options, select the format for the CA certificate.

- **PEM** – Base64 encoded format
- **DER** – ASN.1 binary format
- **PKCS #12** – PKCS #12 format

Click **OK** to add the certificate and return to the **CA Certificates** dialog box.

Repeat this step for each CA certificate you want to add.

14. Click **Next** to continue.
15. By default, UniVerse uses the random (.rnd) file in the current account.

*Note:* The strength of cryptographic functions depends on the true randomness of the keys. We recommend that you use the default .rnd file.

Optionally, you can specify a random file other than the default in the **Random File** dialog box, as shown in the following example:

In the **Random File** text box, enter the full path of the random file to be used for the security context record, or click **Browse** to search for the file location.

Alternatively, you can generate a new random file from a series of seed source files and set that file as the default random file for the security context record.
To create a new random file, click the **New Random File** button. The **New Random File** dialog box appears, as shown in the following example:

In the **File Name** box, enter a name for the new random file, or click **Browse** to select the file location.

From the **File Length** list, select the file length for the new random file.

In the **Random Seed Source Files** box, add one or more seed source files to be used to generate the new random file. To do this, click the **Add** button.
The Random Seed Source File dialog box appears, as shown in the following example:

In the File Name text box, enter the full path of a file to be used as a seed source file in generating the new random file, or click Browse to search for the file location.

Click OK to return to the New Random File dialog box. The selected random seed source file is populated in the Random Seed Source Files box.

You can click the Add button again to add another random seed source file. Otherwise, click OK to close the New Random File dialog box and return to the Random File dialog box. The random file name is now populated in the Random File text box.

In the Random File dialog box, click Next to continue.
16. Optionally, you can specify ciphers to be used for the security context record in the **Ciphers** dialog box, as shown in the following example:

![Security Context Record Wizard]

Ciphers enable you to identify which cipher suites should be supported for the context. This affects the cipher suites and public key algorithms supported during the SSL/TLS handshake and subsequent data exchanges.

When a context is created, its cipher suites are set to SSLv3 suites supported by the SSL version you selected.

In the **Ciphers** text box, enter the cipher suite for the security context record.

The **CipherSpecs** parameter is a string containing `cipher-spec` separated by colons. An SSL cipher specification in `cipher-spec` is composed of four major attributes as well as several, less significant attributes. For detailed information about Cipher Suites, see “UniVerse BASIC Extensions.”

Click **Next** to continue.
17. Optionally, you can specify a certificate revocation list in the Certificate Revocation List dialog box, as shown in the following example:

Click the Add button to add a certificate revocation list (CRL) file to the revocation list for the security context record.
The **Add Certificate Revocation** dialog box appears, as shown in the following example:

In the **Certificate File** box, enter the full path of the file containing the certificate revocation list (CRL), or click **Browse** to search for the file location.

From the **Format** list, select the format for the CRL file:
- **PEM** – Base64 encoded format
- **DER** – ASN.1 binary format
- **PKCS #12** – PKCS #12 format

Click **OK** to add the CRL file and return to the **Certificate Revocation List** dialog box. The selected CRL file is now populated in the certificate revocation list.

Repeat this step if you want to add another CRL file to the certificate revocation list for the security context record.

Click **Next** to continue.
18. Set a password for the security context record in the Password dialog box, as shown in the following example:

In the Password for SCR text box, enter a password for the security context record you are creating. Reenter the password in the Confirm Password for SCR text box.

Click Next to create the security context record.

19. Check the message indicating the status of creating the security context record in the Review Status and Finish dialog box.

If the security context record was created successfully, the dialog box contains the message “SCR record was added/updated successfully.”

Otherwise, the message “Failed to save SCR” is displayed. You can click the Back button to return to previous dialog boxes and correct the error.

Click Finish to close the Security Context Record Wizard and return to the Configure SSL for Servers editor.
Configuring SSL for UniVerse servers

After you create a security context record, you need to configure SSL for UniVerse servers. UniVerse servers process requests by various U2 clients, including UniObjects (UO), UniObjects for Java (UOJ), ODBC, OLEDB, wIntegrate, and others.

A security context record contains all SSL-related properties necessary for the server to establish a secured connection with an SSL client. The properties include the server’s private key certificate, client authentication flag and strength, and trusted entities, among others.

To configure SSL for UniVerse servers:

1. In the Configure SSL for Servers editor, select the Server Configuration tab, as shown in the following example:
2. In the **Server Configuration** tab, click the **Add** button to add an SSL configuration entry.

3. Define the properties of the SSL configuration entry in the **Add SSL Configuration** dialog box, as shown in the following example:

From the **Service Name** list, select a service name.

From the **SCR Database** list, select the database in which the security context record is stored.

The full path of the selected database is populated in the **Path** box.

From the **SCR Record** list, select the security context record for this SSL configuration entry.

In the **Password Seed** text box, enter the password for this SSL configuration record. Reenter the password in the **Confirm Password** text box.

Click **OK** to add the SSL configuration entry and return to the **Configure SSL for Servers** editor.
Chapter 3: Using SSL with UniObjects for Java

Overview of SSL technology .......................................................... 3-3
Software requirements .................................................................. 3-4
Configuring the database server for SSL .................................... 3-5
Creating a secure connection ...................................................... 3-7
  Direct connection. ...................................................................... 3-8
  Establishing the connection ..................................................... 3-10
  Proxy tunneling ........................................................................ 3-11
  Externally secure ...................................................................... 3-13
Managing keys and certificates for a UOJ Client and a proxy server 3-18
  Importing CA certificates Into UOJ client trustfile .................. 3-18
  Generating client certificates .................................................. 3-19
  Managing key file and trust file for the proxy server ............... 3-20
This chapter explains how to use SSL (Secure Socket Layer) with UniObjects for Java (UOJ). The topics covered include:

- "Overview of SSL technology"
- "Software requirements"
- "Configuring the database server for SSL"
- "Creating a secure connection"
Overview of SSL technology

Secure Sockets Layer (SSL) is a transport layer protocol that provides a secure channel between two communicating programs over which arbitrary application data can be sent securely. It is by far the most widely deployed security protocol used on the World Wide Web.

Although it is most widely used in applications to secure web traffic, SSL actually is a general protocol suitable for securing a wide variety of other network traffic that is based on TCP, such as FTP and Telnet.

SSL provides server authentication, encryption and message integrity. It optionally also supports client authentication.

This document assumes that users who want to use this facility have some basic knowledge of public key cryptography.

For more information on the implementation of SSL with UniData and UniVerse, refer to Developing UniBasic Applications manual for UniData and the Guide to UniVerse Basic for UniVerse. Also see Chapter 1, “Chapter 1: Using SSL with the CallHTTP and socket interfaces.”
Software requirements

You must have the following applications installed and configured on the client machine.

- JDK (Java Development Kit) 1.4 or higher
- UniObjects for Java version 2.0.0 or higher
Configuring the database server for SSL

First, you need to create a Server Security Context Record (SCR).

A SCR contains all SSL related properties necessary for the server to establish a secured connection with an SSL client. The properties include the server’s private key, certificate, client authentication flag and strength, and trusted entities. For more information, see UniBasic Extensions and Chapter 1, “Chapter 1: Using SSL with the CallHTTP and socket interfaces.”

The SCR can be generated by directly calling the UniData or UniVerse Security API from a BASIC program, or alternatively, by invoking U2 Extensible Admin.

The SCR is encrypted by a password and saved in a UniData or UniVerse security file with a unique ID. The path, password and ID of the SCR for a UOJ server are important in the following descriptions.

In order to enable SSL for UniObjects for Java on the database server, you need to bind an SCR to the U2 service that runs on the server. You do this by editing the .unisecurity configuration file in the common unishared directory. On UNIX systems, you can determine the location of the unishared directory by entering “cat ./unishared.” On Windows platforms, the default location can be found by examining the following registry record:

```
HKEY_LOCAL_MACHINE\SOFTWARE\ROCKET SOFTWARE\unishared
OR
HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\ROCKET SOFTWARE\unishared
```

For each SSL-enabled service there should be one line in the .unisecurity file, which has the following format:

```
<service_name> <scr_database_path> <scr_ID> <scr_password>
```

For UniObjects for Java, the service name is uvcs, for example:

```
uvcs c:\myscrdb myscr “a long passphrase”
```

You can share the same SCR for different services, or use a different SCR for each service.
**Note**: The `<service_name>` must match the server as defined in `unishared/unirpc/unirpcservices`. 
Creating a secure connection

There are three different modes you can use to establish a secure SSL session with a UniData or UniVerse database server for UniObjects for Java.

- Direct Connection - This method is completely secure. In this mode, the SSL session is established directly between the UOJ client and the UniData or UniVerse database server.
- Proxy Tunneling - This method is completely secure. In this mode, the connection is created through a proxy server. The proxy server provides tunneling for the data exchange between the UOJ client and the UniData or UniVerse database server. Since the proxy server does not decrypt data packets, there is no session multiplexing performed.
- Externally Secure Proxy - The security of this method is reliant on the external proxy. In this mode, the externally secure SSL session is established between the UOJ client and a proxy server. The connection between the proxy server and the UniData or UniVerse database server is not a secure connection. A typical application for this type of connection would be in the case where both the proxy server and UniData or UniVerse database server are behind a firewall. Thus, the unsecured connection between the proxy and database server does not compromise security. In this mode, session multiplexing can be achieved.

The first step is to create a UniSession object by calling the openSession method of the UniJava object. The signature of the method is shown in the following example.

```java
public UniSession openSession(int sslmode) throws UniSessionException
```

The `sslmode` parameter can be one of the following values:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct Connection</td>
<td>UniObjectTokens.SECURE_SESSION</td>
</tr>
<tr>
<td>Proxy Tunnel</td>
<td>UniObjectTokens.SECURE_SESSION</td>
</tr>
<tr>
<td>Secure External Proxy</td>
<td>UniObjectTokens.EXTERNALLY_SECURE_PROXY_SESSION</td>
</tr>
</tbody>
</table>
Next, determine which of the following connection types you wish to use for the secure connection.

**Direct connection**

When creating a secure connection, there are three components that you must consider. They are SSL Socket Factory, Cipher Suites and Keyfile and Trustfile Parameters. You can define these parameters by creating and setting properties of the UniSSLDescriptor object associated with the secure session and setting some system variables.

- **SSL Socket Factory** - Secure Socket Factories encapsulate details for creating and initially configuring secure socket connections. The SSLSocketFactory object is a concrete implementation of the abstract SocketFactory class provided with JSSE in the javax.net package. It acts as a factory for creating secure sockets. You can define your own SSLSocketFactory object with the setSSLSocketFactory method of the UniSSLDescriptor object. If you pass a null parameter to this method, the system defaults will be used. Another way to use the system defaults is to set the UniSSLDescriptor object to null by calling the setSSLDescriptor method of the UniSession object with a null parameter.

- **Cipher Suites** - Define your own available Cipher Suites with the setEnabledCipherSuites method of UniSSLDescriptor. If you pass a null parameter to this method, the system defaults will be used.

- **Keyfile and Trustfile Parameters** - System Variables must be created to define locations of the keyfile, trustfile and the password to access these files. This step is required for any secure connection. For more information about the Java key file and trust file, see “Managing keys and certificates for a UOJ Client and a proxy server” on page 3-18.

If uniojbects.UniSSLDescriptor is set to null, the system will use the system defaults for SSLSocketFactory and default Cipher suites.

Once you have created the session object, to specify your own SSLSocketFactory object and/or define available cipher suites, you need to create the uniojbects.UniSSLDescriptor using the constructor with the following signature.

```java
public UniSSLDescriptor (void)
```
Once created, you need to call the `setSSLSocketFactory` method to set the SSL Socket Factory and `setEnabledCipherSuites` to set the available cipher suites and then pass this object to the session.

Calling the `setSSLSocketFactory` method with the signature shown in the following example will set SSLSocketFactory.

```java
public void setSSLSocketFactory(SSLsocketFactory ssf)
```

Calling the `setEnabledCipherSuites` method with the signature shown in the following example sets CipherSuites.

```java
public void setEnabledCipherSuites(String [] cs)
```

Whether you specify your own Socket Factory and Cipher Suites or use the system defaults, you still need to specify the system variables for the location and password for the keyfile and the trustfile as shown in the following table:

<table>
<thead>
<tr>
<th>System Variable</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.net.sslTrustStore</td>
<td>Defines the location of the trustfile.</td>
</tr>
<tr>
<td>javax.net.sslKeyStore</td>
<td>Defines the location of the keyfile.</td>
</tr>
<tr>
<td>javax.net.sslKeyStorePassword</td>
<td>Defines the password for the keyfile.</td>
</tr>
</tbody>
</table>

The trustfile (also called truststore), is a file that holds a set of keys and certificates. In fact, the keyfile (also called key store) has exactly the same format. The keyfile provides credentials for the UniObjects for Java client and the trustfile verifies credentials from the UniVerse UniObjects for Java server. If the UniVerse UniObjects for Java servers do not request client authentication, you do not need to configure a key file. On the other hand, you must have a trust file available.

You can use tools such as IBM’s keyman utility and Sun’s keytool to create and maintain the keyfile and trustfile. The keytool utility is installed with Oracle’s JDK. For more information on keytool, see http://download.oracle.com/javase/1.4.2/docs/tooldocs/windows/keytool.html. The default location for the trustfile (truststore) is $JREHOME/lib/security/jssecacerts. If the file does not exist, the system assumes that the trustfile is $JREHOME/lib/security/cacerts. There is no default location for the keyfile (keystore).
Establishing the connection

Once you have set the secure parameters for the session, you can connect by calling the connect method of the **UniSession** object as you would in any normal, nonsecure session.

The following code example demonstrates how to create a secure Direct Connection with the database server.

Database server Connection Properties:

- **U2 host**: localhost
- **user name**: "test"
- **password**: "new.pass"
- **accountpath**: "demo"

Security Properties are:

- **keyfile path**: "testkeys"
- **keyfile password**: "new.pass"
- **trustfile path**: "testtrust"
- **trustfile password**: "new.pass"
String U2host = "localhost";
String username = "test";
String password = "new.pass";
String accountpath = "demo";
String keyfilepath = "testkeys";
String keyfilepwd = "new.pass";
String trusfilepath = "testkeys";

// First, let’s instantiate our new UOJ application
uvJava = new UniJava();

// Now, let's open up a session
UniSession demoSession =
    uvJava.openSession(UniObjectsTokens.SECURE_SESSION);

demoSession.setHostPort(UniRPCTokens.UNIRPC_DEFAULT_PORT);
demoSession.setHostName(U2host);
demoSession.setUserName(username);
demoSession.setPassword(password);
demoSession.setAccountPath(accountpath);

// Now we'll set locations for the keystore and truststore and a password for the keystore
System.setProperty("javax.net.sslTrustStore", "testtrust");
System.setProperty("javax.net.sslKeyStore", "testkeys");
System.setProperty("javax.net.sslKeyStorePassword", "new.pass");
demoSession.setSSLDescriptor(null);
demoSession.connect();

**Proxy tunneling**

The process for using the Proxy Tunneling method is basically the same as the Direct Connection method. The only difference is that the connection is tunnelled through a proxy server which passes messages between the client and database server. There are no additional parameters to configure but the proxy server should be properly configured.

You need to set the PROXY_SSL_FLAG parameter in the `uniproxy.config` file to true, so the proxy server will listen for secure connections. See "Externally secure" on page 3-13 for more information on editing the `uniproxy.config` file.

The following example demonstrates how to create a secure connection with the database server through a Proxy Tunneling server.
The U2 connection properties are:

U2 host: localhost
user name: "test"
password: "new.pass"
accountpath: "demo"

Proxy server properties are:

Proxy host - localhost
Proxy token - "password1"

Security properties are:

keyfile path: "testkeys"
keyfile password: 
trustfile path: "testkeys"
"new.pass"
String U2host = "localhost";
String username = "test";
String password = "new.pass";
String accountpath = "demo";
String proxyhost = "localhost";
String proxytoken = "password1";
String keyfilepath = "testkeys";
String keyfilepwd = "new.pass";
String trusfilepath = "testkeys";

int sslmode = UniObjectsTokens.SECURE_SESSION;

// Instantiate our new Uni/Java application
UniJava uvJava = new UniJava();

// First, let's open up a session
UniSession demoSession = uvJava.openSession(sslmode);
demoSession.setHostName(U2Host);
demoSession.setHostPort(UniRPCTokens.UNIRPC_DEFAULT_PORT);
demoSession.setUserName(username);
demoSession.setPassword(password);
demoSession.setAccountPath(accountPath);
demoSession.setProxyHost(proxyhost);
demoSession.setProxyPort(UniRPCTokens.UNIRPC_DEFAULT_PROXY_PORT);
demoSession.setProxyToken(proxytoken);

// Set system variables for locations of the keystore and truststore and a password for the keystore
System.setProperty("javax.net.sslTrustStore", "testtrust");
System.setProperty("javax.net.sslKeyStore", "testkeys");
System.setProperty("javax.net.sslKeyStorePassword.", "new.pass");

// use default SSLSocketFactory object
demoSession.setSSLDescriptor(null);
demoSession.connect();

**Externally secure**

This method requires that you define the properties described in the `uniproxy.config` file.
You must set the following parameters for SSL for UOJ configuration.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROXY_SSL_FLAG</td>
<td>This parameter enables or disables externally secure connections. Its value can be true or false. When set to true, the proxy server will start a new thread that listens on PROXY_SSL_PORT for externally secure connections. This parameter must be set to true for both Proxy Tunneling and Externally Secure modes. The default setting is false.</td>
</tr>
<tr>
<td>PROXY_SSL_ONLy_FLAG</td>
<td>If this parameter is set to true, the proxy only allows secure connections to pass through to the database server. The default setting is false.</td>
</tr>
<tr>
<td>PROXY_SSL_PORT</td>
<td>This parameter defines the port on which the proxy server should listen for externally secure connections.</td>
</tr>
<tr>
<td>SSL_KEY_FILE</td>
<td>This parameter specifies the location of the keyfile (keystore).</td>
</tr>
<tr>
<td>SSL_TRUST_FILE</td>
<td>This parameter specifies the location of the trustfile (truststore).</td>
</tr>
<tr>
<td>SSL_KEY_FILE_TYPE</td>
<td>This parameter specifies the type of the proxy server keyfile type. It can be either JKS or JCEKS. The default value is JKS.</td>
</tr>
<tr>
<td>SSL_TRUST_FILE_TYPE</td>
<td>This parameter specifies the type of the proxy server trustfile. It can be either JKS or JCEKS. The default value is JKS.</td>
</tr>
</tbody>
</table>
The following example demonstrates how to create an Externally Secure connection with the database server.

- The keyfile (keystore) that contains credentials (keys and certificate) for the proxy server is called "testkeys" and is located in the current proxy directory.
- The keyfile type is JKS.
- The proxy server should authenticate all UOJ clients.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| SSL_PWD_METHOD                | This parameter defines the method in which password for the keystore is specified. This parameter can take the following values:  
  - **DIRECT** - When this value is selected, the password is specified as the SSL_KEY_FILE_PWD's value.  
  - **USER_DEFINED** - When you select this value, the parameter, SSL_KEY_FILE_PWD contains a description of how to call a user defined java method that will generate the password. In this case, the value for these properties consists of three fields separated by the underscore character, "_". The first field is a parameter for the method and should be of type String. The second field is a method name and a third field defines a class name. This mode provides better security for protecting the passwords. However, keep in mind that it may be possible that the password algorithm can be reverse engineered.  
  - **INTERACTIVE** - When you select this value, the proxy server prompts the user to enter a password for the keyfile and trustfile interactively during the startup. This mode provides the most password security but cannot support proxy auto-restart.  

| SSL_KEY_FILE_PWD              | This parameter contains information depending on settings defined in the SSL_PWD_METHOD.                                                                                                                      |
| SSL_CLIENT_AUTHENTICATION     | This parameter specifies whether or not the proxy will ask for a client certificate during the SSL handshake.                                                                                                 |
The trustfile (truststore) that contains trusted certificates is called "testtrust" and is located in the current proxy directory.

- The trustfile type is JKS.
- The passwords for the keystore and truststore should be entered interactively.
- The proxy port for listening for externally secure connections is 31452.

The proxy configuration for this example is as follows:

```properties
PROXY_SSL_FLAG=true
PROXY_SSL_PORT=31452
SSL_KEY_FILE=testkeys
SSL_TRUST_FILE=testtrust
SSL_KEY_FILE_TYPE=JKS
SSL_TRUST_FILE_TYPE=JKS
SSL_PWD_METHOD=INTERACTIVE
SSL_CLIENT_AUTHENTICATION=true
```

database server: localhost

```plaintext
user name: newuser
password: new.pass
accountpath: demo
```

Proxy server properties are:

```plaintext
Proxy host: localhost
Proxy token: password1
```

Security properties are:

```plaintext
keyfile path: testkeys
keyfile password: new.pass
trustfile path: testtrust
```
The following sample program uses the above properties:

```java
String U2host = localhost;
String username = newuser;
String password = new.pass;
String accountpath = demo;
String proxyhost = localhost;
String proxytokend = password1;
String keyfilepath = testkeys;
String keyfilepwd = new.pass;
String trustfilepath = testkeys;

int sslmode = UniObjectsTokens.EXTERNALLY_SECURE_PROXY_SESSION;

// Instantiate our new Uni/Java application
UniJava uvJava = new UniJava();

// First, let's open up a sessions
UniSession demoSession = uvJava.openSession(sslmode);

demoSession.setHostName(U2Host);

demoSession.setHostPort(UniRPCTokens.UNIRPC_DEFAULT_PORT);

demoSession.setUserName(username);

demoSession.setPassword(password);

demoSession.setAccountPath(accountPath);

demoSession.setProxyHost(proxyhost);

demoSession.setProxyPort(UniRPCTokens.UNIRPC_DEFAULT_SSL_PROXY_PORT);

demoSession.setProxyToken(proxytoken);

// Set locations for the keystore and truststore and a password for the
keystore
System.setProperty(javax.net.sslTrustStore, testtrust);
System.setProperty(javax.net.sslKeyStore, testkeys);
System.setProperty(javax.net.sslKeyStorePassword, new.pass);

// use default SSLSocketFactory object

demoSession.setSSLDescriptor(null);

demoSession.connect();
```
Managing keys and certificates for a UOJ Client and a proxy server

When a server establishes a secure session with a client, it passes its certificate to the client for authentication. The client usually has a list of trusted certificates that it uses to verify server credentials. If the client cannot verify the server certificate through its trusted certificates, it rejects the connection. Optionally, a server may also require a client to authenticate itself by providing the server with a valid client certificate. In the case where the server cannot verify the client certificate, the secure connection is not established. A list of trusted certificates that is used to verify credentials usually resides in a trust file, and private keys and certificates providing credentials are kept in the keyfile.

A UOJ client should provide the system with a location of trustfile and keyfile and also the keyfile password by setting system properties.

The JDK usually contains a program that works with key files and trust files. In Oracle’s implementation of the JDK, this utility is called keytool. In IBM’s JDK implementation it has an additional tool called the ikeyman utility. All examples from this chapter use the keytool utility. For a complete description of keytool utility, see http://docs.oracle.com.

Importing CA certificates Into UOJ client trustfile

In general, a server’s certificate is issued by a trusted third party called a Certificate Authority (CA), whose certificate (CA certificate) is used to sign the server certificate. In order for a client to verify a server’s certificate, the UOJ client should import the trusted server’s CA certificate into its trust file, if it is not already stored there.

Suppose we have a trusted server CA certificate in the file cacert.pem, the client’s trustfile is called testtrust, and the access password for the trust file is passphrase. By executing the following command, you can import the certificate into the trust file.

```
keytool -import file cacert.pem -keystore testtrust -storepass passphrase
```
Generating client certificates

In the case where the database server or the proxy server requires client authentication, the client certificate should be generated and installed into the client’s key file. Complete the following steps below to generate and install the certificate for the client.

1. Generate a key pair consisting of a public key and a private key. The following command in the keytool utility generates an RSA type key pair, as well as a self-signed certificate in the key file.

   ```
   keytool -genkey -keystore testkeys -storepass passphrase -keyalg RSA
   ```

2. Create a certificate request. The following command in the keytool utility creates a certificate request in the file javacert.req. Enter appropriate answers for all prompts by the utility.

   ```
   keytool -certreq keystore testkeys -storepass passphrase -file javacert.req
   ```

3. Send the certificate request to a Certificate Authority (CA). The javacert.req file containing the certificate request should be sent to a valid Certificate Authority that will approve it and send back the certificate chain. We assume that the certificate chain is returned in the file javacert.pem file. A file javacert.pem can be exported to the client key file.

   If you choose to use the UniData BASIC API or the U2 Extensible Admin tool to generate certificates for requests, or if the CA described in the previous paragraph returns its CA certificate separately, the server CA certificate should be separately installed into the client’s keystore before generated certificates are installed there. The CA Certificate must be imported into the key file using an alias, as described in the following example.

   ```
   keytool -import -file cacert.pem -keystore testkeys -storepass passphrase -alias ca
   ```

   Where cacert.pem contains the CA certificate and ca is the name of the alias.

4. Replace your own certificate with the newly created CA-signed certificate in the keyfile. The following command in the keytool utility will replace the self-signed certificate with the newly generated one.

   ```
   keytool -import -file javacert.pem -keystore testkeys -storepass passphrase
   ```
Managing key file and trust file for the proxy server

The key file and trust file for the proxy server should be managed by a standard key and certificate utility, such as Oracle’s keytool or IBM’s ikeyman utility.

The same procedures described in the previous section also apply.
Chapter

Chapter 4: SSL support in
UniVerse clients
Support for secure connections in UniVerse client . . . . . . . 4 -3
In this chapter . . . . . . . . . . . . . . . . . .
4-3
UCI configuration parameters for ODBC and UniOLEDB . . . . 4 -4
Secure connection parameters . . . . . . . . . . . .
4-4
Changing UCI configuration parameters . . . . . . . .
4-6
UniObjects ConnectionString parameters . . . . . . . . . . 4 -9
About SSL property lists . . . . . . . . . . . . . . . 4 -11
Creating and maintaining SSL property lists . . . . . . .
4-11
List encryption . . . . . . . . . . . . . . . . .
4-11
Loading and decrypting an SSL property list . . . . . . . 4-12
SSL properties . . . . . . . . . . . . . . . . . . 4-12
Using the U2 SSL configuration editor . . . . . . . . . . . 4 -20
Working with SSL property lists . . . . . . . . . . . 4-20
Starting the U2 SSL configuration editor. . . . . . . . . 4-20
Creating a new SSL property list . . . . . . . . . . . . . 4 -24
Task 1: Assign name, password, SSL version, and store type to property
list . . . . . . . . . . . . . . . . . . . . . . 4-25
Task 2: Specify certificates, private key and password, certificate
revocation list, and cipher suites . . . . . . . . . . . 4-29
Task 3: Specify authentication properties . . . . . . . . 4-33
Editing an existing SSL property list . . . . . . . . . . . 4 -38
To edit an existing SSL property list: . . . . . . . . . . 4-38
Deleting an SSL property list . . . . . . . . . . . . . . 4 -50
To delete an SSL property list: . . . . . . . . . . . . 4-50
Copying an SSL property list . . . . . . . . . . . . . . 4 -51
To copy an SSL property list: . . . . . . . . . . . . . 4-51
Renaming an SSL property list . . . . . . . . . . . . . 4 -54
To rename an SSL property list: . . . . . . . . . . . . 4-54

4


Using the trace feature .......................................................... 4-56
   To use the Trace feature: .................................................. 4-56
Using the Console/Problems view ........................................... 4-57
   To use the Console view: .................................................. 4-57
   To use the Problems view: ............................................... 4-58
Support for secure connections in UniVerse client

UniVerse supports the ability of client applications to make secure connections to the database server through Secure Sockets Layer (SSL). SSL is a transport layer protocol that provides a secure channel between two communicating programs over which application data can be transmitted securely. It is the most widely implemented security protocol on the World Wide Web.

SSL provides for server authentication, encryption, and message integrity. It can also support client authentication.

UniVerse uses CallHTTP and the Sockets API. As a security protocol implemented with both of these APIs, SSL enables commercial applications to securely process sensitive data, such as credit card transactions.

In this chapter

This chapter contains the following topics related to secure connections:

Secure connection parameters

UniVerse contains parameters to support a client application’s secure connection to the UniVerse database server:

- UCI configuration parameters for ODBC and UniOLEDB
- UniObjects ConnectionString parameters

SSL property lists

One of the parameters specifies the SSL property list to be used for a secure connection. The property list defines the characteristics of the secure connection. This version of the product includes the U2 SSL Configuration Editor, a graphical user interface (GUI) tool for creating and maintaining SSL property lists.

- About SSL property lists
- Using the U2 SSL configuration editor

This chapter assumes that users who want to use SSL have a basic knowledge of public key cryptography.
UCI configuration parameters for ODBC and UniOLEDB

Client applications can use UniVerse ODBC or UniOLEDB to access UniVerse data sources through entries in the UCI configuration file (uci.config) on the client machine. The uci.config file contains connection parameters needed to route requests to the UCI server.

When ODBC or UniOLEDB attempts to connect to a data source, UniVerse ODBC or UniOLEDB reads the UCI configuration file to determine the connection parameters. There are three UCI configuration parameters that enable a client application to make a secure connection to the UniVerse database.

Secure connection parameters

The UCI configuration parameter indicates whether an application requires a secure connection. If so, two other parameters provide the SSL property list and password to be used. These parameters are detailed in the following table.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECUREMODE</td>
<td>Indicates whether the application requires a secure connection for exchange of data with the database server:</td>
<td>False</td>
</tr>
<tr>
<td></td>
<td>True – Secure connection</td>
<td></td>
</tr>
<tr>
<td></td>
<td>False – Nonsecure connection</td>
<td></td>
</tr>
</tbody>
</table>

Configuration Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSLPROPERTYLIST</td>
<td>The name of the SSL property list to be used to verify properties of the secure connection.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A value for this option applies only if the value of the SECUREMODE option is True.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If SECUREMODE is True and you do not specify a value for SSLPROPERTYLIST, the default SSL property list in the Registry For 32-bit platforms:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HKEY_LOCAL_MACHINE/SOFTWARE/Rocket Software/UniClient/SPL/myspl is used.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For 64-bit platforms:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HKEY_LOCAL_MACHINE//SOFTWARE/WoW6432Node/Rocket Software/UniClient/SPL/myspl</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For further information on property lists, see “About SSL property lists.”</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For instructions on creating and maintaining property lists, see “Using the U2 SSL configuration editor.”</td>
<td></td>
</tr>
<tr>
<td>SSLPROPERTYPASSWORD</td>
<td>The password for the specified SSL property list.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A value for this option applies only if the value of the SECUREMODE option is True.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The password is used to derive an internal decryption key to decrypt the list. If a password is not specified, an internal default decryption key is used.</td>
<td></td>
</tr>
</tbody>
</table>
Changing UCI configuration parameters

If the client application requires a secure connection to the database server, you can enable the connection by using either of the following methods:

- Adding secure session parameters in the uci.config file
- Setting secure session input variables in SQLSetConnectOption

Adding secure session parameters in the uci.config file

The UCI Config Editor tool accepts three parameters for secure connections. To enable a secure connection, you can use the UCI Config Editor or a text editor to add the secure connection parameters in the uci.config file as follows:

```plaintext
[ODBC DATA SOURCES]
<localud>
DBMSTYPE = UNIVERSE
NETWORK = TCP/IP
SERVICE = uvserver
HOST = localhost
SECUREMODE = True
SSLPROPERTYLIST = mylist
SSLPROPERTYPASSWORD = mypassword
```

For information about the UCI Config Editor, see the Administrative Supplement for Client APIs.

Setting secure session input variables in SQLSetConnectOption

Alternatively, you can set values for the secure session input variables in SQLSetConnectOption. The SQLSetConnectOption enables an application to control the behavior of a connection. The SECURE_MODE, SSLPROPERTY_LIST, and SSLPROPERTY_PASSWORD variables set values for a secure connection, as detailed below.

**Syntax**

RETCODE SQLSetConnectOption (hdbc, fOption, vParam, szParam)
Input variables
The following table describes the input variables.

<table>
<thead>
<tr>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDBC</td>
<td>hdbc</td>
<td>Connection handle.</td>
</tr>
<tr>
<td>UWORD</td>
<td>fOption</td>
<td>Option to be set.</td>
</tr>
<tr>
<td>UDWORD</td>
<td>vParam</td>
<td>A 32-bit value associated with fOption when fOption is SQL_EMPTY_NULL, SQL_TXN_ISOLATION, SQL_DATA_MODEL.</td>
</tr>
<tr>
<td>UCHAR *</td>
<td>szParam</td>
<td>Text value associated with fOption when fOption is SECURE_MODE, SQL_OS_UID, SQL_OS_PWD, SQL_LIC_DEV_SUBKEY, any of the SQL_UVNLS options, SSLPROPERTY_LIST, or SSLPROPERTY_PASSWORD.</td>
</tr>
</tbody>
</table>

SQLSetConnectOption Input Variables

szParam values for secure connections
The szParam values for secure connections are as follows:

<table>
<thead>
<tr>
<th>If fOption is...</th>
<th>szParam is...</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECURE_MODE</td>
<td>A value indicating whether the application requires a secure connection for exchange of data with the database server: True or False.</td>
</tr>
</tbody>
</table>

szParam Values
<table>
<thead>
<tr>
<th>If <code>fOption</code> is...</th>
<th><code>szParam</code> is...</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL_PROPERTY_LIST</td>
<td>The name of the SSL property list to be used to verify properties of the secure connection. A value for this option applies only if the value of the SECURE_MODE option is True. If SECURE_MODE is True and you do not specify a value for SSL_PROPERTY_LIST, the default SSL property list in the Registry <code>&lt;UniClient&gt;/SPL/myspl</code> is used, where <code>&lt;UniClient&gt;</code> is shorthand for the following Registry key: For 32-bit platforms: HKEY_LOCAL_MACHINE/SOFTWARE/Rocket Software/UniClient For 64-bit platforms: HKEY_LOCAL_MACHINE/SOFTWARE/WoW6432Node/Rocket Software/UniClient/SPL/myspl</td>
</tr>
<tr>
<td>SSL_PROPERTY_PASSWORD</td>
<td>The password for the specified SSL property list. A value for this option applies only if the value of the SECURE_MODE option is True. The password is used to derive an internal decryption key to decrypt the list. If a password is not specified, an internal default decryption key is used.</td>
</tr>
</tbody>
</table>
UniObjects ConnectionString parameters

The UniObjects session object properties include a ConnectionString property. In previous versions, the ConnectionString property was used only to specify the server process to be used for the session. It now provides a second set of parameters to specify a secure connection.

- If you want to connect to a server other than the one specified by the DatabaseType property, enter the name of the server process (such as defcs, udcs, or uvcs).
  
  If you do not specify a server process in the ConnectionString property, the server specified in the DatabaseType property is used.

- If the client application does not require a secure connection to the database server, secure connection parameters are not required.

If the client application requires a secure connection to the database server, enter values for the following parameters:

```
SecureMode=[True | False]
```

where

- True specifies a secure session.
- False specifies a nonsecure session.

For a secure session, this component is required. If you do not enter a value for the SecureMode parameter, the default value of False is assumed and any values specified for the SSLPropertyList and SSLPropertyPassword parameters are ignored.
SSLPropertyList = list-name

where
list-name is the name of the SSL property list to be used to establish the secure connection.

A value for this parameter applies only if SecureMode=True. If SecureMode=True and you do not specify a value for SSLPropertyList, the default SSL property list in the Registry <UniClient>/SPL/myspl is used,

where
<UniClient> is shorthand for the following Registry key:
HKEY_LOCAL_MACHINE/SOFTWARE/Rocket Software/UniClient

SSLPropertyPassword = password-string

where
password-string is the password for the specified SSL property list.

A value for this parameter applies only if SecureMode=True. The password is used to derive an internal decryption key to decrypt the list. If a password is not specified, an internal default decryption key is used.

Use a semicolon as a separator between values. The following example shows server process and secure connection parameters in the Connection-String property:

udcs;SecureMode=True;SSLPropertyList=myList;
SSLPropertyPassword=myPassword
About SSL property lists

An SSL property list is an ASCII text string that represents the properties for a secure connection and is stored in the Windows registry. These properties define the characteristics and behaviors of the secure connection.

Creating and maintaining SSL property lists

Although the property list is an ASCII text string, you should never edit it directly. UniVerse provides the U2 SSL Configuration Editor for use in creating, editing, or deleting an SSL property list. Using this tool ensures that the list is properly saved to (or deleted from) the Windows Registry. For detailed instructions, see “Using the U2 SSL configuration editor.”

List encryption

An SSL property list may contain sensitive information such as the password to a private key or the location of a certificate authority (CA) certificate. For this reason, it is saved in encrypted form to the Windows Registry at:

HKEY_LOCAL_MACHINE/SOFTWARE/ROCKET
SOFTWARE/UniClient/SPL

The U2 SSL Configuration Editor uses an algorithm to encrypt the list.

If you do not assign your own password to the list, the algorithm uses an internal default password to generate the encryption key for the list. Because the internal default password is fixed, the algorithm always produces the same encryption key from this password. Consequently, anyone who uses the U2 SSL Configuration Editor can access and read the contents of your SSL property list.

For increased security, we strongly recommend that you assign your own password to the SSL property list. In this case, the same algorithm uses your unique password as the seed for generating an encryption key. The resulting encryption key is unique, so only users who know the password can access the list and read its contents.
Loading and decrypting an SSL property list

Before the SSL handshake takes place, the SSL property list must be loaded into memory and decrypted. After the list has been decrypted, it is supplied in plain text form to a UniVerse internal function that handles the SSL handshake.

When the property list is in decrypted form (only internally in UniVerse), each property is stored on a separate line in the file, as shown below:

propertyName=propertyValue

SSL properties

This section describes each property supported in the SSL_PROPERTY_LIST to define the characteristics and behaviors of a secure connection.

SSLVersion={SSLv3 | TLSv1}

Optional. Default is SSLv3.

This property specifies the preferred protocol version.

<table>
<thead>
<tr>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSLv3</td>
<td>This is most widely used protocol.</td>
</tr>
<tr>
<td>TLSv1</td>
<td>This is the newer protocol. Most newer applications support it, but some older applications may not.</td>
</tr>
</tbody>
</table>

Protocol Versions
CertificateStoreType={U2 | Windows}

Optional. Default is U2.

This property specifies the type of certificate stores to be used for all certificates issued for the secure connection.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>U2</td>
<td>All certificates specified in this file are PEM or DER-format OS-level files.</td>
</tr>
</tbody>
</table>
| Windows | All certificates specified in this list are looked up from the native Windows certificate store. Generally, a CA certificate is looked up from Windows CA and ROOT stores, while MyCertificate is looked up from MY or personal stores. In Microsoft’s terminology, these certificate stores are system stores: a collection of physical certificate stores that reside in the Windows Registry. UniVerse looks up these stores from both of the following Registry locations:  
- CERT_SYSTEM_STORE_CURRENT_USER  
- CERT_SYSTEM_STORE_LOCAL_MACHINE |

Certificate Store Types

CACertificate=<cert-path>[;<cert-path>...]

Each property value string can contain multiple CA certificate paths, with paths separated by a semicolon (;) as shown above. Specifying multiple CACertificate properties is allowed.

U2 certificate store type

<cert-path> is the path of the certificate file that is used as a CA certificate. The format of the certificate can be either PEM or DER. (However, see the CertificatePath property for additional information on how U2 loads certificates when performing the SSL handshake.) With the U2 type, if a CA certificate chain is required, you have the choice of specifying multiple CACertificate properties, or, for PEM-format certificates, concatenating the certificate files into one single file (using OS-level editor or command line) and specifying the concatenated file once.
Windows certificate store type

Specify the same “friendly name” or “Common name” that is used for the certificate in the certificate store. With the Windows type, specify only one certificate, which should be the most immediate CA certificate (the one used directly to sign the certificate to which authentication is to be performed).

A certificate chain is automatically established and used in an SSL session. Note that the above description is based on the assumption that a correct and complete trust relationship, for example, certificate chain, exists in the Windows certificate store for the certificate involved. If a complete chain cannot be formed, an error is reported. This also applies to other certificate-related properties described below.

MyCertificate=<cert-path>

Optional for client SSL property list; default is none. Required for server SSL property list.

U2 certificate store type

Note that if you specify this property, you must also specify the MyPrivateKey and PrivateKeyPassword properties. The format of the certificate can be either PEM or DER.

Windows certificate store type

Specify the same “friendly name” or “Common name” that is used for the certificate in the certificate store. Note that when you import a Windows store type certificate to the MY store, you must associate an exportable private key with it by selecting the Exportable private key check box.

See also ClientAuthentication (below).

MyPrivateKey=<key-path>

Applicable to U2 certificate store type only. Required if you entered a value in My Certificate.
This property specifies the path for the file that contains the private key associated with MyCertificate. The format of the key file can be either PEM or DER.

When an SSL property list is created, the private key is loaded into memory and validated against its corresponding certificate (My Certificate). If it passes validation, the key is stored with the SSL property list. This validation feature is designed to enhance the security and protection of the user’s private key.

After the SSL property list has been created, you do not need to keep the private key file on your hard disk. You can store the key file safely on offline media until the next time you want to edit the SSL property list.

See also ClientAuthentication (below).

**PrivateKeyPassword=<pass-phrase>**

Applicable to U2 certificate store type only. Required if you specified a value for MyCertificate.

This property specifies the password for the private key file.

See also ClientAuthentication (below).

**CRL=<cert-path>**

Optional. Default is none. Specifying multiple CRL properties is allowed.

This property specifies the Certificate Revocation List (CRL) to be used for this secure connection.

The CRL is a special certificate published by certificate authority (CA); it contains the serial numbers of certificates revoked by the CA. During the SSL session handshake, the incoming server certificate is checked against the CRL to verify that it has not been revoked before other verification is performed.

The format of the CRL can be PEM or DER.

**AuthenticationDepth=<level>**

Optional. Default is 5.
This property determines the level at which to stop UniVerse’s verification process in authentication processing. The default setting of 5 is a sufficient depth in most cases. If you set the depth for fewer levels of authentication than actually employed for the certificate, the certificate will not pass authentication.

**CipherSuite=\textless cipher-suite-string\textgreater**

Optional. Default is all ciphers supported by the OpenSSL open source library.

This property specifies a suite of ciphers to be used in a specific order in the SSL handshake.

**TrustedPeerName=\textless trusted-peer-name-string\textgreater**

Optional. Default is none. Specifying multiple TrustedPeerName properties is allowed.

\textless trusted-peer-name-string\textgreater is in the format of 
\textless peer-name\textgreater[;\textless peer-name\textgreater[;\textless peer-name\textgreater[...]]

This property tells UniVerse that it needs to perform additional checking in verifying the incoming certificate. If you do not specify TrustedPeerName, the incoming certificate is considered valid when the CA certificate has verified it. However, if you specify TrustedPeerName, a further check is performed to verify that the incoming certificate’s SubjectAltName extension or CommonName subject field matches one of the specified TrustedPeerName.

TrustedPeerName can be either a fully specified name (such as admin@us.xyz.com) or a wildcard name. Two wildcard characters are supported:

\%
Match any character strings

_ 
Match one character

For example, \%@us.xyz.com matches both admin@us.xyz.com and market@us.xyz.com, while admi_@us.xyz.com matches admin@us.xyz.com only.
**AuthenticationStrength=[STRICT | GENEROUS]**

Optional. Default is STRICT.

STRICT authentication requires the following:

- The incoming server certificate is a well-formed X.509 certificate.
- A complete CA certificate chain exists and verifies the incoming server certificate.
- Peer name checking (if specified) is successfully performed.

GENEROUS authentication requires only the following:

- The incoming server certificate is a well-formed X.509 certificate.
- Peer name checking (if specified) is successfully performed.

*Note:* GENEROUS authentication is not highly secure. We recommend using it in test environments only.

**CertificatePath=[DEFAULT | RELATIVE | PATH=<path> | ENV=<env-var>]**

Optional. Applicable to U2 certificate store type only.

When you specify a certificate by the CACertificate, MyCertificate, or CRL property, the value for that property is registered internally. When loading the certificate into memory to establish an SSL connection, UniVerse uses this registered path by default to retrieve the certificate.

The CertificatePath property allows you to specify different locations in which to search the certificates. Note that this property applies to all certificates specified in the file.

Four options are available:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT</td>
<td>Specifies the above-described behavior. This option is the default.</td>
</tr>
<tr>
<td>RELATIVE</td>
<td>UniVerse looks for the certificate in the current directory under which the client process is running.</td>
</tr>
</tbody>
</table>
**ClientAuthentication**=[TRUE | FALSE]

Optional. Default is FALSE.

This property should be specified for a server SSL property list only.

If the value is TRUE, the SSL server using this property list requires client authentication during the SSL handshake. It asks the client to send its certificate.

If TRUE, UniVerse treats the SSL property list as a server property list. Consequently, you must also specify MyCertificate, MyPrivateKey (for the U2 certificate store type only), PrivateKeyPassword, and CACertificate or the SSL property list will not be created.

**RandomFileLocation**=<directory-path>

Optional. Default is “.” (the current directory).

This property specifies the directory in which the client stores random data for the use of SSL operations. The directory should be specified as an absolute path (for example, D:\mysys\work). The directory must currently exist and be writeable.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PATH:&lt;path&gt;</td>
<td>&lt;path&gt; is a user-specified path for loading certificates specified in this SSL property list. It can be either an absolute path or a relative path. The default path is C:\U2\UniDK\certs. With this path, the behavior is the same as that of the DEFAULT option.</td>
</tr>
<tr>
<td>ENV:&lt;env-var&gt;</td>
<td>&lt;env-var&gt; is an environment variable name. With this option, the client process uses the value of the environment variable as the path to load the certificates. Note that UniVerse looks up the environment variable for a client process only once when the first SSL connection is made and its value is cached for later reference by that process.</td>
</tr>
</tbody>
</table>
By default, random data is stored in your UVHOME directory or the directory in which a client process runs. If you want to control where the random data is stored (for example, to limit users' access to the random data by storing it in a directory that has restricted permissions), you should use this property to specify the desired directory.

The random data file named U2SSL.rnd is created in the specified directory.
Using the U2 SSL configuration editor

The U2 SSL Configuration Editor is a graphical user interface (GUI) tool for creating and managing SSL property lists.

Working with SSL property lists

Use the U2 SSL Configuration Editor to create, edit, delete, copy, or rename an SSL property list. Using the tool ensures that the list is properly saved to (or deleted from) the Registry.

Starting the U2 SSL configuration editor

The U2 SSL Configuration Editor program files are placed in a subfolder under the Programs folder when you install UniVerse. This section explains how to navigate to the tool and start it. It also describes the layout of the U2 SSL Configuration Editor window.
Starting the U2 SSL configuration editor:

From the Start menu, choose All Programs -> Rocket U2 -> SSL Config Editor. The U2 SSL Configuration Editor window appears.

Components of this window are described below.

Main Menu

At the top of the U2 SSL Configuration Editor window are four menus:

<table>
<thead>
<tr>
<th>Menu</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>Options for opening, closing, saving, printing, and performing other tasks for managing SSL property lists.</td>
</tr>
</tbody>
</table>

U2 SSL Configuration Editor Main Menu
Shortcut toolbar

Under the main menu is a toolbar with shortcuts for the most common tasks. Roll the mouse over a shortcut tool to see a brief description of the task.

<table>
<thead>
<tr>
<th>Menu</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edit</td>
<td>Options for performing standard Windows file edit actions, including undo, redo, cut, copy, paste, and delete.</td>
</tr>
<tr>
<td>Window</td>
<td>Options for controlling the view and navigation of panes in the U2 SSL Configuration Editor window.</td>
</tr>
<tr>
<td>Help</td>
<td>Options for accessing help.</td>
</tr>
</tbody>
</table>

U2 SSL Configuration Editor Main Menu (Continued)
The **U2 SSL Configuration Editor** window is divided into three panes:

<table>
<thead>
<tr>
<th>Pane</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>U2 SSL Property Explorer (left)</td>
<td>Use this pane to view the directory structure of SSL property lists and copy, rename, or delete existing SSL property lists.</td>
</tr>
<tr>
<td>Editor view (upper right)</td>
<td>This pane contains a <strong>Welcome</strong> tab with information about using the U2 SSL Configuration Editor.</td>
</tr>
<tr>
<td>Console/Problems view (lower right)</td>
<td>This pane contains two tabs: <strong>Console</strong> for viewing error and informational messages and a log of transactions performed in Trace mode. <strong>Problems</strong> for details on any problems encountered while creating, editing, deleting, or performing other operations on an SSL property list.</td>
</tr>
</tbody>
</table>

From the main window, you can perform the following tasks to manage SSL property lists:
- Creating a new SSL property list
- Editing an existing SSL property list
- Deleting an SSL property list
- Copying an SSL property list
- Renaming an SSL property list
- Using the trace feature
- Using the Console/Problems view
Creating a new SSL property list

This section takes you through the process of creating an SSL property list, defining all the properties of a secure connection.

The Create a New U2 SSL Property List dialog box provides a form for entering these properties, helping you input the required information. The requirements are based on whether the SSL property list is for the use of a client or a server, and on the certificate store type.

The properties are grouped on three pages of the dialog box. The instructions for creating a new SSL property list are broken down into tasks, with one task for each page of the dialog box:

- Task 1: Assign name, password, SSL version, and store type to property list
- Task 2: Specify certificates, private key and password, certificate revocation list, and cipher suites
- Task 3: Specify authentication properties
Task 1: Assign name, password, SSL version, and store type to property list

1. In the U2 SSL Configuration Editor window, select File -> New. The Create a New SSL Property List dialog box appears.

![Create a New SSL Property List dialog box]

On the SSL Property List Name, Password, SSL Version, and Store Type page of this dialog box, you define the basic properties of the SSL property list.

2. In the Property list name box, enter a unique name for the SSL property list to be created.
3. Optional. We strongly recommend that you establish a password for the SSL property list. An algorithm is applied to your password to derive a unique encryption key for the list. To access a password-protected list, users must enter the password as the key to decrypt the list and view its plaintext contents. If you do not assign a password to the list, the algorithm uses a fixed internal default password to generate the encryption key. The key produced in this manner never varies and anyone who uses the U2 SSL Configuration Editor can access the list and view its contents.

In the **Password** box, enter a password for the SSL property list. There are no limitations on length or restrictions on characters allowed; however, the length of the password and randomness of the characters contribute to its relative security. Use a password that is difficult to guess and share it only with users who need to access the list.

4. If you entered a password for the SSL property list, you must verify the password. In the **Re-enter password** box, type the same password again.

5. UniVerse supports SSL version 3 and TLS version 1. Under **SSL version**, select the version of the protocol to be used for this secure connection:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSLv3</td>
<td>This is the default setting. It is the most widely used protocol.</td>
</tr>
<tr>
<td>TLSv1</td>
<td>This is the newer protocol. Most newer applications support it, but some older applications may not.</td>
</tr>
</tbody>
</table>

**SSL Versions**
6. Under **Certificate store type**, select the type of certificate stores to be used for all certificates issued for this secure connection:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>U2</td>
<td>This is the default setting. Use this setting if all certificates that apply to this secure connection are PEM or DER format OS-level files.</td>
</tr>
</tbody>
</table>
| Windows | All certificates for this connection are looked up from the native Windows certificate store. Generally, a CA certificate is looked up from Windows CA and ROOT stores, while My Certificate is looked up from MY stores. In Microsoft’s terminology, these certificate stores are *system stores*: a collection of physical certificate stores that reside in the Windows Registry. UniData looks up these stores from both of the following Registry locations:  
  - `CERT_SYSTEM_STORE_CURRENT_USER`
  - `CERT_SYSTEM_STORE_LOCAL_MACHINE` |

**Certificate Store Type**
Choose one of the following actions:

- To discard your entries and cancel the process of creating an SSL property list, click **Cancel**.
- Otherwise, to continue defining properties of the new SSL property list, click **Next**.

The **Certificates, Private Key and Password, CRL, and Cipher Suites** page of the **Create a New SSL Property List** dialog box appears.

On this page of the dialog box, you specify the path of a certificate, set the private key and password if applicable, specify the path of the certificate revocation list (CRL), and specify cipher suites to be used in the handshake.
Task 2: Specify certificates, private key and password, certificate revocation list, and cipher suites

1. If applicable, in the CA certificate box, enter the path of the file to contain a certificate authority (CA) certificate for this secure connection, or click Browse to find the path. See specifics for the certificate store type below.

**U2 certificate store type:**

Specify the path of the certificate file that is used as a CA certificate. The format of the certificate can be either PEM or DER. With the U2 type, you can specify multiple certificate paths, separating each with a semicolon (;

If a CA certificate chain is required, you have the choice of specifying multiple certificate files in the CA certificate box, or, for PEM-format certificates, concatenating the certificate files into one single file (using OS-level editor or command line) and specifying the concatenated file once.

**Windows certificate store type:**

Specify the same “friendly name” or “Common name” that is used for the certificate in the certificate store. With the Windows type, specify only one certificate, generally the most immediate CA certificate (the one used directly to sign the certificate to which authentication is to be performed).

A certificate chain is automatically established and used in an SSL session. Note that the above description is based on the assumption that a correct and complete trust relationship, for example, a complete CA certificate chain, exists in the Windows certificate store for the certificate involved. If a complete chain cannot be formed, an error is reported. This also applies to other certificate-related properties.

2. Optional for a client SSL property list; required for a server SSL property list.

In the My Certificate box, enter the path for your certificate for this secure connection, or click Browse to find the path. See specifics for the certificate store type below.
U2 certificate store type:

Note that if you specify a path in My Certificate for a server SSL property list, you must also enter values for Private key and Private key password. The format of the certificate can be either PEM or DER.

Windows certificate store type:

Specify the same “friendly name” or “Common name” that is used for the certificate in the certificate store. Note that when you import a Windows store type certificate into the MY store, you must associate an exportable private key with it by selecting the Exportable private key check box.

3. Applicable to the U2 certificate store type only. Required if you entered a value in My Certificate.

   In the Private key box, enter the path for the file that contains the private key associated with My Certificate, or click Browse to find the path. The format of the key file can be either PEM or DER.

   When an SSL property list is created, the private key is loaded into memory and validated against its corresponding certificate (My Certificate). If it passes validation, the key is stored with the SSL property list. This validation feature is designed to enhance the security and protection of the user’s private key.

   After the SSL property list has been created, you do not need to keep the private key file on your hard drive. You can store the key file safely on external media until the next time you want to modify properties of the SSL property list.

4. Applicable to the U2 certificate store type only. Required if you entered a value in My Certificate.

   In the Private key password box, enter the password for the private key file.
5. Optional. In the CRL box, enter the path of a certificate revocation list (CRL) to be used for this secure connection, or click Browse to find the path. You can specify multiple CRL paths, separating each with a semicolon (;).

The CRL is a special certificate published by the certificate authority (CA), containing the serial numbers of certificates that the CA has revoked. During the SSL session handshake, the incoming server certificate is checked against the CRL to verify that the certificate has not been revoked before other verification is performed.

The format of the CRL can be either PEM or DER.

6. Optional. In the Cipher Suites box, specify a suite of ciphers to be used in a specific order in the SSL handshake. If you make no entry, the default of all ciphers supported by the OpenSSL open source library applies.

7. Choose one of the following actions:
   - To return to the previous page of the dialog box, click Back.
   - To discard your entries and cancel the process of creating an SSL property list, click Cancel.
   - Otherwise, to continue defining properties of the new SSL property list, click Next.
The **Authentication Properties** page of the **Create a New SSL Property List** dialog box appears.

On this page of the dialog box, you specify properties related to peer authentication for the secure connection.
Task 3: Specify authentication properties

1. Optional. In the Trusted peers box, enter the name of a trusted peer as detailed below.

   This property tells UniVerse that additional checking needs to be performed in verifying the incoming certificate. If you leave this box blank, the incoming certificate is considered valid when the CA certificate has verified it. However, if you specify a trusted peer name, a further check is performed to verify that the incoming certificate’s SubjectAltName extension or CommonName subject field matches that of the trusted peer.

   The trusted peer name can be either a fully specified name (such as admin@us.xyz.com) or a wildcard name. Two wildcard characters are supported:

   %    Match any character string
   _    Match one character

   For example, %@us.xyz.com matches both admin@us.xyz.com and market@us.xyz.com, while admi_@us.xyz.com matches admin@us.xyz.com only.

   You can enter the names of multiple trusted peers, separating each with a semicolon (;).

2. Optional. In the Random file location box, enter the absolute path of the directory in which UniVerse stores random data for the use of SSL operations, or click Browse to find the path. For example, D:\mysys\work is an absolute path. The directory must currently exist and be writable. The default is “.” (the current directory).

   By default, random data is stored in your UVHOME directory or the directory in which a client process runs. If you want to control where the random data is stored (for example, to limit users’ access to the random data by storing it in a directory that has restricted permissions), use this property to specify the desired directory.

   When the SSL property list is created, the random data file named U2SSL.rnd is created in the directory specified here.
3. Optional. In the **Authentication depth** list, select the level at which to stop UniVerse’s verification process in authentication processing. The default setting is 5, which is a sufficient depth in most cases. If you set the authentication depth for fewer levels of authentication than actually employed for the certificate, the certificate will not pass authentication.

4. Applicable to a server SSL property list only. Optional.
   Under **Client authentication**, if the SSL server using this property list requires client authentication during the SSL handshake, select the **Require client authentication** check box. A server that requires client authentication asks the client to send its certificate as an additional security measure.

   If you select this check box, UniVerse treats the SSL property list as a server property list. For a server property list, you must also specify these properties:
   - CA certificate
   - My Certificate
   - Private key (U2 certificate store type only)
   - Private key password (U2 certificate store type only)

   If you leave a required property blank, the U2 SSL Configuration Editor issues an error message after you click Finish, and redisplays the first page on which you need to enter missing information.
5. Optional. Under **Authentication strength**, select the appropriate option for this secure connection.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| Strict | This is the default setting. Strict authentication requires that the following conditions be met:  
- The incoming server certificate is a well-formed X.509 certificate.  
- A complete CA certificate chain exists and verifies the incoming server certificate.  
- Peer name checking (if specified) is successfully performed. |
| Generous | Generous authentication requires only that the incoming server certificate is a well-formed X.509 certificate.  
**Note:** Generous authentication is not highly secure. We recommend using it in test environments only. |

### Authentication Strength

6. Applicable to U2 certificate store type only. Optional.

When you specify a certificate by the CA certificate, My Certificate, or CRL property, the value for that property is registered internally. When the certificate is loaded into memory to establish an SSL connection, UniVerse uses this registered path by default to retrieve the certificate.

The **Certificate path** property allows you to specify different locations in which to search the certificates. Note that this property applies to all certificates in the file.

Under **Certificate path**, select one of the following options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>Specifies the above-described behavior.</td>
</tr>
<tr>
<td>Relative</td>
<td>UniVerse looks for the certificate in the current directory under which the client process is running.</td>
</tr>
</tbody>
</table>

### Certificate Path Options
Certificate Path Options (Continued)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>Enter the path for loading certificates specified in this property list, or click <strong>Browse</strong> to find the path. This can be either an absolute path or a relative path. The default path is C:\U2\UniDK\certs. With this path, the behavior is the same as that of the <strong>Default</strong> option.</td>
</tr>
<tr>
<td>Environment Variable</td>
<td>Enter an environment variable name. With this option, the value of the environment variable is used as the path in which to load the certificates. Note that UniVerse looks up the environment variable for a client process only the first time the process makes an SSL connection; the value of the environment variable is cached for later reference by that process.</td>
</tr>
</tbody>
</table>
7. Choose one of the following actions:

- To return to the previous page of the dialog box, click **Back**.
- To discard your entries and cancel the process of creating an SSL property list, click **Cancel**.
- Otherwise, to finish entry of properties and create the SSL property list, click **Finish**.

The U2 SSL Configuration Editor tool checks your entries to ensure that you have input all required properties. The requirements are based on whether this is a client or server SSL property list, and on the selected certificate store type.

If you left a required property blank or entered conflicting or inconsistent values in related properties, when you click **Finish** the U2 SSL Configuration Editor issues an error message and redispays the first page on which you to need to enter information.

If the tool finds no errors, the program creates the new SSL property list, saving it in encrypted form to the Windows Registry at:

For 32-bit platforms:

HKEY_LOCAL_MACHINE/SOFTWARE/ROCKET SOFTWARE/UniClient/SPL

For 64-bit platforms:

HKEY_LOCAL_MACHINE/SOFTWARE/WoW6432Node/Rocket Software/UniClient/SPL/myspl
Editing an existing SSL property list

This section takes you through the process of editing an existing SSL property list, changing the properties of a secure connection.

To edit an existing SSL property list:

1. In the U2 SSL Configuration Editor window, open the U2 SSL Property Explorer pane if it is not already displayed. To open this pane, choose Window -> Show View -> U2 SSL Property Explorer.

2. In the U2 SSL Property Explorer pane, double-click the name of the SSL property list to be edited. The Property List Password dialog box appears.

3. If the selected SSL property list has an associated password, enter the password and click OK. Otherwise, if the property list has no associated password, leave the box blank and click OK.

   Note: If the SSL property list does not have an associated password, you can rename the list and enter a password during this process. For instructions, see “Renaming an SSL property list” on page 54.
The SSL property list opens in the Editor view in the upper right pane of the **U2 SSL Configuration Editor** window.

The Editor view is split into two components: the **Property List** on the left side and the **Property Editor** on the right.

4. In the **Property List**, select the line containing a property value to be changed. The **Property Editor** displays information for the selected property.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Property</td>
<td>Display only. This box contains the name of the property as it is stored in the U2 SSL Configuration Editor program. Property names cannot be changed.</td>
</tr>
<tr>
<td>Description</td>
<td>Provides guidelines and tips for setting the value of this property.</td>
</tr>
<tr>
<td>Value</td>
<td>Initially displays the current value of the property. In this box, you can change the value of the selected property.</td>
</tr>
</tbody>
</table>
The following table provides information on changing the value of each SSL property. This table lists properties in the order in which they appear in the **Property List** on the left side of the Editor view.

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
</table>
| SSLVersion        | UniVerse supports SSL version 3 and TLS version 1. Select the version of the protocol to be used for this secure connection:  
- **SSLv3** – This is the default setting. It is the most widely used protocol.  
- **TLSv1** – This is the newer protocol. Most newer applications support it, but some older applications may not.  
To apply this change, click **OK**. |
| CertificateStoreType | Select the type of certificate stores to be used for all certificates issued for this secure connection.  
- **U2** – This is the default setting. Use this setting if all certificates that apply to this secure connection are PEM or DER format OS-level files.  
- **Windows** – All certificates for this connection are looked up from the native Windows certificate store. Generally, a CA certificate is looked up from Windows CA and ROOT stores, while My Certificate is looked up from MY stores.  
In Microsoft’s terminology, these certificate stores are *system stores*: a collection of physical certificate stores that reside in the Windows Registry. UniVerse looks up these stores from both of the following Registry locations:  
**CERT_SYSTEM_STORE_CURRENT_USER**  
**CERT_SYSTEM_STORE_LOCAL_MACHINE**  
To apply this change, click **OK**. |
<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CACertificate</td>
<td>Enter the path of the file to contain a certificate authority (CA) certificate for this secure connection, or click <strong>Browse</strong> to find the path. See specifics for the certificate store type below.</td>
</tr>
</tbody>
</table>

**U2 certificate store type:**

Specify the path of the certificate file that is used as a CA certificate. The format of the certificate can be either PEM or DER. With the U2 type, you can specify multiple certificate paths, separating each with a semicolon (;).

If a CA certificate chain is required, you have the choice of specifying multiple certificate files, or, for PEM-format certificates, concatenating the certificate files into one single file (using OS-level editor or command line) and specifying the concatenated file once.

**Windows certificate store type:**

Specify the same “friendly name” or “Common name” that is used for the certificate in the certificate store. With the Windows type, specify only one certificate path, generally the most immediate CA certificate (the one used directly to sign the certificate to which authentication is to be performed).

A certificate chain is automatically established and used in an SSL session. Note that the above description is based on the assumption that a correct and complete trust relationship, for example, a complete CA certificate chain, exists in the Windows certificate store for the certificate involved. If a complete chain cannot be formed, an error is reported. This also applies to other certificate-related properties.

To apply this change, click **OK**.

---

*Editing Property Values (Continued)*
<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MyCertificate</td>
<td>Optional for a client SSL property list; required for a server SSL property list. Enter the path for your certificate for this secure connection, or click Browse to find the path. See specifics for the certificate store type below.</td>
</tr>
<tr>
<td></td>
<td><strong>U2 certificate store type:</strong></td>
</tr>
<tr>
<td></td>
<td>Note that if you specify a path in MyCertificate for a server SSL property list, you must also enter values for MyPrivateKey and PrivateKey-Password. The format of the certificate can be either PEM or DER.</td>
</tr>
<tr>
<td></td>
<td><strong>Windows certificate store type:</strong></td>
</tr>
<tr>
<td></td>
<td>Specify the same “friendly name” or “Common name” that is used for the certificate in the certificate store. Note that when you import a Windows store type certificate into the MY store, you must associate an exportable private key with it by selecting the Exportable private key check box.</td>
</tr>
<tr>
<td></td>
<td>To apply this change, click OK.</td>
</tr>
</tbody>
</table>
Applicable to the U2 certificate store type only. Required if you entered a value in **MyCertificate**.
Enter the path for the file that contains the private key associated with My Certificate, or click **Browse** to find the path. The format of the key file can be either PEM or DER.

When an SSL property list is created, the private key is loaded into memory and validated against its corresponding certificate (My Certificate). If it passes validation, the key is stored with the SSL property list. This validation feature is designed to enhance the security and protection of the user’s private key.

After the SSL property list has been created, you do not need to keep the private key file in memory. You can store the key file safely on media until the next time you want to modify properties of the SSL property list.

To apply this change, click **OK**.

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MyPrivateKey</td>
<td>Applicable to the U2 certificate store type only. Required if you entered a value in <strong>MyCertificate</strong>. Enter the path for the file that contains the private key associated with My Certificate, or click <strong>Browse</strong> to find the path. The format of the key file can be either PEM or DER. When an SSL property list is created, the private key is loaded into memory and validated against its corresponding certificate (My Certificate). If it passes validation, the key is stored with the SSL property list. This validation feature is designed to enhance the security and protection of the user’s private key. After the SSL property list has been created, you do not need to keep the private key file in memory. You can store the key file safely on media until the next time you want to modify properties of the SSL property list. To apply this change, click <strong>OK</strong>.</td>
</tr>
<tr>
<td>PrivateKeyPassword</td>
<td>Applicable to the U2 certificate store type only. Required if you entered a value in <strong>MyCertificate</strong>. Enter the password for the private key file. To apply this change, click <strong>OK</strong>.</td>
</tr>
</tbody>
</table>

**Editing Property Values (Continued)**
TrustedPeerName

Optional. Enter the name of a trusted peer as detailed below.

This property tells UniVerse that additional checking needs to be performed in verifying the incoming certificate. If you leave this box blank, the incoming certificate is considered valid when the CA certificate has verified it. However, if you specify a trusted peer name, a further check is performed to verify that the incoming certificate’s SubjectAltName extension or CommonName subject field matches that of the trusted peer.

The trusted peer name can be either a fully specified name (such as admin@us.xyz.com) or a wildcard name. Two wildcard characters are supported:

%  Match any character string

_  Match one character

For example, %@us.xyz.com matches both admin@us.xyz.com and market@us.xyz.com, while admi_@us.xyz.com matches admin@us.xyz.com only.

You can enter the names of multiple trusted peers, separating each with a semicolon (;).

To apply this change, click OK.
Authentication-Strength

Optional. Select the appropriate authentication strength option for this secure connection:

- **STRICT** – This is the default setting. Strict authentication requires that the following conditions be met:
  - The incoming server certificate is a well-formed X.509 certificate.
  - A complete CA certificate chain exists and verifies the incoming server certificate.
  - Peer name checking (if specified) is successfully performed.

- **GENEROUS** – This strength requires only that the incoming server certificate is a well-formed X.509 certificate. Note that generous authentication is not highly secure. We recommend its use in test environments only.

To apply this change, click **OK**.
CertificatePath Applicable to U2 certificate store type only. Optional.

When you specify a certificate by the CACertificate, MyCertificate, or CRL property, the value for that property is registered internally. When the certificate is loaded into memory to establish an SSL connection, UniVerse uses this registered path by default to retrieve the certificate.

The CertificatePath property allows you to specify different locations in which to search the certificates. Note that this property applies to all certificates in the file. Select one of the following options:

- **DEFAULT** – Specifies the above-described behavior.
- **RELATIVE** – UniVerse looks for the certificate in the current directory under which the client process is running.
- **ENV** – Enter an environment variable name. With this option, the value of the environment variable is used as the path in which to load the certificates. Note that UniVerse looks up the environment variable for a client process only the first time the process makes an SSL connection; the value of the environment variable is cached for later reference by that process.
- **PATH** – Enter the path for loading certificates specified in this property file, or click Browse to find the path. This can be either an absolute path or a relative path. The default path is C:\U2\UniDK\certs. With this path, the behavior is the same as that of the Default option.

To apply this change, click OK.
<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CipherSuite</td>
<td>Optional. Specify a suite of ciphers to be used in a specific order in the SSL handshake. If you make no entry, the default of all ciphers supported by the OpenSSL open source library applies. To apply this change, click OK.</td>
</tr>
<tr>
<td>AuthenticationDepth</td>
<td>Optional. Enter the level at which to stop UniVerse’s verification process in authentication processing. The default setting is 5, which is a sufficient depth in most cases. If you specify a depth with fewer levels of authentication than actually employed for the certificate, the certificate will not pass authentication. To apply this change, click OK.</td>
</tr>
<tr>
<td>CRL</td>
<td>Optional. Enter the path of a certificate revocation list (CRL) to be used for this secure connection, or click Browse to find the path. You can specify multiple CRL paths, separating each with a semicolon (;). The CRL is a special certificate published by the certificate authority (CA), containing the serial numbers of certificates that the CA has revoked. During the SSL session handshake, the incoming server certificate is checked against the CRL to verify that the certificate has not been revoked before other verification is performed. The format of the CRL can be either PEM or DER. To apply this change, click OK.</td>
</tr>
</tbody>
</table>

Editing Property Values (Continued)
ClientAuthentication

Applicable to a server SSL property list only. Select the appropriate option for this secure connection:

- **true** – Use this setting if the SSL server using this property list requires client authentication during the SSL handshake. A server that requires client authentication asks the client to send its certificate as an additional security measure.

  If you select true, UniVerse treats the SSL property list as a server property list. For a server property list, you must also specify these properties:
  - CACertificate
  - MyCertificate
  - MyPrivateKey (U2 certificate store type only)
  - PrivateKeyPassword (U2 certificate store type only)

  If you leave a required property blank, the U2 SSL Configuration Editor issues an error message after you click Finish, and redisplays the first page on which you need to enter missing information.

- **false** – Use this setting if the SSL server does not require client authentication.

To apply this change, click **OK**.
5. When you have finished making changes to the properties in this SSL property list, take one of the following actions:

- To save your changes to the list, click the Save button in the Property List panel.

- To save your changes as a new SSL property list, click the Save As button in the Property List panel. The Property List Name and Password dialog box appears. Enter a unique name for the new list, enter a password, and re-enter the password. Click OK.

---

**Property** | **Value**
---|---
RandomFileLocation | Optional. Enter the absolute path of the directory in which UniVerse stores random data for the use of SSL operations, or click **Browse** to find the path. For example, D:\mysys\work is an absolute path. The directory must currently exist and be writable. The default is "." (the current directory).

By default, random data is stored in the UVHOME directory or the directory in which a client process runs. If you want to control where the random data is stored (for example, to limit users’ access to the random data by storing it in a directory that has restricted permissions), use this property to specify the desired directory.

When the SSL property list is created, the random data file named U2SSL.rnd is created in the directory specified here.

To apply this change, click **OK**.
Deleting an SSL property list

This section shows you how to delete an SSL property list. It is important that you use the U2 SSL Configuration Editor to perform this task so the file is properly deleted from the Windows Registry.

To delete an SSL property list:

1. In the U2 SSL Configuration Editor window, open the U2 SSL Property Explorer pane if it is not already displayed. To open this pane, choose Window ? Show View ? U2 SSL Property Explorer.
2. In the U2 SSL Property Explorer pane, select the SSL property list to be deleted.
3. Click the X button. The Property List Password dialog box appears.
4. If the selected SSL property list has an associated password, enter the password and click OK. Otherwise, if the property list has no associated password, leave the box blank and click OK.
5. The Please Confirm dialog box appears. The message states that you are about to delete an SSL property list and requests your confirmation to proceed.

   If you want to cancel the deletion, click Cancel.
   Otherwise, if you want to complete the procedure and delete the SSL property list, click OK. The SSL property list is deleted from the Registry.
Copying an SSL property list

This section details the steps for copying an SSL property list. The copy function allows you to create a new list from an existing list.

You can use this function for two different purposes:

- Create a list that is similar to the original – When you have a new list, you can edit its properties, specifying the characteristics of a secure connection that is similar to the connection defined by the original list.
- Rename an existing list and assign it a password – If an existing list has no password or you want to change its password, you can use this function to rename the list and assign a new password. You can then delete the original list if it is no longer needed.

Do not copy an SSL property list by any method other than the U2 SSL Configuration Editor. You must use this tool so the list is entered properly in the Registry.

To copy an SSL property list:

1. In the U2 SSL Configuration Editor window, open the U2 SSL Property Explorer pane if it is not already displayed. To open this pane, choose Window > Show View > U2 SSL Property Explorer.
2. In the U2 SSL Property Explorer pane, right-click the SSL property list to be copied.
3. Select the **Copy** option. The **Property List Password** dialog box appears.

![Property List Password dialog box](image)

4. To continue with the copy procedure,
   - If the SSL property list to be copied has an associated password, enter the password and click **OK**.
   - If the property list has no associated password, leave the box blank and click **OK**.
     
     The Console displays the message “List 'listname' has been copied successfully.”

   Otherwise, to cancel the copy procedure, click **Cancel**.

5. The next task is to paste the copied list in the folder. In the **U2 SSL Property Explorer** pane, right-click the U2 SSL Property Lists folder.
6. Select the **Paste** option. The **Property List Name and Password** dialog box appears.

![Property List Name and Password dialog box](image)

- **Enter name for new property list:**
  - CopyOfabcd

- **Enter password for property list “abc” (if no password, click OK):**

- **Re-enter password:**

7. In the **Enter name for new property list** box, the system-generated name for the new list is highlighted. Enter a unique name for the new list.

8. Optional. In the **Enter password for property list** box, assign a password to the new list. To increase the level of security, we strongly recommend that you establish a password for the SSL property list.

9. If you entered a password for the SSL property list, you must verify the password. In the **Re-enter password** box, type the same password again.

10. Take one of the following actions:
    - To paste the new list into the selected folder, click **OK**.
    - The Console displays the message “New list has been created successfully.”
    - To cancel the paste procedure, click **Cancel**.
Renaming an SSL property list

This section provides instructions for renaming an SSL property list. The rename function allows you to change the name of an existing list by overwriting the old name.

Do not rename an SSL property list by any method other than the U2 SSL Configuration Editor. You must use this tool so the list is entered properly in the Registry.

To rename an SSL property list:

1. In the U2 SSL Configuration Editor window, open the U2 SSL Property Explorer pane if it is not already displayed. To open this pane, choose Window ?Show View ?U2 SSL Property Explorer.
2. In the U2 SSL Property Explorer pane, right-click the SSL property list to be renamed.
3. Select the Rename option. The Property List Name and Password dialog box appears.

![Property List Name and Password dialog box](image)

4. In the Enter name for new property list box, enter a unique name for the list.
5. To continue with the rename procedure,
   - If the SSL property list to be renamed has an associated password, enter the password and click **OK**.
   - If the property list has no associated password, leave the box blank and click **OK**.

   The Console displays the message “List ‘old_listname’ has been renamed to ‘new_listname’.”

   Otherwise, to cancel the rename procedure, click **Cancel**.
Using the trace feature

The U2 SSL Configuration Editor provides a Trace feature for recording all operations performed through the tool on SSL property lists. The events of these operations are written to a file named U2SSLConfig.log and also displayed in the Console pane.

You can use the log to track activity on the lists and to troubleshoot any problems that may arise when performing operations on the lists.

The log is located by default in your C:\temp folder. If you have no \temp folder, the log is written to the \tmp folder. If no \tmp folder exists, the program creates a \temp folder. The file name for the log cannot be changed.

When you initially open the U2 SSL Configuration Editor, Trace mode is turned off by default. This section contains instructions for turning Trace mode on and off.

To use the Trace feature:

1. In the U2 SSL Configuration Editor window, choose File.
2. If the Trace option is not check-marked, select it.
   When Trace mode is active, the Trace option is preceded by a check mark on the menu.
3. With Trace mode turned on, perform operations on SSL property lists as you normally would. The events of these operations are recorded in the log.
4. To turn off Trace mode, choose File ?Trace.
5. Navigate to the folder containing the log and open the file to view its contents.
Using the Console/Problems view

The lower right pane of the U2 SSL Configuration Editor window provides two views that help you manage the tasks performed on SSL property lists:

- **Console** for viewing error/informational messages and a log of transactions performed in Trace mode.
- **Problems** for details on any problems encountered while creating, editing, deleting, or performing other transactions on SSL property lists.

You can switch back and forth from Console view to Problems view, or close and open a view as needed.

**To use the Console view:**

1. In the U2 SSL Configuration Editor window, open the **Console** view if it is not already displayed. To open this view, choose **Window -> Show View -> Console**.

2. Optional. If you want to keep a log of transactions performed on SSL property lists, turn on Trace mode. If you need instructions, see “Using the trace feature” on page 56.
3. Perform transactions on SSL property lists as you normally would. Messages and results from these transactions are displayed in the Console. If Trace mode is active, a log of transactions is displayed in the Console.

4. To close this view, click X on the Console tab.

**To use the Problems view:**

5. In the U2 SSL Configuration Editor window, open the Problems view if it is not already displayed. To open this view, choose Window > Show View > Problems.
6. Perform transactions on SSL property lists as you normally would. The details of any problems encountered are displayed in the Problems view.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>A description of a problem encountered while performing transactions on an SSL property list. Each problem is listed on a separate line.</td>
</tr>
<tr>
<td>Resource</td>
<td>The name of the file that stores messages regarding the problem.</td>
</tr>
<tr>
<td>&lt;MSG_PRBL_HOLDER&gt;</td>
<td>The folder that contains the file with messages about the problem.</td>
</tr>
<tr>
<td>Location</td>
<td>The path of the message file for the problem.</td>
</tr>
</tbody>
</table>

7. To close this view, click X on the Problems tab.
Chapter 5: Automatic Data Encryption

Overview .................................................. 5-4
Features ..................................................... 5-4
Automatic Data Encryption components .......... 5-6
  Master key ............................................... 5-6
  The key store .......................................... 5-7
  The metadata store .................................... 5-7
  Encryption keys and wallets ....................... 5-8
  Encryption engine ..................................... 5-9
  Password policies ..................................... 5-9
  Management utilities and tools ..................... 5-10
Setting up Automatic Data Encryption .......... 5-11
  New installation of UniVerse ....................... 5-11
  Upgrade installation from UniVerse 10.3 and later releases .... 5-11
  Upgrade installation from UniVerse 10.2 ............. 5-12
  Setting up the master key ........................... 5-14
  Creating the key store ............................... 5-15
  Configuring password policies ..................... 5-16
Performing Automatic Data Encryption tasks ..... 5-18
  No-echo Automatic Data Encryption command mode ...... 5-18
  TCL commands vs. the Extensible Administration Tool .... 5-18
  Creating keys and wallets ........................... 5-18
  Deleting encryption keys and wallets ............... 5-19
  Granting and revoking access ....................... 5-20
  Encrypting files ...................................... 5-21
  Encryption modes ..................................... 5-21
  Encryption parameters .............................. 5-23
  Encryption algorithms ............................... 5-23
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encrypting a file</td>
<td>5-24</td>
</tr>
<tr>
<td>Decrypting a file</td>
<td>5-25</td>
</tr>
<tr>
<td>Reencrypting a file</td>
<td>5-26</td>
</tr>
<tr>
<td>Encrypting the @ID field</td>
<td>5-27</td>
</tr>
<tr>
<td>Encrypting indexes</td>
<td>5-28</td>
</tr>
<tr>
<td>Finding information about encrypted files</td>
<td>5-31</td>
</tr>
<tr>
<td>Accessing data in encrypted files</td>
<td>5-33</td>
</tr>
<tr>
<td>Passing key activation and decryption disablement to a PHANTOM process</td>
<td>5-39</td>
</tr>
<tr>
<td>Changing key passwords</td>
<td>5-39</td>
</tr>
<tr>
<td>Managing Automatic Data Encryption</td>
<td>5-41</td>
</tr>
<tr>
<td>Adding or changing the master key password</td>
<td>5-41</td>
</tr>
<tr>
<td>Changing the master key</td>
<td>5-44</td>
</tr>
<tr>
<td>Backing up the key store and metadata store</td>
<td>5-45</td>
</tr>
<tr>
<td>Retagging a key store</td>
<td>5-48</td>
</tr>
<tr>
<td>Managing password policies</td>
<td>5-49</td>
</tr>
<tr>
<td>Automatic Data Encryption and UniVerse high availability</td>
<td>5-52</td>
</tr>
<tr>
<td>A step-by-step guide for high availability</td>
<td>5-52</td>
</tr>
<tr>
<td>Methods for encrypting files marked or intended to be marked for UniVerse high availability</td>
<td>5-54</td>
</tr>
<tr>
<td>UniVerse encryption algorithms</td>
<td>5-58</td>
</tr>
<tr>
<td>Encryption commands</td>
<td>5-59</td>
</tr>
<tr>
<td>CREATE. ENCRYPTION. KEY</td>
<td>5-59</td>
</tr>
<tr>
<td>CREATE. ENCRYPTION. WALLET</td>
<td>5-60</td>
</tr>
<tr>
<td>DELETE. ENCRYPTION. WALLET</td>
<td>5-60</td>
</tr>
<tr>
<td>WALLET. ADD. KEY</td>
<td>5-61</td>
</tr>
<tr>
<td>WALLET. REMOVE. KEY</td>
<td>5-62</td>
</tr>
<tr>
<td>LIST. ENCRYPTION. WALLET</td>
<td>5-62</td>
</tr>
<tr>
<td>DELETE. ENCRYPTION. KEY</td>
<td>5-63</td>
</tr>
<tr>
<td>LIST. ENCRYPTION. KEY</td>
<td>5-64</td>
</tr>
<tr>
<td>GRANT. ENCRYPTION. KEY</td>
<td>5-64</td>
</tr>
<tr>
<td>REVOKE. ENCRYPTION. KEY</td>
<td>5-65</td>
</tr>
<tr>
<td>ENCRYPT. FILE</td>
<td>5-66</td>
</tr>
<tr>
<td>REENCRYPT. FILE</td>
<td>5-68</td>
</tr>
<tr>
<td>ENCRYPT. INDEX</td>
<td>5-69</td>
</tr>
<tr>
<td>DECRYPT. FILE</td>
<td>5-70</td>
</tr>
</tbody>
</table>
REENCRYPT.FILE ........................................ 5-72
DECRYPT.INDEX ....................................... 5-73
REENCRYPT.INDEX .................................... 5-73
LIST.ENCRYPTION.FILE .............................. 5-74
ACTIVATE.ENCRYPTION.KEY ......................... 5-75
DEACTIVATE.ENCRYPTION.KEY ....................... 5-76
DISABLE.DECRYPTION .................................. 5-76
ENABLE.DECRYPTION .................................. 5-77
The CHANGE.ENCRYPTION.PASSWORD ................ 5-78

UniVerse BASIC encryption commands ............... 5-80
ACTIVATEKEY ......................................... 5-80
DEACTIVATEKEY ...................................... 5-81
DISABLEDEC ........................................... 5-82
ENABLEDEC .............................................. 5-84
STATUS function changes ............................ 5-85

Automatic Data Encryption utilities reference .... 5-87
Protecting master keys with multiple, changeable passwords 5-87
Defining a master key .................................. 5-87

The encman utility .................................... 5-90
Generating a key store ................................ 5-90
Removing a key store .................................. 5-90
Importing and exporting key store and store metadata 5-91
Resynching tags ....................................... 5-92
Setting up password policies ........................ 5-93
Generating the uvrolf password file ................ 5-96

Password policies ..................................... 5-97
Overview

Over the past decades, data, be it of customer, marketing, or product development plans, has become the most valuable asset of any business. As business relies more and more on computer and web-based information systems to perform its critical functions, data security has increasingly evolved into a topic of interest for management and IT staff alike.

Driven by government regulations such as the Health Insurance Portability and Accountability Act (HIPAA), by industry-wide standards such as the Payment Card Industry – Data Security Standard (PCI-DSS), by the increase in problems like malicious computer attacks and identity theft, and by general demand from customers and vendors, companies are looking for ways to better protect their data resources against unauthorized access and disclosure. Data encryption is a popular and powerful tool to help address the data security problem.

Rocket Software’s UniData and UniVerse database products (collectively known as “U2”) provide sophisticated encryption technologies to aid businesses in meeting data security requirements. These requirements can conveniently be divided between securing “data in transit” (data moving between client and server, or across a public network) and securing “data at rest” (data stored on disk media, backup tapes, CD-ROM, or other static media). The U2 database products provide data encryption options appropriate for addressing both of these requirements.

Rocket Software’s solution to address the requirement for securing “data in transit” is to provide SSL support in the U2 engines and other client products. For securing “data at rest,” Rocket Software offers two solutions: the U2 Security API and Automatic Data Encryption (ADE). The U2 Security API is a set of UniVerse BASIC and UniBasic functions that address data confidentiality and integrity, and allow an application to control how data is encrypted and exchanged with third-party products, which Automatic Data Encryption is a comprehensive solution to satisfy the user’s need for securing “data at rest.”

Features

Automatic Data Encryption has the following features:
- Tightly integrated into the UniData and UniVerse engines
- Support in UniData and UniVerse components including clients, backup utilities, transaction logging, and replication
- Robust key and password management
- Flexible encryption modes
- Easy to manage by Graphical User Interface (GUI) tools and utilities

Once you set up Automatic Data Encryption, data will automatically be encrypted when it is written to encrypted files, and automatically decrypted when read from the files. Your application does not require significant changes to take advantage of Automatic Data Encryption, and you will be able to access encrypted data from UniData or UniVerse clients.

Currently, UniVerse Automatic Data Encryption only encrypts hashed data files. UniVerse does not encrypt directory files, system log files, dictionary files, or system temporary files. In addition, you cannot encrypt the VOC file. For UniVerse Transaction Logging and uvbackup, encrypted data (also known as cipher text) is stored in log files. For UniVerse Replication, UniVerse sends the stored cipher text from the publisher to the subscribers without decrypting in transit.

Although Automatic Data Encryption is easy to set up and use, you must take caution because a poorly planned or deployed encryption solution could be insecure and lead to long term problems. Users who want to deploy a secure and reliable encryption-enabled application must understand the technology behind Automatic Data Encryption, the relationship between various Automatic Data Encryption components, the trade-offs of different Automatic Data Encryption options, the possible impact on performance and storage space, and ways to protect and restore Automatic Data Encryption functions in case of system or environmental incidents.

The remainder of this chapter will prepare you with necessary information to fully understand Automatic Data Encryption and make correct and efficient use of its features.
Automatic Data Encryption components

This section describes the components of Automatic Data Encryption.

Master key

Automatic Data Encryption requires a master key to function. The master key is the most important secret of an Automatic Data Encryption based solution. It is used to protect other Automatic Data Encryption components, such as the key store and the metadata store. It is also required to perform certain Automatic Data Encryption management functions, such as exporting and importing the key store. The master key is also used to derive actual encryption keys. It is involved in all aspects of Automatic Data Encryption operations.

UniVerse stores the master key in your UniData or UniVerse installation’s license file in encrypted format. To enable Automatic Data Encryption on a UniData or UniVerse system, you must first set up the master key. You do this by executing the uvregen utility. See “Setting up the master key” on page 14 for more information.

Beginning at UniVerse 11.1, you can protect the master key with up to two passwords. By setting up two passwords and having different people keep them, you can achieve higher security in certain situations. A service provider can also the two-password scheme to satisfy certain compliance requirements by keeping one password with the provider and one with the customer to ensure no one party can modify the master key. You can also use the uvregen utility or the Extensible Administration Tool (XAdmin) to add or change the master key passwords. If the master key is protected by a password, you must supply the password wherever a master key is required in order to perform certain Automatic Data Encryption management functions.

How you set up your master key also depends on how your encrypted files are going to be shared among all of your data servers. If you want to share encrypted data among different servers, for example through data replication, you must perform an identical Automatic Data Encryption setup on these systems, including the master key.
The key store

The key store is a UniData or UniVerse hashed file that stores all of your encryption keys and wallets. Each key or wallet has a record in the key store, with its contents encrypted by the master key. It is also fortified against unauthorized access or modification. It can only be accessed through Automatic Data Encryption commands and tools. You cannot delete records from, copy records to, or otherwise modify the key store using TCL or UniVerse BASIC commands. The key store is meant to be used on the system on which it was created. If you physically move it to another system, UniVerse will deny access, unless you have the permission to overwrite the protection. For more information, see “Retagging a key store.”

The name of the key store is &KEYSTORE&, and it resides in the $UVHOME directory along with its dictionary. The dictionary contains definitions of all publicly accessible attributes.

UniVerse does not automatically create the key store during installation. After setting up the master key, you must then create the key store using the encman utility. You can also use the encman utility or the Extensible Administration Tool to back up or restore the key store. It is important to plan and perform periodic backups of your key store to prevent data loss.

The metadata store

The Automatic Data Encryption metadata store keeps encryption information for each encrypted file, including its encryption mode, encrypted fields and locations, as well as the algorithms and keys, with its contents encrypted by the master key. The metadata store is also fortified against unauthorized access or modification. It can only be accessed through Automatic Data Encryption commands and tools. You cannot delete records from, copy records to, or otherwise modify the metadata store using TCL or UniVerse BASIC commands. The metadata store is meant to be used on the system on which it was created. If you physically move it to another system, UniVerse will deny access, unless you have the permission to overwrite the protection. For more information, see “Retagging a key store.”

The name of the metadata store is &ENCINFO&, and it resides in the $UVHOME directory. It does not have any publicly accessible attributes.
UniVerse does not automatically create the metadata store during installation. After setting up the master key, you must create the metadata store using the encman utility. You can also use the encman utility or the Extensible Administration Tool to back up or restore it along with the key store. It is important to plan and perform periodic backups of your metadata store to prevent data loss.

**Encryption keys and wallets**

You need encryption keys to encrypt data. You can create encryption keys using TCL commands or the Extensible Administration Tool. By default, only the person who creates the key can use the key. The creator can grant access to other users through a TCL command or the Extensible Administration Tool. Access control is based on the operating system user account or group membership, and is the first layer of protection against unauthorized data access.

For ease of administration, we recommend that key creators and owners be selected as “functional users,” not individuals. In this way, the trail of key ownership is maintained even after individual users are deleted over time.

You can create an encryption key with or without a password. For a key without a password, it is always activated by default for those users who have been granted use of the key, and can be used to encrypt or decrypt data throughout a UniVerse session. For any key with a password, you must explicitly activate the key through a TCL command or UniVerse BASIC before it can be used to encrypt or decrypt data. Under certain circumstances, you may want to deactivate the key immediately after data access is done to further limit the data availability window. Key activation is another layer of protection against unauthorized data access.

Think of an encryption wallet as a bag that holds multiple encryption keys and their passwords. It is a means to manage key activation. When a wallet is activated, all keys in the wallet will be activated. For example, you can put all keys used to encrypt a specific file into a wallet. By activating the wallet, all encrypted data in this file can be accessed. Or you can put keys that belong to a department into a wallet. Then, all data encrypted for this department can be accessed by activating the wallet alone. The wallet functionality is especially valuable as a mechanism for UniVerse client products to access backend encrypted data.
A wallet can also be used to hide encryption key passwords from unnecessary disclosure. You can easily change a wallet’s password without changing the key passwords within the wallet. You can also use a wallet to function as an emergency password, one used in an emergency situation to allow an otherwise unauthorized party to access protected data. For example, a developer may need to look into some encrypted data to fix an application problem. Instead of disclosing the key password, which may have been used to protect other valuable data, you create a temporary wallet to contain this key and disclose the wallet password to the developer. After the developer finished his task, you delete the wallet, thus making the data once again inaccessible to the developer. Without wallets, you either risk unauthorized data disclosure, or you must reencrypt your data.

Encryption keys and wallet passwords are subject to Automatic Data Encryption password policies, which control the complexity, age, and history of passwords. To comply with regulations, standards, or internal policies, you may need to periodically change encryption key passwords. Beginning at UniVerse 11.1, you can do so using TCL commands or the Extensible Administration Tool.

Encryption engine

Automatic Data Encryption uses OpenSSL as its encryption engine. OpenSSL is a solid, secure open source package that provides crypto services inside many of today’s powerful applications. It supports many standard-based cryptographic algorithms, include ADE< DES, MD5 and SHA1.

UniData and UniVerse have been using OpenSSL for their SSL and encryption solutions since early 2000. It is seamlessly integrated into both database engines with enhanced tracing facility, and its invocations are transparent to users.

Password policies

Prior to UniVerse 11.1, Automatic Data Encryption key and wallet passwords did not have any restrictions. However, as data security becomes increasingly important and regulations, standards, and internal policies develop, more and more organizations are required to enforce password rules as a measure to increase data security.
Starting at UniVerse 11.1, Automatic Data Encryption supports password policy enforcement for encryption keys, wallets, and the master key. You can set up separate policies for keys, wallets, and the master key password. If you do not set up your own policies, a set of default policies will be enforced by UniVerse. You can explicitly turn off Automatic Data Encryption password policy enforcement.

UniVerse currently supports the following categories of password policies:

- Age policies – Controls the maximum and minimum age of the passwords, as well as how the warning is given before the passwords expire
- Complexity policies – Controls the maximum and minimum length, as well as the composition and pattern of the passwords
- History policies – Controls the reuse of previous passwords

Universe only applies the complexity policies when you create new keys and wallets, or change existing passwords for them. Existing keys created prior to UniVerse 11.1 are not affected. However, the existing keys and wallets are subject to the age and history policies.

UniVerse stores password policies in the .uvpspolicy file in the $UVHOME directory, encrypted by the master key. You should back up this file along with the key store and the metadata store.

You can manage password policies through the encman utility and the Extensible Administration Tool.

**Note:** Policy changes do not affect UniVerse sessions that are already started.

**Management utilities and tools**

Most Automatic Data Encryption management tasks can be performed either by operating system-level utilities or the Extensible Administration Tool. The only exception is Automatic Data Encryption set up. The initial master key and key store creation must be done by the uvregen utility and the encman utility, respectively and in that order. After the initial set up, subsequent management tasks, including master key password set up, key store back up, restore and retag, and password policy management can all be performed by operating system-level utilities and the Extensible Administration Tool.
Setting up Automatic Data Encryption

This section describes how to set up Automatic Data Encryption.

New installation of UniVerse

UniVerse does not automatically set up Automatic Data Encryption during the installation process. To set up Automatic Data Encryption, follow the procedures described in “Setting up the master key” on page 14 and “Creating the key store” on page 15.

Upgrade installation from UniVerse 10.3 and later releases

If you have previously set up Automatic Data Encryption and are installing UniVerse over the current installation directory, UniVerse attempts to migrate your Automatic Data Encryption setup into the new installation. During installation, it keeps your current key store and metadata store intact, and also tries to transfer your existing master key into the new license file.

The installation script automatically calls the uvregen command to check if an existing master key should be migrated. It searches the current directory to see if a .uvconfig file exists. If one exists, uvregen checks if there is an existing master key it can migrate. If any of the following conditions apply, UniVerse does not automatically migrate the master key:

- The source .uvconfig file is from UniVerse 10.2 or earlier
- The source .uvconfig file is not authorized
- The source .uvconfig file or target .uvconfig file is for the UniVerse Personal Edition
- The master key is not set up in the current .uvconfig file
- The source or target license file is not valid to be used on the hardware

If the automatic master key transfer fails, UniVerse will display a message, and you can choose one of the following options:

- Manually set up the master key using uvregen -m <master_key>, or
Manually transfer the master key by executing the following command:

```bash
$UVBIN/uvregen -T <source>,<target>
```

Where `<source>` is the path of the current license file, `<target>` is the new license file. For example:

```bash
/u2/uv/bin/uvregen -T ./.uvconfig.curr,.uvconfig
```

You must use this procedure if you need to change your UniVerse installation serial number for any reason. Having saved the old .uvconfig file and replaced it with the .uvconfig.bak file, transfer the master key using one of the options described above.

**Upgrade installation from UniVerse 10.2**

Besides migrating the master key, you must also convert any data you encrypted in UniVerse 10.2. This involves converting the key store, the metadata store, and the encrypted data files.

If you install UniVerse 11.2 in the same UniVerse 10.2 home directory, the UniVerse installation process tries to convert the key store and the metadata files automatically. You only need to convert your data files with the convencfile utility. However, on some operating systems such as AIX, Solaris, and HP, the automatic procedure will not succeed. A message indicating that the UniVerse license is invalid appears. If you see this message, or if you install UniVerse 11.2 in a different home directory than UniVerse 10.2, complete the following steps to convert your system:

**Note:** If you are sure you did not set up Automatic Data Encryption in your UniVerse 10.2 release, you skip the following instructions. Set up your master key when you have decided to explore Automatic Data Encryption following the instructions described in Setting Up the Master Key.

---

**UniVerse Security Features**

5-12
1. **Transfer the master key**
   
   On operating systems other than Solaris, HP, or AIX, issue the following command:
   
   ```
   uvregen -T <old_uvhome>,<new_uvhome>
   ```
   
   On Solaris, HP, or AIX operating systems, manually set the master key to the same value as your UniVerse 10.2 system, using the following command:
   
   ```
   uvregen -m <master_key>
   ```

2. **Set up the key store**
   
   Copy the &KEYSTORE&, &ENCINFO&, and D_&ENCINFO& files from the old $UVHOME directory to the new $UVHOME directory.
   
   In the new $UVHOME directory, create a new dictionary for the &KEYSTORE& file using the following command:
   
   ```
   loadfile src.u/d_keystore.u “D_&KEYSTORE&”
   ```

3. **Convert the key store**
   
   Convert the UniVerse 10.2 key store and the metadata store to the 11.2 format using the following command:
   
   ```
   convencfile –keystore
   ```

4. **Retag the key store**
   
   Retag the key store using the following command:
   
   ```
   encman -retag -m <master_key>
   ```

5. **Verify the conversion**
   
   Verify that the conversion was successful by logging on to your UniVerse account and executing the following command:
   
   ```
   >LIST.ENCRYPTION.KEY
   ```
   
   All the encryption keys created in the UniVerse 10.2 release should appear.
6. **Convert encrypted files**

   Execute the convencfile command on individual files on the entire account. The following example illustrates running the command on an individual file:
   
   ```
   convencfile -file <filename>
   ```

   The next example illustrates running the command on an entire account:
   
   ```
   convencfile -file <account_directory>
   ```

---

**Setting up the master key**

UniVerse Automatic Data Encryption requires that you set up a master key before you can perform any other Automatic Data Encryption tasks. Execute the uvregen command to define a new master key. Change your working directory to $UVHOME. You must specify the –m option to set up a new master key, as shown in the following example on a Windows platform:

   ```
   C:\UV\bin\uvregen -m abc123
   ```

   There are three types of master keys:

   - `<Master Key string>` – user-defined master key
   - SYSGEN – UniVerse-generated site-specific master key
   - SYSTEM – UniVerse default master key

   If the master key was previously defined, the uvregen utility will prompt for the current master key for verification. At the prompt, enter the master key string, SYSGEN, or SYSTEM.

   If you specify SYSTEM for the master key, UniVerse sets the master key to the system default. If the current master key is not the system default, in order to revert to the system default, you must provide the current master key. If the current master key is password-protected, you must provide the correct passwords to reset the master key.

   Use `@/full_path` to indicate that the master key is stored in a file, as shown in the following example:

   ```
   C:\U2\bin\uvregen -m @/mysecure/mymaster
   ```
If you specify SYSGEN, use the -o option to specify a file name in which to store the generated master key. If you do not specify -o, UniVerse stores the key in the mygenkey file in the directory where you executed the command, which is normally the $UVHOME directory.

You can let uvregen prompt for the master key instead of specifying it on the command line. This may be desirable if you do not want to disclose your key on the command line. To do this, use the following command:

```
C:\U2\UV\bin\uvregn -m PROMPT
```

Follow the prompt to finish setting up the master key.

Beginning with UniVerse 11.1, you can set up the master key with one or two passwords at the same time you set up the master key. For example, the following command sets up a site-specific master key with two passwords:

```
C:\U2\UV\vin\uvregn -m MySiteMaster -P myMekpass1 -P Secondmekpass
```

Use the following command to verify the master key setup:

```
C:\U2\UV\bin\uvregn -z
```

We recommend that the key file be strongly protected or removed from the system after the installation is complete and stored in a safe place.

The maximum length of a master key is 64 characters. The master key should be long and difficult to guess.

**Note:** Make sure you can recover your master key in case of a system crash or natural disaster. If you lose your master key, you may never be able to recover your data.

**Creating the key store**

Once the master key is set up, create the key store and the metadata store.
If you have previously created the key store and metadata store (for example, it was created by the UniVerse 10.2 default setup), and you are sure they are empty or you do not need them anymore, you can remove them from the system by executing the `encman` utility, similar to the following example. To be cautious, you should make copies of the `&KEYSTORE&` and `&ENCINFO&` files before you delete them.

```
C:\U2\UV\bin\encman -delkeystore -f -a
```

You execute the `encman` utility similar to the following example to create the key store and the metadata store:

```
C:\U2\UV\bin\encman -genkeystore
```

**Configuring password policies**

The purpose of password policies is to ensure the quality of the passwords. Compliance requirements by regulations, standards, or company policies often stipulate that passwords conform to a certain set of rules. Beginning at UniVerse 11.1, UniVerse supports user-configurable password policies. You can configure password policies using the `encman` utility or the Extensible Administration Tool. If using the `encman` utility, use the following command to configure password policies:

```
/usr/uv112/bin/encman -passpolicy ALL
```

The utility will guide you through the process to configure password policies for encryption keys, wallets, and master key passwords.

If you do not configure your own password policies, UniVerse enforces a default set of policies. You can turn off password policy enforcement for encryption keys, wallets, or master key passwords collectively or separately. For example, to turn off password policies for all components, enter the following command:

```
/usr/uv112/bin/encman -passpolicy ALL EnforcePolicy 0
```

Password policies do not affect existing passwords, except for the age and history policies, which are always enforced unless you explicitly turn off policy enforcement.

We recommend that password policies be enforced at all times.
For a complete list of password policies supported at UniVerse 11.2, see “Managing password policies” on page 49.
Performing Automatic Data Encryption tasks

This section describes how to perform various tasks for Automatic Data Encryption.

No-echo Automatic Data Encryption command mode

As you will see in the following sections, many Automatic Data Encryption commands require that you enter a password as part of the command syntax. Normally, UniVerse echoes the password on the command line as you type, which exposes the password and is not secure. Worse yet, UniVerse stores the command with the clear-text password in the TCL command stack, which is an unsecured text file, further exposing the risk.

Beginning at UniVerse 11.1, all Automatic Data Encryption commands that require a password have a no-echo mode. UniVerse prompts for the password, as well as other command options, and does not echo the password or store it in the command stack file.

In the following sections, the examples use the echo mode to illustrate command syntax, unless otherwise noted.

TCL commands vs. the Extensible Administration Tool

You can perform most of the Automatic Data Encryption commands described in this section either through a TCL command or the Extensible Administration Tool, unless otherwise noted.

Creating keys and wallets

Before you encrypt data, you must create one or more encryption keys. For example, you can execute the following command to create a key named “mykey” with a password of ”mypassword.”

CREATE.ENCRYPTION.KEY mykey mypassword

If you do not specify the password, UniVerse creates the key without password protection.
It is important to note that, after a key is successfully created, it is also activated for the remaining session. For more details about key activation, see “Activating keys” on page 33.

To create an encryption wallet, execute the following command:

```
CREATE.ENCRYPTION.WALLET mywallet mywltpass
```

Note that an encryption wallet must have a password.

To add a key to the wallet, you can execute the following command. You can only add one key to a wallet at a time:

```
WALLET.ADD.KEY mywallet mywltpass mykey mypassword
```

Only keys with passwords can be added to the wallet.

Execute the following command to remove a key from a wallet:

```
WALLET.REMOVE.KEY mywallet mywltpass mykey my password
```

Execute the following commands to list existing encryption keys and wallets in the key store:

```
LIST.ENCRYPTION.KEY
LIST.ENCRYPTION.WALLET
```

**Deleting encryption keys and wallets**

When you no longer need a key, you can delete it from the key store. Make sure you have already decrypted all the data encrypted with the key. If not, you can re-create the key with its original password, then decrypt the data.

Execute the following command to delete an encryption key:

```
DELETE.ENCRYPTION.KEY mykey mypassword
```

You do not need to specify the password if the key was created without a password.

**Note:** UniVerse will prompt for confirmation to delete the encryption key or wallet. To bypass the confirmation, enter the following command with the FORCE option:

```
DELETE.ENCRYPTION.KEY FORCE mykey mypassword
```
Execute the following command to delete an encryption wallet:

```
DELETE.ENCRYPTION.WALLET mywallet mywltpass
```

**Note:** Deleting an encryption wallet has no effect on encryption keys stored in the wallet.

Generally, only the creator of the encryption key and wallet can delete the key or wallet from the system. However, if a privileged user such as root, Administrator, or uvadm executed the command with the FORCE option, the command will complete successfully.

### Granting and revoking access

When UniVerse creates and encryption key or wallet, it is only accessible by the user that created it. To allow others to use the key or wallet, the person who created the key or wallet must grant access to other users. This user can grant access to a specific user or a group, or grant access to everyone. The grantees cannot grant access to others, since the privilege is reserved exclusively to the user who created the key or wallet. User and group names must be valid on the system at the operating-system level.

Execute the following command to grant access to a single user, in this example, user1:

```
GRANT.ENCRYPTION.KEY mykey mypassword user1
```

To grant access to a single group, in this example group1, prefix an asterisk ("*")) to the group name, as shown in the following example:

```
GRANT.ENCRYPTION.KEY mykey mypassword *group1
```

To grant access to several identities, use a comma to separate the identities, omitting spaces between the identities:

```
GRANT.ENCRYPTION.KEY mykey mypassword
user1,*group1,user2
```

To grant access to everyone, use the PUBLIC keyword, as shown in the following example:

```
GRANT.ENCRYPTION.KEY mykey mypassword PUBLIC
```
To revoke access from a grantee or several grantees, execute the REVOKE.ENCRYPTIONKEY command, as shown in the following example:

REVOKE.ENCRYPTION.KEY mykey mypassword
user1,*group,user2

To grant or revoke access to a wallet, use the same syntax described above.

**Encrypting files**

UniVerse Automatic Data Encryption only encrypts hashed data files. UniVerse does not encrypt directory files, system log files, dictionary files, or system temporary files. In addition, you cannot encrypt the VOC file. For UniVerse Transaction Logging and uvbackup, UniVerse stores encrypted data (also known as cipher text) in the log file. For replication, UniVerse sends cipher text from the publisher to the subscribers.

Before UniVerse 11.1, you could not encrypt indexes. This restriction has been removed at UniVerse 11.1.

**Encryption modes**

In UniVerse Automatic Data Encryption, encryption is defined on a per-file basis. There are two encryption modes you can use to encrypt your data: the whole record mode and the field mode. There are pros and cons for each mode. The decision to use which mode is dependent on the organization and characteristics of your data, as well as the ways your application accesses the data. Before deciding on a particular mode to encrypt your data, you should carefully analyze and thoroughly benchmark your application, and measure the impact on performance and storage space.

The two encryption modes are mutually exclusive. You cannot mix them together to encrypt a single file.

Whichever mode you choose, only actual data fields are encrypted. UniVerse cannot encrypt I-descriptors, since that do not physically exist in a file on disk.
Whole record mode

With the Whole Record mode, all fields, except the @ID field, in a record are encrypted or decrypted as a single unit. Whole record mode has the advantage of lower performance impact, as the number of encryption operations is a major factor for the performance impact, as well as lower space impact. As the cipher text does not need to be encoded after encryption, the space increase is relatively small. The larger your clear-text record, the smaller the ratio of space increase for the cipher text.

The disadvantages of the whole record mode are:

- Before UniVerse 11.1, you could not encrypt indexes. Therefore, any file with an index could not be encrypted with whole record mode. This restriction has been removed at UniVerse 11.1. For more information about index encryption, see “Encrypting indexes.”
- If the encrypted file contains very large records and only a small number of attributes contain sensitive data that actually needs to be encrypted, the performance impact could be more significant compared with only encrypting the sensitive fields.
- If the encrypted file has indexes, all these indexes must also be encrypted as a result of the whole record mode encryption, which could cause further performance degradation.
- To access data in a whole record mode encrypted file, you must decrypt the whole record to access any data within the record. This in turn requires you to disclose the encryption key and password to anyone who needs to access any data from the file, even to those who only need to access the nonsensitive data in the file.

Field mode

With the field mode, you pick the specific fields to encrypt in a file. As a good practice, only fields containing sensitive data, such as credit card numbers, need to be encrypted. The advantages of this mode include flexibility and controllability of keys and passwords. For example, you can encrypt different fields with different keys, which you can create based on your company’s organizational structure. Then, each department can access the data for which it is authorized. Furthermore, only the indexes on the encrypted fields need to be encrypted, possibly reducing encryption overhead.
Before UniVerse 11.1, you could not encrypt indexes. Therefore, any field with an index could not be encrypted with field mode. This restriction has been removed at UniVerse 11.1. For more information about index encryption, see “Encrypting indexes.”.

The disadvantages of the field record mode are:

■ The number of encryptions is a major factor for performance impact. If you encrypt multiple fields in a file, to access a record in the file, you must decrypt all encrypted fields (see Disabling decryption). With field mode, each encrypted field requires a separate encryption or decryption.

■ The impact on storage space is also larger with field mode. To ensure proper handling of the encrypted data, UniVerse must encode each encrypted field, which increases the storage by roughly one third. The additional encoding also consumes more CPU cycles.

**Encryption parameters**

When defining encryption on a file with either mode, you must specify a set of encryption parameters, which includes an encryption algorithm, an encryption key and its password (if specified). For the field mode, you can specify a different set of parameters for each encrypted field.

**Encryption algorithms**

UniVerse supports the following encryption algorithms:

■ AES (AES128, AES192, AES256)
■ DES (DES, DES3)
■ RC2
■ RC4

AES and DES are Federal Information Processing Standards (FIPS) compliant encryption algorithms. Within each group, with the exception of RC4, there are multiple chaining modes (CBC, ECB, OFB, and CFB).
When you encrypt a file, you must specify a specific algorithm to use in encryption. The following table describes valid algorithms for UniVerse decryption:

<table>
<thead>
<tr>
<th>Type of Encryption Desired</th>
<th>Algorithm to Specify</th>
</tr>
</thead>
<tbody>
<tr>
<td>56-bit key DES encryption</td>
<td>des, des-cbc, des-ecb, des-cfb, or des-ofb</td>
</tr>
<tr>
<td>112-bit key ede DES encryption</td>
<td>des_ede, des-ede-cbc, des-ede, des-ede-cfb, or des-ede-ofb</td>
</tr>
<tr>
<td>168-bit key ede DES encryption</td>
<td>des3, des_ede3, des_ede3-cbc, des_ede3-cfb, or des_ede3-ofb</td>
</tr>
<tr>
<td>128-bit key R2 encryption</td>
<td>rc2, rc2-cbc, rc2-ecb, rc2-cfb, or rc2-ofb</td>
</tr>
<tr>
<td>128-bit key RC4 encryption</td>
<td>rc4</td>
</tr>
<tr>
<td>128-bit key AES encryption</td>
<td>aes128, aes-128-cbc, aes-128-cfb, or aes-128-ofb</td>
</tr>
<tr>
<td>192-bit key AES encryption</td>
<td>aes192, aes-192-cbc, aes-192-cfb, aes-192-ofb</td>
</tr>
<tr>
<td>256-bit key AES encryption</td>
<td>aes256, aes-256-cbs, aes-256-ecb, aes-256-cfb, or aes-256-ofb</td>
</tr>
<tr>
<td>Blowfish (variable length key)</td>
<td>bf, bf-cbc, bf-cfb, bf-ecb, bf-ofb, blowfish</td>
</tr>
<tr>
<td>CAST (variable length key)</td>
<td>cast, cast-cbc, cast5-cbc, cast5-cfb, cast5-ecb, cast5-ofb</td>
</tr>
</tbody>
</table>

**UniVerse Encryption Algorithms**

*Note: The algorithm specification is case-insensitive.*

**Encrypting a file**

You encrypt a file using the ENCRYPT.FILE command. You can encrypt multiple fields with one command execution by specifying a set of encryption parameters for each field. You can omit the key password on the command line. The ENCRYPT.FILE command will then prompt for the password. If you use a key to encrypt multiple fields, you only need to specify the password for the first field that uses the key.
To encrypt a file in whole record mode, use the WHOLERECORD keyword, as shown in the following example:

```
ENCRYPT.FILE myfile
WHOLERECORD,aes256,mykey,mypassword
```

Note that there is no space allowed between each part of the encryption parameters.

Use the following command to encrypt two fields, in this example F1 and F2 for file myfile, using two different keys:

```
ENCRYPT.FILE myfile, F1,aes256,mykey1,mypass1
    F2,des3,mykey2,mypass2
```

Note that the encryption specification for each field is separated by a space.

You can resize the encrypted file along with the encryption operation. This is particularly useful considering that encryption may throw off your file sizing, and both the encryption and the resize operations are time consuming. To resize and encrypt at the same time, specify valid resizing parameters before any encryption specifications, as shown in the following example:

```
ENCRYPT.FILE myfile 2 201 2 F1,aes256,mykey1,mypass1
    F2,des3,mykey2,mypass2
```

The above command encrypts fields F1 and F2 in myfile, and at the same time resized the file into type 2, modulo 201, with separation 2.

If the encrypted file has indexes, as a result of the encryption, some or all indexes may become automatically encrypted. See “Encrypting indexes” on page 28 for more information.

## Decrypting a file

The command syntax to decrypt a file is similar to encrypting the file, except that the decryption specification does not require the algorithm parameter. If a file was encrypted with whole record mode, it must be decrypted with the same mode. If a file was encrypted with field mode, each individual encrypted field must be decrypted separately. However, you can decrypt multiple fields in one command execution. You can also resize the file at the same time.
Use the following command to decrypt a file encrypted with the whole record mode:

```
DECRYPT.FILE myfile WHOLERECORD,mykey,mypassword
```

Use the next command to decrypt one field in a file encrypted in field mode:

```
DECRYPT.FILE myfile F1,mykey1,mypassword
```

To decrypt two fields encrypted with the same key and resize it at the same time, use the following command:

```
DECRYPT.FILE 18 101 3 myfile F1,mykey,mypassword F2,mykey
```

File decryption does not affect the status of index encryption. In other words, encrypted indexes remain encrypted after the corresponding fields were decrypted. See “Encrypting indexes” for more information.

### Reencrypting a file

Regulations, standards, or internal policies may require you to periodically reencrypt your encrypted file. This procedure is also known as “rekeying,” where you reencrypt your data with a different key to prevent “key fatigue.”

Prior to UniVerse 11.1, to perform reencryption, you had to first decrypt your data, then encrypt it again. For large files, this process could be very time consuming. Starting with UniVerse 11.1, you can reencrypt your data with just one pass, as shown in the following examples.

Use the following command to reencrypt a single field in a file:

```
REENCRYPT.FILE myfile -D F1,myoldkey,myoldpass -E F1,aes256,mynewkey,mynewpass
```

Specify the decryption parameters after the -D option, and the encryption parameters after the -E option.
The fields or modes you specify after the -D option or the -E option do not have to match. For example, you can decrypt one field, but reencrypt another field. Similarly, you can decrypt two fields, but reencrypt only of them or neither. Or, you can decrypt a whole record mode encrypted file and reencrypt its F1 and F2 fields, as shown in the following example:

```
REENCRYPTFILE myfile -D WHOLERECORD,mykey,mypassword -E F1,aes256,mykey F2,des3,mykey
```

You can also resize the file while reencrypting it by specifying resize parameters before the -D option.

**Encrypting the @ID field**

Prior to UniVerse 11.1, the @ID field could not be encrypted. This restriction is removed in UniVerse 11.1, and the @ID field can be encrypted just as any other field. However, since the @ID is a special field in UniVerse, hashed files and many UniVerse operations depend on it, and there are interactions between the @ID and index encryptions, you must understand the ramifications of encrypting the @ID before you decide you must encrypt it.

After you encrypt the @ID field of a file, you must activate the encryption key in order to successfully execute many TCL commands. If the encryption key is not active, the execution may fail or have an undesired result.

The maximum length for the @ID field is 256 characters. Note that encrypting the @ID field increases the length by at least one third. Make sure that the maximum length is not violated after encryption, or the 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 of a file, UniVerse automatically encrypts all unencrypted index files for the file. This is because the unencrypted index files still contain the @ID values in clear text. Note that at this stage, UniVerse sets the status on indirectly encrypted indexes to “not yet built.” You must use the ENCRYPT.INDEX command to physically encrypt the encrypted index files, or use the BUILD.INDEX command to build and encrypt the index files. ENCRYPT.INDEX uses the same encryption algorithm and key you specified when encrypting the @ID field. You do not need to provide this information again when you encrypt the index files, unless you want to use different algorithms or keys for the indexes.

Another side effect of encrypting the @ID field is that the UniVerse lock table then records the encrypted @ID. When you examining locks, using LIST.READU for example, UniVerse displays cipher text lock IDs, which can be quite confusing.

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

The following example illustrates how to encrypt the @ID field.

Use the following command to encrypt the @ID field along with other fields in the file, using the same key:

```
ENCRYPT.FILE myfile @ID,aes128,mykey,mypassword
F1,aes256,mykey
```

The following example illustrates how to encrypt the @ID field for a file with whole record mode (remember that the whole record mode does not encrypt the @ID field), using different keys:

```
ENCRYPT.FILE myfile @ID,aes256,mykey1,mypass1
WHOLERECORD,aes256,mykey2,mypass2
```

### Encrypting indexes

Prior to UniVerse 11.1, index files could not be encrypted. Any field that had an index could not be encrypted, nor could new indexes be created on an encrypted field. This restriction is removed at UniVerse 11.1.
An index can become encrypted in two ways. You can *directly* encrypt the index on an unencrypted field. Encrypting the index on an encrypted data field may have limited value, but encrypting the index on a virtual field may make sense. You can also create and build an index on an encrypted field. The index *indirectly* becomes encrypted.

There are more scenarios when an index can become indirectly encrypted. When you encrypt the @ID field, all existing unencrypted indexes on the file automatically become encrypted indexes. When you encrypt a file with the whole record mode, all existing unencrypted indexes become encrypted indexes. When you encrypt a field that has an unencrypted index, the index becomes encrypted.

On the other hand, encrypting a data field, including the @ID, or a file in whole record mode does not affect indexes already encrypted.

UniVerse ensures these actions to prevent clear text values of encrypted data fields from being stored in index files under any situation. UniVerse also ensures that encrypted indexes can be used the same way as clear text indexes, which means, among other things, that your queries can still perform effective equal, ordered, or range comparisons based on indexes.

For the indirectly encrypted indexes, you must execute the ENCRYPT.INDEX command to actually 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 because BUILD.INDEX has to physically build the index from scratch, while ENCRYPT.INDEX only needs the encrypt the index file data blocks. Indirectly encrypted indexes inherit encryption parameters from their corresponding encrypted fields (or whole record mode parameters, or @ID encryption parameters, depending on how they were encrypted). You can overwrite the parameters when you execute ENCRYPT.INDEX.

**Note:** 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 directly 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.

Many complexities arise when encrypting the @ID field and an index. Some of the points to consider are:
- Encrypting the @ID field automatically causes all existing indexes to be encrypted. You can have an unencrypted data field with an encrypted index, but you cannot have an unencrypted index if the @ID is encrypted.

- If you decrypt the @ID (or decrypt a whole record mode encrypted file), all encrypted indexes still remain encrypted. You can manually decrypt indexes on nonencrypted fields.

- A data 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, uvrestore, and transaction restart may need to access encryption keys in order to update an index.

The following examples discuss some scenarios of index encryption and its complications.

You encrypt the @ID field of a file which has unencrypted indexes on clear text fields F1 and F2. LIST.INDEX will show the two indexes are encrypted (type DE), and need to be built (Req Enc). You then execute the ENCRYPT.INDEX command to physically encrypt the indexes. Note that you do not have to specify encryption parameters for field F1 and F2, since they inherit the parameters from the encrypted @ID.

```
ENCRYPT.FILE myfile @ID,aes128,mykey,mypassword
LIST.INDEX myfile ALL
ENCRYPT.INDEX myfile F1 F2
```

You can use the following command to encrypt a previously unencrypted index or to change encryption parameters for an index that was implicitly tagged as encrypted, but not yet physically encrypted:

```
ENCRYPT.INDEX myfile F1,aes256,mykey,mypassword
```

If the encrypted index has not been built by the BUILD.INDEX command, the ENCRYPT.INDEX command will only “tag” the index to be encrypted. You must then execute BUILD.INDEX to have it built and physically encrypted. If, on the other hand, the index has already been build. ENCRYPT.INDEX will actually encrypt the index.
When creating an index or building an index on an encrypted field, UniVerse prompts for the encryption key and password if it is not already activated.

To decrypt and index, execute the following command:

    DECRYPT.INDEX myfile F1,mykey,mypassword

Note that only an index on a clear text file can be decrypted, and the file’s @ID must not be encrypted.

You can also reencrypt indexes just as you do with encrypted fields, as shown in the following example:

    REENCRYPT.INDEX myfile -D F1,mykey1,mypass1 -E F1,aes256,mykey2,mypass2

There are other complications related to encrypted indexes and Transaction Logging, uvbackup, uvrestore, and UniVerse Replication. See the related sections later in this chapter.

Finding information about encrypted files

You can find information about encrypted files by executed one of the TCL commands described below, or calling the UniVerse BASIC FILEINFO() function.

The LIST.ENCRYPTION.KEY command displays all keys in the key store with data showing files and fields encrypted by each key.

The LIST.ENCRYPTION.FILE command displays information about an encrypted file, including the algorithm and key used, if decryption is disabled, and similar information about encrypted indexed for each encrypted field.

For example, to display Automatic Data Encryption information on the file myfile, enter the following command:

    LIST.ENCRYPTION.FILE myfile
UniVerse displays information similar to the following example:

- Encryption field F1, algorithm AES256, key mykey.
- Encryption field F2, algorithm AES128, key mykey (decryption disabled).
- Encryption field F3, algorithm AES256, key mykey2.
- Encryption index field F2, algorithm aes-128-cfb, key mykey.
- Encryption index field F3, algorithm aes-128-cfb, key mykey2.

For information about disabled decryption, see “Accessing data in encrypted files.”

You can use the UniVerse BASIC FILEINFO() function to discover Automatic Data Encryption information as well. For UniVerse 11.1, two codes discover encryption information about a specific file:

- FINFO$ENCINFO(22): Returns information about encrypted fields
- FINFO$IDXENCINFO(23): Returns information about encrypted indexes

The following code segment returns a dynamic array named ENC.INFO containing information about a file names myfile:

```
OPEN “myfile” TO MYFILE ELSE STOP “open error”
ENC.INFO = FILEINFO(MYFILE, FINFO$ENCINFO)
```

If the dynamic array is empty, the file is not encrypted. For encrypted files, the dynamic array contains value-mark (@VM) delimited data for each encrypted field. Within the data for each field, subvalue marks (@SM) are used to delimit data. The format of the data is shown in the following example:

```
<loc>:@SM:<key>:@SM:<alg>:@SM:<field>:@VM:<loc>...
```

where:
- loc is the physical attribute number (location) of the encrypted field
- key is the encryption key used
- alg is the algorithm
- field is the name of the field from the dictionary

If the file is encrypted with whole record mode, then <loc> is -1, and field is omitted. Note that for UniVerse 11.1, this array also contains information about the encrypted @ID, always at location 0.
Similarly, the following code example discovers encryption information about indexes:

```plaintext
IDX.ENC.INFO = FILEINFO(MYFILE, FINFO$IDXENCINFO)
```

IDX.ENC.INFO is a dynamic array containing encrypted index information for the file. If it is empty, then no index is encrypted. For encrypted indexes, the dynamic array has value-mark (@VM) delimited data for each encrypted index. Within data for each index, UniVerse uses subvalue marks (@SM) to delimit data. The format of the data is shown in the following example:

```plaintext
<name>:@SM:<key>:@SM:<alg>:@VM:<name>...
```

where:

- `name` is the name of the index field
- `key` is the encryption key used
- `alg` is the algorithm

### Accessing data in encrypted files

This section describes how you access data encrypted by Automatic Data Encryption. The key concepts are key activation and decryption disablement, which have impact on data availability, performance, and application modification requirements.

#### Activating keys

When UniVerse reads encrypted data from a file, it always tries to decrypt it, unless the decryption is explicitly disabled (See “Disabling decryption” on page 35). When UniVerse writes data to a file, it always tries to encrypt data that was defined to be encrypted. Whether the encryption or decryption can be successfully performed depends on if the encryption key is available, or activated. Key activation can be used as an additional layer of data protection against unauthorized access to sensitive data.

Keys without passwords are always automatically activated. For keys with passwords, you must activate them from TCL or from a UniVerse BASIC program. The only exception is if you created a key with a password in a UniVerse session. In this case, the key is automatically activated for that session. You do not need to explicitly activate the key as long as you stay in that session.
To activate and encryption key from TCL, execute the following command:

```
ACTIVATE.ENCRYPTION.KEY mykey mypassword
```

To activate an encryption key from a UniVerse BASIC program, use the ACTIVATEKEY statement, as shown in the following example:

```
ACTIVATEKEY “mykey”,”mypassword” ON ERROR
   PRINT “Key activation failure:”:STATUS()
   Other error processing statements...
END
```

When you activate a wallet, UniVerse activates all keys contained in the wallet. If for any reason UniVerse cannot activate one of the keys in the wallet, then it does not activate any key in the wallet. For example:

```
ACTIVATE.ENCRYPTION.KEY mywallet mywltpass
```

Key activation is local to and effective for the whole UniVerse session. If you activate a key at the TCL level and then start a UniVerse BASIC program, you do not need to activate the key again inside the program. Likewise, if you activate a key inside a UniVerse BASIC program, once the control is back at the TCL level, UniVerse keeps the key activated.

You can deactivate a key or wallet if you no longer need data access. This is useful if your application needs to be very tight on security and you want minimal data access.

Execute the following command the deactivate an encryption key:

```
DEACTIVATE.ENCRYPTION.KEY mykey mypassword
```

Or, in a UniVerse BASIC program, use the DEACTIVATEKEY statement, as shown in the following example:

```
DEACTIVATEKEY “mywallet”,”mywltpass” ON ERROR PRINT 
   “deactivation error “:STATUS()
```
**Disabling decryption**

By default, UniVerse decrypts all encrypted data when it is read. If you encrypt several fields in a file, UniVerse has to perform multiple decryption. Sometimes you do not want to pay the performance penalty of decrypting fields you are not interested in for your current process, or you do not have the authority to all the encryption keys involved. To avoid these potential problems, you can disable decryption on fields you do not need or cannot access. UniVerse will then return the cipher text for the fields for which decryption is disabled.

Use the following command to disable decryption from the TCL command line:

```tcl
DISABLE.DECRYPTION myfile F1
```

Use the next command to disable decryption on several fields in a file:

```tcl
DISABLE.DECRYPTION myfile F1,F2,F3
```

The following command disables decryption on all fields in a file:

```tcl
DISABLE.ENCRYPTION myfile ALL
```

Note that you cannot disable decryption for a file encrypted in whole record mode, or decryption on indexes. For indexes, encryption is always automatic and mandated without exception.

You can also disable decryption from a UniVerse BASIC program by using the DISABLEDEC statement, as shown in the following example:

```basic
FILE = "myfile"
FIELDS = "F1,F2,F3"
DISABLEDEC FILE,FIELDS ON ERROR PRINT "disablement error: ":STATUS()
```

Like key activation, disabling decryption affects the entire UniVerse session.

You can enable decryption on fields for which decryption was disabled using the following command:

```tcl
ENABLE.DECRYPTION myfile F1,F2,F3
```
You can also enable decryption using the ENABLEDEC command in a UniVerse BASIC program, as shown in the following example:

```plaintext
ENABLEDEC "myfile", "ALL" ON ERROR PRINT "enablement error: \":\"STATUS()"
```

We recommend that you enable decryption as soon as the necessary data access is complete, since decryption disablement could cause unforeseen results.

**Querying data**

You query encrypted files the same way you query unencrypted files. If all the keys are activated and no decryption was disabled, you will see the same result as if the files were not encrypted. If, however, some or all of the keys are not activated or decryption on some fields was disabled, UniVerse will display cipher text for those fields for which the encryption keys were not activated or decryption was disabled.

If the file you query has encrypted indexes, the keys used to encrypt the indexes (if different from those used on the file or fields) must also be activated. Otherwise, UniVerse cannot use the index for the query.

Any TCL command accessing encrypted data requires that the encryption keys be activated. For example, the COPY command involves a source file and a target file, both of which can be encrypted files. For a COPY command involving encrypted files to be successful, you must activate all the keys used by both files.

**Accessing data from UniVerse BASIC programs**

In UniVerse BASIC, if any of the required encryption keys are not activated and the decryption on the fields encrypted by the nonactivated key is not disabled, any UniVerse BASIC READ statement will not return any data. This includes all variants of the READ command, including READU, MATREAD, and so forth. This is different from the behavior of query commands from the TCL command line.

Similarly, all keys used to encrypt data in a file must be activated for WRITE statement to be successful to the file.
For READV and WRITEV statements involving an encrypted field, you only need to activate the key used to encrypt the field. The only exception is for a READV statement on a field for which decryption has been disabled. In this case, you do not have to activate the key, and UniVerse will return cipher text.

For READ and WRITE statements, any Automatic Data Encryption related errors are considered fatal. If your READ/WRITE statements do not contain an ON ERROR clause, UniVerse will forcefully terminate the execution of the UniVerse BASIC program. We recommend that you examine your encryption-enabled applications carefully to ensure an ON ERROR clause on all READ and WRITE statements exists and proper error handling is in place.

You can get the Automatic Data Encryption error in two ways, either calling the STATUS() function or examining @SYSTEM.RETURN.CODE. The STATUS() function returns the error code for a specific error. @SYSTEM.RETURN.CODE stores a more detailed system error text. For more information, see the UniVerse BASIC Commands Reference.

**Accessing data from UniVerse clients**

To access encrypted data on UniVerse servers, UniVerse clients must ensure all necessary encryption keys are activated on the servers. Clients can either directly execute the ACTIVATE.ENCRYPTION.KEY command, perform a user-written subroutine on the server, or call UniVerse globally cataloged procedures ACTIVATE_WALLET() and DEACTIVATE_WALLET() to activate and deactivate an encryption key or keys contained in a wallet. Each of these subroutines has the following parameters, which must be supplied in the following order:

- Wallet_ID – The ID of the encryption key wallet
- Wallet_password – The password for the encryption key wallet
- Status – 0 for success, other codes indicate failure
- Error_message – In case of failure, a detailed error message

The following example illustrates calling the ACTIVATE_WALLET() subroutine:

```
call *activate_wallet(wallet_id, wallet_password, wallet_status, wallet_err)
```
The next example illustrates calling the DEACTIVATE_WALLET subroutine:

\[ \text{CALL } *\text{DEACTIVATE_WALLET}(\text{wallet_ID, wallet_password, wallet_status, wallet_err}) \]

For clients that do not use SQL to access the database, you can call these procedures directly from a session object.

If you are using the JDBC Driver for UniVerse, you can access these procedures by executing a CALL statement from a connection object.

If you are using UniObjects for Java, you can access these methods through the subroutine method of a session object.

For clients that use SQL to access the database, such as UniVerse ODBC and UniOLEDB, you can add WALLETID and WALLETPASS to the uci.config file. Adding these parameters enables UniVerse to perform encryption key activation automatically.

If you are using UniObjects with Connection Pooling, the current key activation state of an individual Connection Pool slave process will persist between successive use until the slave process terminates. Any client application accessing the same connection pool could pick up a slave process with active keys, and as a result inherit the current key activation state. If you do not wish the key activation state for a specific connection pool to be persistent, you should deactivate the keys before returning from the method call.

**Working with UV/NET files**

UV/NET files are encrypted on the remote machines, based on the remote machine’s Automatic Data Encryption setup. When accessing data in UV/NET files, encryption keys must be activated on the remote machine. You must also disable decryption on the remote machine. When UniVerse writes to a UV/NET file, it sends data to the remote machine in clear text and encrypts it on the remote machine. When reading data from a UV/NET file, UniVerse decrypts data on the remote machine and sends clear text data to the local machine.
To activate or decrypt an encryption key or wallet on the remote machine, use the same commands with the additional ON clause to specify the remote machine, as shown in the following example:

```
ACTIVATE.ENCRYPTION.KEY myremotekey myremotepass ON myuvnet1
DEACTIVATE.ENCRYTION.KEY myremotekey myremotepass ON myuvnet1
```

Use the following UniVerse BASIC commands to activate or decrypt an encryption key on a remote machine:

```
ACTIVATEKEY "myremotekey", "myremotepass" ON "myuvnet1" ON ERROR...
DEACTIVATEKEY "myremotekey", "myremotepass" ON "myuvnet1" ON ERROR...
```

### Passing key activation and decryption disablement to a PHANTOM process

Prior to UniVerse 11.1, key activation and decryption disablement information was not passed to PHANTOM processes. To access encrypted data, the PHANTOM processes had to activate all necessary keys or disable decryption on related fields.

Beginning with UniVerse 11.1, UniVerse automatically passes key activation and field disablement information to any PHANTOM processes as they are started.

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

**Note:** You cannot use a PHANTOM called from the operating system level which accesses files that are encrypted with keys that are password protected.

### Changing key passwords

Regulations, industry standards, or internal policies often mandate periodic password changes as a measure to ensure information system security.
Prior to UniVerse 11.1, you could not change a password for an encryption key without re-creating the encryption key itself and specifying a new password. If you have used that key to encrypt data, UniVerse had to decrypt the data with the old encryption key and reencrypt it with the new encryption key. Reencryption could be a time consuming process.

Beginning with UniVerse 11.1, 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 change the password using the CHANGE.ENCRYPTION.PASSWORD command, as shown in the following example:

```
CHANGE.ENCRYPTION.PASSWORD mykey myPassw0rd
myNewPa$$word
```

UniVerse subjects the new password to the password policy rules, if they are enforced.

You can also change the password to an encryption wallet, as shown in the following example:

```
CHANGE.ENCRYPTION.PASSWORD mywallet myw1tpass
myneww1tpass
```

By default, when you change the password for a key, the password for that key in all wallets containing that key is also changed. You can choose not to change the password in the wallets by specifying the NOCASCADE option, as shown in the following example:

```
CHANGE.ENCRYPTION.PASSWORD mykey myPassw0rd
myNewPa$$word NOCASCADE
```

As a result of the above command, the password for mykey in wallets containing it remains unchanged. Any future attempt to activate those wallets will fail.

**Note:** Changing the password for the encryption key does not change the key itself. UniVerse always encrypts data with the key material derived when you first created the encryption key. The consequence of this is that if you ever have to re-create the encryption key due to key store corruption and no available backup, you must create the key using the original password as when it was initially created.
Managing Automatic Data Encryption

This section describes administrative tasks for Automatic Data Encryption.

Adding or changing the master key password

You must create a master key before you can perform any other Automatic Data Encryption operations. Some Automatic Data Encryption management operations, such as retagging the key store, exporting and importing a key store, or creating a pass file for warm start, also require that you provide the master key.

Since the master key is vitally important to the security of your Automatic Data Encryption setup, once it is set up it is not convenient to change, you should protect it with maximum precaution. Subjecting the master key to frequent exposure may increase the possibility of leaking it out to unauthorized parties.

Beginning at UniVerse 11.1, 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 and enforced password policy. For more information about password policies, see “Password policies.”

If your master key is protected by a password or passwords, you do not have to provide the master key itself where UniVerse requires a master key. Instead, you must provide all current passwords in order to perform those Automatic Data Encryption management tasks, including changing the password for the master key.

Using passwords to protect the master key can reduce the need to disclose the actual master key. For example, a 24x7 data center had a disk failure at midnight that corrupted the key store. The operator needed to restore the key store from an exported key store file, which required the master key. The off-site security administrator gave the master key password(s) to the operator to perform the restore, and changed the master key password(s) immediately afterwards. Business interruption was kept to a minimum and the security of the master key was not compromised.
Use the uvregen utility to add two passwords to a master key that does not yet have a password, as shown in the following example:

```
C:\U2\UV\bin\uvregen -m MySiteMaster -P myMekpass1 -P Secondm3kpass
```

The number of master key passwords will be changed from 0 to 2.
Are you sure you want to proceed? [Y/N]: Y
New master key password is successfully verified.

If you just want add one password, uvregen will still prompt for the second password. Press Enter to confirm that you want to set just one password, as shown in the following example:

```
D:\U2\UV>uvregen -m SYSTEM -P myMekPass1
Enter Q to quit, or press ENTER if you do not want to enter a password.
Enter second new master key password:
Confirm master key password:
You have not entered a password.
Are you sure you want to continue? [Y/N]: Y
The number of master key passwords will be changed from 0 to 1.
Are you sure you want to proceed? [Y/N]: Y
New master key password is successfully verified.
```
You can change the master key passwords at any time, subject to the enforced password policies. You can also change the number of passwords or cancel all passwords. In the following example, the master key was protected by one password and is changed to a new one:

```
D:\U2\UV>uvregen -m CURRENT
Enter current master key password (or Q to quit): *********
Enter Q to quit, or press ENTER if you do not want to enter a password.
Enter first new master key password: *********
Confirm master key password: *********
Enter second new master key password:
Confirm master key password:
You have not entered a password.
Are you sure you want to continue? [Y/N]: Y
New master key password is successfully verified.
Master key password is going to be changed.
Are you sure you want to proceed? [Y/N]: Y
Master key password change successful.
```

Note that when you execute uvregen to change the master key password you use the keyword CURRENT in place of the actual master key. You can also directly provide the new password on the command line, as shown in the following example:

```
D:\U2\UV>uvregen -m CURRENT -P yNewMekpass2
```

**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, UniVerse checks multiple passwords 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.*

Although recommended, you do not have to protect a master key with a password.
Changing the master key

You must understand the consequence of changing your master key. If you change the master key, your previously encrypted data, including backups and transaction logs, become inaccessible. Your current key store and metadata store also become unusable. Moreover, if you share your encrypted data among multiple servers, you must change the master key on all servers, otherwise, you will no longer be able to share your data.

Before changing your master key, you must decrypt all of your encrypted data, make sure that you can re-create all keys with their initial passwords, and back up your key store in case something goes wrong and you need to restore the current setup. Then, change the master key, restart UniVerse, re-create the key store with all previous encryption keys, and finally reencrypt your data.

This process can be time consuming and is prone to error. We recommend that you not change the master key once your data is encrypted, unless it becomes absolutely necessary.

Use the uvregen utility to change the master key, as shown in the following example:

D:\U2\UV\bin\uvregen -m myNewMek

Warning: changing the Master Key will render your current encrypted data inaccessible. If you already have encrypted data, exit this utility and complete the following steps:
1. Decrypt your data
2. Execute encman to remove key store
3. Shutdown UniVerse RDBMS
4. Execute uvregen to change the Master Key
5. Restart UniVerse RDBMS
6. Execute encman to re-create key store
7. Reencrypt your data

Do you want to continue? [Y/N]: Y

Enter current master key password (or Q to quit): *********

Current master key password are not changed.
UniVerse master key has been set to the new master key specified. UniVerse must be restarted for the new master key to take effect.
UniVerse will ask you to provide the current master key or its passwords, depending on whether the passwords were set up. As with the initial master key setup, you can choose to specify the new key to be the system default or your own key text, let the system generate one for you, or enter the name of a file containing the new master key. You can set up the new master key passwords at the same time.

As a good practice, you should always keep your old and current master keys in a safe place, such as a reputable custodian service, in order to recover from disastrous situations.

**Backing up the key store and metadata store**

It is important to maintain the integrity and availability of your key store. We recommend that you back up your key store and metadata store on a regular basis preferably after every change (for example, as a result of adding a key, changing a password, and so forth). With an up-to-date backup, you can always recover your key store from data loss.

You can also back up your key store in order to migrate encrypted data to another system. For use encrypted data on other systems, the target systems must have an identical Automatic Data Encryption setup, including the master key and key store contents. You can import a backup key store to the target system so you do not have to re-create everything from scratch.

UniVerse always backs up the key store along with the metadata store. For UniVerse 11.1, UniVerse also backs up the password policy file, if it exists.

Use the encman utility to manage backing up and restoring the key store and metadata store. You can also use the Extensible Administration tool to perform these tasks.

The following example illustrates backing up (exporting) the current key store and metadata store to an encrypted file, mykeys:

```
D:\U2\UV\bin\encman -export CURRENT mykeys
Current directory: D:\U2\UV
Enter first master key password (or Q to quit): *********
File D:\U2\UV\mykeys exists, would you like to overwrite it? [y]es, [n]:y
Export keystore/encinfo successful
```
Because the master key in this example has one password, you can use the “CURRENT” keyword in the pace of the actual master key. You can also provide the password on the command line, as shown in the following example:

D:\U2\UV\bin\encman -export CURRENT -P myMekPass1 mykeys

If your master key does not have a password, you must provide the master key itself, or specify the file that contains the master key using the “@” convention.

Additionally, you can add an export-file password to further protect your backup, as shown in the following example:

D:\U2\UV\bin\encman -export CURRENT -P myMekPass1 myexportpass mykeys

To restore or import the key store, the target system must set up the same master key as the one on the source system where you created the backup.

The following example restores (imports) the key store and metadata store:

D:\U2\UV>encman -import CURRENT -P myMekPass1 myexportpass mykeys
Current directory: D:\U2\UV.
Import password policy data...
Password policy file exists, would you like to overwrite it? [y]es, [n]o:y
Import password policy successful.
Import key store...
Key mykey exists, would you like to overwrite it? [y]es, [n]o, [A]ll, [N]one:A
Import key mykey successful.
Import key mykey2 successful.
Import metadata store...
Import encinfo 23701 successful.
Import encinfo 14213 successful.
Import keystore/encinfo successful.
You may need to restore a backup key store to another system, for example, from a Windows system to a UNIX system, where user accounts are named differently from the target system. Encrypted files may also be at different locations. The differences would make the contents of your restored key store out of sync with the current source system. User account names are stored in the key store’s Creator and Grantees fields. If the account names do not match, no key can be used. The file paths are stored in the key store’s Files field. If the paths do not match, UniVerse cannot update correctly.

You can change account names and file paths during a restore (import) operation. For example, you may want to change the account name in the backup from Administrator to root, and change the file path C:\myapp\sales to /myapp/sales. Use the following command to accomplish these tasks:

```
encman -import CURRENT -chown Administrator,root -chpath C:\myapp\sales,/myapp/sales
```

Note that the -chown option changes matching account names in both the Creator and Grantees fields of all encryption keys in the backed up key store, as does the -chpath option.

You can also use the interactive mode of the encman utility to import the key store, as shown in the following example:

```
/usr/uv112/bin/encman -mport
Current directory: /usr/uv112/.
Password for the import file (ENTER to skip, or Q to quit):
Old owner (ENTER to skip):Administrator
New owner:root
Old owner (ENTER to skip):
Old path (ENTER to skip):C:\myapp\sales
New path:/myapp/sales
Old path (ENTER to skip):
Import file (or Q to quit):mykeys
Enter first master key password (or Q to quit):
...
```

As you can see, you can change multiple owners and paths at the same time.
If you have archived encrypted files, it is advisable to export the appropriate keys and record the master key at the same time. These should be stored securely and separate from the archives themselves. This will allow you to retrieve data (through IMPORT) many years in the future, even if the master key and individual encryption keys and passwords on the current system have completely changed. This is particularly appropriate where distributed files are used with date-based distribution algorithms, and individual part files are periodically archived.

Note: Make sure you can recover or re-create your key store and metadata store in case of a system crash or natural disaster. If you cannot recover your key store, you may never be able to recover your data. To ensure this, you may consider saving the master key, encryption keys and their initial passwords (preferably in a secure form), as well as the encrypted files/fields with encryption modes and parameters, to a safe place or a third party custodian. In case you lose your key store and its backup copies, you can still manually re-create your key store and metadata store.

Retagging a key store

To protect your Automatic Data Encryption setup from unauthorized access or misuse, UniVerse has both the key store and the metadata store so that they can only be used on the machine on which they were first created. If someone tries to copy them directly to another machine, they cannot be used in any Automatic Data Encryption operation, and their contents cannot be disclosed.

Your key store may become invalid if you have a hardware failure, failed disk drive, or some other event that causes you to move your data to another machine. You can use the encman utility’s retag option to restore the key store and metadata store back to an operational state.
As with other Automatic Data Encryption operations, the retag operation requires the master key or its passwords. You can supply the master key with the -m option, or passwords with the -P option on the command line. You can also invoke encman in interactive mode and supply the master key or passwords, as shown in the following example:

```bash
D:\U2\UV>encman -retag
Current directory:D:\U2\UV
Enter first master key password (or Q to quit):*********
Processing ENCINFO file...
Processing key 22302...
Processing key 54433...
ENCINFO was properly retagged.
Processing key store file...

Processing key mykey1...
Keystore was properly retagged.
Operation was successful.
```

### Managing password policies

Password policies for encryption keys, wallets, and master keys are set up and enforced separately. You can set up policies using the encman utility or the Extensible Administration tool. The encman utility has an interactive mode and a command line mode. You can set up multiple policies through the interactive mode, or set up one specific policy through the command line mode.

**Note:** Policy changes do not affect UniVerse sessions that are already started.

In the interactive mode, you select a policy by its sequence number and enter an appropriate value. The encman utility validates the value and rejects it if it is invalid. You can then select another policy or save the result.
The following example shows how to set up the MinimumLength policy for encryption keys using the interactive mode:

```
D:\U2\UV>encman -passpolicy KEY
Current directory:D:\U2\UV

Configuring password policy for ADEKey...
Password policy values for ADEKEY:
[1] EnforcePolicy.....: 0
[3] MaximumLength.....: 64
[4] MinimumAge.....: 0
[5] MaximumAge.....: 90
[6] RequiredCharSet.....: 16
[7] ExpirationWarning.....: 14
[8] Complexity....: 0
[9] MinimumHistory.....: 4

Enter policy to change [1-9], or press <ENTER> to continue:2
Enter new value (or [D]efault, or <ENTER> to keep current value:7
Password policy values for ADEKEY:
[1] EnforcePolicy.....: 0
[3] MaximumLength.....: 64
[4] MinimumAge.....: 0
[5] MaximumAge.....: 90
[6] RequiredCharSet.....: 16
[7] ExpirationWarning.....: 14
[8] Complexity....: 0
[9] MinimumHistory.....: 4

Enter policy to change [1-9], or press <ENTER> to continue:
Password policy values for ADEKEY:
[1] EnforcePolicy.....: 0
[3] MaximumLength.....: 64
[4] MinimumAge.....: 0
[5] MaximumAge.....: 90
[6] RequiredCharSet.....: 16
[7] ExpirationWarning.....: 14
[8] Complexity....: 0
[9] MinimumHistory.....: 4
```

Enter policy to change [1-9], or press <ENTER> to continue:
Password policy values for ADEKEY:
[1] EnforcePolicy.....: 0
[3] MaximumLength.....: 64
[4] MinimumAge.....: 0
[5] MaximumAge.....: 90
[6] RequiredCharSet.....: 16
[7] ExpirationWarning.....: 14
[8] Complexity....: 0

Updating policy file...
ADEKey policy configuration completed successfully.

You can use the command line mode to accomplish the same task, as shown in the following example:

```
D:\U2\UV>encman -passpolicy KEY MinimumLength 7
Current directory: D:\U2\UV
Configuring password policy for ADEKey...
Updating policy file...
Updated policies:
Password policy values for ADEKEY:
[1] EnforcePolicy.....: 0
[3] MaximumLength.....: 64
[4] MinimumAge.....: 0
[5] MaximumAge.....: 90
[6] RequiredCharSet.....: 16
[7] ExpirationWarning.....: 14
[8] Complexity.....: 0
[9] MinimumHistory.....: 4
ADEKey policy configuration completed successfully.
```

If you want to set up policies for all categories (key, wallet, and master key) use the ALL keyword, as shown in the following example:

```
C:\U2\UV>encman -passpolicy ALL
```

You can reset the policy back to the system default using the following command:

```
C:\U2\UV>encman -passpolicy ALL default
```

For a detailed description of all password policies, see “Password policies.”
Automatic Data Encryption and UniVerse high availability

There are some points to consider when using Automatic Data Encryption in conjunction with UniVerse high availability options: Transaction Logging and U2 Data Replication.

It is important to remember that high availability components are mostly not aware of Automatic Data Encryption, they do not work together, but they can be made to work alongside each other. The basic rule of thumb is to perform the Automatic Data Encryption set up and configuration on a file outside of the high availability utility. This means encrypting files while they are not activated, published, or subscribed. The process of encrypting a file is not a logged or published operation. UniVerse logs and replicates individual encryptions or decryptions to records on WRITE and READ operations, but not the encryption or decryption of the file. Successfully encrypting or decrypting a file while it is marked for transaction logging or replication can cause massive data inconsistencies and even data loss.

A step-by-step guide for high availability

The basic method for a file that is not encrypted or marked for high availability is to perform the encryption configuration first. After you have encrypted the file to its desired configuration, you can mark the file for high availability, using with Transaction Logging or U2 Data Replication.

If the file is already marked for high availability, you have to deactivate, unpublish or unsubscribe the file before you apply an encryption configuration. It is important to note that all transaction and replication logs that existed prior to the encryption are now useless, and you should discard them immediately. Keeping these logs is dangerous, because if they are applied to a file later, you will have a mix of clear text and cipher text, meaning some data could be lost forever. Because of this, we recommend that you back up your data immediately prior to configuring a file for encryption and high availability.

This is also true if you decrypt a file. UniVerse logs or replicates the raw data, which means the logs contain either cipher text or clear text. If the “live” file contains one and the logs the other, applying the logs is very dangerous.
With the addition of Index Encryption at UniVerse 11.1, there are now more points to consider. If you want to encrypt files that are part of your Transaction Logging solution (it does not affect U2 Data Replication). Due to the nature of index encryption, you must activate encryption keys to perform any rollforward operations, including warmstart (automated recovery at UniVerse startup). The UniVerse Transaction Logging interfaces do not allow for the entry of a key or wallet directly, so you must configure encryption before attempting to rollforward in any way. Use the encman utility for this configuration.

To allow the rollforward process to activate the necessary keys for encrypted index, UniVerse creates a file that the process uses for activation. Use the encman utility to create this file. You can configure the file to contain a specific key, or a set of keys through a wallet or all key on the system.

```
encman -gen_uvrolfpass mkey [-P mkey_pass [-P mkey_pass]] [-k key,kpass | -f file]
```

The following table describes each parameter of the syntax:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mkey</td>
<td>The current master key. If you do not supply this information, UniVerse prompts for the master key.</td>
</tr>
<tr>
<td>mkey_pass</td>
<td>The master key password(s). You must supply one -P argument for each password protecting the master key (0, 1, or 2)</td>
</tr>
<tr>
<td>key</td>
<td>An encryption key or wallet. Each encrypted index of a file activated for transaction logging is required, all necessary keys should be accounted for.</td>
</tr>
<tr>
<td>kpass</td>
<td>The password for the encryption key or wallet you specified.</td>
</tr>
<tr>
<td>file</td>
<td>An input file containing a list of keys and/or wallets and their passwords. Enter this information in the following format: Keyname,password Wallet name, password keyname,password Make sure you do not leave this file on the system.</td>
</tr>
</tbody>
</table>

```
encman -gen_uvrolfpass Parameters
```

If you do not specify the -k or -f options, UniVerse creates the output file using all available keys on the system.
The command described above creates a file in the current directory called uvrolf.pass. This file contains cipher text, and is therefore not readable. You must copy the file to the $UVHOME account directory and rename it with a period prefix, such as .uvrolf.pass.

If you reference all necessary keys in the file in some way, either directly or through a wallet, the rollforward and warmstart processes can access encrypted indexes. Remember, it is only necessary if there are encrypted indexes on your system.

**Methods for encrypting files marked or intended to be marked for UniVerse high availability**

This section describes methods you can use for encrypting files for UniVerse High Availability:

*Encrypting a file that is intended for transaction logging*

Use the following method if the file you are encrypting is not already marked for transaction logging, or U2 Data Replication, but will be marked at a later stage:

1. Encrypt the file with the credentials you desire using the ENCRYPT.FILE command or the Extensible Administration Tool.
2. Activate the file for Transaction Logging

*Encrypting a file that is already marked as active for transaction logging:*

Use the follow method if the file you are encrypting is already activated and part of the Transaction Logging system:

1. Deactivate the file for Transaction Logging.
2. Suspend Transaction Logging.
3. Encrypt the file with the credentials you desire using the ENCRYPT.FILE command or the Extensible Administration tool.
4. Activate the file for Transaction Logging.
5. Back up all of the files marked as active for Transaction Logging, or the entire database, depending on your backup plan.
6. Release, purge, or delete all Transaction Logging log files.
7. Create new Transaction Logging log files.
8. Enable Transaction Logging.

The reason for the additional steps in this method is because the encryption process makes all previously logged transactions against the file obsolete. The previous logs contain the data in clear text, the file, having been encrypted, now contains cipher text. Applying the previous logs to the file after encrypting the file is dangerous and in many circumstances not possible and may lead to data loss.

**Encrypting a file that is intended for U2 Replication**

For U2 Data Replication to successfully replicate files encrypted with Automatic Data Encryption, the following conditions must be met:

- The master key must be identical on each system.
- The encryption keys and their passwords must be identical on each system.

If these conditions are not met, replication will not succeed, or worse, it will appear to succeed but you will not be able to retrieve encrypted data on the subscribing system.

Use the next method if the file you are encrypting is not already marked for U2 Data Replication (or Transaction Logging), but will be at later stage.

1. Encrypt the file with the credentials you desire using the ENCRYPT.FILE command or the Extensible Administration tool.
2. Refresh the subscribing system with a copy of the file from the publishing system.
3. Publish the file on the publishing system.
4. Subscribe the file on the subscribing system.

**Encrypting a file that is already replicated using U2 Data Replication**

For U2 Data Replication to successfully replicate files encrypted with Automatic Data Encryption, the following conditions must be met:

- The master key must be identical on each system.
The encryption keys and their passwords must be identical on each system.

If these conditions are not met, replication will not succeed, or worse, it will appear to succeed but you will not be able to retrieve encrypted data on the subscribing system.

Use the following method if the file you are encrypting is already published an subscribed using U2 Data Replication:

1. Unsubscribe the file on the subscribing system.
2. Unpublish the file on the publishing system.
3. Encrypt the file with the credentials you desire using the ENCRYPT.FILE command or the Extensible Administration tool.
4. Refresh the subscribing system with a copy of the file from the publishing system.
5. Publish the file on the publishing system.
6. Subscribe the file on the subscribing system.

You can use the uvbackup and uvrestore utilities to back up and restore data encrypted with Automatic Data Encryption. UniVerse stores cipher text in the output file created by uvbackup, so the data is protected at all times. The uvrestore utility restores the cipher text to the disk without going through the UniVerse “front end.” This means that UniVerse simply places the cipher text into the file, bypassing the encryption engine. This is true for all types of Automatic Data Encryption, except for encrypted indexes.

Provided there are no encrypted indexes, you can use uvbackup and uvrestore normally using Automatic Data Encryption.

If there are encrypted indexes on your system, the uvrestore utility requires all keys protecting them to be activated. This is done using the -k option on the uvrestore command line. The uvbackup utility does not require the keys to be active, so there is no -k option.

uvrestore [options] -k key,pass [image]
The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>The existing options for the <code>uvrestore</code> command. See the <em>UniVerse Commands Reference</em> for more information.</td>
</tr>
<tr>
<td>image</td>
<td>The input image file to restore.</td>
</tr>
<tr>
<td>key</td>
<td>The name of the encryption key protecting encrypted indexes. You can also supply a wallet name, which we recommend if you require multiple encryption keys.</td>
</tr>
<tr>
<td>pass</td>
<td>The password to the key or wallet.</td>
</tr>
</tbody>
</table>

**Note:** The `-k` option is only required if there are files with encrypted indexes in the backup image.

We recommend that you create an encryption wallet referencing all encryption keys that protect encrypted indexes. Specify this wallet in the `uvrestore` command with the `-k` option.
UniVerse encryption algorithms

UniVerse supports the following encryption algorithms:

- AES (AES128, AES192, AES256)
- DES (DES, DES3)
- RC2
- RC4

AES and DES are Federal Information Processing Standards (FIPS) compliant encryption algorithms. Within each group, with the exception of RC4, there are multiple chaining modes (CBC, ECB, OFB, and CFB).

When you encrypt a file, you must specify a specific algorithm to use in encryption. The following table describes valid algorithms for UniVerse decryption:

<table>
<thead>
<tr>
<th>Type of Encryption Desired</th>
<th>Algorithm to Specify</th>
</tr>
</thead>
<tbody>
<tr>
<td>56-bit key DES encryption</td>
<td>des, des-cbc, des-ecb, des-cfb, or des-ofb</td>
</tr>
<tr>
<td>112-bit key ede DES encryption</td>
<td>des_ede, des-ede-cbc, des-ede, des-ede-cfb, or des-ede-ofb</td>
</tr>
<tr>
<td>168-bit key ede DES encryption</td>
<td>des3, des_ede3, des_ede3-cbc, des_ede3-cfb, or des_ede3-ofb</td>
</tr>
<tr>
<td>128-bit key R2 encryption</td>
<td>rc2, rc2-cbc, rc2-ecb, rc2-cfb, or rc2-ofb</td>
</tr>
<tr>
<td>128-bit key RC4 encryption</td>
<td>rc4</td>
</tr>
<tr>
<td>128-bit key AES encryption</td>
<td>aes128, aes-128-cbc, aes-128-cfb, or aes-128-ofb</td>
</tr>
<tr>
<td>192-bit key AES encryption</td>
<td>aes192, aes-192-cbc, aes-192-cfb, aes-192-ofb</td>
</tr>
<tr>
<td>256-bit key AES encryption</td>
<td>aes256, aes-256-cbs, aes-256-ecb, aes-256-cfb, or aes-256-ofb</td>
</tr>
</tbody>
</table>

UniVerse Encryption Algorithms

*Note: The algorithm specification is case-insensitive.*
Encryption commands

This section lists commands you can use for encrypting and decrypting your data.

**CREATE.ENCRYPTION.KEY**

Use the CREATE.ENCRYPTION.KEY command to create an encryption key in the UniVerse key store. We recommend that you create a password for the key.

**Syntax**

```
CREATE.ENCRYPTION.KEY key.id [password]
```

**Parameters**

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key.id</td>
<td>The encryption key ID.</td>
</tr>
<tr>
<td>password</td>
<td>The password for key.id.</td>
</tr>
</tbody>
</table>

**CREATE.ENCRYPTION.KEY Parameters**

*Note: We suggest that the password you create is a phrase that is hard to guess, but easy to remember, using a combination of ASCII characters and digits. If a password contains a space (“ ”), you must use quotation marks to enclose the password. Passwords are subject to password policies.*

The following example illustrates creating an encryption key using the CREATE.ENCRYPTION.KEY command:

```
> CREATE.ENCRYPTION.KEY sample myuniverse
Create encryption key sample successful.
>
```
CREATE.ENCRYPTION.WALLET

Use the CREATE.ENCRYPTION.WALLET command to create and encryption wallet, which contains encryption keys and passwords.

Syntax

CREATE.ENCRYPTION.WALLET wallet_id [wallet_password]

Parameters

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wallet_id</td>
<td>The encryption wallet ID.</td>
</tr>
<tr>
<td>wallet_password</td>
<td>The password for key.id.</td>
</tr>
</tbody>
</table>

DELETE.ENCRYPTION.WALLET

Use the DELETE.ENCRYPTION.WALLET command to delete an encryption wallet. You must be the owner of the file or logged on as root or a UniVerse Administrator to delete an encryption wallet. If you specify FORCE, UniVerse deletes the encryption wallet even if it is referenced by an encrypted record or field.

Syntax

DELETE.ENCRYPTION.WALLET [FORCE] wallet_id [wallet_password]
Parameters

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORCE</td>
<td>Forces the encryption wallet to be deleted, even if it is referenced by an encrypted record or field.</td>
</tr>
<tr>
<td>key.id</td>
<td>The encryption wallet to delete.</td>
</tr>
<tr>
<td>password</td>
<td>The password for the encryption wallet to delete.</td>
</tr>
</tbody>
</table>

DELETE.ENCRYPTION.KEY Parameters

WALLET.ADD.KEY

Use the WALLET.ADD.KEY command to add a key to an encryption wallet.

Syntax

WALLET.ADD.KEY <wallet_ID> <wallet_password> <key_ID> <key_password>

Parameters

The following parameters describe each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wallet_ID</td>
<td>The ID of the wallet to which you want to add a key.</td>
</tr>
<tr>
<td>wallet_password</td>
<td>The password for the wallet to which you want to add a key.</td>
</tr>
<tr>
<td>key_ID</td>
<td>The key_ID you want to add to the wallet.</td>
</tr>
<tr>
<td>key_password</td>
<td>The password for the key you want to add to the wallet.</td>
</tr>
</tbody>
</table>

Note: Keys that are entered in a wallet must have a password.
**WALLET.REMOVE.KEY**

Use the WALLET.REMOVE.KEY to remove an encryption key from a wallet.

**Syntax**

```
WALLET.REMOVE.KEY [FORCE] <wallet_id> <wallet_password> <key_id> [key_password]
```

**Parameters**

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORCE</td>
<td>If you specify FORCE, a root or uvadmin user can delete an encryption key from an encryption wallet without knowing the key password.</td>
</tr>
<tr>
<td>wallet_ID</td>
<td>The ID of the wallet from which you want to remove an encryption key.</td>
</tr>
<tr>
<td>wallet_password</td>
<td>The password for the encryption key you want to remove from the encryption wallet.</td>
</tr>
<tr>
<td>key_ID</td>
<td>The key_ID you want to remove from the encryption wallet.</td>
</tr>
<tr>
<td>key_password</td>
<td>The password from the key you want to remove from the encryption wallet.</td>
</tr>
</tbody>
</table>

**LIST.ENCRIPTION.WALLET**

Use the LIST.ENCRIPTION.WALLET command to list the existing encryption wallets in the key store. You can also list records in the key store using UniVerse RetriVe commands, such as LIST, LIST.ITEM, SORT, SORT.ITEM, and so forth.
**Note**: The name of the key store file is &KEYSTORE&. Although you can view records from this file using UniVerse Retrieve commands, other UniVerse commands, such as DELETE.FILE and CLEAR.FILE will fail. The ED command will only display encrypted data.

**DELETE.ENCRYPTION.KEY**

Use the DELETE.ENCRYPTION.KEY command to delete a key from the key store. You must be the owner of the key or logged on as root or Administrator to delete an encryption key, and you must provide the correct password. If the key is referenced by any encrypted field or file, deleting the key will fail, unless you specify FORCE.

**Syntax**

```
DELETE.ENCRYPTION.KEY [FORCE] key.id [password]
```

**Parameters**

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORCE</td>
<td>Forces the encryption key to be deleted, even if it is referenced by an encrypted file or field.</td>
</tr>
<tr>
<td>key.id</td>
<td>The encryption key to delete.</td>
</tr>
<tr>
<td>password</td>
<td>The password for the encryption key to delete.</td>
</tr>
</tbody>
</table>

**Example**

The following example illustrates deleting an encryption key using the DELETE.ENCRYPTION.KEY command:

```
>DELETE.ENCRYPTION.KEY sample myuniverse
Do you really want to remove this encryption key? (Y/N) Y
Remove encryption key sample successful.
>
```
LIST.ENCRYPTION.KEY

Use the LIST.ENCRYPTION.KEY command to list the existing keys in the key store. You can also list records in the key store using UniVerse Retrieve commands, such as LIST, LIST.ITEM, SORT, SORT.ITEM, and so forth.

Note: The name of the key store file is &KEYSTORE&. Although you can view records from this file using UniVerse Retrieve commands, other UniVerse commands, such as DELETE.FILE and CLEAR.FILE will fail. The ED command will only display encrypted data. Any attempt to write to the key store will fail, including a UniVerse BASIC WRITE operation or an TCL COPY.

GRANT.ENCRYPTION.KEY

Use the GRANT.ENCRYPTION.KEY command to grant other users access to the encryption key. When a key is created, only the owner of the key has access. The owner of the key can grant access to other users.

Syntax

```
GRANT.ENCRYPTION.KEY key.id [password] {PUBLIC | grantee {grantee...}}
```

Parameters

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>key.id</code></td>
<td>The encryption key.</td>
</tr>
</tbody>
</table>

GRANT.ENCRYPTION.KEY Parameters
Account-based access control and password protection are two ways to protect encryption keys, independent of each other. You must grant access to an encryption key even if it does not have password protection if you want other users to use the key. Conversely, even if you have the correct password for the key, you cannot access it without being granted access.

Example

The following example illustrates granting PUBLIC access to the “sample” encryption key:

```
>GRANT.ENCRYPTION.KEY sample myuniverse PUBLIC
GRANT.ENCRYPTION.KEY to PUBLIC successful.
```

REVOKE.ENCRYPTION.KEY

Use the REVOKE.ENCRYPTION.KEY command to revoke access to the encryption key from other users. When a key is created, only the owner of the key has access. The owner of the key can grant to and then revoke access from other users.

Syntax

```
REVOKE.ENCRYPTION.KEY key.id [password] [PUBLIC | grantee {,grantee...}]
```
Parameters

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>key.id</code></td>
<td>The encryption key for which you are revoking access from another user.</td>
</tr>
<tr>
<td><code>password</code></td>
<td>The password for the encryption key for which you are revoking access from another user.</td>
</tr>
<tr>
<td><code>PUBLIC</code></td>
<td>Revokes PUBLIC access to the encryption key from all users on the system.</td>
</tr>
<tr>
<td><code>grantee</code></td>
<td>Revokes access to the encryption key from the <code>grantee</code> you specify. <code>grantee</code> can be a user name, or a group name. If you specify a group name, prefix the name with an asterisk (&quot;*&quot;). On Windows platforms, you can qualify a group name with a domain name, such as mydomain\users. When you specify a group name, UniVerse revokes access from all users belonging to the group, unless the specific users in the group are granted access separately. Grantees cannot revoke access to the encryption key from other users.</td>
</tr>
</tbody>
</table>

Example

The following example illustrates revoking encryption privileges from PUBLIC for the "sample" encryption key:

```
>REVOKE.ENCRYPTION.KEY sample myuniverse PUBLIC
REVOKE.ENCRYPTION.KEY for PUBLIC successful.
>grant.encryption.key SAMPLE MYUNIVERSE public
```

ENCRYPT.FILE

Use the ENCRYPT.FILE command to create a file in which each record is encrypted.

**Note:** Prior to UniVerse 11.1, you could not use ENCRYPT.FILE with the WHOLERECORD command on files containing indexes. You could not encrypt an index file, or an indexed field. These restrictions have been removed in UniVerse in 11.1.
Syntax

\textbf{ENCRYPT.FILE} \textit{filename} ... \{WHOLERECORD | \textit{filename},\textit{alg},\textit{key},\textit{pass} \} \{\textit{filename},\textit{alg},\textit{key},\textit{pass}\}...

Parameters

ENCRYPT.FILE accepts all parameters of the RESIZE command. If the file you are encrypting is empty, you do not need to specify any of the RESIZE parameters. If the file you are encrypting is not empty, and you know that the file needs resizing because encrypting the file will increase the record size, you should specify the RESIZE parameters. You must specify RESIZE parameters before you specify any of the ENCRYPT.FILE parameters described in the following table.

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{filename}</td>
<td>The name of the file to be encrypted.</td>
</tr>
<tr>
<td>WHOLERECORD</td>
<td>Specifies to encrypt the file in WHOLERECORD mode.</td>
</tr>
<tr>
<td>\textit{filename},\textit{alg},\textit{key},\textit{pass}</td>
<td>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.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{filename}</td>
<td>The name of the field to encrypt.</td>
</tr>
<tr>
<td>\textit{alg}</td>
<td>The algorithm to use for encryption. See “UniVerse encryption algorithms” for a list of valid values.</td>
</tr>
<tr>
<td>\textit{key}</td>
<td>The key ID to use for the field encryption.</td>
</tr>
<tr>
<td>\textit{pass}</td>
<td>The password corresponding to the \textit{key}.</td>
</tr>
</tbody>
</table>
Encrypting a file requires exclusive access to the file, and is very time consuming. During the encryption process, UniVerse creates a temporary file and writes the newly encrypted data to that file. If any errors occur during the encryption process, the command aborts and the original file is left intact.

**Warning:** The ENCRYPT.FILE command can run for a very long time if you are encrypting a file that already contains a large amount of data. All parameters for ENCRYPT.FILE, including the password for each encryption key, can potentially be seen by other users. Therefore, we recommend that you do not specify passwords on the command line but enter them when prompted by ENCRYPT.FILE, or activate them before executing the ENCRYPT.FILE command.

**Example**

The following example illustrates encrypting the CUSTOMER file using the WHOLERECORD option:

```
>ENCRYPT.FILE CUSTOMER.F WHOLERECORD,aes128,sample,myuniverse
ENCRYPT.FILE successful.
```

**REENCRYPT.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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>The name of the file you want to reencrypt.</td>
</tr>
</tbody>
</table>

**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 unencrypted 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.

**ENCRIPT.INDEX**

Beginning at UniVerse 11.1, a new command, ENCRYPT.INDEX, is introduced. This command encrypts the index file associated with a field. It does not rebuild the index.

**Syntax**

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

The following table describes each parameter of the syntax:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>The name of the file for which you want to encrypt the index.</td>
</tr>
<tr>
<td>field</td>
<td>The name of the field you want to encrypt.</td>
</tr>
<tr>
<td>alg</td>
<td>A string containing the cipher name.</td>
</tr>
<tr>
<td>key</td>
<td>The encryption key.</td>
</tr>
<tr>
<td>pass</td>
<td>The password for the encryption key.</td>
</tr>
</tbody>
</table>

**DECRYPT.FILE**

The DECRYPT.FILE command decrypts data in a file or in the fields you specify.

**Syntax**

DECRYPT.FILE filename ... [WHOLERECORD | fieldname],key[,pass] [fieldname,key[,pass]]...

DECRYPT.FILE accepts all RESIZE command parameters. If the file you are decrypting is empty, you do not need to specify any of the RESIZE parameters. If the file you are decrypting is not empty, and you know that the file needs resizing because decrypting the file will decrease the record size, you should specify the RESIZE parameters. You must specify RESIZE parameters before you specify any DECRYPT.FILE parameters described below.
Parameters

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>The name of the file to decrypt.</td>
</tr>
<tr>
<td>WHOLERECORD</td>
<td>Specifies to decrypt the WHOLERECORD mode encrypted file.</td>
</tr>
<tr>
<td>fieldname,key,pass</td>
<td>Specifies the field name to decrypt, and the key, and password to use.</td>
</tr>
<tr>
<td></td>
<td>If you do not specify a password, but created the key using password protection, UniVerse prompts for the password.</td>
</tr>
<tr>
<td></td>
<td>If several fields use the same password, you only have to specify it once, at the first field that uses that key.</td>
</tr>
<tr>
<td>fieldname</td>
<td>The name of the field to decrypt.</td>
</tr>
<tr>
<td>key</td>
<td>The key ID to use for the field decryption.</td>
</tr>
<tr>
<td>pass</td>
<td>The password corresponding to the key.</td>
</tr>
</tbody>
</table>

DECRYPT.FILE Parameters

If the encrypted file was created using the WHOLERECORD keyword, you should specify WHOLERECORD when decrypting the file. If the file was not encrypted using the WHOLERECORD keyword, do not specify WHOLERECORD when decrypting the file.

Warning: The DECRYPT.FILE command can run for a very long time if you are decrypting a file that already contains a large amount of data. All parameters for DECRYPT.FILE, including the password for each encryption key, can potentially be seen by other users. Therefore, we recommend that you do not specify passwords on the command line but enter them when prompted by DECRYPT.FILE, or activate them before executing the DECRYPT.FILE command.

Example

The following example illustrates decrypting a file that was originally encrypted with the WHOLERECORD option.

```
DECRYPT.FILE CUSTOMER.F WHOLERECORD,sample,myuniverse
DECRYPT.FILE successful.
```
REENCRIPT.FILE

Periodic file reencryption using different encryption keys is often needed due
to internal policies or regulation. This process is often referred to as rekeying
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 REENCRIPT.FILE command to rekey a file.

Syntax

REENCRIPT.FILE <filename> <resize options> -D<enclist>
   -E<enclist>

The following table describes each parameter of the syntax.

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

REENCRIPT.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 unencrypted fields in the -D encryption list, or the -E
encryption list contains fields already encrypted by not decrypted by the -D
list, UniVerse generates and error and terminates the process.
**DECRYPT.INDEX**

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:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>The name of the file for which you want to decrypt the index.</td>
</tr>
<tr>
<td>field</td>
<td>The name of the field you want to encrypt.</td>
</tr>
<tr>
<td>key</td>
<td>The encryption key.</td>
</tr>
<tr>
<td>pass</td>
<td>The password for the encryption key.</td>
</tr>
</tbody>
</table>

**REENCRYPT.INDEX**

REENCRYPT.INDEX command decrypts and encrypts 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_spec>
Parameters

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>The name of the file for which you want to decrypt/encrypt the index.</td>
</tr>
<tr>
<td>decrypt_specs</td>
<td>field: The name of the field you want to decrypt.</td>
</tr>
<tr>
<td></td>
<td>key: The encryption key.</td>
</tr>
<tr>
<td></td>
<td>pass: The password for the encryption key.</td>
</tr>
<tr>
<td>encrypt_specs</td>
<td>field: The name of the field you want to encrypt.</td>
</tr>
<tr>
<td></td>
<td>alg: A string containing the cipher name.</td>
</tr>
<tr>
<td></td>
<td>key: The encryption key.</td>
</tr>
<tr>
<td></td>
<td>pass: The password for the encryption key.</td>
</tr>
</tbody>
</table>

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 unencrypted, 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, unless it is also specified with the -D option.

LIST.ENCRIPTION.FILE

Use the LIST.ENCRYPTION.FILE command to display encryption configuration data, such as the fields that are encrypted, the algorithms used, and so forth. This command also displays the fields for which decryption is currently disabled. Beginning with UniVerse 11.1, UniVerse also displays index encryption information.
Syntax

LIST.ENCRYPTION.FILE filename

Example

The following example illustrates the output from the LIST.ENCRYPTION.FILE command:

```
>LIST.ENCRYPTION.FILE CUSTOMER.F
Whole-record encryption, algorithm aes128, key sample.
```

**ACTIVATE.ENCRYPTION.KEY**

Use the ACTIVATE.ENCRYPTION.KEY command to activate a key or a wallet. It is necessary to activate a key if it is protected by a password before you can access any data encrypted by the key.

Syntax

```
ACTIVATE.ENCRYPTION.KEY key.id password [ON hostname]
```

Parameters

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key.id</td>
<td>The key ID or wallet ID to activate. If you provide a Wallet ID, UniVerse activates all keys in the wallet.</td>
</tr>
<tr>
<td>password</td>
<td>The password corresponding to key.id.</td>
</tr>
<tr>
<td>ON hostname</td>
<td>Activates the encryption key on a remote host.</td>
</tr>
</tbody>
</table>

**ACTIVATE.ENCRYPTION.KEY Parameters**

*Note:* You can activate only keys with password protection using this command. Keys that do not have password protection are automatically activated.
DEACTIVATE.ENCRYPTION.KEY

Use the DEACTIVATE.ENCRYPTION.KEY command to deactivate a key or a wallet. This command is useful to deactivate keys to make your system more secure.

**Syntax**

```
DEACTIVATE.ENCRYPTION.KEY key.id password [ON hostname]
```

**Parameters**

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key.id</td>
<td>The key ID or wallet ID to deactivate. If you provide a wallet ID, UniVerse deactivates all keys in the wallet.</td>
</tr>
<tr>
<td>password</td>
<td>The password corresponding to key.id.</td>
</tr>
<tr>
<td>ON hostname</td>
<td>Deactivates the encryption key on a remote host.</td>
</tr>
</tbody>
</table>

**Note**: You can deactivate only keys with password protection with this command. Keys that do not have password protection are automatically activated and cannot be deactivated.

DISABLE.DECRYPTION

Use the DISABLE.DECRYPTION command to turn off decryption on a field or fields you specify.

**Syntax**

```
DISABLE.DECRYPTION filename {<field_list> | ALL}
```
Parameters

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>The name of the file on which you want to disable decryption.</td>
</tr>
<tr>
<td>field_list</td>
<td>A comma-separated list of fields for which you want to disable decryption. Do not enter spaces between the field names.</td>
</tr>
<tr>
<td>ALL</td>
<td>Disables decryption on all encrypted fields in the file.</td>
</tr>
</tbody>
</table>

Example

The following example illustrates disabling decryption on two fields in the CUSTOMER.F file:

```
>DISABLE.DECRYPTION CUSTOMER.F NAME,ZIP
  Disable decryption on field NAME successful.
  Disable decryption on field ZIP successful.
>
```

ENABLE.DECRYPTION

Use the ENABLE.DECRYPTION command to activate decryption on specific fields in a file on which the decryption was previously turned off by the DISABLE.DECRYPTION command.

Syntax

```
ENABLE.DECRYPTION filename {<field_list> | ALL}
```
Parameters

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>The name of the file on which you want to enable decryption.</td>
</tr>
<tr>
<td>field_list</td>
<td>A comma-separated list of fields for which you want to enable decryption. Do not enter spaces between the field names.</td>
</tr>
<tr>
<td>ALL</td>
<td>Enables decryption on all encrypted fields in the file.</td>
</tr>
</tbody>
</table>

Example

The following example illustrates enabling decryption of two fields in the CUSTOMER.F file:

```
> ENABLE.DECRYPTION CUSTOMER.F NAME,ZIP
Enable decryption on field NAME successful.
Enable decryption on field ZIP successful.
>
```

The CHANGE.ENCRYPTION.PASSWORD

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

Syntax

```
CHANGE.ENCRYPTION.PASSWORD ID existing_password new_password
[NOCASCADE]
```
The following table describes each parameter of the syntax:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>The name of the encryption key or wallet ID.</td>
</tr>
<tr>
<td>existing_password</td>
<td>The current password for the encryption key or wallet.</td>
</tr>
<tr>
<td>new_password</td>
<td>The new password for the encryption key or wallet.</td>
</tr>
<tr>
<td>NOCASCADE</td>
<td>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.</td>
</tr>
</tbody>
</table>

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.
UniVerse BASIC encryption commands

This section describes the UniVerse BASIC commands for use with encryption and decryption.

**ACTIVATEKEY**

Use the ACTIVATEKEY command to activate a key. It is necessary to activate a key if you want to supply a password for key protection.

**Syntax**

```
ACTIVATEKEY <key.id>, <password> [ON <hostname>] [ON ERROR <statements>]
```

**Parameters**

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>key.id</code></td>
<td>The key ID or wallet ID to activate. If you provide a Wallet ID, UniVerse activates all keys in the wallet.</td>
</tr>
<tr>
<td><code>password</code></td>
<td>The password corresponding to <code>key.id</code>.</td>
</tr>
<tr>
<td><code>ON hostname</code></td>
<td>The name of the remote host on which you want to activate the encryption key.</td>
</tr>
<tr>
<td><code>ON ERROR statements</code></td>
<td>If you specify ON ERROR statements and an error occurs, UniVerse executes the statements following the ON ERROR clause. Otherwise, UniVerse executes the next statement.</td>
</tr>
</tbody>
</table>

**ACTIVATEKEY Parameters**

*Note: You can activate only keys with password protection with this command. Keys that do not have password protection are automatically activated.*
DEACTIVATEKEY

Use the DEACTIVATEKEY command to deactivate a key or a wallet. This command is useful to deactivate keys to make your system more secure.

Syntax

```
DEACTIVATEKEY <key.id>, <password> [ON <hostname>] [ON ERROR <statements>]
```

Parameters

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key.id</td>
<td>The key ID or wallet ID to deactivate. If you provide a wallet ID, UniVerse deactivates all keys in the wallet.</td>
</tr>
<tr>
<td>password</td>
<td>The password corresponding to key.id.</td>
</tr>
<tr>
<td>ON hostname</td>
<td>The name of the remote host on which you want to deactivate the encryption key.</td>
</tr>
<tr>
<td>ON ERROR statements</td>
<td>If you specify ON ERROR statements and an error occurs, UniVerse executes the statements following the ON ERROR clause. Otherwise, UniVerse executes the next statement.</td>
</tr>
</tbody>
</table>

DEACTIVATEKEY Parameters

**Note:** You can deactivate only keys with password protection with this command. Keys that do not have password protection are automatically activated and cannot be deactivated.
Status codes

The ACTIVATEKEY and DEACTIVATEKEY statements return the following STATUS codes regarding key and wallet operations:

<table>
<thead>
<tr>
<th>STATUS Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Operation successful.</td>
</tr>
<tr>
<td>1</td>
<td>Key is already activated or deactivated. This applies to a single key, not a wallet operation.</td>
</tr>
<tr>
<td>2</td>
<td>Operation failed. This applies to a single key, not a wallet operation.</td>
</tr>
<tr>
<td>3</td>
<td>Invalid key or wallet ID or password.</td>
</tr>
<tr>
<td>4</td>
<td>No access to the key or wallet.</td>
</tr>
<tr>
<td>5</td>
<td>Invalid key ID or password in a wallet.</td>
</tr>
<tr>
<td>6</td>
<td>No access to one of the keys in the wallet.</td>
</tr>
<tr>
<td>9</td>
<td>Other error.</td>
</tr>
</tbody>
</table>

ACTIVATEKEY and DEACTIVATEKEY Status Codes

Example

The following example illustrates deactivating an encryption key:

```
DEACTIVATEKEY "sample", "myuniverse" ON ERROR PRINT "Unable to deactivate key"
```

DISABLEDEC

Use the DISABLEDEC command to turn off decryption on a file or fields you specify.

Syntax

```
DISABLEDEC <filename> [, <multilevel-filename>] [ALL \n | <field_list>] [ON ERROR <statements>]
```
Parameters

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>The name of the file on which you want to disable decryption.</td>
</tr>
<tr>
<td>ALL</td>
<td>If you specify ALL, UniVerse will disable decryption on all encrypted fields of this file.</td>
</tr>
<tr>
<td>field_list</td>
<td>A comma-separated list of fields for which you want to disable decryption. Do not enter spaces between the field names.</td>
</tr>
<tr>
<td>ON ERROR</td>
<td>If you specify ON ERROR statements and an error occurs, UniVerse executes the statements following the ON ERROR clause. Otherwise, UniVerse executes the next statement.</td>
</tr>
</tbody>
</table>

DISABLEDEC Parameters

STATUS codes

DISABLEDEC has the following STATUS codes:

<table>
<thead>
<tr>
<th>STATUS Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error, operation successful.</td>
</tr>
<tr>
<td>1</td>
<td>Decryption is already disabled.</td>
</tr>
<tr>
<td>2</td>
<td>General operation failure, such as an open file error.</td>
</tr>
<tr>
<td>3</td>
<td>File is not an encrypted file.</td>
</tr>
<tr>
<td>4</td>
<td>Attempting operation on a WHOLERECORD encrypted file.</td>
</tr>
<tr>
<td>5</td>
<td>Field(s) is not an encrypted field.</td>
</tr>
<tr>
<td>6</td>
<td>Cannot locate information to disable decryption.</td>
</tr>
<tr>
<td>7</td>
<td>Field is not a valid field in this file.</td>
</tr>
</tbody>
</table>

DISABLEDEC STATUS Codes
**Example**

The following example illustrates disabling decryption on two fields in a file using a quoted string:

```
DISABLEDEC "CUSTOMER.F","NAME,PHONE" ON ERROR PRINT "Unable to disable decryption"
```

The next example illustrates disabling decryption on two fields using variables:

```
CUST="CUSTOMER.F"
FIELDS="NAME,PHONE"
DISABLEDEC CUST,FIELDS ON ERROR PRINT "Unable to disable decryption"
```

**ENABLEDEC**

Use the ENABLEDEC command to activate decryption on a file or fields you specify.

**Syntax**

```
ENABLEDEC <filename> [, <multilevel-filename>], {ALL | <field_list>} [ON ERROR <statements>]  
```

**Parameters**

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>The name of the file on which you want to enable decryption.</td>
</tr>
<tr>
<td>ALL</td>
<td>If you specify ALL, UniVerse enables decryption on all encrypted fields of this file.</td>
</tr>
<tr>
<td>field_list</td>
<td>A comma-separated list of fields for which you want to enable decryption. Do not enter spaces between the field names.</td>
</tr>
<tr>
<td>ON ERROR</td>
<td>If you specify ON ERROR statements and an error occurs, UniVerse executes the statements following the ON ERROR clause. Otherwise, UniVerse executes the next statement.</td>
</tr>
</tbody>
</table>
**STATUS codes**

ENABLEDEC has the following STATUS codes:

<table>
<thead>
<tr>
<th>STATUS Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error, operation successful.</td>
</tr>
<tr>
<td>1</td>
<td>Decryption is already enabled.</td>
</tr>
<tr>
<td>2</td>
<td>General operation failure, such as an open file error.</td>
</tr>
<tr>
<td>3</td>
<td>File is not an encrypted file.</td>
</tr>
<tr>
<td>4</td>
<td>Attempting operation on WHOLERECORD encrypted file.</td>
</tr>
<tr>
<td>5</td>
<td>Field(s) is not an encrypted field.</td>
</tr>
<tr>
<td>6</td>
<td>Cannot locate information to enable decryption.</td>
</tr>
<tr>
<td>7</td>
<td>Field is not a valid field in this file.</td>
</tr>
</tbody>
</table>

**ENABLEDEC Status Codes**

**Example**

The following example illustrates enabling decryption on two fields in a file using a quoted string:

```basic
ENABLEDEC "CUSTOMER.F", "NAME, PHONE" ON ERROR PRINT "Unable to enable decryption"
```

The next example illustrates enabling decryption on two fields using variables:

```basic
CUST="CUSTOMER.F"
FIELDS=NAME, PHONE
ENABLEDEC CUST, FIELDS ON ERROR PRINT "Unable to enable decryption"
```

**STATUS function changes**

The following changes have been made to the UniVerse BASIC STATUS function:

- For UniVerse BASIC READ statements, STATUS() returns 5 to indicate that an encryption error occurred during the READ operation.
For UniVerse BASIC WRITE statements, STATUS() returns -9 to indicate that an encryption error occurred during the WRITE operation.

When an encryption error occurs, a READ/WRITE statement will execute statements following the ON ERROR clause, if an ON ERROR clause is specified. If there is no ON ERROR clause, the UniVerse BASIC program will abort.

FILEINFO() function

Two new values have added to the FILEINFO() function to obtain encryption information:

- FINFO$ENCINFO(22) – If the file is encrypted, UniVerse returns a dynamic array containing the following multivalued data:
  - For a file encrypted with the WHOLERECORD option:
    -1@SVM@<key-id>@SVM<algorithm>
  - For a file encrypted at the field level, returns the following for each field:
    location
    number@SVM@<keyid>@SVM<algorithm>@SVM@<field_name>
  - If the file is not encrypted, UniVerse returns an empty string.

- FINFO$IDXENCINFO(23) – If the index is encrypted, UniVerse returns the following information about the index, separated by subvalue (@SVM) marks:
  index_name@SVM>key_id@SVM>algorithm

If the master key is not protected by passwords, you must specify the master key string. Do not specify the –P option.
Automatic Data Encryption utilities reference

This section describes utilities available with automatic data encryption.

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 UniVerse 11.1, 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.” 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.

Defining a master key

UniVerse Data Encryption requires that you set a master key before it runs. Use the uvregen command to define a new master key. You must specify -m to set up a new master key, as shown in the following example:

```
C\U2\UV>\U2\uv\bin\uvregen -m abc123
UniVerse master key has been set to the new master key specified.
```

There are three types of master keys:

- <Master Key String> – User-defined master key
- SYSGEN – UniVerse-generated site-specific master key
SYSTEM – UniVerse default master key

If the master key was previously defined, the uvregen utility will prompt for a new master key, or prompt for the current master key for verification. At the prompt, enter either the Master Key string, SYSGEN, or SYSTEM.

If you specify SYSTEM for the master key, UniVerse changes the master key to the system default. In order to revert to the system default, you must provide the current master key.

Use @/full_path to indicate that the master key is stored in a file, as shown in the following example:

@/mysecure/mymaster

If you specify SYSGEN, use the -o option to specify a file name to store the generated master key. If you do not specify -o, UniVerse stores the key in the mygenkey file in the directory where you executed the command.

We recommend that the key file is strongly protected, or removed from the system after the installation is complete and stored in a safe place.

The maximum length of a master key is 64 characters. The master key should be long and difficult to guess.

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 describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-m &lt;master_key&gt;</td>
<td>The master key you want to define.</td>
</tr>
<tr>
<td>CURRENT</td>
<td>Specify CURRENT if you are changing the passwords for the current master key.</td>
</tr>
<tr>
<td>-P &lt;password_A&gt;</td>
<td>You can specify up to two passwords for the master key. If you do not enter -P on the command line, or only enter -P once, UniVerse prompts for the remaining new passwords.</td>
</tr>
<tr>
<td>-P &lt;password_B&gt;</td>
<td>If you are resetting the password(s) for a master key, uvregen 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.</td>
</tr>
<tr>
<td></td>
<td>If the password you specify begins with &quot;@&quot; it specifies that a file contains the master key.</td>
</tr>
</tbody>
</table>

For more information on using the -m parameters, refer to the uvregen command.
The encman utility

The encman utility enables you to manage data encryption. You can create a key store through this utility.

You must be logged in as root or Administrator to run this command.

Generating a key store

To generate a key store, use the -genkeystore option.

Syntax

encman [ -genkeystore ] [-n ]

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-n</td>
<td>Specifies to not create the &amp;ENCINFO&amp; file.</td>
</tr>
</tbody>
</table>

encman -genkeystore Parameters

Removing a key store

To remove a key store, use the -delkeystore option.

Syntax

encman [ -delkeystore ] [-f ] [-a]
The following table describes the parameter of the syntax:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| -f        | Deletes the key store without prompting for confirmation.  
  *Note:* Using this operation is dangerous. If you have encrypted files, data cannot be retrieved unless you recreate the key store and keys used by these files. |
| -a        | Remove the key store and the &ENCINFO& file. |

**Importing and exporting key store and store metadata**

To assist with disaster recovery or system migration, use the export and import options. These options also back up and restore the &ENCINFO& file.

*Note:* The import operation will fail if the target system has a different master key than the original system. This is because you cannot access data files encrypted with keys contained in the old key store, even if you restore the key store and metadata store, unless the master key on the target system is the same as that on the source system. You must set up the same system master key on the target system prior to importing the 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.

To export encryption metadata to a file, use the export option. The import operation re-creates the key store on the running system. The running system can be the same system where you performed the export operation, or a different system.
If the `<master key>` begins with “@”, UniVerse treats the rest of the string as the name of the file that contains the master key. If the current system master key is the system default, specify ‘SYSTEM’ on the command line. If `<master key>` or `<password>` contain spaces or other nonalphanumeric characters, you must place `<master key>` or `<password>` in quotation marks.

To import metadata to a file, use the import option:

```
encman -import <master key> [<password>] [chown
<owner>,<newowner> [<owner>,<newowner>] ] -chpath <path>,<new path> [<path>,<new path>]] <filename>
```

You must provide the `<master key>` from the old system to execute the import option, which was provided when executing the export operation. If you specified `<password>` when exporting the data, you must provide that password when executing the import option.

The `-chown` option allows you to change all references of `<owner>` to `<new owner>`, including the key creator and grantees in a key record.

The `-chpath` option allows you to change the file path in the key reference attribute of a key record to a new path. You must specify the full path. UniVerse changes all instances of `<path>` to `<new path>`. For example, if FILEA has the following path:

```
/home/disk1/acct1/FILEA
```

and you execute the following `-chpath` option:

```
... -chpath /home/disk1/acct1,/usr/disk1,acct1
```

FILEA will have the following path:

```
/usr/disk1/acct1/FILEA
```

### Resynching tags

The `&ENCINFO&` and `&KEYSTORE&` files each contain system-specific tags. These tags stop the files from being accessed if they are stolen and placed on another system.
In the case of a system migration or disaster recovery, use the -retag option of the encman utility to retag the key store and the metadata store. You can use the encman utility with the -P option to unlock the master key to use in a retag operation, as shown in the following example:

```
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.

The master key you specify must match the master key currently effective on the system. If you do not specify the master key, UniVerse prompts for it.

Inputting the master key does not echo to the screen. If the master key you specify begins with the "@" symbol, UniVerse uses the text following "@" as a key file for the encman utility to search for the actual key.

You should use "SYSTEM" for the system default master key.

**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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>Manager the password policy to encryption keys, wallets, and master key.</td>
</tr>
<tr>
<td>KEY</td>
<td>Manage the password policy to only encryption keys.</td>
</tr>
<tr>
<td>WALLET</td>
<td>Manage the password policy to only wallet encryption keys.</td>
</tr>
<tr>
<td>MASTERKEY</td>
<td>Manage the password policy to only the master key.</td>
</tr>
<tr>
<td>policy_name</td>
<td>The password policy you want to change. Valid values are:</td>
</tr>
<tr>
<td></td>
<td>EnforcePolicy</td>
</tr>
<tr>
<td></td>
<td>MinimumLength</td>
</tr>
<tr>
<td></td>
<td>MaximumLength</td>
</tr>
<tr>
<td></td>
<td>MinimumAge</td>
</tr>
<tr>
<td></td>
<td>MaximumAge</td>
</tr>
<tr>
<td></td>
<td>RequiredCharSet</td>
</tr>
<tr>
<td></td>
<td>ExpirationWarning</td>
</tr>
<tr>
<td></td>
<td>Complexity</td>
</tr>
<tr>
<td></td>
<td>MinimumHistory</td>
</tr>
</tbody>
</table>

You can specify “default” as the value for policy_name. If you specify “default,” UniVerse sets all policy names to the system defaults.

| policy_value | The value for the policy you are defining. |
|             | You can specify “default” as the value for policy_value. 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:\U2\UU>bin\encman -passpolicy
Current directory: C:\U2\UU.
Configure policy for [null, IKley, IWallet, or IMasterKey]?
```

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

```
C:\U2\UU>bin\encman -passpolicy
Current directory: C:\U2\UU.
Configure policy for [null, IKley, IWallet, or IMasterKey]? A

Configuring password policy for ADEKey...

Password policy values for ADEKey:
[1] EnforcePolicy........: 1
[4] MinimumAge............: 1
[5] MaximumAge............: 90
[6] RequiredCharSet........: 7
[7] ExpirationWarning.....: 14
[8] Complexity............: 3
[9] MinimumHistory........: 8
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.
Generating the uvrolf password file

Beginning at UniVerse 11.1, the gen_uvrolfpass option has been added to the encman utility to generate the uvrolf password file. This option creates a uvrolf.pass file in the current directory. When you copy this file to $UVHOME/.uvrolf.pass, all the encryption keys held in this file are automatically activated during the roll forward process, including warm start, media recovery, and rolling a file forward.

Syntax

```
```

Parameters

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>masterkey</td>
<td>The master key.</td>
</tr>
<tr>
<td>-P mek_pass</td>
<td>The password for the master key.</td>
</tr>
<tr>
<td>-k key,pass</td>
<td>Use the -k option to add the encryption keys and corresponding passwords to the uvrolf.pass file.</td>
</tr>
<tr>
<td>-f filename</td>
<td>Use the -f option to add the keys and corresponding passwords from filename to the uvrolf.pass file.</td>
</tr>
</tbody>
</table>

encman -gen_uvrolfpass Parameters

If you do not specify either the -k or the -f option, UniVerse writes the master key and corresponding passwords to the uvrolf.pass file, and activates all keys involved during the roll forward process automatically.
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 “.uvpspolicy.” The file has the following characteristics:

- You manage this password policy file from U2 Extensible Admin 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:

<table>
<thead>
<tr>
<th>Policy</th>
<th>Values</th>
</tr>
</thead>
</table>
| 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 date after which the encryption key password expires. The default value is 90. Value values are \(2^{31} - 1\). 0 indicates never expire. |

Supported Password Policies
<table>
<thead>
<tr>
<th>Policy</th>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
</table>
| RequiredCharSet | The character set for the password. The default values are:          | - 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)  
SPECIAL characters are all printable nonalphanumeric characters except a single quotation mark (‘), double quotation marks (“”) and a backslash (\).
Nonprintable characters are not permitted.  
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.  
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:  
- Pass7834  
- mypass12/23  
- PMOD89$12  
The actual values are:  
- REQUIRE_1  16  
- REQUIRE_2  32  
- REQUIRE_3  48  
- REQUIRE_4  64  |
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. UniVerse then uses the system defaults to enforce password compliance.

<table>
<thead>
<tr>
<th>Policy</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExpirationWarning</td>
<td>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.</td>
</tr>
<tr>
<td>Complexity</td>
<td>0 – Do not perform various checks to ensure the strength of the password.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>2 – SUCCESSION – Checks if there are more than two of the same character in succession.</td>
</tr>
<tr>
<td></td>
<td>You can use a bitwise OR to combine these values together to specify a combination of characters.</td>
</tr>
<tr>
<td>MinimumHistory</td>
<td>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.</td>
</tr>
</tbody>
</table>
Chapter 6: Using the U2 Extensible Administration Tool for Automatic Data Encryption

- Encryption key management ........................................... 6-5
  - Adding an encryption key ........................................... 6-5
  - Viewing encryption key details ................................... 6-6
  - Deleting an encryption key ........................................ 6-7
  - Granting privileges ................................................ 6-8
  - Revoking privileges ............................................... 6-10
  - Changing an encryption key password .......................... 6-12
- Encryption wallet management ...................................... 6-14
  - Creating an encryption wallet .................................... 6-15
  - Viewing encryption wallet details ............................... 6-16
  - Granting privileges to the encryption wallet .................. 6-17
  - Revoking privileges from the encryption wallet .............. 6-19
  - Changing wallet passwords ...................................... 6-21
  - Deleting an encryption wallet ................................... 6-22
  - Adding a key to an encryption wallet ........................... 6-23
  - Deleting a key from an encryption wallet ....................... 6-24
- Managing encryption .................................................. 6-25
  - Encrypting a file .................................................. 6-25
  - Decrypting a file .................................................. 6-30
  - Reencrypting a file ............................................... 6-32
  - Encrypting an index ............................................... 6-37
  - Decrypting an index ............................................... 6-39
  - Reencrypting an index ............................................ 6-41
  - Managing the key store .......................................... 6-45
  - Exporting a key store ............................................ 6-48
  - Importing a key store ............................................ 6-50
  - Managing master keys ............................................ 6-52
Changing the master key . . . . . . . . . . . . . . 6-54
Managing password policies . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6-57
You can use U2 Extensible Admin to manage data encryption on your system.

From the U2 Extensible Admin main window in the Admin portion of the tool, select **Data Encryption**. The **Data Encryption** window appears, as shown in the following example:
Encryption key management

This section describes how to manage encryption keys through U2 Extensible Admin.

Adding an encryption key

To create an encryption key, click Add. The New Encryption Key dialog box appears, as shown in the following example:

Enter the name of the encryption key in the Key Name box. Although not required, you can enter a password for the new key in the Password box. Reenter the password in the Confirm Password box.

After you create the encryption key, it appears in the Keys area of the Data Encryption dialog box.
Viewing encryption key details

To view details about an encryption key, click the key for which you want to view details. A dialog box similar to the following example appears:
Deleting an encryption key

To delete an encryption key, from the Data Encryption dialog box, click the encryption key you want to delete, then click Delete. The following dialog box appears:

![Keys dialog box](image)

If the encryption key contains a password, enter it in the Password box. If you want to delete the encryption key, click OK. If not, click Cancel.
Granting privileges

To grant privileges to the encryption key to a user or group, click **Grant**. The **Grant Encryption Key Access** dialog box appears, as shown in the following example:

To grant Public privileges, click the **Public** check box.

If you want to display global users on the system, click the **Show Global Users** check box. Select the users to which you want to grant access. You can select multiple users by pressing the CTRL key and the users to which you want to grant access, or click **Select All** to select all users.
If you want to display global groups on the system, click the **Show Global Groups** check box. Select the groups to which you want to grant access. You can select multiple groups by pressing the CTRL key and the groups to which you want to grant access, or click **Select All** to select all groups.

Alternatively, you can enter the users to which you want to grant access in the **Enter Users** box, separated by commas. You can also enter the groups to which you want to grant access in the **Enter Groups** box, separated by commas.

When you have selected the users and groups to which you want to grant privileges, click **Finish**.

When you have finished selecting the users or groups to which you want to grant privileges, click **Finish** to save your changes, or click **Cancel** to exit without saving changes.
Revoking privileges

To revoke privileges from an encryption key from a user or group, click **Revoke**. The **Revoke Encryption Key Access** dialog box appears, as shown in the following example:

To revoke privileges from Public users, click the **Public** check box.
In the **Users** portion of the dialog box, click the users from which to want to revoke privileges. You can select multiple users by pressing the CTRL key while selecting users. If you want to revoke privileges from all users listed, click **Select All**.

In the **Groups** portion of the dialog box, click the groups from which you want to revoke privileges. You can select multiple groups by pressing the CTRL key while selecting groups. If you want to revoke privileges from all groups listed, click **Select All**.

Alternatively, you can enter the users from which you want to revoke access in the **Enter Users** box, separated by commas. You can also enter the groups from which you want to revoke access in the **Enter Groups** box, separated by commas.

When you have selected the users and groups from which you want to revoke privileges, click **Finish**.
Changing an encryption key password

To change the password for an encryption key, click Change Password. A dialog box similar to the following example appears:

Enter the name of the key for which you want to change the password in the Key Name box.

Enter the old password for the key in the Password box.

Enter the new password for the key in the New Password box. Reenter the password in the Confirm New Password box.
By default, if a key is included in an encryption wallet and you change the password for the key, the password for that key in the wallet is also changed. If you do not want to change the password of the key in the wallet, select the **No Cascade** check box.
Encryption wallet management

To manage Encryption Wallets, from the Data Encryption dialog box, click the Wallets tab. A dialog box similar to the following example appears:
Creating an encryption wallet

To create an encryption wallet, click **Add**. The **New Encryption Wallet** dialog box appears, as shown in the following example:

![New Encryption Wallet dialog box](image)

Enter the name for the new encryption wallet in the **Wallet Name** box.

Enter the password for the encryption wallet in the **Password** box. Reenter the password in the **Confirm Password** box. Click **Finish**.
Viewing encryption wallet details

To view details about an encryption wallet, click the wallet for which you want to view details. A dialog box similar to the following example appears:
Granting privileges to the encryption wallet

To grant privileges so other users can access the encryption wallet, select the wallet for which you want to grant access, then click **Grant**. The **Grant Encryption Wallet Access** dialog box appears, as shown in the following example:

Enter the password for the encryption wallet in the **Password** box.

If you want to grant public access to the encryption wallet, select the **Public** check box, then click **Finish**.
If you want to display global users on the system, click the **Show Global Users** check box. Select the users to which you want to grant access. You can select multiple users by pressing the CTRL key and the users to which you want to grant access, or click **Select All** to select all users.

If you want to display global groups on the system, click the **Show Global Groups** check box. Select the groups to which you want to grant access. You can select multiple groups by pressing the CTRL key and the groups to which you want to grant access, or click **Select All** to select all groups.

Alternatively, you can enter the users to which you want to grant access in the **Enter Users** box, separated by commas. You can also enter the groups to which you want to grant access in the **Enter Groups** box, separated by commas.

When you have selected the users and groups to which you want to grant privileges, click **Finish**.
Revoking privileges from the encryption wallet

To revoke privileges from the encryption wallet, select the wallet for which you want to revoke access, then click **Revoke**. The **Revoke Encryption Wallet Access** dialog box appears, as shown in the following example:

Enter the password for the encryption wallet in the **Password** box.

In the **Users** portion of the dialog box, click the users from which you want to revoke privileges. You can select multiple users by pressing the CTRL key while selecting users. If you want to revoke privileges from all users listed, click **Select All**.
In the **Groups** portion of the dialog box, click the groups from which you
want to revoke privileges. You can select multiple groups by pressing the
CTRL key while selecting groups. If you want to revoke privileges from all
groups listed, click **Select All**.

Alternatively, you can enter the users from which you want to revoke access
in the **Enter Users** box, separated by commas. You can also enter the groups
from which you want to revoke access in the **Enter Groups** box, separated by
commas.

When you have selected the users and groups from which you want to
revoke privileges, click **Finish**.
Changing wallet passwords

To change the password for an encryption wallet, click Change Password. The Change Password dialog box appears, as shown in the following example:

Enter the password for the encryption wallet in the Password box.

Enter the new password for the encryption wallet in the New Password box. Reenter the password in the Confirm New Password box, then click Finish.
Deleting an encryption wallet

To delete an encryption wallet, click **Delete**. A dialog box similar to the following example appears:

![Image of a dialog box asking 'Do you really want to delete wallet 'testwallet'? with a text field for 'Wallet Name' and a 'Password' field.]

To delete the encryption wallet, enter the password for the encryption wallet in the **Password** box, then click **OK**.
Adding a key to an encryption wallet

To add a key to an encryption wallet, select the wallet to which you want to add a key, then in the Included Keys portion of the Wallets dialog box, click Add. A dialog box similar to the following example appears:

![Wallets dialog box](image)

Enter the password to the encryption wallet in the Wallet Password box.

Select the Key Name to add to the wallet from the Key Name list. Enter the password to the encryption key in the Key Password box, then click OK.

*Note: Any encryption key added to a wallet must have a password.*
Deleting a key from an encryption wallet

To delete a key from an encryption wallet, from the Included Keys portion of the Wallet dialog box, select the key you want to delete from the wallet, then click Delete. A dialog box similar to the following example appears:

Enter the password to the encryption wallet in the Wallet Password dialog box. Enter the password to the key you are deleting from the wallet in the Key Password dialog box. If you want UniVerse to forcefully delete the key from the encryption wallet, select the Force check box.

Click OK to delete the encryption key from the encryption wallet.
Managing encryption

This section describes how to encrypt a file, decrypt a file, encrypt an index file, decrypt an index file, reencrypt a file, and reencrypt an index file.

Encrypting a file

To encrypt a file or fields in a file, check the File Encryption tab. A dialog box similar to the following example appears:
In the **Accounts** area of the screen, expand the account where you want to encrypt files. Select the file you want encrypt. A dialog box similar to the following example appears:

![Encrypting a file in whole record mode](image)

**Encrypting a file in whole record mode**

The U2 Extensible Admin tool populates the full path for the data file and dictionary file you selected.
Click Encrypt. The **File Encryption** dialog box appears, as shown in the following example:

- **Whole record** – If you want to encrypt each field (except the @ID field) in the record, select **Whole Record Mode**.
Encrypt Info – Define the following information in the Whole Record Encryption Parameters area if you are encrypting an entire record:

- **Algorithm** – Select the algorithm to use for encrypting the record. For a list of valid algorithms, see UniVerse encryption algorithms in Chapter 5, “Chapter 5: Automatic Data Encryption.”
- **Key** – Select the encryption key you want to use when encrypting the data from the Key Name list.
- **Password** – Enter the password corresponding the encryption key, if one exists.

Click **Apply**. UniVerse encrypts every field for every record in the file.

**Note:** Whole Record mode does not encrypt the @ID field.

**Encrypting a file in field mode**

If you want the encrypt specified fields in the file, select **Field Mode**. Define the following information to encrypt specific fields in a record:
Field Encryption Parameters – Define the parameters to use when encrypting the field. For a list of valid parameters, see ENCRYPT.FILE in Chapter 5, “Chapter 5: Automatic Data Encryption.”

Click the name of the field you want to encrypt, then click Set. The File Encryption dialog box appears, as shown in the following example:

Algorithm – Select the algorithm to use when encrypting the field. For a list of valid algorithms, see UniVerse encryption algorithms in Chapter 5, “Chapter 5: Automatic Data Encryption.”

Key Name – Select the key to use when encrypting the file.

Password – Enter the password corresponding to the key.

When you have defined all the fields you want to encrypt, click Finish.

You can select multiple fields by holding the CTRL key and selecting the fields you want to encrypt. If you do so, UniVerse applies the field encryption parameters you defined to each field you select.

Note: If a file has index fields as a result of encryption, some index fields may become encrypted, as shown by a check mark in the “Is Encrypted” field. These indexes must be built to complete the encryption. The “Build” column for these fields will show “Is Required.” You must click Encrypt Index to complete the process. For more information, see “Encrypting an index” for more information.
Decrypting a file

To decrypt a file or fields in a file, check the Decrypt tab. A dialog box similar to the following example appears:

![File Encryption](image)

**Decrypting a record in whole record mode**

If you want to decrypt the entire record, click Whole Record Mode.
Whole Record Encryption Parameters – The parameters to use when decrypting the file. For a list of valid parameters, see DECRYPT.FILE in Chapter 5, “Chapter 5: Automatic Data Encryption.”

Define the following information in if you are decrypting an entire record:

- **Key Name** – Select the encryption key you want to use when decrypting the data from the Key Name list.
- **Password** – Enter the password corresponding to the encryption key, if one exists.

Click **Apply**. UniVerse decrypts every field for every record in the file.

Decrypting a file in field mode

Define the following information to decrypt specific fields in a record:

- **Parameters** – The parameters to use when decrypting the file. For a list of valid parameters, see DECRYPT.FILE in Chapter 5, “Chapter 5: Automatic Data Encryption.”

- **Field Encryption Parameters** – Select the name of the field you want to decrypt, then click **Set**. The Field Encrypt Info dialog box appears, as shown in the following example:

![Field Encryption Dialog](image)

- **Key Name** – Select the key to use when decrypting the file.
- **Password** – Enter the password corresponding to the key.

When you have defined all the fields you want to decrypt, click **Finish**.
Reencrypting a file

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

When reencrypting a file, you first need to specify the decryption actions, then you specify the encryption actions. You can reencrypt a file from Whole Record Mode to Field Mode or from Field Mode to Whole Record Mode.
Select the file for which you want the reencrypt date, then click **Re-encrypt**. The **File Re-encryption** dialog box appears, as shown in the following example:

![File Re-encryption dialog box](image)

**Reencrypting a file in whole record mode**

Define the following information to decrypt a file in Whole Record Mode.
Select **Whole Record Mode**, then click **Next**. A dialog box similar to the following example appears.
Whole Record Encryption Parameters – The parameters to use when decrypting the file. For a list of valid parameters, see DECRYPT.FILE in Chapter 5, “Chapter 5: Automatic Data Encryption.”

Define the following information in the Whole Record Encryption Parameters area if you are decrypting an entire record:

- **Algorithm** – Select the algorithm to use for encrypting the record. For a list of valid algorithms, see UniVerse encryption algorithms in Chapter 5, “Chapter 5: Automatic Data Encryption.”
- **Key Name** – Select the encryption key you want to use when decrypting the data from the Key Name list.
- **Password** – Enter the password corresponding the encryption key, if one exists.

Click Next.

Reencrypting a file in field mode

Define the following information to decrypt specific fields in a record:

- **Field Encryption Info** – Select the name of the field you want to decrypt, then click Set. The File Decryption dialog box appears, as shown in the following example:

  ![File Decryption Dialog](example)

  - **Key Name** – Select the key to use when decrypting the file.
  - **Password** – Enter the password corresponding to the key.

Click Next.
Specifying encryption parameters

The U2 Extensible Admin tool next prompts for the encryption parameters. For information about these parameters, see “Encrypting a file.”

When you have defined the encryption parameters, click Finish. UniVerse decrypts and encrypts the data you specified.
Encrypting an index

To encrypt an index on a data file, from the Data Encryption dialog box, click Encrypt Index. A dialog box similar to the following example appears:
The indexed fields for the file you specified appear in the **File Encryption Parameters** area. Select the index fields you want to encrypt, and click **Set**. The **File Encryption** dialog box appears, as shown in the following example:

![File Encryption Dialog Box](image)

In the **Algorithm** box, select the algorithm you want to use to encrypt the index field.

In the **Key Name** box, enter the name of the key you want to use to encrypt in the index field. Enter the password for the key in the **Password** box, if on exists. Click **OK**.

Click **Finish** to encrypt the index field.
Decrypting an index

To decrypt an index, from the Data Encryption dialog box, click Decrypt Index. A dialog box similar to the following example appears:
Click the index field you want to decrypt. A dialog box similar to the following example appears:

![File Decryption Dialog Box]

Enter the name of the key for the encrypted index field in the **Key Name** dialog box. Enter the corresponding password, if one exists, in the **Password** box, then click **OK**.

Click **Finish**. UniVerse decrypts the index field you specified.
Reencrypting an index

To reencrypt an index, from the Data Encryption dialog box, click Reencrypt Index. A dialog box similar to the following example appears:
Click the index field you want to decrypt. A dialog box similar to the following example appears:

![File Decryption](image)

Enter the name of the key for the encrypted index field in the **Key Name** dialog box. Enter the corresponding password, if one exists, in the **Password** box, then click **OK**.
Click **Next**. A dialog box similar to the following example rs:
The indexed fields for the file you specified appear in the **File Encryption Parameters** area. Select the index fields you want to encrypt, then click **Set**. The **File Encryption** dialog box appears, as shown in the following example:

![File Encryption Dialog Box](image)

In the **Algorithm** box, select the algorithm you want to use to encrypt the index field.

In the **Key Name** box, enter the name of the key you want to use to encrypt in the index field. Enter the password for the key in the **Password** box, if on exists. Click **OK**.

Click **Finish** to reencrypt the index field.
Managing the key store

This section describes how to manage the key store from U2 Extensible Admin. From the **Data Encryption** dialog box, click the **Key Store** tab. A dialog box similar to the following example appears:
Retagging a key store

To retag a key store, select **Retag**. Enter the current master key in the **Current Master Key** box, or click **Browse** to locate the file where the master key is stored.

Click **Retag**. UniVerse retags the key store displays information about the operation, as shown in the following example:
For more information about retagging a key store, see “Retagging a key store” in Chapter 5, “Chapter 5: Automatic Data Encryption”.

Note: Retagging a key store requires the master key. If the master key is password protected, you must provide the correct password or passwords. If the master key is not password protected, you must either enter the master key in the “Current Master Key” box, or enter the file containing the master key, starting with the “@” character followed by the file name, or click Browse to locate the file.
Exporting a key store

You can export the key store to another location. To export the key store, click Export. A dialog box similar to the following example appears:

![Data Encryption - localhost(Windows):UniVerse](image)

Enter the master key in the Current Master Key box, or click to choose the location of the master key.
Enter the full path to the file where you want to export the key store, or click **Browse** to browse for the location of the file.

Enter the password for the exported &KEYSTORE& file, if desired, in the **Password for Key Store File** box. Reenter the password in the **Confirm Password** box, then click **Export**.

UniVerse exports the contents of &KEYSTORE& file, the &ENCINFO& file, and the password policy file in the key store file you specify.

**Note:** Exporting a key store requires the master key. If the master key is password protected, you must provide the correct password or passwords. If the master key is not password protected, you must either enter the master key in the **Current Master Key** box, or enter the file containing the master key, starting with the “@” character followed by the file name, or click **Browse** to locate the file.
Importing a key store

You can import a key store from another location. To import a key store, click Import. A dialog box similar to the following example appears:

![Image of import dialog box]

Enter the master key in the Current Master Key box, or click to choose the location of the master key.
Enter the full path to the file where the key store you want to import resides, or click Browse to browse for the location of the file.

Enter the password for the key store file, if required, in the Password for Key Store File box. Reenter the password in the Confirm Password box, then click Export.

UniVerse imports the contents of the key store file into the &KEYSTORE& file, the &ENCINFO& file, and the password policy file.

**Note:** Importing a key store requires the master key. If the master key is password protected, you must provide the correct password or passwords. If the master key is not password protected, you must either enter the master key in the “Current Master Key” box, or enter the file containing the master key, starting with the “@” character followed by the file name, or click Browse to locate the file.

You can also replace the user/group name and/or the file path in the imported key store file with a new name and/or path specified in the Change Owner and Change Path boxes. For more information, see Chapter 5, “Chapter 5: Automatic Data Encryption.”
Managing master keys

You can add or change passwords for the master key, or change the master key itself through U2 Extensible Admin. From the **Data Encryption** dialog box, click **Master Key**. The following dialog box appears:
To add or change a password for the master key, select the **Add Passwords** check box. A dialog box similar to the following example appears:

If you are using a system generated master key, select **System Default**. Otherwise, enter the master key.
If your master key has passwords, one or two password dialog boxes will appear instead of the **Current Master Key** box. You must then provide the correct password or passwords.

Enter the first new password for the master key in the **New Password 1** box. Reenter the password in the **Confirm New Password 1** box. If you want to add a second password, enter it in the **New Password 2** box. Reenter the second password in the **New Password 2** box.

Click **Update Master Key** to add passwords to the master key.

### Changing the master key

To change the master key, select the **Change Master Key** check box. A dialog box similar to the following example appears:

![Screen capture of UniVerse security features](image)

Enter the current masterkey in the **Current Master Key** box. Enter the password for the current master key in the **Master Key Password 1** box. If the master key has to passwords, enter the second password in the **Master Key Password 2** box.
If your master key does not have a password, you must enter the master key in the **Current Master Key** box, or enter the file containing the master key starting with the “@” character followed by the file name, or click **Browse** to locate the file.

If you want to use the system default master key, select **System Default**.

If you want to use a system generated master key, select **System Generated**.

If you want to enter your own master key, select **Key Text**. The **Master Key** dialog box appears, as shown in the following example:

![Master Key dialog box](image)

Enter the new master key in the **New Master Key** box. Reenter the new master key in the **Confirm New Master Key** box, then click **OK**.

If the master key resides in an operating system-level file, select **Key File**. The following dialog box appears:

![Master Key file dialog box](image)

Enter the full path of the file where the master key resides, or click **Browse** to locate the file, then click **OK**.
Click **Update Master Key** to create the new master key.

*Note*: If you change a master key, you must stop and restart UniVerse to have the new master key file take effect. See Chapter 5, "Chapter 5: Automatic Data Encryption," for more information about master keys.
Managing password policies

To manager password policies for encryption, click **Password Policies**. The following dialog box appears:

In the **Policy Components** portion of the dialog box, select the type of component the password policy is for. Valid values are:

- **Key** – The password policy applies only to the encryption key.
- **Wallet** – The password policy applies only to the encryption wallet.
- **Master Key** – The password policy applies only to the master key.
Select the component for which you want to change the password policy. A dialog box similar to the following example appears:

<table>
<thead>
<tr>
<th>Policy Name</th>
<th>Policy Value</th>
<th>Default</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnforcePolicy</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>MinimumLength</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>64</td>
</tr>
<tr>
<td>MaximumLength</td>
<td>32</td>
<td>32</td>
<td>6</td>
<td>64</td>
</tr>
<tr>
<td>MinimumAge</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>90</td>
</tr>
<tr>
<td>MaximumAge</td>
<td>90</td>
<td>90</td>
<td>1</td>
<td>Unlimited</td>
</tr>
<tr>
<td>RequiredCharSet</td>
<td>7</td>
<td>7</td>
<td>1</td>
<td>64</td>
</tr>
<tr>
<td>ExpirationWarning</td>
<td>14</td>
<td>14</td>
<td>0</td>
<td>30</td>
</tr>
<tr>
<td>Complexity</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>MinimumHistory</td>
<td>8</td>
<td>8</td>
<td>0</td>
<td>24</td>
</tr>
</tbody>
</table>

Click the value you want to change and enter the new value.

After you have updated the policy values, click **Update**, or click **Restore Defaults** to restore all values to the system defaults.

For more information about password policies, see “Password policies” in Chapter 5, “Chapter 5: Automatic Data Encryption.”
Chapter 7: Database Audit Logging

Overview .............................................. 7-3
The Audit Log file ................................. 7-4
   The Audit Log file contents ................. 7-6
Resources ............................................ 7-10
Audit events ........................................ 7-11
   System events .................................. 7-11
   Data events .................................... 7-12
   User event ..................................... 7-13
   DAT.QUERY.RESULTSET event record .... 7-14
   SYS.LOGON event ............................... 7-14
Configuring UniVerse for Audit Logging .... 7-15
   uvconfig file parameters .................... 7-15
   AUDIT_LOG_LOC ................................. 7-15
   AUDIT_LOG_MAX ................................ 7-15
   AUDIT_LOG_ENC ................................ 7-16
   The audit configuration file ................. 7-16
Audit policies .................................... 7-18
   Syntax terms .................................. 7-18
   Global policies ................................ 7-19
   Policy rules ................................... 7-23
Examples ............................................. 7-25
   Audit configuration examples ............... 7-25
   Event group policy examples ............... 7-26
   Account and file policy ...................... 7-26
   Account and file policy using environment variable .................................. 7-26
   Process policy ................................. 7-27
Program policy ........................................ 7-27
Specific policy without predefined event group policy .. 7-28
RESULTSET event ......................................... 7-28
Excluding system audit files from Audit Logging .......... 7-28
UniVerse BASIC changes .................................. 7-29
RetrieVe and UniVerse SQL implications................. 7-31
The audman utility ....................................... 7-32
Options ..................................................... 7-32
Database auditing tracks the usage of database resources and related authentication and authorization operations. Database auditing helps to satisfy compliance requirements from Health Insurance Portability and Accountability Act (HIPAA), Health Information Technology for Electronic and Clinical Health Act (HITECH), Payment Card Industry (PCI) and the Sarbanes-Oxley Act (SOX). It should be used in conjunction with other system or application-level audit facilities.

UniVerse Audit Logging is a component of UniVerse security. UniVerse databases are composed of data objects and system programs, which are resources to the UniVerse Audit Logging subsystem. When UniVerse accesses a resource, it triggers an audit event and an audit log record may be generated, depending on the audit configuration.

UniVerse Audit Logging is designed to be:

- Comprehensive – Covers all types of resources and operations.
- Flexible – Can be configured according to event types and through various policies, as well as before or after starting the system.
- Secure – The configuration file is encrypted and can be protected by a password, if desired. The Audit Log file is protected from illegal use and you can also encrypt its content.

The UniVerse Audit Logging implementation provides the following features:

- Classifies events and resources and audits them based on the classification
- Enables you to configure the location and number of audit files before UniVerse starts.
- Allows you to customize U2 Database Auditing without having to stop and restart UniVerse.
- Writes audit records to a UniVerse hashed file or group of files.
- Protects the audit file against unauthorized access and modification.
Overview

The Audit subsystem is an integral part of the system. You cannot completely disable the Audit subsystem. UniVerse always audits certain security related events. UniVerse ships with a default audit configuration, which has one active audit log file in the audit subdirectory of UVHOME, and, except for certain system security related events, no other events are enabled for auditing by default.

UniVerse ensures that the audit log files are valid and ready at startup. It loads the audit configuration into shared memory for use by other UniVerse processes.

If you change the audit configuration while UniVerse is running using the Audit Management utility (audman) or XAdmin, the UniVerse shared memory management daemon handles the audit configuration reload request by refreshing shared memory with the new audit configuration. The changes take effect for new and existing UniVerse processes. UniVerse also allows you to change the audit configuration when UniVerse is not running. In this case, UniVerse does not notify the shared memory management daemon.

The shared memory management daemon also periodically starts a background process to sweep staged audit records into active audit log files. UniVerse temporarily saves these records in a staging subdirectory by some standalone UniVerse utilities or daemon processes that do not have access to UniVerse hashed files, or they are created when UniVerse is not running.

Use the audman utility or XAdmin to control certain audit system behaviors, such as suspending and resuming active log files, clearing an audit log file, or notifying the shared memory management daemon to reload configuration data which takes effect immediately for existing and new UniVerse processes.

UniVerse audit log files are UniVerse hashed files that are protected by the system. You can use query commands such as LIST, SELECT or SQL SELECT, to query the contents of the files. Any attempt to change the contents of these files through UniVerse is blocked.

Note: Care should be taken when deciding which events to audit. The more events and more objects (files, processes, users, etc.) you select to audit, the more impact on performance.
The Audit Log file

Initially, UniVerse creates the audit log file as a 64-bit dynamic file with a block size of 4K and a modulo of 5000, making the file size approximately 20 MB. Depending on how you configure auditing, this file can grow very rapidly. You should regularly monitor the size of the audit log file, archive it regularly, and purge it when necessary. If the audit log file is full, you could lose newly generated audit records, and subsequent processing could stop.

UniVerse creates the log files with the shared memory management (uvsmm) daemon the first time you start UniVerse after installation, and verifies their existence and validity when you subsequently start UniVerse. UniVerse loads the dictionary of each log file based on $UVHOME/src.u/d_u2audlog.u.

If you modify an item in the U2-created audit log dictionary, it will be overwritten the next time UniVerse restarts. To avoid this, you should create your own item rather than modifying the existing item.

After creating and verifying the log file, the shared memory management daemon creates a file entry for each audit log file in the VOC file located in UVHOME. The ID of each entry is &AUDLOGn&, where n is the corresponding audit log file number.

If any of these steps fail when starting UniVerse, UniVerse may write error messages to the uvsmm.log file or the uvsmm.errlog file, and the system may fail to start.

You can access the Audit Log file just as you access any other UniVerse hashed file, but you cannot write to the file or delete it using UniVerse utilities, TCL commands, UniVerse SQL commands, or UniVerse BASIC.

Audit Log staging area

UniVerse creates a default “staging” directory in the audit subdirectory of UVHOME. A UniVerse daemon or a standalone utility may deposit its audit log records temporarily before they are loaded to the active audit log files. If you configure the audit log location to be created in a directory other than the default, UniVerse creates another staging directory under that location for use by the UniVerse daemons and standalone utilities.
UniVerse periodically loads the records in the staging directory to the audit log files when UniVerse is running. By default, this operation takes place every 120 seconds. You can change this time using the audman utility, but you can never set it to less than the shared memory management daemon check time, set by uvsmm.

**Audit Logging records**

Each audit record contains predefined data fields that include the event name and type, the resource information, user information, and other data. The audit record fields are not configurable.

For events regarding file record access, UniVerse only logs the record ID. For UniVerse BASIC READV and WRITEV statements, UniVerse logs the location of the data as well as the record ID. If the record ID is sensitive information, such as a social security number or credit card number, you should encrypt the audit log file.
The Audit Log file contents

The fields in the Audit Log file are not configurable. Each Audit log record contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
</table>
| @ID   | The time stamp and process ID in the following format: <date>.<time>.<ticks>.<pid>.<seq#>  
  Date – days from January 1, 1970 UTC.  
  Time – seconds from 0:00:00 of the date.  
  Ticks – Integer representing the microseconds within a second.  
  Pid – The event trigger process ID.  
  Seq # – Sequential audit record number within each process.  
  The @ID is composed in such a way that it guarantees the uniqueness of the audit record and facilitates the investigation and reconstruction of past events. |
| EventName | For SYS or DAT events – Predefined event name.  
  For USR events – USR.EVENT |
| Origin | The event originator, for example, the name of the executable such as uvsh, encman, uvregan, uvapi_slave, and so forth. You can also define the origin by the application for USR events. |
| Program | The UniVerse BASIC program. |
| Account | The account name if available, or the current directory. |
| User | The name of the user who triggers the event. |
| IPAddress | If available, the host name and/or the IP address of the client that connects to the UniVerse server. |
| File | The file name for DATA events and some SYS.COMMAND or UTILITY events. |
| RecordID | The record ID for DATA events and some SYS.COMMAND or UTILITY events. |
The following optional fields may or may not be stored in the audit log record. Certain events, such as USR EVENT, may create audit log records that have them.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OldData</td>
<td>The data before a change was made.</td>
</tr>
<tr>
<td>NewData</td>
<td>The data after a change was made.</td>
</tr>
</tbody>
</table>

### Field Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>Event dependent, for example:</td>
</tr>
<tr>
<td></td>
<td>- For SYS.ADE, the specific ADE operation such as CreateKey, and so forth.</td>
</tr>
<tr>
<td></td>
<td>- For SYS.COMMAND, the command name, such as LOGTO.</td>
</tr>
<tr>
<td></td>
<td>- For DAT.QUERY events, the statement name.</td>
</tr>
<tr>
<td></td>
<td>- For other events, this field may be empty.</td>
</tr>
<tr>
<td>Details</td>
<td>A free-form description that provides additional information about the event. This field varies by event, and may be empty.</td>
</tr>
<tr>
<td>Status</td>
<td>The final exit status of running a particular action.</td>
</tr>
<tr>
<td>BeforeAction</td>
<td>0 – post action</td>
</tr>
<tr>
<td></td>
<td>1 – before action</td>
</tr>
<tr>
<td>Consolidation type</td>
<td>1 – Time mode</td>
</tr>
<tr>
<td></td>
<td>2 – Counter mode</td>
</tr>
<tr>
<td></td>
<td>3 – First-time-only mode</td>
</tr>
<tr>
<td>Consolidation start date</td>
<td>The date the consolidation began.</td>
</tr>
<tr>
<td>Consolidation start time</td>
<td>The time the consolidation began.</td>
</tr>
<tr>
<td>Consolidation count</td>
<td>The number of items consolidated.</td>
</tr>
</tbody>
</table>

### Audit Log File Fields (Continued)
The following fields are defined in the Audit Log dictionary file as I-descriptors, whose data is derived from other fields in the dictionary file.

<table>
<thead>
<tr>
<th>I-Descriptor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EventClass</td>
<td>SYS, DAT, or USR</td>
</tr>
<tr>
<td>ProcessID</td>
<td>The process ID.</td>
</tr>
<tr>
<td>Date</td>
<td>Internal representation of the event date.</td>
</tr>
<tr>
<td>Time</td>
<td>Internal representation of the event time.</td>
</tr>
</tbody>
</table>

Audit Log file examples

This section illustrates the contents of the Audit Log file. For this example, we assume that the SYS.QUERY.RESULTSET event is enabled for the VOC file. The contents of the Audit Log record when you run the LIST VOC command is similar to the following example:

```
Audit Log Record ID.......16312.57361.207101.23303.54
Date.....................08/28/2012
Time.....................15:56:01
UTC Event Class.........DAT
Event Name...............DAT.QUERY.COMMAND
User.....................mike2
Process ID...............23303
Status...................1
BeforeAction.............0
Origin...................uvsh
Program..................
Account................../disk2/u2
IP Address..............
Action...................LIST
File......................VOC
Record ID................
Details...................23303-4:LIST VOC
```

The Details field lists the query ID 23303-4. From this ID, you can find all result set records in the audit log for the query by issuing another query similar to the following example:

```
LIST &AUDLOG1& WITH DETAILS LIKE “23303-4...”
```
The next example shows the audit log record for one of the records listed:

<table>
<thead>
<tr>
<th>Audit Log Record ID</th>
<th>Date</th>
<th>Time</th>
<th>UTC Event Class</th>
<th>Event Name</th>
<th>User</th>
<th>Process ID</th>
<th>Status</th>
<th>BeforeAction</th>
<th>Origin</th>
<th>Program</th>
<th>Account</th>
<th>IP Address</th>
<th>Action</th>
<th>File</th>
<th>RecordID</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>16312.57360.24259.23303.53</td>
<td>08/28/2012</td>
<td>15:56:00</td>
<td>DAT</td>
<td>DAT.QUERY.RESULTSET</td>
<td>mike2</td>
<td>23303</td>
<td>0</td>
<td>0</td>
<td>uvsh</td>
<td>/disk2/u2</td>
<td>SELECT.BUFFER-\Keyword-SET.SQL environment</td>
<td>23303-4:</td>
<td>RESULTSET</td>
<td>VOC</td>
<td>SELECT.BUFFER</td>
<td>DETAILS: 23303-4: SELECT.BUFFER-\Keyword-SET.SQL environment</td>
</tr>
</tbody>
</table>
Resources

Resources are logical representations of system objects. A resource can represent many objects, for example, a Utility resource could represent all UniVerse-supplied operating system level utilities. For auditing purposes, there are three classes of resources:

- **System resources** – refers to access control and authorization mechanisms, configuration and metadata, cataloged programs, system daemon processes, command and utilities, and so forth.
- **Data resources** – refers to data file and their schemas, dictionaries, views, indexes, and so forth.
- **User resources** – defined by applications
Audit events

For this document, an event is defined as an action on some predefined system resources. For example, any creation of a hashed file, an execution of a UniVerse BASIC READ statement, and a logon failure by a client through a UniVerse server are all events.

UniVerse always logs some events, for example some system-level or security-related operations, and optionally, ALL actions performed by privileged users, which is a requirement of PCI. All other events are not logged by default, but you can configure events to be logged.

UniVerse classifies the following events:

System events

This section describes System Events:

UniVerse system events

The following table describes the Logon/Logout/Authentication/Session events:

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS.LOGON</td>
<td>A user logon request through one of the UniVerse servers.</td>
</tr>
<tr>
<td>SYS.LOGOFF</td>
<td>A user logs off from a UniVerse server.</td>
</tr>
<tr>
<td>SYS.SESSION.START</td>
<td>A UniVerse session is initiated.</td>
</tr>
<tr>
<td>SYS.SESSION.END</td>
<td>A UniVerse session ended.</td>
</tr>
<tr>
<td>SYS.ADE.ACTIVATEKEY</td>
<td>Automatic Data Encryption key activation and deactivation.</td>
</tr>
<tr>
<td>SYS.COMMAND</td>
<td>A TCL command has been run, other than query commands.</td>
</tr>
<tr>
<td>SYS.UTILITY</td>
<td>Running of any UniVerse utilities.</td>
</tr>
</tbody>
</table>

System Events
### System configuration events

The following table describes the System Configuration event, which is always on and nonconfigurable:

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS.CONFIG.CHANGE</td>
<td>A system-level configuration changed, such as a uvconfig, audit configuration, replication configuration change, and so forth.</td>
</tr>
<tr>
<td>SYS.SECURITY</td>
<td>SQL GRANT, REVOKE, future security operations.</td>
</tr>
<tr>
<td>SYS.ADE</td>
<td>Automatic Data Encryption operations: master key, key store, key creation, key deletion, file encryption, index encryption, password related operations.</td>
</tr>
<tr>
<td>SYS.DAEMON</td>
<td>Starting or stopping of UniVerse background processes, such as U2Rep services.</td>
</tr>
</tbody>
</table>

### Data events

This section describes Data Events.

The following table describes the Data Events.

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAT.QUERY.COMMAND</td>
<td>LIST, SORT, SELECT, SUM, REFORMAT, COUNT, TLOAD, TDUMP, CVIEW SQL, SELECT</td>
</tr>
<tr>
<td>DAT.QUERY.RESULTSET</td>
<td>LIST, SORT, SELECT, SUM, REFORMAT, COUNT, TLOAD, TDUMP, CVIEW SQL, SELECT</td>
</tr>
<tr>
<td>DAT.BASIC.READ</td>
<td>BASIC READ, READV, MATREAD, SELECT, SELECTINDEX, READSEQ, READBLK, BSCAN</td>
</tr>
</tbody>
</table>
This section describes the User Event.

The following table describes the User Event.

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USR.EVENT</td>
<td>User-defined audit event through the new UniVerse BASIC AuditLog() function.</td>
</tr>
</tbody>
</table>

User Event

Some entities are under control of both UniVerse and the operating system. For example, you can modify a configuration file either through UniVerse or directly from an operating system command. If you modify it through UniVerse, an audit event is triggered, but if you modify it from the operating system level, an audit event is not triggered. To fully account for events that occur on a system, both UniVerse Audit Logging and operating system level auditing must be used together.

Some TCL commands actually run an operating system level utility to perform a function. For example, the RESIZE command runs the resize utility. In order to audit under all situations, audit logs are created at both the TCL level and the operating system level when the corresponding utility is run. As a result, a TCL command may trigger 2 audit records, or 4 if the BeforeAction audit is enabled.
DAT.QUERY.RESULTSET event record

If the DAT.QUERY.RESULTSET event is enabled, each record in a query result set creates one audit log file in the audit log file. To identify the result set records with its query statement, UniVerse requires that the DAT.QUERY.COMMAND event be enabled at the same time. If you have not already enabled the DAT.QUERY.COMMAND in the configuration file, UniVerse enables it when UniVerse is started.

UniVerse assigns a “query ID” to each result set query command in the form of “processID-seq#” and puts it before the query statement in the Details field for the query command. For each RESULTSET record, UniVerse stores this query ID in the Details field of that record, along with the actual data returned by the query command.

SYS.LOGON event

On UNIX and Linux systems, the UniVerse telnet server performs the user log on and then starts a shell session. From the shell, you can then access UniVerse. The log on and starting of UniVerse creates separate audit records if so configured. To associate the telnet session with its UniVerse process, UniVerse stores the process ID of the UniVerse telnet service (uvtelnetd) in the SYS.SESSION.START audit record of the UniVerse process so you can track the activities of a telnet client. From the UniVerse process audit record, you can trace back to the corresponding client’s host name, IP address and port.
Configuring UniVerse for Audit Logging

This section describes how to configure UniVerse for Audit Logging.

You can configure UniVerse Audit Logging by editing two files, the uvconfig file and the u2audit.config file. The uvconfig file contains parameters for specifying where to create the audit log file or files, and whether or not the files should be encrypted. The u2audit.config file contains audit policies that control what events to audit and how they should be audited.

uvconfig file parameters

The uvconfig parameters described in this section are used to configure UniVerse Audit Logging.

AUDIT_LOG_LOC

The value of the AUDIT_LOG_LOC parameter specifies the directory in which UniVerse creates audit log files. If this value is not a full path, UniVerse treats it as relative to UVHOME. If you do not specify this directory, UniVerse uses the default directory of $UVHOME/audit.

The files contained in the audit log directory are named U2audlogn, where n is from 1 to the value of the AUDIT_LOG_MAX directory. UniVerse creates the directory with write access only to the root or Administrator user, and all u2audlogn files under it with write access to all users.

AUDIT_LOG_MAX

You may want to specify more than one audit log file so that you can suspend some log files for maintenance, such as backup or purging, while still allowing audit logging to the remaining active log files. After such maintenance, you may resume the suspended log files and suspend the previous active log files for maintenance. Using more log files may also improve system performance by splitting audit log writes to different files. As a shortcoming, using multiple log files complicates the task of discovery or reconstruction of past events, should such requirements arise.
The maximum number of audit log files is 8. The default value is 1.

**AUDIT_LOG_ENC**

Specifies whether or not to encrypt the audit log files. A value of 0 specifies to not encrypt the files, and a value of 1 specifies that the files should be encrypted by the system. UniVerse encrypts the files regardless if Automatic Data Encryption has been set up or not.

The purpose of encrypting the audit log files is to ensure that no sensitive data is stored in clear-text in the audit log files. When UniVerse reads the data in the audit log files, it automatically decrypts the data. However, if you copy or back up your audit log files using operating system level commands, the data remains encrypted.

**The audit configuration file**

Enabling or disabling auditing of events is controlled by a system-wide audit configuration file named u2audit.config located in the $UVHOME directory. This is an encrypted file and can only be changed through the audman utility with the –config option or the Extensible Administration Tool (XAdmin) by users with root, uvadm, or Administrator privileges. If you change the file when UniVerse is running, UniVerse sends a message to the shared memory management daemon to have its contents reloaded into shared memory.

Each line in the u2audit.config file defines an audit policy. The general format is as follows:

```
 policy_name [+ | -]= value
```

UniVerse always tries to read the u2audit.config file when the system starts. If the file is missing or corrupt, UniVerse enables audit logging for ALL events as the default action. This prevents possible malicious activity on the account as nobody can disable auditing without the event being logged, and prevents a DoS type of attack. UniVerse creates a warning message in the uvsmm.log to indicate this situation.

After you enable audit logging, an event triggers the creation of a record in the audit log file. If you encrypt the file, read and query operations automatically decrypt the record.
Writing to the audit log file is always a synchronous operation. UniVerse blocks the event triggering the UniVerse process until the audit log write completes. If a write error happens for any reason, the operation either proceeds or stops, depending on the configuration.
Audit policies

UniVerse provides global policies and event policies. Global policies include:

- On_error
- Privileged_user_audit
- Account
- Event group

Global policies are not associated with any specific event.

Event policies include:

- File
- User
- Process
- Executable
- Program
- BeforeAction
- Status
- Consolidation

Event policies must be associated with a single or a group of events.

Syntax terms

The following terms are used when describing audit policies:

List

A list is composed of objects of the same type, for example, events, processes, programs, users, or files that are separated by either a comma (",") or a vertical bar ("|”). UniVerse allows spaces or tabs between the object names and the separators.
**Operator**

An operator specifies inclusion or exclusion in the form of:

\[
(+ | -)=
\]

- “+=” – adds the policy you specify to the inclusion list
- “-=” – adds the policy you specify to the exclusion list
- “=” – sets the inclusion list only to the policy you specify and removes whatever is currently defined in the exclusion list

**Global policies**

This section describes the global policies available in UniVerse.

**Audit error handling policy**

The Audit Error Handling policy determines what action to take after an auditing processing error occurs. If you specify “stop,” UniVerse processes stop if an error occurs during audit processing. If you specify “ignore,” UniVerse ignores the error and continues processing.

\[
on\_error = \{ignore \mid stop\}
\]

The default is ignore.

**Privileged user policy**

The Privileged User policy specifies whether or not to audit all activities by privileged users defined as root or uvadm users on UNIX and Linux platforms and Administrators group members on Windows platforms.

\[
\text{privileged\_user\_audit} = \{on \mid off\}
\]

The default is off.

**Account policy**

The Account policy defines a default shorthand for a UniVerse account.

\[
\text{account} = \langle\text{account\_name}\rangle:\langle\text{full\_account\_path}\rangle
\]
A full account path is either a path starting from the root directory, such as /disk1/uv/uv112 on UNIX platforms or C:\uv\uv112 on Windows platforms, or a path starting with an environment variable which contains an absolute full path at run time, such as $UVHOME/HS.SALES, where $UVHOME contains /disk1/uv/uv112.

On Windows platforms, the length of <account_name> must be at least 2.

Example

The following example illustrates an account policy that defines MyAccount as a shorthand for “disk1/u2/SALES”:

```
OurEvents=DAT.BASIC.READ, DAT.BASIC.WRITE, DAT.QUERY.COMMAND
account=MyAccount:/disk1/u2/SALES
OurEvents.file+=MyAccount:CUSTOMER, MyAccount:PRODUCTS
```

Think of ‘account’ as a keyword and ‘MyAccount’ as the user-defined alias or a shorthand name.

Event group policy

An event group policy defines a group of events. (+ or no sign: on, -: off)

```
<event_group> [+ | -]= <event_list>
```

Event specific policies

With the exception of beforeAction, the purpose of an event-specific policy is to restrict creating audit records for an event that has been enabled.

The following events are always audited:

- SYS.ADE
- SYS.SECURITY
- SYS.DAEMON
- SYS.CONFIG.CHANGE

UniVerse ignores the events listed above if they appear in a policy.
File policy

The File Policy applies to Data events only.

\[ \{<\text{event\_name}> | <\text{event\_group}>\}.file [+ | -] = <\text{file\_list}> \]

You must specify the files in \(<\text{file\_list}>\) with either a full path or in the form of:

\(<\text{account}>:\<\text{file}>\)

where \(<\text{account}>\) must already be defined by an account policy, and \(<\text{file}>\)
must be a physical file name, not necessarily a VOC entry name. A full path
is either a path starting from the root directory, such as
\(/\text{disk1/\u2/SALES/VOC} \) on UNIX platforms or \(C:\\u2\SALES\VOC\) on
Windows platforms. The path can also begin with an environment variable
that contains an absolute full path at run time, such as \($UVHOME/SALES$\).

UniVerse Auditing is performed based on physical files rather than file
entries in the VOC file. UniVerse also recognizes hard and symbolic links.

Wildcard File Name

One exception to the File Policy rule is that UniVerse allows two forms of
wildcard specification:

- Wildcard file name – you can specify an asterisk ("*") to indicate all
  files.
- Wild card for files in a directory – if you want to specify all files in a
directory, use the form \(<\text{account\_name}>:*\) or \(\text{/.../*}\).

A directory wildcard such as \(/\text{mydir}/*\) does not include the directory itself.

User policy

A User Policy contains user logon names or operating system group names.
You must prefix group names with an asterisk ("*"). For Windows platforms
you may also specify domain names, such as \(\text{MYDOMAIN\\johnd}\) or
*\(\text{MYDOMAIN\USERS}\).

\[ \{<\text{event\_name}> | <\text{event\_group}>\}.user [+ | -] = <\text{user\_group\_list}> \]
Process policy

\{<event_name> | <event_group>.process [+ | -]= <process_list>\}

The process_list contains process IDs. UniVerse ignores invalid process IDs.

Executable policy

\{<event_name> | <event_group>.executable [+ | -]= <executable_list>\}

The executable_list contains UniVerse servers and/or utilities names.

Program policy

\{<event_name> | <event_group>.program {+ | -}= <program_list>\}

In this syntax, program is a UniVerse BASIC program or cataloged procedure started by the RUN command. In other words, you can run the program from the TCL command or EXECUTE it from a UniVerse BASIC program, or embed it in a paragraph. UniVerse ignores invalid programs.

BeforeAction policy

The beforeAction policy creates an audit record before the action is taken. This policy does not apply to some SYS events, such as SYS.LOGON. By default, all event audit logging is done after the event action is completed.

\{<event_name> | <event_group>.beforeAction = {on | off | 1 | 0}\}

The default is off.

Status policy

A Status policy specifies whether to log only successful events or failures, or both. The default is both.

\{<event_name> | <event_group>.status = {success | failure | both}\}
UniVerse usually returns 0 for successful execution of an event or nonzero for failure. However, there are exceptions. For example, when you terminate a query command by “Q” at the prompt, UniVerse sets the status to -1. Therefore, audit logging may not be completely accurate based on the status value being 0 or nonzero.

**Consolidation policy**

A Consolidation policy specifies a time, expressed in seconds, within which multiple events of the same type are consolidated into a single event. UniVerse logs one record with the consolidation mode, start time, and occurrence counter in a group of consolidation information fields. You can also specify a counter for consolidation, then UniVerse records the same type of events only when the specified counter has been reached. You can also specify a “first-only” policy, which means that during a session, UniVerse only logs the first occurrence of an event.

UniVerse performs consolidation based on the program, file, before-action state, and execution status within a UniVerse session.

Only DAT events, USR.EVENT, and SYS.ADE.ACTIVITY events can have a consolidation policy. UniVerse ignores the consolidation policy for all other events, meaning those events trigger immediate audit record logging at the time the events happened if enabled.

When UniVerse detects that an Audit Logging configuration has been reloaded or a UniVerse process is preparing to exit, it flushes all pending consolidation events to the audit log files. You may lose some information when you consolidate, such as record IDs.

```plaintext
{<event_name> | <event_group>}.consolidation = none |
  time:<sec> | counter:<count> | firstonly default: none
```

**Policy rules**

This section describes the rules that apply to policies.

- Policies apply to configurable events only. For example, UniVerse ignores “Always on” system events even if it appears in a configuration file.
The events in event-group policies are meant to be enabled for all users, processes, executables and programs and on all files (for DATA policies), unless further restricted by user, process, file, executable, and program policies.

If you remove an event from a group (using -=), UniVerse disables the event regardless if it is also a member of another previously-defined group, that is, the order of event group definition is important.

If you only specify “exclusion” for a specific policy, it also means that all other entities of this policy are audited. If you specify any “inclusions,” all those that are not matched are not audited. If you “set” any option using only the equal sign (“=”), only the matching ones are audited. All previously specified inclusion or exclusion policies are cancelled.

Events that are not positively defined in any event group, or on the left side of any policies, are not audited, except for all non-configurable events.

UniVerse processes policies in the order you specify them. For example, you must define any event-group policy first before you can reference it in other policies.

UniVerse accepts wild card event names. For example, the asterisk (“*”) represents all configurable audit events. SYS.* represents all configurable SYSTEM events, and SYS.SESSION.* represents all SYSTEM SESSION events, and so forth. Other than group policies, UniVerse does not allow wild card events on the right side.

The user name must be a valid operating system account name. On Windows platforms, it is case insensitive and domain designation is allowed, for instance, SALES\JohnDoe.

When evaluating policies, specific policy has priority over general policy, and exclusion has priority over inclusion. For example, process policy is more specific than user policy, SYS.* is more specific than *, and so forth.
Examples

This section contains several examples of audit configuration files.

Audit configuration examples

Enable auditing for all events

The example of this configuration file enables auditing for all events, users, files, and processes without restriction.

\[
\text{allevents} = * 
\]

Enable auditing for specific events, users, files and processes

The next configuration file example defines two groups of audit events. \textit{Sysevents} includes all system-level events. \textit{Otherevents} includes all data events. For \textit{sysevents} events, UniVerse only logs events triggered by user johnw and mikep. For \textit{otherevents} events, UniVerse only logs events triggered by process 19208 and on the CUSTOMER file. The file policy only applies to DAT events.

\[
\begin{align*}
\text{sysevents} &= \text{SYS}.* \\
\text{sysevents.user} &= \text{johnw,mikep} \\
\text{otherevents} &= \text{DAT}.* \\
\text{otherevents.process} &= \text{19208} \\
\text{otherevents.file} &= /\text{myacct/CUSTOMER} \\
\end{align*}
\]

Audit specific events for all users except specific one

The following configuration file specifies that, for a SYS.LOGON event, UniVerse logs the event for all users excepts johnw and mikep. For all DATA events, UniVerse logs all processes except for the process with ID 19208 or uvapi_slave processes. If the process is privileged AND if the privileged\_user\_audit policy is enabled, UniVerse ignores this policy.

\[
\begin{align*}
\text{SYS\_LOGON.user} &= \text{johnw,mikep} \\
\text{dataevents} &= \text{DAT}.* \\
\text{dataevents.process} &= \text{19208} \\
\text{dataevents.executable} &= \text{uvapi_slave} \\
\end{align*}
\]
**Event group policy examples**

You cannot have an event policy without specifying events to monitor. In the following example, “OurEvents” is properly defined.

```
OurEvents=DAT.BASIC.READ
OurEvents.process=5112014
```

The next example is invalid since “OurEvents” is not defined.

```
OurEvents.process=5112014
```

You can enable a list or group of defined events and not define a policy. In this case, there is no restriction, and each event triggers creation of an audit log record.

```
OurEvents=DAT.BASIC.READ, DAT.BASIC.DELETE, DAT.QUERY.COMMAND
```

**Account and file policy**

This example illustrates an account and file policy.

Line 1 defines a list of events to audit.

Line 2 is an alias or shorthand definition to the account path. This is not required, but if you do not use this, you have to specify the full path of the account.

Line 3 defines a file policy that indicates auditing is required for any of the events defined in line 1 that occur on any of the files specified in line 3.

```
1. OurEvents=DAT.BASIC.READ, DAT.BASIC.WRITE, DAT.QUERY.COMMAND
2. account=MyAccount:/disk1/u2/SALES
3. OurEvents.file+=MyAccount:CUSTOMER, MyAccount:PRODUCTS
```

**Account and file policy using environment variable**

The next example shows an account and file policy using an environment variable.

Line 1 defines a list of events to audit.
Line 2 is an alias or shorthand definition to the account path. In this example, the account path is defined as $UVHOME/HS.SALES. $UVHOME must be defined for the session and contain a valid directory path. This account path is not required, but if you do not use this, you have to specify the full path of the account.

Line 3 is a file policy that indicates auditing is required for any of the events defined in line 1 that occur on any of the files specified in line 3. In this example UniVerse uses another environment variable, $UVDEV.

```plaintext
1. OurEvents=DAT.BASIC.READ, DAT.BASIC.WRITE, DAT.BASIC.DELETE, DAT.QUERY.COMMAND
2. account=MyAccount$UVHOME/SALES
3. OurEvents.file=+MyAccount:CUSTOMER, MyAccount:PRODUCTS, $UVDEV/CUSTOMER
```

**Process policy**

The next example illustrates a process policy.

Line 1 defines a list of events to audit.

Line 2 states UniVerse should audit any UniVerse BASIC READ statement from any program, on any file, performed by pid 5112014, and such actions by any other uv process should not be audited. The pid must be a UniVerse process pid.

```plaintext
1. OurEvents=DAT.BASIC.READ
2. OurEvents.process=5112014
```

**Program policy**

The following example illustrates a program policy.

Line 1 defines a list of events to audit.

Line 2 states that UniVerse should only audit UniVerse BASIC READ statements performed by the RDM.LOG.THIS.ONE program.

```plaintext
1. OurEvents=DAT.BASIC.READ
2. OurEvents.program=RDM.LOG.THIS.ONE
```
Specific policy without predefined event group policy

If you define a policy on a single event, as in “Audit specific events for all users except specific one” you can omit the group definition, as shown in the following example:

```
DAT.BASIC.READ.program=RDM.LOG.THIS.ONE
```

RESULTSET event

In this example, UniVerse enables ResultSet for query commands such as LIST, SELECT, SQL SELECT, on ALL files.

```
Myevents=DAT.QUERY.RESULTSET
```

To restrict this policy to certain files, you need to specify some policies such as those in the following example:

```
Myevents.file=myfile...or
DAT.QUERY.RESULTSET.file=myfile...
```

Excluding system audit files from Audit Logging

In the previous example, Audit Logging logs entries for queries on all files, including the audit log file. This example illustrates how you can restrict logging to a particular file, or on all files and exclude the audit log file.

1. MyEvents=DAT.QUERY.RESULTSET
2. DAT.QUERY.* = <path-to-my-file>
3. DAT.QUERY.* -= <path-to-U2audlog1>

Line 2 specifies to log a specific file.

Line 3 excludes the system audit log file.
UniVerse BASIC changes

To reduce unnecessary or excessive logging, the UniVerse BASIC AuditLog() function has been added at this release. This function allows for application-driven audit. For example, instead of enabling system-wide UniVerse BASIC READ auditing, which could create a huge number of audit log records, you can choose to have your application call this function at a strategic point to have the action recorded in the system audit file.

Syntax

AuditLog(Originator, Action, File, Record, Info, Status, OldData, NewData)

Parameters

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Originator</td>
<td>The ID of the originator of the event.</td>
</tr>
<tr>
<td>Action</td>
<td>The action taken.</td>
</tr>
<tr>
<td>File</td>
<td>The file name to audit.</td>
</tr>
<tr>
<td>Record</td>
<td>The record ID to audit.</td>
</tr>
<tr>
<td>Info</td>
<td>Additional details about this logged action. The content specified in this</td>
</tr>
<tr>
<td></td>
<td>parameter goes into the Details field of the audit log file.</td>
</tr>
<tr>
<td>Status</td>
<td>An integer indicating the status of the logged actions. 0 usually indicates</td>
</tr>
<tr>
<td></td>
<td>success, and nonzero values indicate errors.</td>
</tr>
<tr>
<td>OldData</td>
<td>Optional. The data before the change was made.</td>
</tr>
<tr>
<td>NewData</td>
<td>Optional. The data after the change was made.</td>
</tr>
</tbody>
</table>
All parameters are expressions that evaluates to text strings (Originators, Action, File, and Info) or dynamic arrays (Record, OldData, NewData), except for Status, which must be an integer.

**Note:** OldData and NewData are optional parameters. You can omit them if you don't need to store these values. Also, if you do not need File or RecordID, you can supply empty strings instead.

For example:

```plaintext
OldAddr = Rec.addr
NewAddr = "1234 Main St Cape Town MA 02021"
CALL ChangeAddr("file1", "ID1", Rec, NewAddr)
Status = AuditLog("myappl", "ChangeAddr", "file1", "ID1","replaced billing address", 0, OldAddr, NewAddr)
```
RetrieVe and UniVerse SQL implications

For UniVerse queries using RetrieVe or UniVerse SQL, UniVerse logs the entire statement. For result-set level auditing, where each record in a result set is logged as a separate record, we cannot guarantee that UniVerse can do a complete and accurate job due to the complexity of the query evaluation and reporting. This is because a query statement may contain a TRANS statement and you may wish to log every access to the file contained in the TRANS statement. For UniVerse SQL, multiple files may be joined, aggregate functions may be used, and the final result may not include data from some of the joined files. For these reasons, we recommend that you use the new AuditLog() function to achieve result-set audit information.
The audman utility

Use the audman utility to manage Audit Logging.

Syntax

```
audman

-config [-noreload | -reloadonly | -display [level] | -file path] [-password [pass]] [-newpassword [newpass] [-editor]]

-suspendlog n
-resumelog n
-clearlog n [-force]
-writestagedlog [-silent | [-disable | -enable] [-interval n]]
-checkaudlog file encflag [-silent]
-logstatus
```

Options

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-config</td>
<td>Use the -config option to view, change, verify, or reload the audit configuration.</td>
</tr>
<tr>
<td>-noreload</td>
<td>Do not reload configuration.</td>
</tr>
<tr>
<td>-reloadonly</td>
<td>Reload configuration if there is no syntax error, without getting into editor.</td>
</tr>
</tbody>
</table>

audman Options
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-display [n]</td>
<td>Display current configuration. n determines the detail level.</td>
</tr>
<tr>
<td></td>
<td>0 – display events</td>
</tr>
<tr>
<td></td>
<td>1 – display groups in addition to 0</td>
</tr>
<tr>
<td></td>
<td>2 – display file tables in addition to 0 and 1.</td>
</tr>
<tr>
<td>-editor</td>
<td>Allows users to specify a text editor to override the system’s default editor. The default text editor on an UNIX system is vi and on Windows the default editor is Notepad.</td>
</tr>
<tr>
<td>-file path</td>
<td>The path to the configuration file.</td>
</tr>
<tr>
<td>-password [currpass]</td>
<td>Specifies the current password for the config file.</td>
</tr>
<tr>
<td>-newpassword [newpass]</td>
<td>Add or change a password for the config file,</td>
</tr>
<tr>
<td>-suspendlog n</td>
<td>Suspend the nth audit log from accepting new records.</td>
</tr>
<tr>
<td>-resumelog n</td>
<td>Resume nth audit log.</td>
</tr>
<tr>
<td>-clearlog n</td>
<td>Clear nth audit log. We recommend doing this after successfully suspending logging</td>
</tr>
<tr>
<td>-force</td>
<td>Clears the log without further confirmation.</td>
</tr>
<tr>
<td>-writestagedlog</td>
<td>Write staged log records to audit log file.</td>
</tr>
<tr>
<td>-silent</td>
<td>Suppresses messages.</td>
</tr>
<tr>
<td>-enable</td>
<td>Enable log write.</td>
</tr>
<tr>
<td>-disable</td>
<td>Disable automatic log write.</td>
</tr>
<tr>
<td>-interval n</td>
<td>Set automatic log write interval to n seconds. This value must be more than 0 and less than 65535.</td>
</tr>
</tbody>
</table>

_Option Description (Continued)_

7-33
The -config operation enables you to create and/or change the system audit configuration file. On UNIX or Linux platforms, a vi session appears with the configuration file contents. This appears in notepad on Windows platforms or vi on UNIX platforms. You can then modify the file according to the audit policy syntax rules described above.

You can specify the editor you want to use from the command line or by setting an environment variable. From the command line, enter the following command to change the default editor for this running of audman only:

```
    audman -config -editor name_of_editor
```

Set the U2AUDIT_EDITOR environment variable to change the default editor every time you run the audman utility, as shown in the following example:

```
    U2AUDIT_EDITOR="emacs"
```

**Note**: Do not select an editor that will format the text.

When you save the file, UniVerse checks the syntax and encrypts the file. UniVerse save the configuration file as it was before being changed with a file name as u2audit.config.mmddyyyy.hhmmss. If you make frequent configuration changes, you could accumulate a large number of saved configuration files. UniVerse stores backup file copies of the u2audit.config file in the $UVHOME/audit/config directory named u2auditconfig.

If you specify the -display [level] option, UniVerse displays the current configuration loaded in shared memory. Level can be from 0 to 2, and provides more details:

- 0 – display events

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-checkaudlog</td>
<td>Checks that the file is a valid audit log file.</td>
</tr>
<tr>
<td>file encflag</td>
<td></td>
</tr>
<tr>
<td>-silent</td>
<td>Suppresses messages.</td>
</tr>
<tr>
<td>-logstatus</td>
<td>Displays the status of the current log file.</td>
</tr>
</tbody>
</table>

-**config operation**

The -config operation enables you to create and/or change the system audit configuration file. On UNIX or Linux platforms, a vi session appears with the configuration file contents. This appears in notepad on Windows platforms or vi on UNIX platforms. You can then modify the file according to the audit policy syntax rules described above.

You can specify the editor you want to use from the command line or by setting an environment variable. From the command line, enter the following command to change the default editor for this running of audman only:

```
    audman -config -editor name_of_editor
```

Set the U2AUDIT_EDITOR environment variable to change the default editor every time you run the audman utility, as shown in the following example:

```
    U2AUDIT_EDITOR="emacs"
```

**Note**: Do not select an editor that will format the text.

When you save the file, UniVerse checks the syntax and encrypts the file. UniVerse save the configuration file as it was before being changed with a file name as u2audit.config.mmddyyyy.hhmmss. If you make frequent configuration changes, you could accumulate a large number of saved configuration files. UniVerse stores backup file copies of the u2audit.config file in the $UVHOME/audit/config directory named u2auditconfig.

If you specify the -display [level] option, UniVerse displays the current configuration loaded in shared memory. Level can be from 0 to 2, and provides more details:

- 0 – display events
- display groups in addition to 0
- display file tables in addition to 0 and 1.

When the -config operation completes and it passes the syntax check, UniVerse notifies the shared memory management daemon so the new values are reloaded into shared memory. If you specify the -noreload option, UniVerse does not send the notification. If you specify the -reloadonly option, no editing takes place, but UniVerse checks the syntax before it sends the notification.

The -file option enables you to edit a user-specified audit configuration rather than the default system configuration file, u2audit.config. You must specify a path with -file. UniVerse does not send a reload message to the shared memory management daemon when the editing is done.

You can optionally protect the audit configuration file with a password. If the configuration file has a password, you cannot edit the file without providing the correct password using the -password option. To set or change the password, use the -newpassword option. If you specify the -password or -newpassword option but do not provide a password on the command line, UniVerse prompts for the password.

The purpose of the password is to prevent unauthorized editing of the configuration file. A password is not required to read the configuration file for startup purposes, even if a password exists for the configuration file. The password is only required by the audman -config command for editing and reloading purposes.

**suspendlog, resumelog, and clearlog operations**

UniVerse uses the suspendlog, resumelog, and clearlog operations to manage active audit log files. You can suspend, resume, or clear a specific audit log file using a file number \(n\). \(n\) must be from 1 to the maximum number of audit log files you specify in the uvconfig file. When you suspend an audit log file, UniVerse does not write any new records to that file. You can then perform file maintenance, such as archiving or clearing the audit log files. UniVerse always has at least one active audit log file available. Therefore, if you try to suspend the last active audit file the operation fails.
writestagedlog operation

The writestagedlog operation does two things. You can use this operation to copy staged log records to active audit log files, or you can use it to disable or enable an automatic staged-log sweep. By default, the shared memory management daemon periodically starts a staged-log-write operation every 120 seconds. You can modify the time using the -interval option. The interval value cannot be less that the shared memory management daemon’s check time interval.

checkaudlog operation

The checkaudlog operation validates if the file you specify is a valid audit log file. A valid audit log file must be a UniVerse 64-bit dynamic hash file with proper audit setup in the file itself, and has an encryption setup that matches the current AUDIT_LOG_ENC value in the uvconfig file. If the file you specify does not validate, the audman utility prints an error message. If you specify the -silent option, UniVerse only returns an error code. The shared memory management daemon performs the operation when UniVerse starts. You do not generally have to run this operation.

logstatus operation

The logstatus operation is used to display the status of the current log file. When the logstatus operation is used, the output will look similar to the following example.

```
Audit Logging is licensed and enabled
Audit log location : /eskimo3/uv112/audit
Audit log encryption : no
Number of audit log files : 8

<table>
<thead>
<tr>
<th>Log File</th>
<th>Status</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>active</td>
<td>20520960</td>
</tr>
<tr>
<td>2</td>
<td>suspended</td>
<td>20488192</td>
</tr>
<tr>
<td>3</td>
<td>suspended</td>
<td>20488192</td>
</tr>
<tr>
<td>4</td>
<td>suspended</td>
<td>20488192</td>
</tr>
<tr>
<td>5</td>
<td>suspended</td>
<td>20488192</td>
</tr>
<tr>
<td>6</td>
<td>suspended</td>
<td>20488192</td>
</tr>
<tr>
<td>7</td>
<td>suspended</td>
<td>20488192</td>
</tr>
<tr>
<td>8</td>
<td>suspended</td>
<td>20488192</td>
</tr>
</tbody>
</table>
```