# Universal Broker

User Guide

Universal Products

Version 3.2.0

ub-user-3206

# Universal Broker

## User Guide

## Universal Broker 3.2.0

| Document Name | Universal Broker 3.2.0 User Guide | | | | |
|---|---|---|---|---|---|
| **Document ID** | **ub-user-3206** | | | | |
| **Products** | **z/OS** | **UNIX** | **Windows** | **OS/400** | **HP NonStop*** |
| Universal Broker | √ | √ | √ | √ | √ |
| * Universal Broker 2.1.1 is used on the HP NonStop operating system. | | | | | |

# Stonebranch Documentation Policy

# Summary of Changes

## Changes for Universal Broker 3.2.0 User Guide (ub-user-3206) February 19, 2010

- Added *JOBCTL to the commands for removing *ALLOBJ and/or *SPLCTL authorities from the UBROKER user profile in Section 6.4.3 User Profile.

## Changes for Universal Broker 3.2.0 User Guide (ub-user-3205) September 8, 2009

- Added Section 6.2.3 Configuration Refresh.

### Universal Broker 3.2.0.6

- Specified, in Table 3.1 Universal Broker for z/OS – DD Statements in JCL Procedure, that `UNVDB` and `UNVSPOOL` ddnames are not used if zFS data sets are used.
- Modified the following entries for zFS data sets in Table 3.2 Universal Broker for z/OS - Configuration Options:
  - MOUNT_POINT
  - MOUNT_POINT_MODE
- Added the UNIX_DB_DATA_SET and UNIX_SPOOL_DATA_SET configuration options in the following tables:
  - Table 2.2 Universal Broker Options - Configuration File Editable Only; Recycle Required
  - Table 3.2 Universal Broker for z/OS - Configuration Options
- Added the LOG_FILE_LINES configuration option in the following tables:
  - Table 2.4 Universal Broker Options - UMC and Configuration File Editable; Refresh Required
  - Table 5.1 Universal Broker for UNIX - Configuration Options
- Added zFS data set information in Section 8.3.1 z/OS.

## Changes for Universal Broker 3.2.0 User Guide (ub-user-3204) July 29, 2009

### Universal Broker 3.2.0.1 for OS/400

- Modified document for upgrade from Universal Broker 3.1.1 for OS/400 to Universal Broker 3.2.0 for OS/400, including:
  - Changed the following OS/400 names throughout the document:
    - Universal Broker subsystem name from `UBROKER` to `UNVUBR320`.
    - Universal Broker user profile name from `UBROKER` to `UNVUBR320`.
    - Universal Products installation library name from `UNIVERSAL` to `UNVPRD320`.
    - Universal Products spool library name from `UNVSPOOL` to `UNVSPL320`.
    - Universal Products temporary directory from `UNVTMP` to `UNVTMP320`.
  - Added the following configuration options in Table 6.1 Universal Broker for OS/400 - Configuration Options:
    - ACTIVITY_MONITORING
    - CERTIFICATE_REVOCATION_LIST
    - EVENT_GENERATION
    - MONITOR_EVENT_EXPIRATION
    - PERSISTENT_EVENT_EXPIRATION
    - SERVCIE_BACKLOG
  - Added the following entries in Table 6.3 Universal Broker for OS/400 - UACL Entries:
    - EVENT_ACCESS
    - REMOTE_CONFIG_ACCESS
  - Modified Section 6.4.3 User Profile information in Section 6.4 Security.
  - Added subsection 8.3.4 OS/400 in Section 8.3 Universal Broker Databases.

## Changes for Universal Broker 3.2.0 User Guide (ub-ref-3203) April 1, 2009

- Added DD statement for SAP RFC file used by Universal Connector, as of Universal Connector 3.2.0.1, in:
  - Figure 3.1 Universal Broker for z/OS – JCL procedure
  - Table 3.1 Universal Broker for z/OS – DD Statements in JCL Procedure.

## Changes for Universal Broker 3.2.0 User Guide (ub-user-3202) December 17, 2008

- Modified Removing *ALLOBJ Authority from UNVUBR320 User Profile in Section 6.4.3 User Profile of Chapter 6 Universal Broker for OS/400.
- Added Updating the Universal Broker ACL Entries in Section 4.4.4 Universal Access Control List of Chapter 4 Universal Broker for Windows.

## Changes for Universal Broker 3.2.0 User Guide (ub-user-3201) September 5, 2008

- Added toll-free telephone number for North America in Appendix A Customer Support.

## Changes for Universal Broker 3.2.0 User Guide (ub-user-320) May 16, 2008

### Universal Broker 3.2.0.0

- Added support for the following features:
  - Universal Broker throughput and scalability has been improved. The Broker can now process hundreds of connections simultaneously while maintaining a high transaction rate.
  - Universal Broker is now required on all host on which a Universal Product component executes, including Manager components.
- Added Chapter 2 Features, including:
  - Section 2.3 Universal Broker Configuration Refresh
  - Section 2.4 Remote Configuration
  - Section 2.5 Universal Configuration Manager.
- Added the following configuration options:
  - ACTIVITY_MONITORING
  - BIF_DIRECTORY
  - CERTIFICATE_REVOCATION_LIST
  - EVENT_GENERATION
  - MONITOR_EVENT_EXPIRATION
  - PERSISTENT_EVENT_EXPIRATION
  - SAF_KEY_RING
  - SAF_KEY_RING_LABEL
  - SSL_IMPLEMENTATION
  - SYSTEM_ID
- Added the following UACL entries:
  - EVENT_ACCESS
  - REMOTE_CONFIG_ACCESS
- Deleted the following specification methods for all configuration options:
  - Command Line, Short Form
  - Command Line, Long Form
  - Environment Variable
- Added Configuration File Keyword as a specification method for Windows configuration options.
- Added Chapter 8 Database Administration.

# Contents

# List of Figures

# List of Tables

# Preface

## Document Structure

This document is written using specific conventions for text formatting and according to a specific document structure in order to make it as useful as possible for the largest audience. The following sections describe the document formatting conventions and organization.

### Format

Starting with the Universal Products 3.2.0 release, this Universal Broker 3.2.0 User Guide was created. Formerly, information on Universal Broker (and Universal Control) was documented in the Universal Command User Guide.

Additionally, links to detailed information in a companion document, the Universal Broker 3.2.0 Reference Guide, have been created in this user guide.

In order for the links between these documents to work correctly:

- Place the documents in the same folder.
- In Adobe Reader / Adobe Acrobat, de-select **Open cross-document link in same window** in the **General** category of your **Preferences** dialog (selected from the **Edit** menu).

# Conventions

Specific text formatting conventions are used within this document to represent different information. The following conventions are used.

## Typeface and Fonts

`This Font` identifies specific names of different types of information, such as file names or directories (for example, `\abc\123\help.txt`).

## Command Line Syntax Diagrams

Command line syntax diagrams use the following conventions:

| Convention | Description |
|---|---|
| **`bold monospace font`** | Specifies values to be typed verbatim, such as file / data set names. |
| *`italic monospace font`* | Specifies values to be supplied by the user. |
| [ ] | Encloses configuration options or values that are optional. |
| { } | Encloses configuration options or values of which one must be chosen. |
| \| | Separates a list of possible choices. |
| . . . | Specifies that the previous item may be repeated one or more times. |
| **BOLD UPPER CASE** | Specifies a group of options or values that are defined elsewhere. |

Table P.1 Command Syntax

## Operating System-Specific Text

Most of this document describes the product in the context of all supported operating systems. At times, it is necessary to refer to operating system-specific information. This information is introduced with a special header, which is followed by the operating system-specific text in a different font size from the normal text.

**z/OS**

This text pertains specifically to the z/OS line of operating systems.

This text resumes the information pertaining to all operating systems.

### Tips from the Stoneman

Look to the Stoneman for suggestions
or for any other information
that requires special attention.

**Stoneman's Tip**

## Vendor References

References are made throughout this document to a variety of vendor operating systems. We attempt to use the most current product names when referencing vendor software.

The following names are used within this document:

- **z/OS** is synonymous with IBM z/OS and IBM OS/390 line of operating systems.
- **Windows** is synonymous with Microsoft's Windows 2000 / 2003 / 2008, Windows XP, Windows Vista, and Windows 7 lines of operating systems. Any differences between the different systems will be noted.
- **UNIX** is synonymous with operating systems based on AT&T and BSD origins and the Linux operating system.
- **OS/400** is synonymous with IBM OS/400, IBM i/5, and IBM i operating systems.
- **AS/400** is synonymous for IBM AS/400, IBM iSeries, and IBM System i systems.

Note: These names do not imply software support in any manner. For a detailed list of supported operating systems, see the Universal Products 3.2.0 Installation Guide.

## Document Organization

This document is organized into the following chapters:

- Overview (Chapter 1)
  General architectural and functional overview of Universal Broker.
- Features (Chapter 2)
  Product features as they pertain to the entire family of products for all operating systems.
- Universal Broker for z/OS (Chapter 3)
  Description of Universal Broker specific to the z/OS operating system.
- Universal Broker for Windows (Chapter 4)
  Description of Universal Broker specific to the Windows operating system.
- Universal Broker for UNIX (Chapter 5)
  Description of Universal Broker specific to the UNIX operating system.
- Universal Broker for OS/400 (Chapter 6)
  Description of Universal Broker specific to the OS/400 operating system.
- Universal Broker for HP NonStop (Chapter 7)
  Description of Universal Broker specific to the HP NonStop operating system.
- Database Administration (Chapter 8)
  Information about Universal Broker database administration.
- Customer Support (Appendix A)
  Customer support contact information for Universal Broker (and all Universal Products).

# Chapter 1
# Overview

## 1.1 Introduction

Universal Broker manages Universal Products components.

A component is a program that provides a well-defined service. For example, the Universal Command Server and Universal Command Manager each are components.

Universal Broker is not aware of the service that a component provides; it knows only that it is a component. To Universal Broker, all components are equal.

Universal Broker provides the following management tasks:

1. Receive requests to start components on behalf of a user. The user can be another component or a person.
2. Track all components it has started.
3. Report on all components it has started.
4. Receive requests to restart components.

Universal Broker is required on all systems running a Universal Products component.

# Chapter 2 Features

## 2.1 Overview

This chapter provides information on Universal Broker features that apply to all operating systems.

- Configuration
- Configuration Files
- Universal Broker Configuration Refresh
- Remote Configuration
- Universal Configuration Manager
- Network Data Transmission
- Universal Spool
- Universal Access Control List
- Message and Audit Facilities
- X.509 Certificates

## 2.2 Configuration

If local requirements require a change in product configuration, there are multiple methods of configuration available. Product configuration consists of specifying options that control product behavior and resource allocation.

An example of configurable product behavior option is the specification for whether or not data transferred over the network is compressed. An example of a configurable resource allocation option is the directory location in which a product creates its log files.

Each option is comprised of a pre-defined keyword that identifies the option and one or more values for that option. The format of the keyword depends on the configuration method being used (see Section 2.2.1 Configuration Methods).

Although there are many different configuration options for each product, Universal Products – in general – are designed to require minimal configuration and administration. The default values for the options work very well in most environments.

### 2.2.1 Configuration Methods

All Stonebranch Inc. Universal Products provide consistent and flexible methods of configuration. Depending on the product, and the operating system on which it is being run, configuration can be performed via one or more of the following methods:

- Command line
- Command line file
- Environment variables
- Configuration file

The command line, command line file, and environment variables methods let you set options and preferences for a single execution of a product.

The configuration file method lets you set default options and preferences for all executions of a product.

#### Universal Broker Configuration Method

Universal Broker, and all Universal Products servers, are configurable only by modifying their configuration files (see Section 2.2.2 Configuration Files). They are not configurable via command line, command line file, or environmental variables.

## 2.2.2 Configuration Files

Configuration files specify system-wide configuration values.

Most Universal Products have some options that can be specified only in a configuration file. Other options can be overridden by individual command executions. (The Stonebranch, Inc. documentation for each product identifies these options.)

There are three ways to modify a configuration file:

1. Via text editor.
2. Remotely, via the Universal Enterprise Controller's Universal Management Console client application (see Section 2.4 Remote Configuration).
3. For Universal Products for Windows, via graphical user interface (see Section 2.5 Universal Configuration Manager).

**z/OS**

Configuration files are members of a PDSE. The data set record format is fixed or fixed block with a record length of 80. No line numbers can exist in columns 72-80. All 80 columns are processed as data.

All configuration files are installed in the **UNVCONF** library.

See Section 2.2.3 Configuration Files Syntax for the configuration file syntax.

**UNIX**

Configuration files are regular text files on UNIX.

Universal Broker searches for the configuration files in a fixed list of directories. It uses the first configuration file that it finds in its search. The directories are listed below in the order they are searched:

| Directory | Notes |
|---|---|
| /etc/opt/universal | |
| /etc/universal | Installation default. |
| /etc/stonebranch | Obsolete as of version 2.2.0. |
| /etc | |
| /usr/etc/universal | |
| /usr/etc/stonebranch | Obsolete as of version 2.2.0. |
| /usr/etc | |

Table 2.1 UNIX Configuration File Directory Search

See 2.2.3 Configuration Files Syntax for the configuration file syntax.

### Windows

Configuration files reside in **`%ALLUSERSPROFILE%\Application Data\Universal\conf`**, where **`%ALLUSERSPROFILE%`** is an environment variable that resolves by default to **`C:\Documents and Settings\All Users`** on Windows 2000 / XP / Server 2003 and **`C:\ProgramData`** on Windows Vista / Server 2008.

Although configuration files can be edited with any text editor (for example, Notepad), the Universal Configuration Manager application, accessible via the Control Panel, is the recommended way to set configuration options.

The Universal Configuration Manager provides a graphical interface and context-sensitive help, and helps protect the integrity of the configuration file by validating all changes to configuration option values (see Section 2.5 Universal Configuration Manager).

### OS/400

The configuration files on OS/400 are stored in a source physical file named **`UNVCONF`** in the **`UNVPRD320`** library. The files can be edited with a text editor.

See Section 2.2.3 Configuration Files Syntax for the configuration file syntax.

### HP NonStop

The configuration files on HP NonStop are stored as EDIT files, file code 101, within the **`$SYSTEM.UNVCONF`** subvolume. The files can be edited with the EDIT editor.

See Section 2.2.3 Configuration Files Syntax for the configuration file syntax.

## Universal Products Components Configuration Files

Universal Broker maintains the configuration files for all Universal Products components that it manages. The components do not read the configuration files themselves (except for Universal Enterprise Controller, which directly reads its own configuration file).

When a component starts, it first registers with its local Universal Broker. As part of the registration process, Universal Broker returns the configuration data to the component.

Universal Broker reads the configuration files at initial start-up and, thereafter, whenever it is refreshed; that is, when either of the following occurs:

- Universal Broker is recycled (stopped and restarted).
- Universal Broker receives a REFRESH command from Universal Control.
- Univeral Broker is refreshed by Universal Enterprise Controller (via Universal Management Console).

**Windows**

- Universal Broker is refreshed by Universal Configuration Manager.

Any changes made to a configuration file are not in effect until Universal Broker is recycled or receives a REFRESH command (see Section 2.3 Universal Broker Configuration Refresh).

## Universal Broker Configuration File

As with all Universal Products, all Universal Broker options can be modified by editing the configuration file directly.

However, unlike other Universal Products, not all Universal Broker options can be modified via the Universal Management Console (UMC). In UMC, these options are read-only.

Additionally, Universal Broker must be recycled (stopped and restarted) in order for some modified options to be updated in Universal Broker memory. These options do not take effect when Universal Broker is refreshed (see Section 2.3 Universal Broker Configuration Refresh).

## 2.2.3 Configuration Files Syntax

Configuration files are text files that can be edited with any available text editor.

The following rules apply for configuration file syntax:

- Options are specified in a keyword / value format.
- Keywords can start in any column.
- Keywords must be separated from values by at least one space or tab character.
- Keywords are not case sensitive.
- Keywords cannot contain spaces or tabs.
- Values can contain spaces and tabs, but if they do, they must be enclosed in single ( **'** ) or double ( **"** ) quotation marks. Repeat the enclosing characters to include them as part of the value.
- Values case sensitivity depends on the value being specified. For example:
  - Directory and file names are case sensitive.
  - Pre-defined values (such as **yes** and **no**) are not case sensitive.
- Each keyword / value pair must be on one line.
- Characters after the value are ignored.
- Newline characters are not permitted in a value.
- Values can be continued from one line to the next either by ending the line with a:
  - Plus ( **+** ) character, to remove all intervening spaces.
  - Minus ( **-** ) character, to preserve all intervening spaces between the end of the line being continued and the beginning of the continuing line.

  Ensure that the line continuation character is the last character on a line.
- Comment lines start with a hash (**#**) character.
- Blank lines are ignored.

Note: If an option is specified more than once in a configuration file, the last option specified is used.

## 2.3 Universal Broker Configuration Refresh

After a Universal Broker configuration has been modified, the Universal Broker must be refreshed in order for the modified values to take effect. Refreshing a Universal Broker directs it to read its configuration data and update its current configuration settings.

There are four ways in which Universal Broker can be refreshed:

1. Recycle (stop and restart) Universal Broker
   The configuration values for Universal Broker, and all components that Universal Broker manages, are refreshed.
2. Issue a Universal Control REFRESH command
   Universal Broker reads all configuration information, including its own information, and then refreshes itself with this information.
3. Modify options via Universal Management Console (UMC)
   Universal Broker is refreshed automatically.

**Windows**

4. Modify options via Universal Configuration Manager
   Universal Broker is refreshed automatically.

Some Universal Broker options can be modified only by editing the configuration file. For these modifications to take effect, Universal Broker must be recycled (see Section 2.3.1 Configuration File Editable Only, Recycle Required).

All other Universal Broker options can be modified either by editing the configuration file, via the UMC, or via the Universal Configuration Manager. Depending on the option, for a modification to take effect:

- Universal Broker must be recycled (see Section 2.3.2 UMC and Configuration File Editable, Recycle Required).
- Universal Broker must be refreshed:
  - By issuing a REFRESH command, if the modifications are made in the configuration file.
  - Automatically, if the modifications are made via UMC or the Universal Configuration Manager .

  (See Section 2.3.3 UMC and Configuration File Editable, Refresh Required.)

## 2.3.1 Configuration File Editable Only, Recycle Required

Table 2.2, below, identifies Universal Broker options that you can modify only by editing the Universal Broker configuration file, and for which Universal Broker must be recycled in order for the modified values to be used.

These options are not updated when Universal Broker is refreshed.

(In Universal Management Console, these options are Read-Only.)

| Option | Description |
|---|---|
| BIF_DIRECTORY | Broker Interface File directory that specifies where Universal Broker will create its interface file. |
| COMPONENT_DIRECTORY | Component definition file directory. |
| INSTALLATION_DIRECTORY | Base directory where product is installed. |
| MOUNT_POINT | HFS or zFS database mount directory. |
| MOUNT_POINT_MODE | HFS or zFS permission mode for MOUNT_POINT. |
| NLS_DIRECTORY | UMC and UTT file directory. |
| PID_FILE_DIRECTORY | PID file location. |
| SMF_EXIT_LOAD_LIBRARY | UNVACTRT SMF exit load library. |
| SPOOL_DIRECTORY | Spool file directory. |
| SYSTEM_ID | Universal Broker running on a system (O/S image). |
| UCMD_STC_SUPPORT | Support for Universal Command started tasks. |
| UNIX_DB_DATA_SET | HFS or zFS data set used for the Universal Broker's databases. |
| UNIX_SPOOL_DATA_SET | HFS or zFS data set used for the Universal Broker's spool. |

Table 2.2 Universal Broker Options - Configuration File Editable Only; Recycle Required

**Stoneman's Tip**

If the PID_FILE_DIRECTORY value is modified, the UNIX script that starts/stops/restarts the Universal Broker, `ubrokerd`, also must be modified to indicate the location of the Broker's PID file.

If `ubrokerd` is not modified, it will not know the Process ID of the executing Universal Broker. Thus, it will not be able to return status information of the executing Universal Broker sucessfully.

## 2.3.2 UMC and Configuration File Editable, Recycle Required

Table 2.3, below, identifies Universal Broker options that you can modify by editing the Universal Broker configuration file or via the Universal Management Console, and for which Universal Broker must be recycled in order for the modifications to take effect.

**Windows**

If the options are modified via the Universal Configuration Manager, Universal Broker must be recycled.

These options are not updated when Universal Broker is refreshed.

| Option | Description |
|---|---|
| CA_CERTIFICATES | Path to PEM-formatted trusted CA X.509 certificates. |
| CERTIFICATE | Path to Broker's PEM-formatted X.509 certificate. |
| CERTIFICATE_REVOCATION_LIST | Path to PEM-formatted CRL. |
| COMPONENT_PORT | TCP/IP port used for Broker-Component communications. |
| PRIVATE_KEY | Path to Broker's PEM formatted RSA private key. |
| PRIVATE_KEY_PWD | Password for the Broker's PRIVATE_KEY. |
| SAF_KEY_RING | SAF certificate key ring name. |
| SAF_KEY_RING_LABEL | SAF certificate key ring label. |
| SERVICE_BACKLOG | Service interface backlog size for pending connection requests. |
| SERVICE_IP_ADDRESS | TCP/IP address on which the Broker listens. |
| SERVICE_PORT | TCP/IP port number on which the Broker listens. |
| SSL_IMPLEMENTATION | SSL implementation to be used for network configuration. |

Table 2.3 Universal Broker Options - UMC and Configuration File Editable; Recycle Required

## 2.3.3 UMC and Configuration File Editable, Refresh Required

Table 2.4, below, identifies Universal Broker options that you can modify by editing the Universal Broker configuration file or via Universal Management Console, and for which Universal Broker only needs to be refreshed in order for the modifications to take effect.

- If the options are modified by editing the Universal Broker configuration file, a Universal Control REFRESH command must be issued.
- If the options are modified via Universal Management Console, Universal Broker is refreshed automatically.

**Windows**

If the options are modified via the Universal Configuration Manager, Universal Broker is refreshed automatically.

| Option | Description |
|---|---|
| ACTIVITY_MONITORING | Specification for whether or not product activity monitoring events are generated. |
| CODE_PAGE | Text translation code page. |
| CTL_SSL_CIPHER_LIST | SSL cipher list for the control sessions. |
| DNS_CACHE_TIMEOUT | Time-out for DNS cache. |
| EVENT_GENERATION | Events to be generated as persistent events. |
| LOG_DIRECTORY | Log file directory. |
| LOG_FILE_LINES | Total number of lines to be written to the log file before the log file is wrapped. |
| MESSAGE_DESTINATION | Location where messages are written. |
| MESSAGE_LANGUAGE | Language of messages written. |
| MESSAGE_LEVEL | Level of messages written. |
| MONITOR_EVENT_EXPIRATION | Duration of a monitoring event record in the Universal Broker local UES database. |
| PERSISTENT_EVENT_EXPIRATION | Duration of a persistent event record in the Universal Broker local UES database. |
| RUNNING_MAX | Maximum number of simultaneous components. |
| TMP_DIRECTORY | Directory for temporary files. |
| TRACE_DIRECTORY | Directory for trace files. |
| TRACE_FILE_LINES | Maximum number of lines written to the trace file. |
| TRACE_TABLE | Memory trace table specification. |
| WORKING_DIRECTORY | Broker’s working directory. |

Table 2.4 Universal Broker Options - UMC and Configuration File Editable; Refresh Required

## 2.4 Remote Configuration

Universal Products can be configured remotely by Universal Enterprise Controller using the Universal Management Console (UMC) client application, and can be "locked down" so that they *only* can be remotely configured.

UMC instructs the Universal Broker of a remote Universal Agent to modify the configurations of the Universal Products components managed by that Universal Broker.

Universal Broker supports remote configuration in either of two modes:

1. Unmanaged Mode
2. Managed Mode

### 2.4.1 Unmanaged Mode

Unmanaged mode is the default mode of operations for Universal Broker. It allows a Universal Broker – and the Universal Products managed by that Universal Broker – to be configured either:

- Locally, by editing configuration files.
- Remotely, via Universal Management Console (UMC).

The system administrator for the machine on which a Universal Agent resides can use any text editor to modify the configuration files of the various local Universal Products.

Via UMC, selected users can modify all configurations of any Universal Agent, including the local Universal Agent. UMC sends the modified data to the Universal Broker of that agent, which Universal Broker then uses to update the appropriate configuration files.

If UMC sends modifications for a Universal Broker configuration, Universal Broker validates the modified data before it accepts it. If the data fails validation, Universal Broker does not update its configuration file.

If UMC sends modification to the configuration of any other Universal Products component, the Universal Broker updates the appropriate configuration file. The component will use this new configuration at its next invocation.

Note: If errors or invalid configuration values are updated via UMC for a component other than Universal Broker, the component may not run successfully until the configuration has be corrected.

## 2.4.2 Managed Mode

When a Universal Broker is operating in managed mode, the configuration information for all Universal Products components managed by that Universal Broker is "locked down." Universal Broker stores the information in a database file located within its specified spool directory. The information can be modified only via Universal Management Console (UMC).

From this point on, Universal Broker uses the database file – not the configuration files – to access configuration information. Any configuration changes made to the components – via UMC – are placed in the database file. Therefore, as long as Universal Broker stays in managed mode, the configuration files may no longer contain current or valid configuration information.

If managed mode is de-selected for the Universal Broker, it reads the database file where it stored the configuration information. Universal Broker uses this information to create and/or update configuration files for the components.

- If a configuration file exists in the configuration directory, it is overwritten.
- If a configuration file does not exist, it is created.

Note: Because of remote configuration and the desire to be able to "lock down" all product configurations, Universal Broker – and all Universal Products servers – no longer support the command line and environmental variables methods of specifying configuration options.

### Selecting Managed Mode

The managed mode of operations for Universal Broker is selected via the Universal Enterprise Controller Administration client application.

(See the Universal Enterprise Controller 3.2.0 Client Applications User Guide for specific information on how to select managed mode.)

Figure 2.1, below, illustrates remote configuration for one Universal Agent in managed mode and one Universal Agent in unmanaged mode.

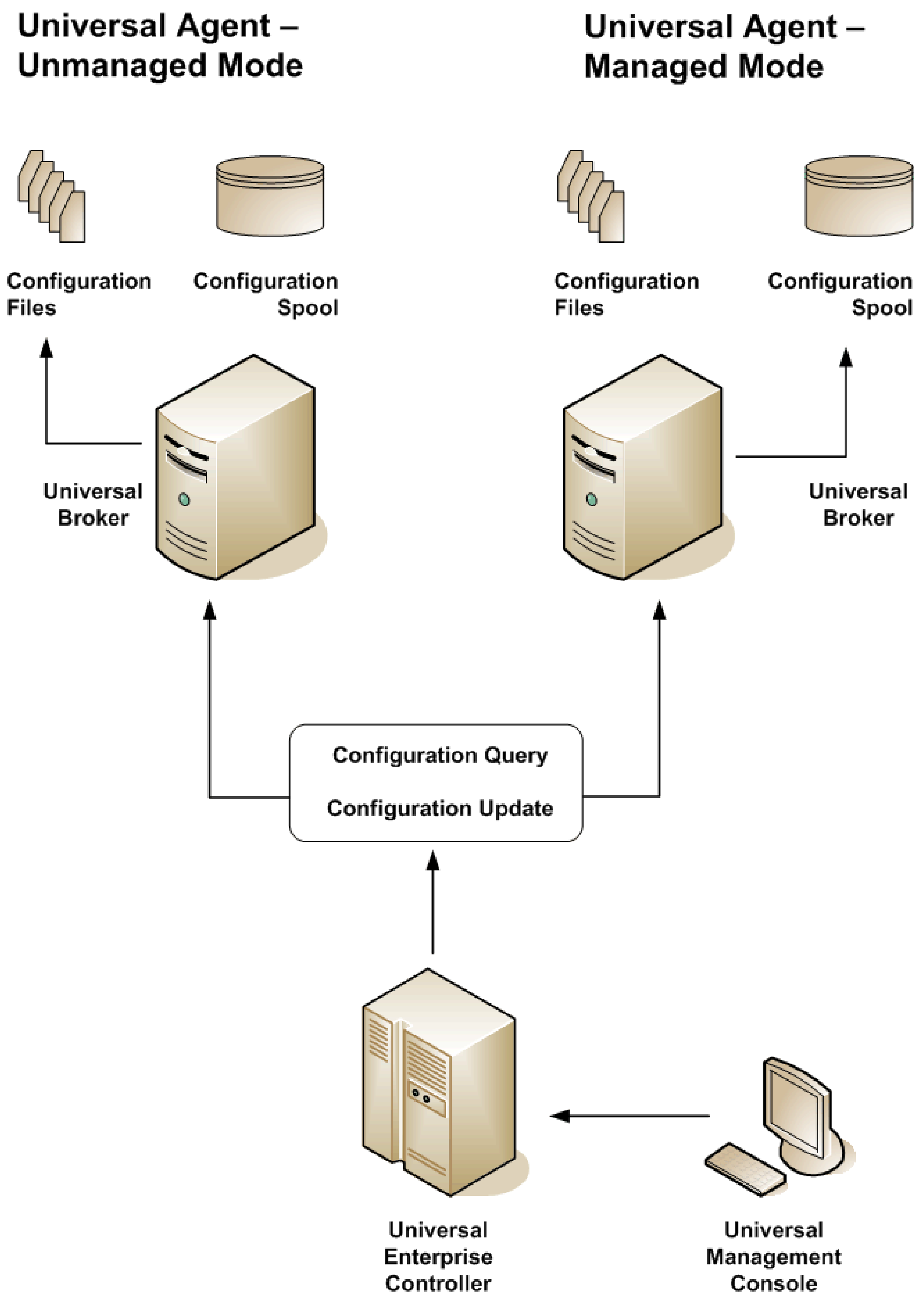


Figure 2.1 Remote Configuration - Unmanaged and Managed Modes of Operation

## 2.4.3 Universal Broker Start-up

At Universal Broker start-up, in both managed and unmanaged modes, the Universal Broker configuration file is always read.

### Unmanaged Mode

At Universal Broker start-up in unmanaged mode, Universal Broker reads the configuration files of all Universal Products components into its memory. The Universal Broker configuration file is used to define the Universal Broker configuration, just as all configuration files are used in unmanaged mode. Universal Broker updates its memory from the configuration files whenever Universal Control issues a REFRESH request.

### Managed Mode

At Universal Broker start-up in managed mode, the Universal Broker configuration file points Universal Broker to the location of the configuration spool file, from which the Broker retrieves configuration information for all Universal Products. Universal Broker updates its memory from the configuration spool file and, automatically, after changes are made via UMC.

If more configuration information than needed is included in the Universal Broker configuration file at Universal Broker start-up, Universal Broker will update its running configuration with the information that it retrieved from the spool file. The configuration file that was used at start-up is made obsolete.

## 2.5 Universal Configuration Manager

Universal Configuration Manager is a graphical user interface application for Universal Products for Windows. It enables you to configure all of the Universal Products for Windows components.

Universal Configuration Manager is the recommended method of specifying configuration data that will not change with each command invocation. It helps protect the integrity of the configuration file by validating all changes to configuration option values.

### 2.5.1 Availability

Universal Configuration Manager is installed automatically on the Windows operating system as part of every Universal Products for Windows installation.

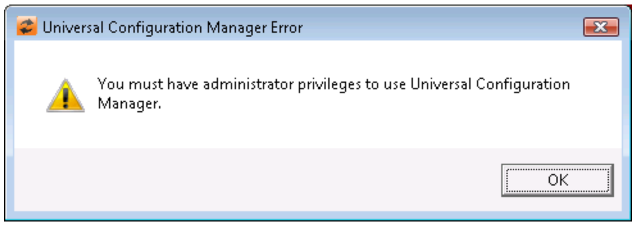
It is available to all user accounts in the Windows Administrator group.

**Windows Vista**

When opening the Universal Configuration Manager for the first time on Windows Vista, two new operating system features, the Program Compatibility Assistant (PCA) and User Account Control (UAC), may affect its behavior.

With these two features enabled, the expected Universal Configuration Manager behavior is as follows:

1. Universal Configuration Manager may issue the following error:

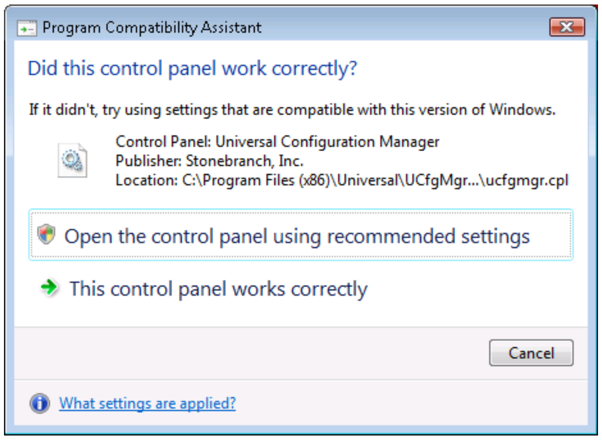

Universal Configuration Manager Error

You must have administrator privileges to use Universal Configuration Manager.

Figure 2.2 Universal Configuration Manager Error dialog - Windows Vista

2. Click **OK** to dismiss the error message.

   The Windows Vista Program Compatibility Assistant (PCA) displays the following dialog:


Program Compatibility Assistant

Did this control panel work correctly?

If it didn't, try using settings that are compatible with this version of Windows.

Control Panel: Universal Configuration Manager
Publisher: Stonebranch, Inc.
Location: C:\Program Files (x86)\Universal\UCfgMgr...\ucfgmgr.cpl

Open the control panel using recommended settings
This control panel works correctly
Cancel
What settings are applied?

Figure 2.3 Windows Vista - Program Compatibility Assistant

3. To continue, select **Open the control panel using recommended settings**. This instructs the PCA to "shim" (Microsoft term) the Configuration Manager, establishing it as an application that requires elevated privileges.

   Windows Vista User Account Control (UAC) then displays a prompt seeking permission to elevate the logged-in account's access token.
4. Select **Continue** to give the account full administrative privileges.

   Subsequent attempts to open Universal Configuration Manager should result only in the UAC prompt.

## 2.5.2 Accessing the Universal Configuration Manager

To access the Universal Configuration Manager:

1. Click the **`Start`** icon at the lower left corner of your Windows operating system screen to display the Start menu.
2. Click (Settings/) Control Panel on the Start menu to display the Control Panel screen.
3. Select the Universal Configuration Manager icon to display the Universal Configuration Manager screen (see Figure 2.4).

**Windows XP, Windows Vista, Windows Server 2008**

Newer versions of Windows support a Control Panel view that places applet icons within categories. This "category view" may affect the location of the Universal Configuration Manager icon.

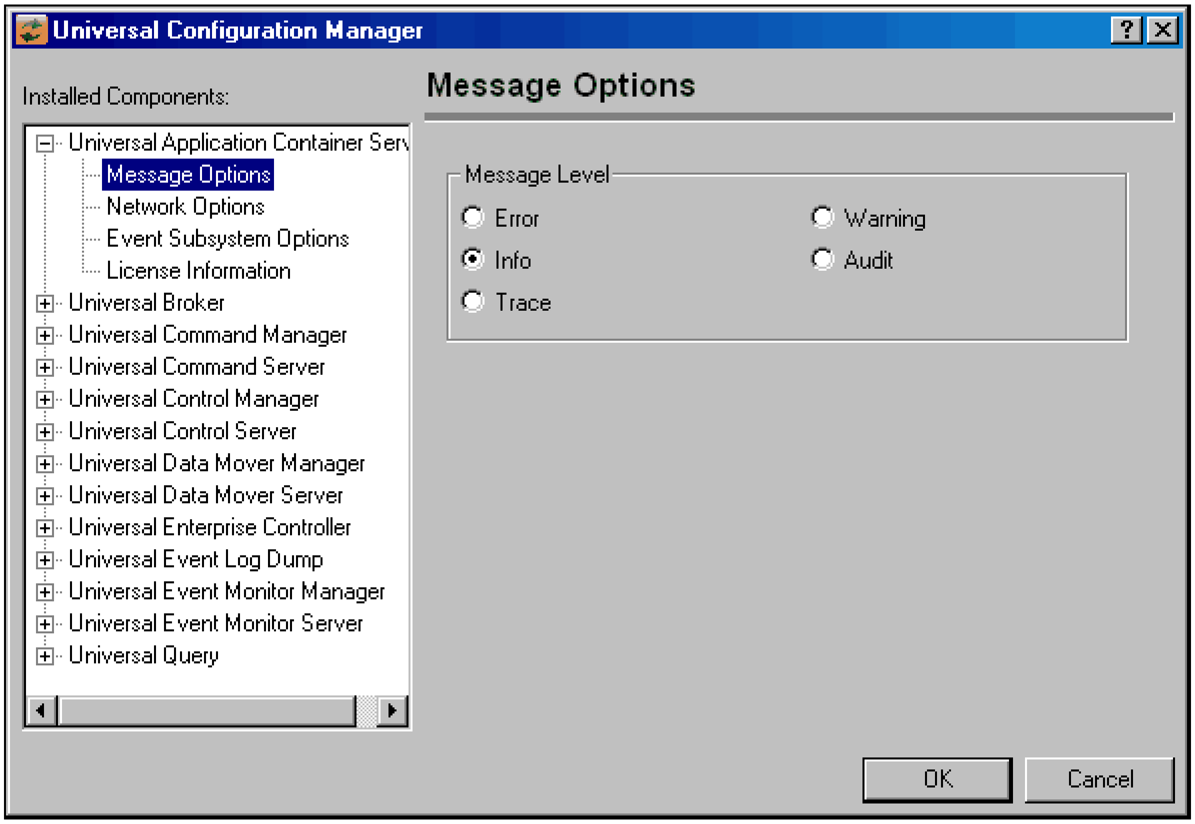
For example, the Windows XP Category View places the Universal Configuration Manager icon under the **`Other Control Panel Options`** link. Windows Vista and Windows Server 2008 place the icon within the **`Additional Options`** category.

If you have trouble locating the Universal Configuration Manager icon, simply switch to the Classic View to display all Control Panel icons at the same time.

**64-bit Windows Editions**

The Windows Control Panel places icons for all 32-bit applets under the **`View x86 Control Panel Icons`** (or, on newer versions, the **`View 32-bit Control Panel Icons`**) category, even when the Classic View is enabled.

When using the Category View, look for the 32-bit Control Panel applet icons in the **`Additional Options`** category.


Universal Configuration Manager
Installed Components:
Universal Application Container Serv
Message Options
Network Options
Event Subsystem Options
License Information
Universal Broker
Universal Command Manager
Universal Command Server
Universal Control Manager
Universal Control Server
Universal Data Mover Manager
Universal Data Mover Server
Universal Enterprise Controller
Universal Event Log Dump
Universal Event Monitor Manager
Universal Event Monitor Server
Universal Query
Message Options
OK
Cancel

Message Level: Error Warning [✓ Info] Audit Trace

Figure 2.4 Universal Configuration Manager

Each Universal Configuration Manager screen contains two sections:

1. Left side of the screen displays the Installed Components tree, which lists:
   - Universal Products components currently installed on your system.
   - Property pages available for each component (as selected), which include one or more of the following:
     - Configuration options
     - Access control lists
     - Licensing information
     - Other component-specific information
2. Right side of the screen displays information for the selected component / page.

By default, Universal Configuration Manager displays the first property page of the first component in the Installed Components tree.

## 2.5.3 Navigating through Universal Configuration Manager

To display general information about a component, click the component name in the Installed Components list.

To display the list of property pages for a component, click the **+** icon next to the component name in the Installed Components list.

To display a property page, click the name of that page in the Installed Components list.

If a property page has one or more of its own pages, a **+** icon displays next to the name of that property page in the Installed Components list. Click that **+** icon to display a list of those pages.

In Figure 2.4, for example:

- List of property pages is displayed for Universal Application Container Server.
- Message Options property page has been selected, and information for that property is displayed on the right side of the page.
- No **+** icons next to any of the property pages indicates that they do not have one or more of their own property pages.

## 2.5.4 Modifying / Entering Data

On the property pages, modify / enter data by clicking radio buttons, selecting from drop-down lists, and/or typing in data entry fields.

Some property pages provide panels that you must click in order to:

- Modify or adjust the displayed information.
- Display additional, modifiable information.

Note: You do not have to click the **OK** button after every modification or entry, or on every property page on which you have modified and/or entered data. Clicking **OK** just once, on any page, will save the modifications and entries made on all pages – and will exit Universal Configuration Manager (see Section 2.5.5 Saving Data.)

### Rules for Modifying / Entering Data

The following rules apply for the modification and entry of data:

- Quotation marks are not required for configuration values that contain spaces.
- Edit controls (used to input free-form text values) handle conversion of any case sensitive configuration values. Except where specifically noted, values entered in all other edit controls are case insensitive.

## 2.5.5 Saving Data

To save all of the modifications and entries made on all of the property pages, click the **OK** button at the bottom of any property page. The information is saved in the configuration file, and Universal Broker is automatically refreshed.

Clicking the **OK** button also exits the Universal Configuration Manager. (If you click **OK** after every modification, you will have to re-access Universal Configuration Manager.)

To exit Universal Configuration Manager without saving any of the modifications / entries made on all property pages, click the **Cancel** button.

## 2.5.6 Accessing Help Information

Universal Configuration Manager provides context-sensitive help information for the fields and panels on every Universal Products component options screen.

To access Help:

1. Click the question mark ( **?** ) icon at the top right of the screen.
2. Move the cursor (now accompanied by the **?**) to the field or panel for which you want help.
3. Click the field or panel to display Help text.
4. To remove the displayed Help text, click anywhere on the screen.

**Windows Vista, Windows Server 2008**

The Universal Configuration Manager’s context-sensitive help is a WinHelp file, which Windows Vista and Windows Server 2008 do not support.

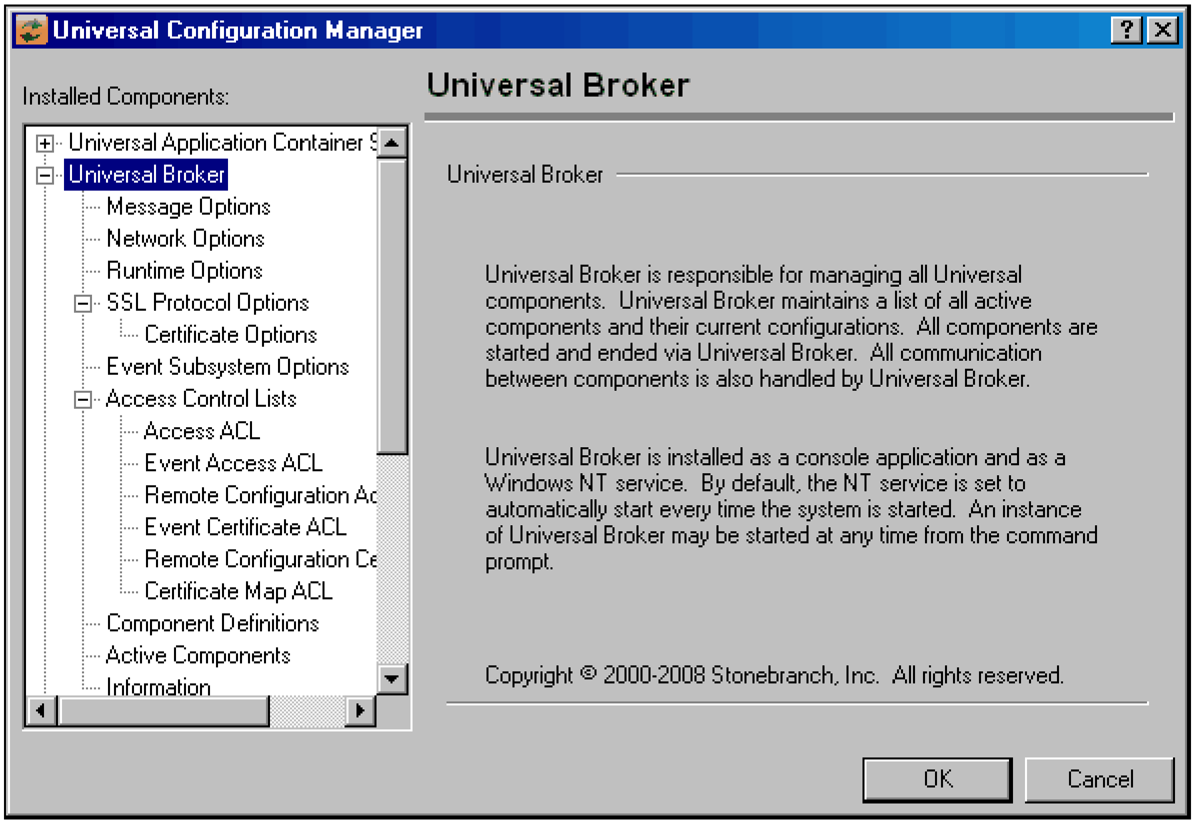
Microsoft offers the 32-bit WinHelp engine as a separate download from its website. If you require access to the Universal Configuration Manager’s context-sensitive help, simply download and install the WinHelp engine.

## 2.5.7 Universal Broker Installed Component

Figure 2.5 illustrates the Universal Configuration Manager screen for the Universal Broker.

The Installed Components list identifies all of the Universal Broker property pages.

The text describes the selected component, Universal Broker.



Universal Broker is responsible for managing all Universal components. Universal Broker maintains a list of all active components and their current configurations. All components are started and ended via Universal Broker. All communication between components is also handled by Universal Broker.

Universal Broker is installed as a console application and as a Windows NT service. By default, the NT service is set to automatically start every time the system is started. An instance of Universal Broker may be started at any time from the command prompt.

Copyright © 2000-2008 Stonebranch, Inc. All rights reserved.

Figure 2.5 Universal Configuration Manager - Universal Broker

## 2.6 Network Data Transmission

Distributed systems, such as Universal Command, communicate over data networks. All Stonebranch products communicate using the TCP/IP protocol. The UDP protocol is not used for any product data communication over a network.

The Universal Products suite can utilize one of two network protocols:

1. Secure Socket Layer version 3 (SSLv3) provides the highest level of security available. SSL is a widely used and accepted network protocol for distributed software applications that are required to address all aspects of secure data transfer on private and public networks.
2. Universal Products version 2 (UNVv2) legacy protocol is provided for backward compatibility with previous versions of Universal Products.

The following sections discuss each of the protocols.

In addition to the network protocol used to transmit data, Universal Products application protocol is discussed as well.

### 2.6.1 Secure Socket Layer Protocol

Universal Products implement the SSL protocol using the OpenSSL library or the IBM z/OS System SSL library, available on the z/OS operating system. The most recent SSL standard is version3. A subsequent version was produced changing the name to Transport Layer Security version 1 (TLSv1). TLSv1 is the actual protocol used by Universal Products. TLSv1 is more commonly referred to simply as SSL and the term SSL is used throughout the rest of this documentation to mean TLSv1 unless otherwise noted.

The SSL protocol addresses the major challenges of communicating securely over a potentially insecure data network. The following sections discuss the issue of data privacy and integrity, and peer authentication.

#### Data Privacy and Integrity

Data sent over the network that should remain private must be encrypted in a manner so that unauthorized persons cannot determine what the original data contained regardless of their level of expertise, access to network resources, amount of data captured, and amount of time they have. The only party that should be able to read the data is the intended recipient.

Data integrity must be protected from errors in transmission and malicious users. Data integrity checks insures that what was sent is exactly what is received by the recipient. Without integrity checks, there is no guarantee.

As data is transmitted over the network, it passes through media and hardware of unknown quality that may erroneously change bits of data without warning. Additionally, although data may be encrypted, there is nothing stopping a malicious person from changing the data while it is transmitted over the network. The changed data may or may not be detected by the recipient depending on what changed and how it is processed. It may be accepted as valid data, but the information it represents is now erroneous.

Encryption algorithms are used to encrypt data into an unreadable format. The encryption process is computationally expensive. There are a variety of encryption algorithms, some of which perform better than others. Some algorithms offer a higher level of security than others. Typically, the higher level of security requires more computational resources.

Message digest algorithms are used to produce a Message Authentication Code (MAC) that uniquely identifies a block of data. (Message digest algorithms are often referred to as MACs and can be used synonymously in most contexts.) The sender computes a MAC for the data being sent based on a shared secret key that the sender and receiver hold. The sender sends the data and the MAC to the receiver. The receiver computes a new MAC for the received data based on the shared secret key. If the two MACs are the same, data integrity is maintained. Otherwise, the data is rejected, as it has been modified.

The SSL standard defines a set of encryption and message digest algorithms, referred to as cipher suites, that ensure data privacy and data integrity. Cipher suites pair encryption algorithms with appropriate message digest algorithms; the two algorithms cannot be specified individually.

All Universal Products support a subset of the complete SSL cipher suites defined by the standard. The cipher suite name is formatted as an encryption algorithm abbreviation followed by the message digest algorithm abbreviation.

Table 2.5, below, identifies the supported cipher suites.

| Cipher Suite Name | Description |
|---|---|
| RC4-SHA | 128-bit RC4 encryption with SHA-1 message digest. |
| RC4-MD5 | 128-bit RC4 encryption with MD5 message digest. |
| AES256-SHA | 256-bit AES encryption with SHA-1 message digest. |
| AES128-SHA | 128-bit AES encryption with SHA-1 message digest. |
| DES-CBC3-SHA | 128-bit Triple-DES encryption with SHA-1 message digest. |
| DES-CBC-SHA | 128-bit DES encryption with SHA-1 message digest. |
| NULL-SHA | No encryption with SHA-1 message digest. |
| NULL-MD5 | No encryption with MD5 message digest. |

Table 2.5 Supported SSL Cipher Suites

Universal Products support one additional cipher suite name that is not part of the SSL protocol. The NULL-NULL cipher suite turns SSL off completely and instead uses the Universal Products Protocol (UNVv2) described below.

## Peer Authentication

When communicating with a party across a data network, how do you insure that the party you are communicating with (your peer) is who you believe? A common form of network attack is a malicious user representing themselves as another user or host.

Peer authentication insures that the peer is truly who they identify themselves as. Peer authentication applies to users, computer programs and hardware systems.

SSL uses X.509 certificates and public and private keys to identify an entity. An entity may be a person, a program, or a system. A complete description of X.509 certificates is beyond the scope of this documentation. Section 2.10 X.509 Certificates provides an overview to help get the reader oriented to the concepts, terminology and benefits.

For additional details, the following web site is recommended:

http://www.faqs.org/rfcs/rfc3280.html

## 2.6.2 Universal Products Protocol

The Universal Products protocol (UNVv2) is a proprietary protocol that securely and efficiently transports data across data networks. UNVv2 is used in Universal Products prior to version 3 and will be available in future versions.

UNVv2 addresses data privacy and integrity. It does not address peer authentication.

### Data Privacy and Integrity

Data privacy is insured with data encryption algorithms. UNVv2 utilizes 128-bit RC4 encryption for all data encryption.

Data integrity is insured with message digest algorithms. UNVv2 utilizes 128-bit MD5 MAC's for data integrity. UNVv2 referred to data integrity as data authentication.

Encryption and integrity may be enabled and disabled on an individual bases.

Encryption keys are generated using a proprietary key agreement algorithm. A new key is created for each and every network session.

## 2.6.3 Universal Products Application Protocol

Universal Product components use an application-layer protocol to exchange data messages. The protocol has the following characteristics:

- Low-Overhead
- Secure
- Extensible
- Configurable Options

The following sections refer to two categories of data transmitted by Universal Products:

- Control data (or messages) consists of messages generated by Universal Products components in order to communicate with each other. The user of the product has no access to the control data itself.
- Application data (or messages) consists of data that is transmitted as part of the requested work being executed. For example, standard input and output data of jobs Universal Command executes. The data is created by the job and read or written by Universal Command on behalf of the job.

### Low-Overhead

The protocol is lightweight, in order to minimize its use of network bandwidth. The product provides application data compression options, which reduces the amount of network data even further.

There are two possible compression methods:

- **ZLIB** method offers the highest compression ratios with highest CPU utilization.
- **HASP** method offers the lowest compression ratios with lowest CPU utilization.

Note: Control data is not compressed. Compression options are available for application data only.

### Secure

The protocol is secure. All control data exchanged between Universal Products components are encrypted with a unique session key and contain a MAC. The encryption prevents anyone from analyzing the message data and attempting to circumvent product and customer policies. Each session uses a different encryption key to prevent "play back" types of network attacks, where messages captured from a previous session are replayed in a new session. This applies to both network protocols: SSL and UNVv2.

The security features used in the control messages are not optional. They cannot be turned off. The security features are optional for application data sent over the network.

The data encryption options affect the application data being sent over the network. Special fields, such as passwords, are always encrypted. The encryption option cannot be turned off for such data.

## Extensible

The message protocol used between the Universal Products components is extensible. New message fields can be added with each new release without creating product component incompatibilities. This permits different component versions to communication with each other with no problems. This is a very important feature for distributed systems, since it is near impossible to upgrade hundreds of servers simultaneously.

New encryption and compression algorithms can be added in future releases without loosing backward compatibility with older releases. After a network connection is made, connection options are negotiated between the two Universal Products programs. The options negotiated include which encryption and compression algorithms are used for the session. Only algorithms that both programs implement are chosen in the negotiation process. The negotiation process permits two different program versions to communicate.

## 2.6.4 Configurable Options

The network protocol can be configured in ways that effect compression, encryption, code pages, and network delays.

The following configuration options are available on many of the Universal Products:

### CODE_PAGE

The CODE_PAGE option specifies the code page translation table used to translate network data from and to the local code page for the system on which the program is executing.

A codepage table is a text file that contains two columns. It maps local single byte character codes to two-byte UNICODE character codes.

Code pages are located in the product National Language Support (NLS) directory or library. New code pages may be created and added to the NLS directory or library. The CODE_PAGE option value is simply the name of the code page file without any file name extension if present.

### CTL_SSL_CIPHER_LIST

The CTL_SSL_CIPHER_LIST option specifies one or more SSL cipher suites that are considered accptable by the network to use for the control session, which is used for component internal communication. (Acceptable means what is determined as acceptable by your site.)

The SSL protocol uses cipher suites to specify the combination of encryption and message digest algorithms used for a session. An ordered list of acceptable cipher suites can be specified in a most to least order of preference.

An example cipher suite list is **`RC4-MD5,RC4-SHA,AES128-SHA`**. In this example, the **`RC4-MD5`** cipher suite is the most preferred and **`AES128-SHA`** is the least preferred.

When a manager and server first connect, they perform an SSL handshake. The handshake negotiates the cipher suite used for the session. The manager and server each have a cipher suite list and the first one in common is used for the session.

A list of acceptable cipher suites is useful in distributed software solutions that may cross many organizational and application boundaries, each of which has their own acceptable security requirements. Instead of having to choose one cipher suite for all distributed components, the software components can be configured with their own list of acceptable cipher suites, based on their local security requirements. When a high level of security is required, the higher CPU-consuming cipher suite is justified. When lower level of security is acceptable, a lower CPU-consuming cipher suite may be used. As long as the manager has both cipher suites in its list, it can negotiate either cipher suite with servers of different security levels.

### DATA_AUTHENTICATION

The DATA_AUTHENTICATION option specifies whether or not the network data is authenticated. Data authentication verifies that the data did not change from the point it was sent to the point it was received.

Data authentication also is referred to as a data integrity in this document.

Data authentication occurs for each message sent over the network. If a message fails authentication, the network session is terminated and both programs end with an error.

The DATA_AUTHENTICATION option is applicable to the UNVv2 proprietary protocol only. SSL always performs authentication.

### DATA_COMPRESSION

The DATA_COMPRESS option specifies that network data be compressed.

Compression attempts to reduce the amount of data to a form that can be decompressed to its original form. The compression ratio is the original size divided by the compressed size. The compression ratio value will depend on the type of data. Some data compress better than others.

Two methods of compression are available:

- ZLIB method provides the highest compression ratio with the highest use of CPU
- HASP method provides the lowest compress ratio with the lowest use of CPU.

Whether or not compression is used and which compression method is used depends on several items:

- Network bandwidth. If network bandwidth is small, compression may be worth the cost in CPU.
- CPU resources. If CPU is limited, the CPU cost may not be worth the reduced bandwidth usage.
- Data compression ratio. If the data does not compress well, it is probably not worth CPU cost. If the data ratio is high, the CPU cost may be worth it.

### DATA_ENCRYPTION

The DATA_ENCRYPTION option specifies whether or not network data is encrypted.

Encryption translates data into a format that prevents the original data from being determined. Decryption translates encrypted data back into its original form.

The type of encryption performed depends on the network protocol being used, SSL or UNVv2.

Data encryption does increase CPU usage. Whether or not encryption is used depends on the sensitivity of the data and the security of the two host systems and the data network between the hosts.

## DATA_SSL_CIPHER_LIST

The DATA_SSL_CIPHER LIST option specifies one or more SSL cipher suites that are acceptable to use for network communications on the data session, which is used for standard I/O file transmission.

(See CTL_SSL_CIPHER_LIST in this section.)

## DEFAULT_CIPHER

The DEFAULT_CIPHER option specifies the SSL cipher suite to use, since SSL protocol requires a cipher suite, if the DATA_ENCRYPTION option is set to NO (see Table 2.5 Supported SSL Cipher Suites). The default DEFAULT_CIPHER is NULL-MD5 (no encryption, MD5 message digest).

All SSL cipher suites have a message digest. The message digest ensures that the data sent is the data received. Without a message digest, it is possible for bits of the data packet to get changed without being noticed.

## KEEPALIVE_INTERVAL

The KEEPALIVE_INTERVAL option specifies how often, in seconds, a keepalive message (also commonly known as a heartbeat message) is sent between a manager and server. A keepalive message ensures that the network and both programs are operating normally. Without a keepalive message, error conditions can arise that place one or both programs in an infinite wait.

A keepalive message is sent from the server to the manager. If the server does not receive a keepalive acknowledgement from the manager in a certain period of time (calculated as either 2 x NETWORK_DELAY or the KEEPALIVE_INTERVAL, whichever is higher), the server considers the manager or network as unusable. How the server processes a keepalive timeout depends on what fault tolerant features are being used. If no fault tolerant features are being used, the server ends with an error. The manager expects to receive a keepalive message in a certain period of time (calculated as the KEEPALIVE_INTERVAL + 2 x NETWORK_DELAY).

## NETWORK_DELAY

The NETWORK_DELAY option provides the ability to fine tune Universal product's network protocol. When a data packet is sent over a TCP/IP network, the time it takes to reach the other end depends on many factors, such as, network congestion, network bandwidth, and the network media type. If the packet is lost before reaching the other end, the other end may wait indefinitely for the expected data. In order to prevent this situation, Universal Products time out waiting for a packet to arrive in a specified period of time. The delay option specifies this period of time.

NETWORK_DELAY specifies the maximum acceptable delay in transmitting data between two programs. Should a data transmission take longer than the specified delay, the operation ends with a time out error. Universal Products will consider a time out error as a network fault.

The default NETWORK_DELAY value is 120 seconds. This value is reasonable for most networks and operational characteristics. If the value is too small, false network time outs could occur. If the value is too large, programs will wait a long period of time before reporting a time out problem.

## SIO_MODE

The SIO_MODE option specifies whether the data transmitted over the network is processed as text data or binary data.

Text data is translated between the remote and local code pages. Additionally, end of line representations are converted

Text translation operates in two modes: direct and UCS. The default is direct. The direct translation mode exchanges code pages between Universal Products components to build direct translation tables. Direct translation is the fastest translation method when a significant amount (greater then 10K) of text data is transmitted. The code page exchange increases the amount of data sent over the network as part of the network connection negotiation. UCS translation does not require the exchange of code pages. For transactions that have little text data transmission, this is the fastest.

Binary data is transmitted without any data translation.

## 2.7 Universal Spool

Universal Broker maintains component server data in the Universal Spool, simply referred to as the spool. Universal Command stores user process standard I/O files (standard input, standard output, and standard error) in the spool when spooling has been activated in the Server.

Universal Spool is implemented as a set of databases. Universal Broker and Universal Command Server remove the database records automatically when they are no longer required. No database maintenance jobs are required.

Universal Broker and server components are the only programs that access the spool. No user access is required. The operating system's file system security should be used to prevent all access to the spool except for the broker and server. The broker and server require full control permissions to the spool in order to add, delete, update and read database files.

All standard I/O files written to the spool are encrypted to insure data privacy.

## 2.8 Universal Access Control List

Many Universal Products utilize the Universal Access Control List (UACL) feature as an extra layer of security to the services they offer. The UACL determines if a request is denied or allowed to continue and can assign security attributes to the request.

This section describes the UACL capabilities in general, non-component specific terms. See the appropriate component security sections for complete details on how a component utilizes the UACL feature.

The following Universal Product components use the UACL feature:

- Universal Broker uses UACLs to permit or deny TCP/IP connections based on the remote host IP address.

  See the UACL section for each operating system in this user guide for complete details.
- Universal Command Server uses UACLs to permit or deny Manager access based on the Managers IP address and user ID, and to control whether or not the Manager request requires user authentication.

  See the Universal Command 3.2.0 User Guide for complete details.
- Universal Control Server uses UACLs to permit or deny Manager access based on the Managers IP address and user ID, and to control whether or not the Manager request requires user authentication.

  See the Universal Control chapter of the Universal Products Utilities 3.2.0 User Guide for complete details.
- Universal Data Mover Server uses UACLs to permit or deny Manager access based on the Managers IP address and user ID.

  See the Universal Data Mover 3.2.0 User Guide for complete details.

## 2.8.1 UACL Configuration

The method used to configure UACL rules is platform dependent. The following sections discuss each of the methods.

### z/OS

All UACL rules are defined in library UNVCONF, member ACLCFG00. The Universal Broker allocates the UACL configuration data set to ddname UNVACL.

The UACL file syntax is the same as all other Universal Products z/OS configuration files. See Section 2.2.3 Configuration Files Syntax for details.

### UNIX

All UACL rules are defined in one file, **`uacl.conf`**. This file is required for products utilizing UACL rules; otherwise, the product will not start. The configuration file consists of zero or more UACL entries.

The UACL file syntax is the same as all other Universal UNIX configuration files. See Section 2.2.3 Configuration Files Syntax for details.

### Windows

All UACL rules are stored in the configuration file.

UACL entries for each component are maintained using the Universal Configuration Manager (see Section 2.5 Universal Configuration Manager).

### OS/400

All UACL rules are defined in file **`unvconf`** and member **`uacl`**. This file is required for products utilizing UACL rules, else the product will not start. The configuration file consists of zero or more UACL entries.

The UACL file is searched for in the same manner as all other product configuration files. See Section 2.2.2 Configuration Files for information on how configuration files are located.

The UACL file syntax is the same as all other Universal Products for OS/400 configuration files. See Section 2.2.3 Configuration Files Syntax for details.

### HP NonStop

All UACL rules are defined in one file, **`uaclcfg`**.  This file is required for products utilizing UACL rules, else the product will not start. The configuration file consists of zero or more UACL entries.

The UACL file is located within the same subvolume as all other product configuration files.

The UACL file syntax is the same as all other Universal HP NonStop configuration files. See Section 2.2.3 Configuration Files Syntax for details.

## 2.8.2 UACL Entries

UACL entries are composed of two parts: type and rule.

- Type identifies the Universal Products component for which the rule applies. For example, the Universal Broker product utilizes UACL rules of type **`ubroker_access`**.
- Rule defines the client's identity and the client's request for which the entry pertains and the security attributes it enforces.

UACL configuration file syntax is the same as all other configuration files, where the configuration file keyword corresponds to the UACL type part and the configuration file value corresponds to the UACL rule part.

The entire rule part of the UACL entry must be enclosed in quotation characters, not just a sub-field of the rule, if a space or tab is part of the value.

The correct syntax would be as follows:

```
ubroker_access "prod.host.name,allow"
```

For each client that connects and sends a request, Broker and Server components search UACL entries to find the best match for the client identity and the client request. Entries are searched in the order they are listed. The first entry found stops the search.

Note: There is no limit to the number of UACL entries that can be specified.

### Client Identification

Rule matching is based on the client identity and the client request.

There are two client identification methods:

1. X.509 certificate authentication.
2. Client IP address and reported user account.

## X.509 Certificate Authentication

X.509 certificates identify an entity. An entity can be a program, person, or host computer. When an X.509 certificate is authenticated, it authenticates that the entity is who it claims to be.

X.509 certificates are utilized in UACL entries by first mapping a client certificate to a UACL certificate identifier. The certificate identifier then is used in the UACL entries. A certificate identifier provides for:

1. Concise representation of certificates in UACL entries. There are a large number of certificate fields that may be used and many of the fields have lengthy, tedious naming formats. A certificate map only needs to be defined once and then the concise certificate identifier can be used in the UACL entries.
2. Mapping of one or more certificates to a single certificate identity. A group of entities that share a common security access level may be represented by one certificate identity reducing the number of UACL entries to maintain.

UACL certificate map entries are searched sequentially (that is, top to bottom) matching the client certificate to each entry until a match is found. The certificate map defines a set of X.509 certificate fields that may be used as matching criteria.

Table 2.6, below, defines the certificate map matching criteria.

| Criteria | Description |
|---|---|
| SUBJECT | Matches the X.509 **`subject`** field. The **`subject`** field is formatted as an X.501 Distinguished Name (DN). A DN is a hierarchical list of attributes referred to as Relative Distinguished Names (RDNs).<br>RDNs are separated with a comma (`,`) by default. If a different separator is required (perhaps one of the RDN values uses a comma), start the DN with the different separator character. Valid separators are slash (`/`), comma (`,`) and period (`.`).<br>Many RDN values can be used in a DN. Some of the most common values are:<br>• C Country name<br>• CN Common name<br>• L Locality<br>• O Organization<br>• OU Organizational Unit<br>• ST State<br>The RDN attributes must be listed in the same order as they are defined in the certificate to be considered matched.<br>A partial DN can be specified. All certificates that have a **`subject`** name that matches up to the last RDN are considered a match. This permits a group of certificates to be matched.<br>The RDN attribute values can include pattern matching characters. An asterisk (`*`) matches 0 or more characters and a question mark (`?`) matches one character.<br>Some example of SUBJECT values are:<br>• **`subject=”C=US,ST=Georgia,O=Acme,CN=Road Runner”`**<br>• **`subject=”C=US,ST=Georgia,O=Acme,CN=Road * ”`**<br>• **`subject=”C=US,ST=Georgia,O=Acme,CN=Road ?unner”`**<br>Whether an RDN value is case sensitive or not depends on the format in which the value is stored. The certificate creator has some control over which format is used. All formats except for **`printableString`** are case sensitive. |

| Criteria | Description |
|---|---|
| EMAIL | Matches the X.509 **emailAddress** attribute of the **subject** field and **rfc822Name** of the **subjectAltName** extension value. Both fields format the email address as an RFC 822 **addr-spec** in the form of **identifier@domain**.<br>The attribute values may include pattern matching characters. An asterisk (`*`) matches 0 or more characters and a question mark (`?`) matches one character.<br>Some example EMAIL values are:<br>• **email=user1@acme.com**<br>• **email=*@acme.com**<br>• **email=user?@acme.com**<br>RFC 822 names are not case sensitive. |
| HOSTNAME | Matches the following X.509 fields in the order listed:<br>1. **dNSName** of the **subjectAltName** extension value.<br>2. **commonName** (`CN`) RDN attribute of the **subject** field's DN value.<br>Some example HOSTNAME values are:<br>• **hostname=bigfish.acme.com**<br>• **hostname=*.acme.com**<br>The values are not case sensitive. |
| IP ADDRESS | Matches the X.509 **iPAddress** field of the **subjectAltName** extension value.<br>An example IPADDRESS value is:<br>• **ipaddress=10.20.30.40** |
| SERIAL NUMBER | Matches the X.509 **serialNumber** value.<br>The value can be specified in a hexadecimal format by prefixing the value with **0x** or **0X**, otherwise, the value is considered a decimal format. For example, the value **0x016A392E7F** would be considered a hexadecimal format.<br>An example SERIALNUMBER value is:<br>• **serialnumber=0x7a2d52cbae** |

Table 2.6 Certificate Map Matching Criteria

If a certificate map rule is found that matches the client certificate, the rule's identifier is assigned to the client's request. The certificate identifier is then used in matching certificate-based UACL entries.

Table 2.7, below, defines the certificate identifier field as used in UACL entries.

| Criteria | Description |
|---|---|
| CERTID | Matches the certificate identifier defined by the certificate map entry. The CERTID value has the following syntax:<br>• An asterisk (`*`) matches 0 or more characters and a question mark (`?`) matches one character. For example, **AB*M** matches **ABCDM** and **ABM**. **AB?M** matches **ABCM**, but not **ABCDM**.<br>• The comparison is case insensitive.<br>• Pattern matching characters, such as the asterisk and question mark, are included in the text to be matched by prefixing them with a forward slash (`/`) character. For example, **A/*B** matches **A*B**. **A//B** matches **A/B**. |

Table 2.7 Certificate Identifier Field

## Client IP Address Identification

TCP/IP provides a method to obtain a client's IP address. The IP address typically identifies the host computer on which the client is executing. There are exceptions to this though. Networks can be configured with Network Address Translation (NAT) systems between the client and the Broker that hides the client's IP address. In addition to the client IP address, Universal Products clients provide a user account name with which they are executing that is used to further refine the client's identity.

UACL entries are searched matching the client's IP address and user account to each entry until a match is found.

Table 2.8, below, defined possible matching criteria for IP address and user account client identification.

| Criteria | Description |
|---|---|
| HOST | Matches the TCP/IP address of the remote user.<br>The HOST value has the following syntax:<br>• Dotted numeric form of an IP address. For example, **`10.20.30.40`**.<br>• Dotted numeric prefix of the IP addresses. For example, **`10.20.30.`** matches all IP addresses starting with **`10.20.30`**. The last dot (**`.`**) is required.<br>• A `net/mask` expression. For example, **`131.155.72.0/255.255.254.0`** matches IP address range **`131.155.72.0`** through **`131.155.73.255`**. The `mask` and the host value are AND'ed together. The result must match `net`.<br>Note: Contact your network administrator for calculation of the correct net / mask expression.<br>• Host name for an IP address. For example, **`sysa.abc.com`**.<br>• Host name suffix for a range of IP addresses. For example, **`.abc.com`** matches all host names ending with **`abc.com`**, such as, **`sysa.abc.com`**. The first dot (**`.`**) is required.<br>• A value of **ALL** matches all IP addresses. The value must be uppercase. |
| REMOTE_USER | Matches the user name with which the remote user is executing as on the remote system.<br>The REMOTE_USER value has the following syntax:<br>• An asterisk (`*`) matches 0 or more characters and a question mark (`?`) matches one character. For example, **`AB*M`** matches **`ABCDM`** and **`ABM`**. **`AB?M`** matches **`ABCM`** but not **`ABCDM`**.<br>• Control code `/c` switches off case-sensitivity and `/C` switches on case-sensitivity matching. The default is on. For example, **`/cABC`** matches **`abc`**. **`/ca/Cbc`** matches **`Abc`** but not **`ABC`**.<br>• Pattern matching characters, such as the asterisk and question mark, are included in the text to be matched by prefixing them with a forward slash (`/`) character. For example, **`A/*B`** matches **`A*B`**. **`A//B`** matches **`A/B`**. |

Table 2.8 Client IP Address - Matching Criteria

## Request Identification

In addition to the client identity being used to search for UACL entries, the client's request may be part of the matching criteria. The exact request fields used is dependent on the component's UACL entry type.

Table 2.9, below, lists a complete set of the request fields that are possible. See each component's UACL entry definitions for further details.

| Criteria | Description |
|---|---|
| LOCAL_USER | Matches the local user name with which the remote user is requesting to execute as on the local host. LOCAL_USER value has the following syntax:<br>• An asterisk (`*`) matches 0 or more characters and a question mark (`?`) matches one character. For example, **AB*M** matches **ABCDM** and **ABM**. **AB?M** matches **ABCM** but not **ABCDM**.<br>• Control code `/c` switches off case-sensitivity and `/C` switches on case-sensitivity matching. The default is on. For example, **/cABC** matches **abc**. **/ca/Cbc** matches **Abc** but not **ABC**.<br>• Pattern matching characters, such as the asterisk and question mark, are included in the text to be matched by prefixing them with a forward slash (`/`) character. For example, **A/*B** matches **A*B**. **A//B** matches **A/B**.<br>• Variable name **$RMTUSER** can be included in the value. The variable name itself is not case sensitive. **$RMTUSER** and **$rmtuser** are the same. The **$RMTUSER** variable value is the user name with which the remote user is executing. It is the same value used in matching the REMOTE_USER field.<br>A space character delimits the variable name, or it can be enclosed in parentheses (for example, **$(RMTUSER)**), in which case it is delimited by the right parenthesis. This is useful if it is immediately followed by text.<br>For example, if the remote user name is **TOM**, a LOCAL_USER value of **$RMTUSER** will match if the local user name requested is also **TOM**. A LOCAL_USER value of **$(RMTUSER)01** will match if the local user name requested is **TOM01**.<br>**Windows**<br>The LOCAL_USER value is not case sensitive since Windows user account names are not. |
| REQUEST_TYPE | Matches the type of request a Universal Command Manager is requesting. The REQUEST_TYPE value has the following syntax:<br>• An asterisk ( ***** ) matches 0 or more characters and a question mark ( **?** ) matches one character. For example, **AB*M** matches **ABCDM** and **ABM**. **AB?M** matches **ABCM** but not **ABCDM**.<br>• The comparison is case insensitive.<br>• Pattern matching characters, such as the asterisk and question mark, are included in the text to be matched by prefixing them with a forward slash (`/`) character. For example, **A/*B** matches **A*B**. **A//B** matches **A/B**. |

| Criteria | Description |
|---|---|
| REQUEST_NAME | The REQUEST_NAME field matches the name of a Universal Command Manager is request. The REQUEST_NAME value has the following syntax:<br>• An asterisk ( ***** ) matches 0 or more characters and a question mark ( **?** ) matches one character. For example, **AB*M** matches **ABCDM** and **ABM**. **AB?M** matches **ABCM** but not **ABCDM**.<br>• Case sensitivity depends on the REQUEST_TYPE and the operating system on which the Universal Command Server is executing. See the Server's Security section for the operating system in question.<br>• Control code **/c** switches off case-sensitivity and **/C** switches on case-sensitivity matching. The default is on. For example, **/cABC** matches **abc**. **/ca/Cbc** matches **Abc** but not **ABC**.<br>• Control code **/s** normalizes spaces and **/S** does not normalize spaces. Space normalization removes preceding and trailing spaces as well as reduce consecutive multiple spaces to a single space. The default is no space normalization. For example, **/sa b c** matches **a   b      c**. **/Sa b c** matches **a b c** but not **a      bc**.<br>• Pattern matching characters, such as the asterisk and question mark, are included in the text to be matched by prefixing them with a forward slash ( **/** ) character. For example, **A/*B** matches **A*B**. **A//B** matches **A/B**. |

Table 2.9 Request Fields

## Certificate-Based and Non Certificate-Based UACL Entries

Universal Products components that support X.509 certificates define their UACL entries in two varieties:

1. Certificate-based entries
2. Non certificate-based entries

The two entry types are distinguished by their name. For example, **`cmd_cert_access`** is the certificate-based form of the entry and **`ucmd_access`** is a non certificate-based entry . All entries follow the same format.

Certificate-based UACL entries are searched under the following conditions:

- Client provides an X.509 certificate that matches a certificate map entry.

Non certificate-based UACL entries are searched under the following conditions:

- Client provides an X.509 certificate and no certificate map entry matches.
- Client does not provide an X.509 certificate.

Either the certificate-based UACL entries or the non certificate-based UACL entries are searched, but not both.

## 2.9 Message and Audit Facilities

All Universal Products have the same message facilities. Messages - in this context - are text messages written to a console, file, or system log that:

1. Document the actions taken by a program.
2. Inform users of error conditions encountered by a program.

This section describes the message and audit facilities that are common to all Universal Products. (See the individual Universal Product documentation for additional details.)

### 2.9.1 Message Types

There are six types (or severity levels) of Universal Products messages. (The severity level is based on the type of information provided by those messages.)

1. Audit messages document the configuration options used by the program's execution and resource allocation details. They provide complete description of the program execution for auditing and problem resolution.
2. Informational messages document the actions being taken by a program. They help determine the current stage of processing for a program. Informational messages also document statistics about data processed.
3. Warning messages document unexpected behavior that may cause or indicate a problem.
4. Error messages document program errors. They provide diagnostic data to help identify the cause of the problem.
5. Alert messages document a notification that a communications issue, which does not disrupt the program or require action, has occurred.
6. Diagnostic messages document diagnostic information for problem resolution.

The MESSAGE_LEVEL configuration option in each Universal Product component lets you specify which messages are written (see Section 2.9.3 Message Levels).

## 2.9.2 Message ID

Each message is prefixed with a message ID that identifies the message.

The message ID format is **`UNVnnnnl`**, where:

- **`nnnn`** is the message number.
- **`l`** is the message severity level:
  - **`A`** (Audit)
  - **`I`** (Informational)
  - **`W`** (Warning)
  - **`E`** (Error)
  - **`T`** (alerT)
  - **`D`** (Diagnostic)

Note: The Universal Products 3.2.0 Messages and Codes document identifies all messages numerically, by product, using the **`nnnn`** message number.

## 2.9.3 Message Levels

Each Universal Product includes a MESSAGE_LEVEL configuration option that lets you select which levels (that is, severity levels) of messages are to be written.

- *Audit* specifies that all audit, informational, warning, and error messages are to be written.
- *Informational* specifies that all informational, warning, and error messages are to be written.
- *Warning* specifies that all warning and error messages are to be written.
- *Error* specifies that all error messages are to be written.
- *Trace* specifies that a trace file is created, to which data used for program analysis will be written. The trace file name and location are Universal Product dependent (see the appropriate Universal Product documentation for details).

  (Trace should be used only at the request of Stonebranch, Inc. Customer Support.)

Note: Diagnostic and Alert messages are always written, regardless of the level selected in the MESSAGE_LEVEL option.

## 2.9.4 Message Destinations

The location to which messages are written is the message destination.

Some Universal Products have a MESSAGE_DESTINATION configuration option that specifies the message destination. If a program is used only from the command line or batch job, it may have only one message destination, such as standard error. Valid destination values will depend on the host operating system.

**z/OS**

Universal Products on z/OS run as batch jobs or started tasks. Batch jobs do not provide the MESSAGE_DESTINATION option. All messages are written to the SYSOUT ddname.

Started task message destinations are listed in the table below.

| Destination | Description |
|---|---|
| LOGFILE | Messages are written to ddname UNVLOG.<br>All messages written to log files include a date and time stamp and the program's USS process ID. |
| SYSTEM | Messages are written to the console log as WTO messages. |

**UNIX**

Message destinations are listed in the table below.

| Destination | Description |
|---|---|
| STDERR | Messages are written to standard error. This destination is most useful for console commands. |
| LOGFILE | Messages are written to a log file. Not all programs provide this destination.<br>The recommended directory for log files is **`/var/opt/universal/log`**. This can be changed with the LOG_DIRECTORY option. All messages written to log files include a date and time stamp and the program's process ID. |
| SYSTEM | Messages are written to the **`syslog`** daemon. Not all programs provide this destination.<br>Universal programs that execute as daemons write to the **`syslog`**'s daemon facility. All messages include the programs process ID. If an error occurs writing to the **`syslog`**, the message is written to the system console. |

### Windows

Message destinations are listed in the table below.

| Destination | Description |
|---|---|
| STDERR | Messages are written to standard error. This destination is most useful for console commands. |
| LOGFILE | Messages are written to a log file. Not all programs provide this destination.<br>Log files are written to product specific log directories, which can be modified with the LOG_DIRECTORY option. All messages written to log files include a date and time stamp and the program's process ID. |
| SYSTEM | Messages are written to the Windows Application Event Log. |

### OS/400

Message destinations are listed in the table below.

| Destination | Description |
|---|---|
| STDERR | Messages are written to standard error. A batch job's standard error file is allocated to the print file QPRINT. |
| LOGFILE | Messages are written to the job's job log. |
| SYSTEM | Messages are written to the system operator message queue QSYSOPR. |

### HP NonStop

Message destinations are listed in the table below.

| Destination | Description |
|---|---|
| STDERR | Messages are written to standard error. |
| LOGFILE | Messages are written to a log file. Not all programs provide this destination.<br>Log files are written the **`$SYSTEM.UNVLOG`** subvolume. All messages written to log files include a date and time stamp and the program's process ID. |

## 2.10 X.509 Certificates

A certificate is an electronic object that identifies an entity. It is analogous to a passport in that it must be issued by a party that is trusted by all who accept the certificate. Certificates are issued by trusted parties called Certificate Authorities (CA's). For example, VeriSign Inc. is a CA that most parties trust. We all have faith that a trusted CA takes the necessary steps to confirm the identity of a user before issuing the user a certificate.

Certificate technology is based on public/private key technology. There are a few different types of public/private keys: RSA, DH, and DSS. As their name denotes, the private key must be kept private, like a password. The public key can be given to anyone or even published in a newspaper.

A property of public/private keys is that data encrypted with one can be decrypted only with the other. Therefore, if someone wants to send you a secret message, they encrypt the data with your public key, which everyone has. However, since you are the only one with your private key, you are the only one who can decrypt it. If you want to send someone message, such as a request for $100,000 purchase, you can "sign" it with your private key.

Note: Signing does not encrypt the data. Once a person receives your request, that person can verify it is from you by verifying your electronic signature with your public key.

A certificate ties a statement of identity to a public key. Without the public key, the certificate is meaningless. Possession of a certificate alone does not prove your identity. You must have the corresponding private key. The two together prove your identity to any third party that trusts the CA that issued your certificate. This is a key point; if you do not trust the CA that signed a certificate, you cannot trust the certificate.

Since certificates originally were designed to be used for internet authentication, global directory technologies were developed to make them available via the internet. This directory technology is known as X.500 Directory Access Protocol. Later LDAP was introduced by Netscape to make it Lightweight Directory Access Protocol.

X.500 divides the world into a hierarchical directory. A person's identity is located by traversing down the hierarchy until it reaches the last node. Each node in the hierarchy consists of a type of object, such as a country, state, company, department, or name.

## 2.10.1 Sample Certificate Directory

Figure 2.6, below, provides a sample diagram of a small X.500 directory.

```
                                C=US
                                  |
                   +-----------------------------+
                   |                             |
              ST=Georgia                    ST=Florida
                   |                             |
           O=Stonebranch, Inc.              O=Acme, Inc.
                   |                             |
               OU=Sales                      OU=Sales
                   |                             |
        +---------------------+                CN=Joe Buck
        |                     |
CN=Joe Black            CN=Joe Simpson
```

Figure 2.6 X.500 Directory (Sample)

The keywords listed on each node are referred to as a Relative Distinguished Name (RDN). A person is identified by a Distinguished Name (DN). The DN value for Joe Black is **`C=US/ST=Georgia/O=Stonebranch, Inc./OU=Sales/CN=Joe Black`**.

A certificate is composed of many fields and possible extensions. Many of the most popular fields are specified as X.500 DN values.

## 2.10.2 Sample X.509 Certificate

Figure 2.7, below, illustrates a sample X.509 version 3 certificate for Joe Buck at the Acme corporation.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            01:02:03:04:05:06:07:08
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=US, ST=Florida, O=Acme, Inc., OU=Security, CN=CA
Authority/emailAddress=ca@acme.com
        Validity
            Not Before: Aug 20 12:59:55 2004 GMT
            Not After : Aug 20 12:59:55 2005 GMT
        Subject: C=US, ST=Florida, O=Acme, Inc., OU=Sales, CN=Joe Buck
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:be:5e:6e:f8:2c:c7:8c:07:7e:f0:ab:a5:12:db:
                    fc:5a:1e:27:ba:49:b0:2c:e1:cb:4b:05:f2:23:09:
                    77:13:75:57:08:29:45:29:d0:db:8c:06:4b:c3:10:
                    88:e1:ba:5e:6f:1e:c0:2e:42:82:2b:e4:fa:ba:bc:
                    45:e9:98:f8:e9:00:84:60:53:a6:11:2e:18:39:6e:
                    ad:76:3e:75:8d:1e:b1:b2:1e:07:97:7f:49:31:35:
                    25:55:0a:28:11:20:a6:7d:85:76:f7:9f:c4:66:90:
                    e6:2d:ce:73:45:66:be:56:aa:ee:93:ae:10:f9:ba:
                    24:fe:38:d0:f0:23:d7:a1:3b
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Subject Alternative Name:
                email:joe.buck@acme.com
    Signature Algorithm: md5WithRSAEncryption
        a0:94:ca:f4:d5:4f:2d:da:a8:6d:e3:41:6e:51:83:57:b3:b5:
        31:95:32:b6:ca:7e:d1:4f:fb:01:82:db:23:a0:39:d8:69:71:
        31:9c:0a:3b:ce:f6:c6:e2:5c:af:23:f0:d7:ee:87:3e:8a:7b:
        40:03:39:64:a1:8c:29:7d:5b:99:93:fa:23:19:e1:e4:ac:4d:
        13:0f:de:ad:51:27:e3:4e:4b:9f:40:4c:05:fd:f2:82:09:3e:
        46:05:f0:ad:cc:f7:78:25:3e:11:f8:ca:b6:df:f7:37:57:9b:
        63:00:d0:b5:b5:18:ec:38:73:d2:85:a3:c7:24:21:47:ee:f2:
        8c:0d
```

Figure 2.7 X.509 Version3 Certificate (Sample)

Note: The contents of a certificate file does not look like the information in Figure 2.7, which is produced by a certificate utility using the certificate file as input. Certificates can be saved in multiple file formats, so their file contents will look very different.

## Certificate Fields

A certificate is composed of many fields.

Figure 2.8, below, describes the main fields.

| Field or Section | Description |
|---|---|
| Version | X.509 certificates come in two versions: 1 and 3. |
| Serial Number | CA is required to provide each certificate it issues a unique serial number. The serial number is not unique for all certificates, only for the certificates issued by each CA. |
| Issuer | DN name of the CA that issued the certificate. |
| Validity | Starting and ending date for which this certificate is valid. |
| Subject | Identity of the certificate. A certificate may identify a person or a computer. In this case, the certificate identifies Joe Buck in the Sales organization of the Acme company in the state of Florida in the United States. |
| Public Key | Public key associated with the certificate identity. |
| X509v3 Extensions | X.509 version 3 introduced this section so that additional certificate fields may be added. In this case, the identity's email address is included as a Subject Alternative Name field.<br>Note: This section is not available in X.509 version 1. |
| Signature | CA's digital signature of the certificate. |

Figure 2.8 Certificate Fields

## 2.10.3 SSL Peer Authentication

The SSL protocol utilizes X.509 certificates to perform peer authentication. For example, a Universal Command Manager may want to authenticate that it is connected to the correct Broker.

Peer authentication is performed by either one or both of the programs involved in the network session. If a Manager wishes to authenticate the Broker to which it connects, the Broker will send its certificate to the Manager for the Manager to authenticate. Should the Broker wish to authenticate the Manager, the Manager sends its certificate to the Broker.

Certificate authentication is performed in the following steps:

1. Check that the peer certificate is issued by a trusted CA.
2. Check that the certificate has not been revoked by the CA.
3. Check that the certificate identifies the intended peer.

If a step fails, the network session is terminated immediately.

### Certificate Verification

The Universal Product must be configured with a list of trusted CA certificates. When a peer certificate is received, the trusted CA certificates are used to verify that the peer certificate is issued by one of the trusted CA's.

The trusted CA certificate list must be properly secured so that only authorized accounts have update access to the list. Should the trusted CA list become compromised, there is a possibility that an untrusted CA certificate was added to the list.

The CA certificate list configuration option is CA_CERTIFICATES. It specifies a PEM formatted file that contains one or more CA certificates used for verification.

Should a peer certificate not be signed by a trusted CA, the session is immediately terminated.

### Certificate Revocation

After a certificate is verified to have come from a trusted CA, the next step is to check if the CA has revoked the certificate. Since a certificate is held by the entity for which it identifies, a CA cannot take a certificate back after it is issued. Therefore, when a CA needs to revoke a certificate for some reason, it issues a list of revoked certificates referred to as the Certificate Revocation List (CRL). A program that validates certificates must have access to the latest CRL issued by the CA.

The CERTIFICATE_REVOCATION_LIST configuration option specifies the PEM-formatted file that contains the CRL. This option is available in all Universal Products that utilize certificates.

## Certificate Identification

When a certificate has been validated as being issued by a trusted CA, and not revoked by the CA, the next step is to check that it identifies the intended peer.

A Universal Product Manager validates a Broker certificate by the Broker host name or IP address or the certificate serial number. The VERIFY_HOST_NAME configuration option is used to specify the host name or IP address that is identified in the Broker certificate. Each certificate signed by a CA must have a unique serial number for that CA. The VERIFY_SERIAL_NUMBER option is used to specify the serial number in the Broker certificate.

Should certificate identification fail, the session is immediately terminated.

Universal Brokers work differently than the Managers. A Broker maps a peer certificate to a certificate ID. The certificate map definitions are part of the Universal Access Control List (UACL) definitions. At that point, the certificate ID is used by UACL definitions to control access to Broker and Server services.

## Certificate Support

Many certificate authority applications, also known as Public Key Infrastructure (PKI) applications, are available. Universal Products should be able to utilize any certificate in a PEM format file. PEM (Privacy Enhanced Mail) is a common text file format used for certificates, private keys, and CA lists.

Universal Products support X.509 version 1 and version 3 certificates.

Although implementing a fully featured PKI infrastructure is beyond the scope of Universal Products and this documentation, some assistance is provided using the OpenSSL toolkit (http://www.openssl.org).

Universal Products on most of the supported platforms utilize the OpenSSL toolkit for its SSL and certificate implementation. OpenSSL is delivered on most UNIX distributions and Windows distributions are available on the OpenSSL web site.

Universal Products supports z/OS System SSL on the IBM z/OS operating system as well as OpenSSL. System SSL interfaces directly with the RACF security product for certificate access. All certificates, CA and user certificates, and private keys must be stored in the RACF database to use System SSL.

The Universal Product suite includes an X.509 certificate utility, Universal Certificate, to create certificates for use in the Universal Product suite. See the Universal Certificate chapter in the Universal Products Utilities 3.2.0 User Guide for details.

# Chapter 3 Universal Broker for z/OS

## 3.1 Overview

This chapter provides information on using the Universal Broker, specific to the z/OS operating system.

It describes how to execute and configure the Universal Broker.

### 3.1.1 Environment

Universal Broker for z/OS executes as a started task.

The UBROKER program utilizes the z/OS UNIX System Services environment.

## 3.2 Usage

### 3.2.1 Start Universal Broker

To start Universal Broker, execute the **START** console command:

```
START UBROKER[,UPARM='options']
```

Broker options are described in Section 3.3.3 Configuration Options.

### 3.2.2 Stop Universal Broker

To stop Universal Broker, execute the **STOP** console command:

```
STOP UBROKER
```

## 3.2.3 JCL Procedure

Figure 3.1, below, illustrates the JCL procedure for the Universal Broker started task. **UBROKER** is the member name of this JCL procedure in the Universal Products sample library (**SUNVSAMP**).

```
//UBROKER  PROC HLQ=#SHLQ.UNV,
//             DBHLQ=#PHLQ.UNV,
//             PHLQ=#PHLQ.UNV,
//             SAPRFC=USPRFC00,
//             RGN=50M,
//             UPARM=,
//             LEPARM=
//*
//S1       EXEC PGM=UBROKER,REGION=&RGN,
//             PARM='ENVAR(TZ=EST5EDT) &LEPARM/&UPARM'
//STEPLIB  DD  DSN=&HLQ..SUNVLOAD,
//             DISP=SHR
//UNVCONF  DD  DSN=&PHLQ..UNVCONF,
//             DISP=SHR
//UNVCOMP  DD  DSN=&PHLQ..UNVCOMP,
//             DISP=SHR
//UNVRFC   DD  DSN=&PHLQ..UNVCONF(&SAPRFC),
//             DISP=SHR
//UNVNLS   DD  DSN=&HLQ..SUNVNLS,
//             DISP=SHR
//UNVTMPL  DD  DSN=&HLQ..SUNVTMPL,
//             DISP=SHR
//UNVCREF  DD  DSN=&PHLQ..UNVCREF,
//             DISP=SHR
//UNVDB    DD  DSN=&DBHLQ..UNVDB,
//             DISP=SHR
//UNVSPOOL DD  DSN=&DBHLQ..UNVSPOOL,
//             DISP=SHR
//UNVTRACE DD  DSN=&PHLQ..UNVTRACE,
//             DISP=SHR
//UNVTRMDL DD  DSN=&PHLQ..MDL,
//             DISP=SHR
//UNVLOG   DD  SYSOUT=*,HOLD=YES
//SYSPRINT DD  SYSOUT=*,HOLD=YES     -- standard output
//SYSOUT   DD  SYSOUT=*,HOLD=YES     -- standard error
//CEEDUMP  DD  SYSOUT=*,HOLD=YES     -- LE dumps
//SYSUDUMP DD  SYSOUT=*,HOLD=YES     -- system dumps
//SYSIN    DD  DUMMY                 -- standard input
```

Figure 3.1 Universal Broker for z/OS – JCL procedure

## 3.2.4 DD Statements used in JCL Procedure

Table 3.1, below, describes the DD statements used in the Universal Broker for z/OS JCL procedure illustrated in Figure 3.1.

| ddname | DCB Attributes | Mode | Description |
|---|---|---|---|
| STEPLIB | DSORG=PO, RECFM=U | input | Universal Products load library containing the program being executed. |
| UNVCONF | DSORG=PS, RECFM=(F, FB, V, VB) | input | Universal Broker configuration member. |
| UNVCOMP | DSORG=PO, RECFM=(F, FB, V, VB) | input | Universal Broker component definition PDS. |
| UNVRFC | DSORG=PS, RECFM=(F, FB, V, VB) | input | SAP RFC file used by Universal Connector. |
| UNVNLS | DSORG=PO, RECFM=(F, FB, V, VB) | input | Universal Products national language support library. Contains message catalogs and code page translation tables. |
| UNVTMPL | DSORG=PO, RECFM=(V, VB) | input | Universal Products configuration template library. |
| UNVCREF | DSORG=PO, RECFM=(F, FB, V, VB) | input | Universal Command Server command reference PDS. |
| UNVDB | DSNTYPE=HFS | input, output | Universal Broker database. Note: This ddname is not used if zFS data sets are used instead of HFS data sets. |
| UNVSPOOL | DSNTYPE=HFS | input, output | Universal Products spool database. Note: This ddname is not used if zFS data sets are used instead of HFS data sets. |
| UNVTRACE | DSORG=PO, RECFM=(F, FB, V, VB), LRECL=256 or above. | output | Universal Products trace PDS. This ddname is used only if UNVTRMDL is not defined. |
| UNVTRMDL | DSORG=PS, RECFM=(F,FB,V,VB), LRECL=256 or above. | output | Universal Products trace model data set. The data set name is used as the high-level qualifier of the dynamically allocated trace data sets. |
| UNVLOG | DSORG=PS, RECFM=(F,FB,V,VB), LRECL=256 or above. | output | Universal Broker message destination ddname when option MESSAGE_DESTINATION value is LOGFILE. |
| SYSPRINT | DSORG=PS, RECFM=(F, FB, V, VB) | output | Standard output file for the UBROKER program. |
| SYSOUT | DSORG=PS, RECFM=(F, FB, V, VB) | output | Standard error file for the UBROKER program. |
| SYSIN | DSORG=PS, RECFM=(F, FB, V, VB) | input | Standard input file for the UBROKER program. |

Table 3.1 Universal Broker for z/OS – DD Statements in JCL Procedure

## 3.3 Configuration

This section describes the Universal Broker for z/OS configuration options.

See Section 2.2.1 Configuration Methods for details on Universal Products configuration methods.

### 3.3.1 Configuration Input

Universal Broker reads configuration options only from the Universal Broker configuration file.

See Section 2.2.1 Configuration Methods for complete details on configuration methods and command input for Universal Products.

### 3.3.2 Configuration File

The configuration file provides the simplest method of specifying configuration values that will not change with each invocation.

The Universal Broker configuration file is allocated to ddname **UNVCONF**.

See Section 2.2.1 Configuration Methods for details on Universal Products configuration methods.

## 3.3.3 Configuration Options

Table 3.2, below, identifies all of the Universal Broker for z/OS configuration options.

Each Option Name is a link to detailed information about that option in the Universal Broker 3.2.0 Reference Guide.

| Option Name | Description |
|---|---|
| ACTIVITY_MONITORING | Specification for whether or not product activity monitoring events are generated. |
| CA_CERTIFICATES | Path to PEM formatted trusted CA X.509 certificates. |
| CERTIFICATE | Path to Broker's PEM formatted X.509 certificate. |
| CERTIFICATE_REVOCATION_LIST | Path to PEM formatted CRL. |
| CODE_PAGE | Text translation code page. |
| CTL_SSL_CIPHER_LIST | SSL cipher list for the control sessions. |
| DNS_CACHE_TIMEOUT | Time-out for DNS cache. |
| EVENT_GENERATION | Events to be generated as persistent event records. |
| MESSAGE_DESTINATION | Location where messages are written. |
| MESSAGE_LANGUAGE | Language of written messages. |
| MESSAGE_LEVEL | Level of messages written. |
| MONITOR_EVENT_EXPIRATION | Duration of a monitoring event record in the Universal Broker local UES database. |
| MOUNT_POINT | HFS or zFS database mount directory. |
| MOUNT_POINT_MODE | HFS or zFS permission mode for MOUNT_POINT. |
| PERSISTENT_EVENT_EXPIRATION | Duration of a persistent event record in the Universal Broker local UES database. |
| PRIVATE_KEY | Path to Broker's PEM formatted RSA private key. |
| PRIVATE_KEY_PWD | Password for the Broker's PRIVATE_KEY. |
| RUNNING_MAX | Maximum number of simultaneous components. |
| SAF_KEY_RING | SAF certificate key ring name. |
| SAF_KEY_RING_LABEL | SAF certificate key ring label. |
| SERVICE_BACKLOG | Service interface backlog size for pending connection requests. |
| SERVICE_IP_ADDRESS | TCP/IP address on which the Broker listens. |
| SERVICE_PORT | TCP/IP port number on which the Broker listens. |
| SMF_EXIT_LOAD_LIBRARY | UNVACTRT SMF exit load library. |
| SSL_IMPLEMENTATION | SSL implementation. |
| SYSTEM_ID | Broker running on a system (O/S image). |
| TMP_DIRECTORY | z/OS UNIX directory name for temporary files. |

| Option Name | Description |
|---|---|
| TRACE_FILE_LINES | Maximum number of lines written to the trace file. |
| TRACE_TABLE | Memory trace table specification. |
| UCMD_STC_SUPPORT | Support for Universal Command started tasks. |
| UNIX_DB_DATA_SET | HFS or zFS data set used for the Universal Broker's databases. |
| UNIX_SPOOL_DATA_SET | HFS or zFS data set used for the Universal Broker's spool. |

Table 3.2 Universal Broker for z/OS - Configuration Options

## 3.3.4 Configuration Refresh

Refreshing Universal Broker configuration directs Universal Broker to read its configuration data and update its current configuration settings.

Some configuration settings can be updated by refreshing Universal Broker while it is running. Other settings can be updated only when Universal Broker is refreshed by being recycled (stopped and restarted).

For information on the different methods of refreshing Universal Broker, and a list of options that is updated via each method, see Section 2.3 Universal Broker Configuration Refresh).

### Configuration Refresh via Universal Cntrol

One method of refreshing Universal Broker is via the Universal Control Manager **REFRESH** command.

For example, the z/OS version of the Universal Control Manager REFRESH command is:

```
//jobname  JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*********************************************************************
//* (c) Copyright 2001-2008, Stonebranch, Inc.  All rights reserved.
//*
//* Stonebranch, Inc.
//* Universal Control
//*
//* Description
//* -----------
//* This sample demonstrates the use of the UCTL program to refresh
//* a running component on host dallas.
//*
//* Make the following modifications as required by your local
//* environment:
//*
//* - Modify the JOB statement as appropriate.
//* - Change all '#HLQ' to the high-level qualifier of the
//*   Universal Command data sets.
//* - If not already done, modify the JCL procedure UCTLPRC
//*   as required by your local environment.
//********************************************************************
```

Figure 3.2  Universal Broker for z/OS – REFRESH Command Example (1 of 2)

```
//*
//         JCLLIB ORDER=#HLQ.UNV.SUNVSAMP
//*
//STEP1    EXEC UCTLPRC
//SYSIN    DD  *
 -refresh -host dallas
/*
```

Figure 3.3  Universal Broker for z/OS – REFRESH Command Example (2 of 2)

This example refreshes the Broker configuration on host **dallas**.

(See the Universal Products Utilities 3.2.0 User Guide for details on the Universal Control.)

The REFRESH command directs the Broker to take the following actions:

| Step | Procedure |
|---|---|
| Step 1 | Read its configuration file.<br>The Broker refreshes configuration options (see Table 2.4). |
| Step 2 | Read all component definitions found in ddname **UNVCONF**. The Broker replaces all component definitions with the newly read component definitions. New component definitions are added and deleted component definitions are removed. |
| Step 3 | Read the Universal Access Control List configuration file allocated to ddname **UNVACL**.<br>The Broker replaces its UACL entries with the newly read entries. |

## 3.4 Component Management

Universal Broker is aware only of Universal Products components that have been defined. It is the responsibility of Universal Broker to start, stop, and query these defined components.

One of the steps in the installation of a component is defining it to the local Universal Broker. These component definitions provide Universal Broker with the necessary information that it needs to manage the components.

### 3.4.1 Component Definitions

Component definitions are text files that define Universal Products components to the Universal Broker. All z/OS component definition files are located in the Universal Broker component definition library **UNVCOMP** allocated to the **UNVCOMP** ddname.

The syntax of a component definition file is the same as the Universal Broker configuration file. See Section 2.2.3 Configuration Files Syntax for detailed syntax information.

Table 3.3, below, identifies all of the options that comprise Universal Products for z/OS component definitions.

Each **Option Name** is a link to detailed information about that component definition option in the Universal Broker 3.2.0 Reference Guide.

| Option Name | Description |
|---|---|
| AUTOMATICALLY_START | Specification for whether the component automatically starts by the Universal Broker at start-up time or only on demand. |
| COMPONENT_NAME | Name by which clients know the component. |
| COMPONENT_TYPE | Type of component. |
| CONFIGURATION_FILE * | Component's configuration file name. |
| RUNNING_MAXIMUM | Maximum number of this component that can run simultaneously. |
| START_COMMAND * | Component program member name. |
| WORKING_DIRECTORY * | Path used as the working directory of the component. |
| * These options are required in the component definitions. | |

Table 3.3 Universal Products for z/OS - Component Definition Options

## 3.5 Security

Universal Broker is designed to be a secure system. As the level of security rises, so does the administrative complexity of the system. Universal Broker has balanced the two to avoid the administrative complexity with a minimum sacrifice to security.

Universal Broker security concerns are:

1. Access to Universal Broker data sets.
2. User account with which the Universal Broker executes.
3. Privacy and integrity of transmitted network data.

### 3.5.1 File Permissions

At a minimum, only trusted user accounts should have write access to the Universal Broker installation data sets. This most likely means only the administrators should have write access. For maximum security, only trusted accounts should have read access to these data sets.

In addition, Universal Broker requires update access to the HFS data base files allocated to ddname UNVDB and UNVSPOOL.

### 3.5.2 Configuration Files

Only trusted user accounts should have write, create or delete access to the Broker configuration files or any of the directories in the configuration file directory search list.

## 3.5.3 Universal Access Control List

The Universal Broker uses the Universal Access Control List (UACL) file as an extra layer of security. The UACL file contains Universal Broker entries that contain Access Control List (ACL) rules that permit or deny access to the Universal Broker.

See Section 2.8 Universal Access Control List for details on the Universal Access Control List feature.

The Universal Broker reads in the UACL entries when the program is started. If the UACL file is changed, the new entries can be activated by recycling the Broker or by sending the Universal Broker a Universal Control REFRESH command that will instruct the Universal Broker to reread all its configuration files including the UACL file (see Figure 3.3 Universal Broker for z/OS – REFRESH Command Example (2 of 2)).

(See the Universal Products Utilities 3.2.0 User Guide for details on Universal Control.)

## 3.5.4 UACL Entries

The syntax of a UACL entry file is the same as the Universal Broker configuration file. See Section 2.2.3 Configuration Files Syntax for detailed syntax information.

Table 3.4 identifies all UACL entries for Universal Broker for z/OS.

Each **UACL Entry Name** is a link to detailed information about that UACL entry in the Universal Broker 3.2.0 Reference Guide.

| UACL Entry Name | Description |
|---|---|
| UBROKER_ACCESS | Allows or denies access to Universal Broker services. |
| CERT_MAP | Maps a client X.509 certificate to a certificate identifier. |
| EVENT_ACCESS | Controls which Universal Enterprise Controller has read and delete access to the Universal Event Subsystem event data maintained by the Universal Broker. |
| REMOTE_CONFIG_ACCESS | Authorizes update access to the product configuration files and setting of the configuration managed mode of the Broker. |

Table 3.4 Universal Broker for z/OS - UACL Entries

## 3.5.5 UACL Examples

The following set of rules authorize the Universal Enterprise Controller at address 10.20.30, with update access to the product configuration files and setting of the configuration managed mode of the Broker, and denies all other connections.

```
remote_config_access     10.20.30.,allow,allow
remote-config_access     ALL,deny,deny
```

The following set of rules permit connections for the subnet 10.20.30 and denies all other connections.

```
ubroker_access     10.20.30.,allow
ubroker_access     ALL,deny
```

The following set of rules permit connections from host 10.20.30.40 and 10.20.30.50 and denies all other connections.

```
ubroker_access     10.20.30.40,allow
ubroker_access     10.20.30.50,allow
ubroker_access     ALL,deny
```

The following set of rules map X.509 certificates to certificate identifiers.

```
cert_map           id=joe,subject="/C=US/ST=Georgia/O=Acme, Inc./
                                   OU=Sales/CN=Joe Black"
```

# Chapter 4 Universal Broker for Windows

## 4.1 Overview

This chapter provides information on the Universal Broker, specific to the Windows operating system.

It describes how to execute and configure the Universal Broker.

### 4.1.1 Environment

Universal Broker can be executed in two different environments:

1. Console Application
2. Windows Service

Differences between these environments are described in the following sections.

## 4.1.2 Console Application

The **`ubroker`** command starts Universal Broker as a console application.

Enter **`ubroker`** either from the:

- Command Prompt window
- **`Run`** dialog (Select **`Run...`** from the Windows **`Start`** menu.)

### Console Security

Universal Broker inherits its user account from the user that starts it. The Broker itself does not require any additional permissions or rights other than the default ones granted to the Windows group user.

However, components started by the Broker also run with the same user account as the Broker. Some components may require permissions or rights other than those granted to the user account that started the Broker.

Refer to the Security sections in the User Guides of the components that you want to run for details on their security requirements.

For additional information regarding the security requirements of Universal Broker, see Section 4.4 Security.

## 4.1.3 Windows Service

Universal Broker is installed as a Windows service that starts automatically when the system is started. Windows provides a utility called **`Services`** that is used to interact with and manage all installed services. **`Services`** is an item in the Administrative Tools program group, which is accessible from the Control Panel.

### Service Security

The Universal Broker service must execute with the Local System account. The Local System account provides sufficient permissions and rights for the Broker.

Note: The Local System account does not provide access to network resources, such as network drives or printers.

## 4.2 Configuration

This section describes the Universal Broker for Windows configuration options.

See Section 2.2.1 Configuration Methods for details on Universal Products configuration methods.

### 4.2.1 Configuration Options

Table 4.1, below, identifies all of the Universal Broker for Windows configuration options. Each Option Name is a link to detailed information about that option in the Universal Broker 3.2.0 Reference Guide.

| Option Name | Description |
|---|---|
| ACTIVITY_MONITORING | Specification for whether or not product activity monitoring events are generated. |
| CA_CERTIFICATES | Path to PEM formatted trusted CA X.509 certificates. |
| CERTIFICATE | Path to Broker's PEM formatted X.509 certificate. |
| CERTIFICATE_REVOCATION_LIST | Path to PEM formatted CRL. |
| CODE_PAGE | Text translation code page. |
| COMPONENT_PORT | TCP/IP port used for Broker-Component communications. |
| CTL_SSL_CIPHER_LIST | SSL cipher list for the control sessions. |
| DNS_CACHE_TIMEOUT | Time-out for DNS cache. |
| EVENT_GENERATION | Events to be generated as persistent event records. |
| INSTALLATION_DIRECTORY | Base directory where product is installed. |
| LOG_DIRECTORY | Directory where log files are created. |
| MESSAGE_DESTINATION | Location where messages are written. |
| MESSAGE_LANGUAGE | Language of written messages. |
| MESSAGE_LEVEL | Level of messages written. |
| MONITOR_EVENT_EXPIRATION | Duration of a monitoring event record in the Universal Broker local UES database. |
| NLS_DIRECTORY | Location of UMC and UTT files. |
| PERSISTENT_EVENT_EXPIRATION | Duration of a persistent event record in the Universal Broker local UES database. |
| PRIVATE_KEY | Path to Broker's PEM formatted RSA private key. |
| PRIVATE_KEY_PWD | Password for the Broker's PRIVATE_KEY. |
| RUNNING_MAX | Maximum number of simultaneous components. |
| SERVICE_BACKLOG | Service interface backlog size for pending connection requests. |
| SERVICE_IP_ADDRESS | TCP/IP address on which the Broker listens. |
| SERVICE_PORT | TCP/IP port number on which the Broker listens. |

| Option Name | Description |
|---|---|
| SPOOL_DIRECTORY | Spool file directory. |
| TMP_DIRECTORY | Temporary file directory. |
| TRACE_DIRECTORY | Trace file directory. |
| TRACE_FILE_LINES | Maximum number of lines written to the trace file. |
| TRACE_TABLE | Memory trace table specification. |
| WORKING_DIRECTORY | Broker's working directory. |

Table 4.1 Universal Broker for Windows - Configuration Options

## 4.2.2 Configuration Refresh

Refreshing Universal Broker configuration directs Universal Broker to read its configuration data and update its current configuration settings.

Some configuration settings can be updated by refreshing Universal Broker while it is running. Other settings can be updated only when Universal Broker is refreshed by being recycled (stopped and restarted).

For information on the different methods of refreshing Universal Broker, and a list of options that is updated via each method, see Section 2.3 Universal Broker Configuration Refresh).

When any of the options that can be refreshed (see Table 2.4) are updated using the Universal Configuration Manager (see Section 2.5 Universal Configuration Manager), a refresh command is sent to Universal Broker, and its configuration is refreshed automatically.

The refresh command directs Universal Broker to take the following actions:

| Step | Description |
|---|---|
| Step 1 | Read its configuration file.<br>Universal Broker refreshes its configuration options. |
| Step 2 | Read all component definitions found in the component definition directory. The Broker replaces all component definitions with the newly read component definitions. New component definitions are added and deleted component definitions are removed. |
| Step 3 | Read the Universal Access Control List configuration entries from the registry.<br>The Broker replaces its UACL entries with the newly read entries. |

### Configuration Refresh via Universal Control

One method of refreshing Universal Broker is via the Universal Control Manager **REFRESH** command.

The Windows version of the **REFRESH** command issued from a console window is:

```
uctl -refresh -host hostname –port port
```

This example refreshes the configuration of Universal Broker running on the host **hostname** and listening on the port **port**.

(For detailed information on Universal Control, see the Universal Products Utilities 3.2.0 User Guide.)

The **REFRESH** command directs Universal Broker to take the same actions as when Universal Configuration Manager issues its refresh command.

## 4.3 Component Management

Universal Broker is aware only of Universal Products components that have been defined to it. It is the responsibility of Universal Broker to start, stop, and query these defined components.

One of the steps in the installation of a component is defining it to the local Universal Broker. These component definitions provide Universal Broker with the necessary information that it needs to manage the components.

### 4.3.1 Component Definitions

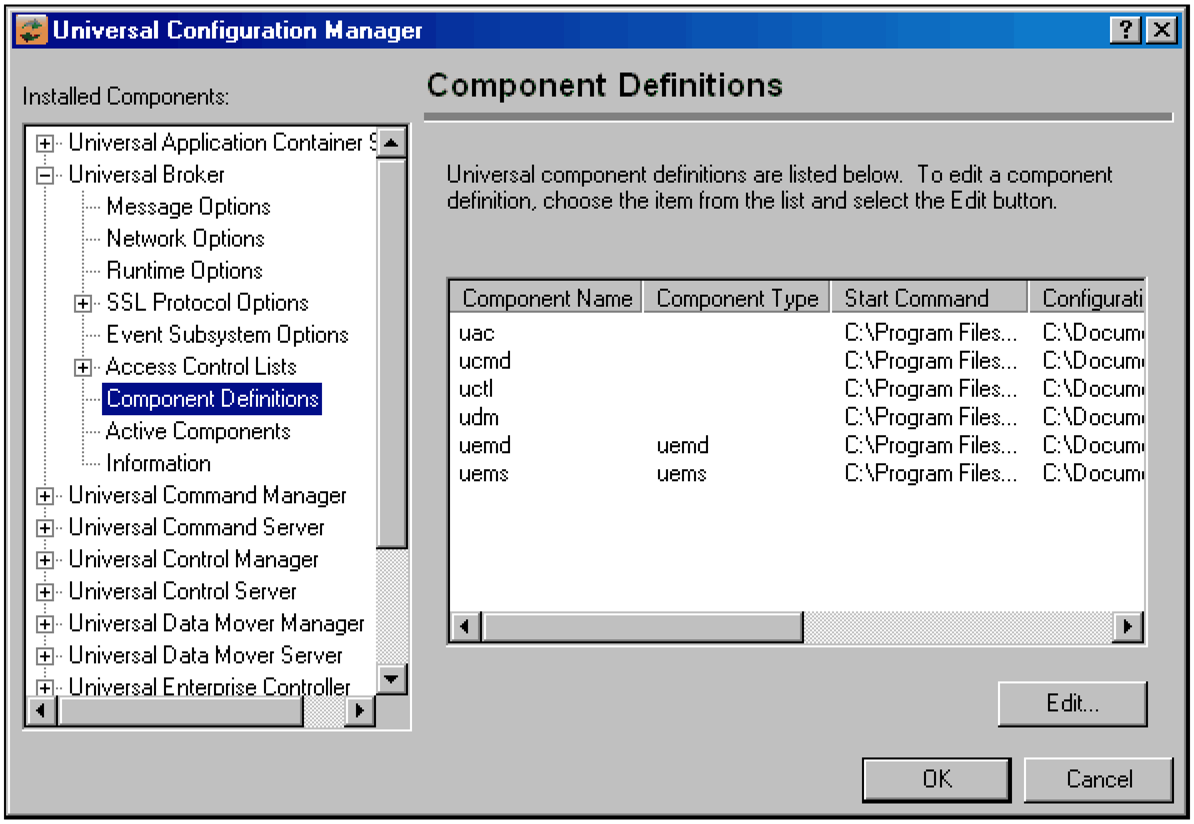
Component definitions are text files that define Universal Products components to the Universal Broker.

Component definition files reside in **`%ALLUSERSPROFILE%\Application Data\Universal\comp`**, where **`%ALLUSERSPROFILE%`** is an environment variable that resolves by default to **`C:\Documents and Settings\All Users`** on Windows 2000/XP/Server 2003 and **`C:\ProgramData`** on Windows Vista/Server 2008.

The syntax of a component definition file is the same as the Universal Broker configuration file. See Section 2.2.3 Configuration Files Syntax for detailed syntax information.

Although component definition files can be edited with any text editor (for example, Notepad), the Universal Configuration Manager application is the recommended way to edit component definitions for Windows (see Section 2.5 Universal Configuration Manager).

Note: The component definitions for all Universal Products are identified in the Component Definitions property page of the Universal Broker (see Figure 4.1, below).



Component Definitions

Universal component definitions are listed below. To edit a component definition, choose the item from the list and select the Edit button.

| Component Name | Component Type | Start Command | Configurati |
|---|---|---|---|
| uac | | C:\Program Files... | C:\Docum |
| ucmd | | C:\Program Files... | C:\Docum |
| uctl | | C:\Program Files... | C:\Docum |
| udm | | C:\Program Files... | C:\Docum |
| uemd | uemd | C:\Program Files... | C:\Docum |
| uems | uems | C:\Program Files... | C:\Docum |

Figure 4.1 Universal Configuration Manager - Component Definitions

Table 4.2, below, identifies all of the options that comprise Universal Products for Windows component definitions.

Each **Option Name** is a link to detailed information about that component definition option in the Universal Broker 3.2.0 Reference Guide.

| Option Name | Description |
|---|---|
| AUTOMATICALLY_START | Specification for whether the component automatically starts by the Universal Broker at start-up time or only on demand. |
| COMPONENT_NAME | Name by which clients know the component. |
| COMPONENT_TYPE | Type of component. |
| CONFIGURATION_FILE * | Component's configuration file name. |
| RUNNING_MAXIMUM | Maximum number of this component that can run simultaneously. |
| START_COMMAND * | Command that starts the component. |
| WORKING_DIRECTORY * | Path used as the working directory of the component. |
| * These options are required in the component definitions. | |

Table 4.2 Universal Products for Windows - Component Definition Options

## 4.4 Security

Universal Broker is designed to be a secure system. As the level of security increases so does the administrative complexity of the system. Universal Broker has balanced the two to avoid the administrative complexity with a minimum sacrifice to security.

Universal Broker security concerns are:

1. Access to Universal Broker files and directories
2. Access to Universal Broker configuration options
3. Account with which Universal Broker executes
4. Privacy and integrity of transmitted network data

### 4.4.1 File Permissions

At a minimum, only trusted user accounts should have write permission to the Universal Broker installation directory, subdirectories, and all of the files within them. This most likely means only the administrator group should have write access. For maximum security, only trusted accounts should have read, write, and execute permissions to these directories and files.

When the Broker is run as a Windows service, it does not modify any file in its installation directory or subdirectories. When the Broker runs as a console application with the message destination option set to log file, the Broker must have full control of the log subdirectory and all `.LOG` files within it.

### 4.4.2 Configuration Files

Only trusted user accounts should have write, create, or delete access to the Universal Broker configuration files.

Although configuration files can be edited with any text editor (for example, Notepad), the Universal Configuration Manager application, accessible via the Control Panel, is the recommended way to set configuration options.

The Universal Configuration Manager provides a graphical interface and context-sensitive help, and helps protect the integrity of the configuration file by validating all changes to configuration option values (see Section 2.5 Universal Configuration Manager).

### 4.4.3 Universal Broker User Account

Universal Broker and many of the server components must execute with the Local System account. Running Universal Broker as a Windows service using the Local System account provides all the rights required by Universal Broker and any component that it start. The system account is the default Windows account for Windows services.

### 4.4.4 Universal Access Control List

Universal Broker uses the Universal Access Control List (UACL) as an extra layer of security. The UACL contains Broker entries that contain Access Control List (ACL) rules that permit or deny access to the Broker.

See Section 2.8 Universal Access Control List for details on the Universal Access Control List feature.

Universal Broker reads the UACL entries when the program is started. If the UACL file is changed, the new entries can be activated either by:

- Stopping and starting Universal Broker.
- Sending Universal Broker a Universal Control REFRESH command, which instructs Universal Broker to reread all of its configuration files, including the UACL file.

Note: Although the UACL file, like all configuration files, can be edited with any text editor (for example, Notepad), the Universal Configuration Manager application, accessible via the Control Panel, is the recommended way to change UACL entries (see Section 2.5 Universal Configuration Manager).

Via this method, a REFRESH command is sent to Universal Broker, and any new entries take effect immediately. There is no need to stop and restart the Broker in order for the changes to be applied.

## UACL Entries

The syntax of a UACL entry file is the same as the Universal Broker configuration file. See Section 2.2.3 Configuration Files Syntax for detailed syntax information.

Table 4.3 identifies all Universal Broker for Windows UACL entries. Each **UACL Entry Name** is a link to detailed information about that UACL entry in the Universal Broker 3.2.0 Reference Guide.

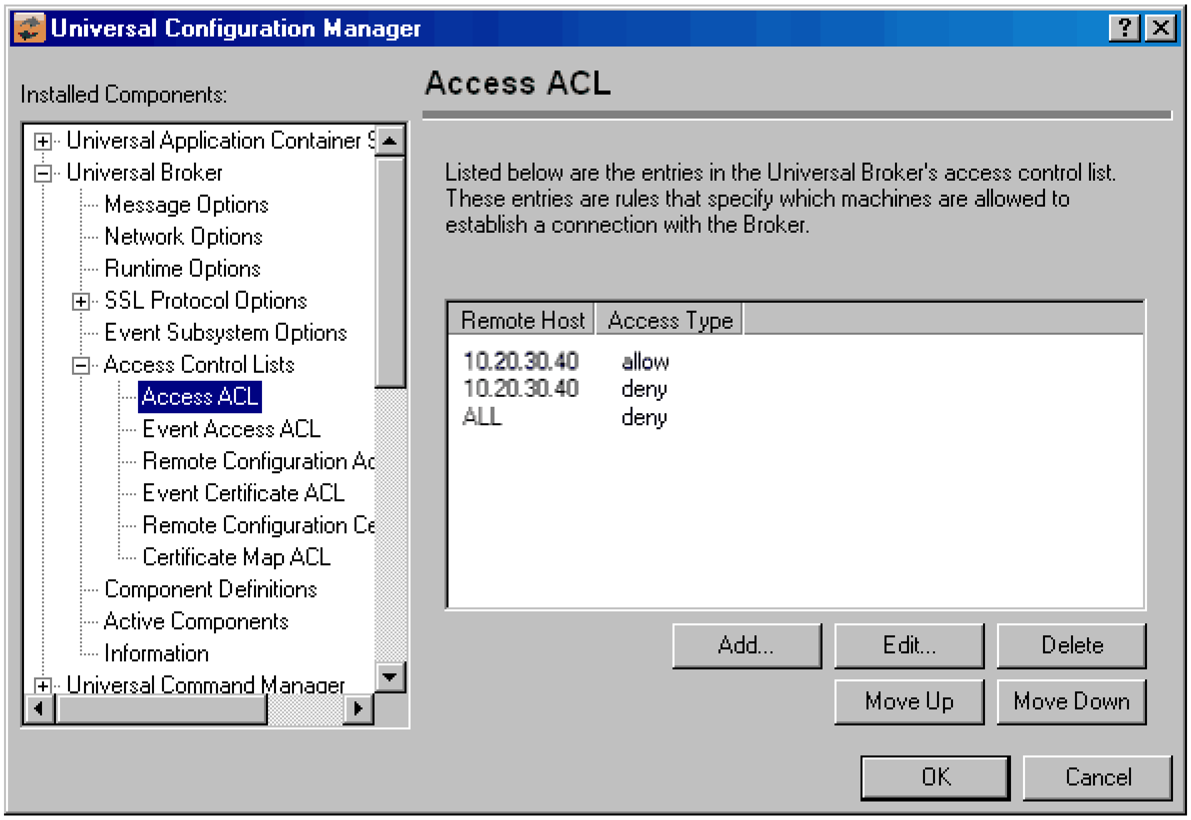
| UACL Entry Name | Description |
|---|---|
| UBROKER_ACCESS | Allows or denies access to Universal Broker services |
| CERT_MAP | Maps a client X.509 certificate to a certificate identifier. |
| EVENT_ACCESS | Controls which Universal Enterprise Controller has read and delete access to the Universal Event Subsystem event data maintained by the Universal Broker. |
| REMOTE_CONFIG_ACCESS | Authorizes update access to the product configuration files and setting of the configuration managed mode of the Broker. |

Table 4.3 Universal Broker for Windows - UACL Entries

## Updating the Universal Broker ACL Entries

Although UACL files can be edited with any text editor (for example, Notepad), the Universal Configuration Manager application, accessible via the Control Panel, is the recommended way to update UACL entries (see Section 2.5 Universal Configuration Manager). From there, ACL entries can be added, changed, deleted or sorted (rules are applied in the order in which they are listed).

Figure 4.2, below, illustrates an example.



Listed below are the entries in the Universal Broker's access control list. These entries are rules that specify which machines are allowed to establish a connection with the Broker.

| Remote Host | Access Type |
|---|---|
| 10.20.30.40 | allow |
| 10.20.30.40 | deny |
| ALL | deny |

Figure 4.2 Universal Configuration Manager - Universal Broker - Access ACL

# Chapter 5 Universal Broker for UNIX

## 5.1 Overview

This chapter provides information on using the Universal Broker, specific to the UNIX operating system.

It describes how to execute and configure the Universal Broker.

### 5.1.1 Environment

Universal Broker can be executed in two different environments:

- Daemon
- Console Application

Differences between the environments are described in the following sections.

Only one instance of the Universal Broker can execute at any one time. A PID file is used to help ensure that there is only one active instance; it is a locking mechanism that prevents the execution of a second Broker. The PID file, **`ubroker.pid`**, is created in directory **`/var/opt/universal`** by default. If the PID file is in the PID directory, it is assumed that a Broker instance is executing.

## 5.1.2 Daemon

Universal Broker can run as a UNIX daemon process. This is the preferred method of running the Broker. A daemon start-up script is provided to manage the starting and stopping of the Broker daemon. The startup script utilizes the PID file to ensure that only one instance of the Broker is executing at any one time. For this reason, the start-up script should be used to start and stop the Broker.

Note: Although they have the same name, the Broker daemon start-up script should not be confused with the actual Broker daemon program file.

- Startup script is installed in the primary Broker directory (that is, **`./universal/ubroker`**).
- Program file is installed in the Broker's **`bin`** directory (that is, **`./universal/ubroker/bin`**).

```
ubrokerd { start | stop | status | restart }
```

Figure 5.1 Universal Broker for UNIX - Daemon Startup Script Syntax

Figure 5.2, below, describes the command line arguments to the Universal Broker daemon start-up script.

| Command | Description |
|---|---|
| `start` | Starts the Universal Broker daemon. Only one instance of Universal Broker can run at any given time, so if the Broker already is running, the command fails and the script returns. |
| `stop` | Stops the Universal Broker daemon. If the Broker daemon is not running, the script simply returns. |
| `status` | Returns the status of the Universal Broker daemon, either *running* or *stopped*. If the daemon is running, the script displays its process ID. |
| `restart` | Performs a **`stop`** request followed by a **`start`** request. |

Figure 5.2 Universal Broker - Command Line Arguments to Daemon Startup Script

### Daemon Security

When a daemon is started at system initialization, it is started as user **`root`**. The root user ID provides sufficient authority for the Broker and any component it may start.

If the daemon is started with a non-root user ID, the environment is the same as if it was started as a console application. (See Section Console Security for more details.)

## 5.1.3 Console Application

The `ubroker` command starts Universal Broker as a console application.

### Console Security

Universal Broker runs with the same user ID as the user who starts it. The Broker does not require superuser rights. It only requires access to its installation directory and files, which often are created by the superuser account when the product is installed.

However, components started by the Broker also run with the same user ID as the Broker. Some of these components may require superuser rights.

Refer to the Security sections in the User Guides of the components that you want to run for details on their security requirements.

## 5.2 Configuration

This section describes the Universal Broker configuration options.

See Section 2.2.1 Configuration Methods for details on Universal Products configuration methods.

### 5.2.1 Configuration File

The configuration file provides the simplest method of specifying configuration values that will not change with each command invocation.

The Universal Broker configuration file is named **`ubroker.conf`**. This file can be edited manually with any text editor.

See Section 2.2.2 Configuration Files for details on the location of Universal Products configuration files.

## 5.2.2 Configuration Options

Table 5.1, below, identifies all of the Universal Broker for UNIX configuration options. Each Option Name is a link to detailed information about that option in the Universal Broker 3.2.0 Reference Guide.

| Option Name | Description |
|---|---|
| ACTIVITY_MONITORING | Specification for generation of product activity monitoring events. |
| BIF_DIRECTORY | Broker Interface Directory where Universal Broker will create its broker interface file. |
| CA_CERTIFICATES | Path to PEM formatted trusted CA X.509 certificates. |
| CERTIFICATE | Path to Broker's PEM formatted X.509 certificate. |
| CERTIFICATE_REVOCATION_LIST | Path to PEM formatted CRL. |
| CODE_PAGE | Text translation code page. |
| COMPONENT_DIRECTORY | Component definition file directory. |
| CTL_SSL_CIPHER_LIST | SSL cipher list for the control sessions. |
| DNS_CACHE_TIMEOUT | Time-out for DNS cache. |
| EVENT_GENERATION | Events to be generated as persistent event records. |
| INSTALLATION_DIRECTORY | Base directory where product is installed. |
| LOG_DIRECTORY | Log file directory. |
| LOG_FILE_LINES | Total number of lines to be written to the log file before the log file is wrapped. |
| MESSAGE_DESTINATION | Location where messages are written. |
| MESSAGE_LANGUAGE | Language of written messages. |
| MESSAGE_LEVEL | Level of messages written. |
| MONITOR_EVENT_EXPIRATION | Duration of a monitoring event record in the Universal Broker local UES database. |
| NLS_DIRECTORY | UMC and UTT file directory. |
| PERSISTENT_EVENT_EXPIRATION | Duration of a persistent event record in the Universal Broker local UES database. |
| PID_FILE_DIRECTORY | PID file location. |
| PRIVATE_KEY | Path to Broker's PEM formatted RSA private key. |
| PRIVATE_KEY_PWD | Password for the Broker's PRIVATE_KEY. |
| RUNNING_MAX | Maximum number of simultaneous components. |
| SERVICE_BACKLOG | Service interface backlog size for pending connection requests. |
| SERVICE_IP_ADDRESS | TCP/IP address on which the Broker listens. |
| SERVICE_PORT | TCP/IP port number on which the Broker listens. |
| SPOOL_DIRECTORY | Spool file directory. |
| TMP_DIRECTORY | Temporary file directory. |

| Option Name | Description |
|---|---|
| TRACE_DIRECTORY | Trace file directory. |
| TRACE_FILE_LINES | Maximum number of lines written to the trace file. |
| TRACE_TABLE | Memory trace table specification. |
| WORKING_DIRECTORY | Broker's working directory. |

Table 5.1 Universal Broker for UNIX - Configuration Options

## 5.2.3 Configuration Refresh

Refreshing Universal Broker configuration directs Universal Broker to read its configuration data and update its current configuration settings.

Some configuration settings can be updated by refreshing Universal Broker while it is running. Other settings can be updated only when Universal Broker is refreshed by being recycled (stopped and restarted).

For information on the different methods of refreshing Universal Broker, and a list of options that is updated via each method, see Section 2.3 Universal Broker Configuration Refresh).

### Configuration Refresh via Universal Control

One method of refreshing Universal Broker is via the Universal Control Manager **REFRESH** command.

The UNIX version of the Universal Control Manager REFRESH command is:

```
uctl -refresh -host dallas
```

This example refreshes Universal Broker configuration on host **dallas**.

(For detailed information on Universal Control, see the Universal Products Utilities 3.2.0 User Guide.)

The REFRESH command directs Universal Broker to take the following actions:

| Step | Description |
|---|---|
| Step 1 | Read its configuration file **ubroker.conf**.<br>Universal Broker refreshes the following configuration options:<br>• MESSAGE_LANGUAGE<br>• RUNNING_MAX |
| Step 2 | Read all component definitions found in the component definition directory. The Broker replaces all component definitions with the newly read component definitions. New component definitions are added and deleted component definitions are removed. |
| Step 3 | Read the Universal Access Control List configuration file **uacl.conf**.<br>The Broker replaces its UACL entries with the newly read entries. |

## 5.3 Component Management

Universal Broker is aware only of Universal Products components that have been defined. It is the responsibility of Universal Broker to start, stop, and query these defined components.

One of the steps in the installation of a component is defining it to the local Universal Broker. These component definitions provide Universal Broker with the necessary information that it needs to manage the components.

### 5.3.1 Component Definitions

Component definitions are text files that define Universal Products components to the Universal Broker. All UNIX component definition files are located in the Universal Broker component definition directory (specified with the COMPONENT_DIRECTORY configuration option).

The syntax of a component definition file is the same as the Universal Broker configuration file. See Section 2.2.3 Configuration Files Syntax for detailed syntax information.

Table 5.2, below, identifies all of the options that comprise Universal Products for UNIX component definitions.

Each **Option Name** is a link to detailed information about that component definition option in the Universal Broker 3.2.0 Reference Guide.

| Option Name | Description |
|---|---|
| AUTOMATICALLY_START | Specification for whether the component automatically starts by the Universal Broker at start-up time or only on demand. |
| COMPONENT_NAME | Name by which clients know the component. |
| COMPONENT_TYPE | Type of component. |
| CONFIGURATION_FILE * | Component's configuration file name. |
| RUNNING_MAXIMUM | Maximum number of this component that can run simultaneously. |
| START_COMMAND * | Command that starts the component. |
| WORKING_DIRECTORY * | Path used as the working directory of the component. |
| * These options are required in the component definitions. | |

Table 5.2 Universal Products for UNIX - Component Definition Options

## 5.4 Security

Universal Broker is designed to be a secure system. As the level of security rises, so does the administrative complexity of the system. Universal Broker has balanced the two to avoid the administrative complexity with a minimum sacrifice to security.

Universal Broker security concerns are:

1. Access to Universal Broker files and directories.
2. Access to Universal Broker configuration files.
3. User account with which the Universal Broker executes.
4. Privacy and integrity of transmitted network data.

### 5.4.1 File Permissions

At a minimum, only trusted user accounts should have write permission to the Universal Broker installation directory, subdirectories, and all files within. This most likely means only the administrators should have write access. For maximum security, only trusted accounts should have read, write or execute permissions to these directories and files.

All files that the Broker creates or updates are located in the `/var/opt/universal`. The Broker does not need write access to its installation directory or subdirectories.

Universal Broker requires full control (read, write, remove, and add) of the `/var/opt/universal` directory and its subdirectories. The Broker creates spool files, trace files, and log files in this directory. Users accounts other then the administrator accounts do not require access to this directory.

The Broker configuration options can be changed to use directories other then `/var/opt/universal`. If this is the case, the same permissions must be set up for these specified directories.

### 5.4.2 Configuration Files

Only trusted user accounts should have write, create or delete access to the Broker configuration files or any of the directories in the configuration file directory search list.

## 5.4.3 Universal Broker User ID

A component started by Universal Broker inherits the same user ID as Universal Broker. However, the component can have different security requirements than Universal Broker.

For example, although Universal Broker itself does not require super-user privileges, Universal Command Server may require super-user authority.

Since the component inherits its user ID from Universal Broker, either:

- Universal Broker must be running as root.
- Universal Command Server program must be owned by root and have the set user ID on execution permission.

If Universal Broker is started as a daemon at system startup time, it is started with a user ID of root. Universal Broker and all its components then will have sufficient authority.

## 5.4.4 Universal Access Control List

Universal Broker uses the Universal Access Control List (UACL) file as an extra layer of security. The UACL file contains Universal Broker entries that contain Access Control List (ACL) rules that permit or deny access to Universal Broker.

See Section 2.8 Universal Access Control List for details on the Universal Access Control List feature.

Universal Broker reads in the UACL entries when the program is started. If the UACL file is changed, the new entries can be activated either by:

- Stopping and starting Universal Broker
- Sending Universal Broker a Universal Control REFRESH command, which instructs Universal Broker to reread all its configuration files, including the UACL file.

The UNIX REFRESH command is: **`uctl –refresh –host BROKER-IPADDR`**.

(See the Universal Products Utilities 3.2.0 User Guide for complete details on Universal Control.)

## 5.4.5 UACL Entries

The syntax of a UACL entry file is the same as the Universal Broker configuration file. See Section 2.2.3 Configuration Files Syntax for detailed syntax information.

Table 5.3 identifies all Universal Broker for UNIX UACL entries. Each **UACL Entry Name** is a link to detailed information about that UACL entry in the Universal Broker 3.2.0 Reference Guide.

| UACL Entry Name | Description |
|---|---|
| UBROKER_ACCESS | Allows or denies access to Universal Broker services. |
| CERT_MAP | Maps a client X.509 certificate to a certificate identifier. |
| EVENT_ACCESS | Controls which Universal Enterprise Controller has read and delete access to the Universal Event Subsystem event data maintained by the Universal Broker. |
| REMOTE_CONFIG_ACCESS | Authorizes update access to the product configuration files and setting of the configuration managed mode of the Broker. |

Table 5.3 Universal Broker for UNIX - UACL Entries

## 5.4.6 UACL Examples

The following set of rules is reqired to allow Universal Management Console to access Universal Broker.

```
remote_config_access     10.20.30.,allow
remote-config_access     ALL,deny
```

The following set of rules permit connections for the subnet 10.20.30 and denies all other connections.

```
ubroker_access     10.20.30.,allow
ubroker_access     ALL,deny
```

The following set of rules permit connections from host 10.20.30.40 and 10.20.30.50 and denies all other connections.

```
ubroker_access     10.20.30.40,allow
ubroker_access     10.20.30.50,allow
ubroker_access     ALL,deny
```

The following set of rules map X.509 certificates to certificate identifiers.

```
cert_map           id=joe,subject="/C=US/ST=Georgia/O=Acme, Inc./"
```

# Chapter 6 Universal Broker for OS/400

## 6.1 Overview

This chapter describes how to execute and configure the Universal Broker, specific to the OS/400 operating system.

### 6.1.1 Environment

Universal Broker executes within its own OS/400 subsystem, named **UNVUBR320**. The **UNVUBR320** subsystem provides a self-contained environment in which Universal Broker can be managed. The **UNVUBR320** subsystem description (object type ***SBSD**) is named **UNVUBR320**.

The **UNVUBR320** subsystem contains several entries that define the subsystem environment. The two most visible are:

- Autostart entry
- Pre-start job entries

The subsystem autostart entry defines what jobs are started automatically when the subsystem is started. The **UNVUBR320** subsystem defines one autostart entry, **UNVUBR320**. The **UBROKER** job executes with the job description **UBROKER** (object type ***JOBD**) and user profile **UNVUBR320** (object type ***USRPRF**). Only one instance of the **UBROKER** job, which runs continuously, can be active at any one time.

The subsystem pre-start job entries define jobs that are in an initialized state. They are not executing but are ready to accept a request and execute at any time. Pre-starting jobs before they are required improves the overall throughput of the subsystem jobs.

Universal Broker jobs running under **UNVUBR320** use the **UBROKER** job queue and class located in the product installation library. See the Universal Products 3.2.0 Installation Guide for additional information.

The Universal Command (UCMD) Server jobs log all significant events to the **UBROKER** job log. However, by default, OS/400 does not keep job logs unless the job terminates due to an error. As a result, important information relevant to server errors may be discarded when the **UBROKER** job is shut down normally.

To preserve the server-related information, the UBROKER job description specifies Message Logging as *4 0 *MSG*. The **UBROKER** job's job log will be sent automatically to the output queue and printer device designated in the **UBROKER** job description, which is located in the Universal Products installation library, **UNVPRD320** (by default).

In some very large organizations with heavy **UBROKER** usage, the job log may fill. By default, OS/400 jobs are stopped when the job log fills. To ensure continuous **UBROKER** operation, Universal Products sets the job log to wrap. (See Chapter 6 OS/400 Installation in the Universal Products 3.2.0 Installation Guide for additional information.)

## 6.1.2 Commands

The following O/S commands help manage the **UNVUBR320** subsystem.

### Start Subsystem Command (STRSBS)

Starts the Universal Broker subsystem, **UNVUBR320**.

```
STRSBS UNVPRD320/UNVUBR320
```

Figure 6.1 Universal Broker for OS/400 - Subsystem Start Command

### End Subsystem Command (ENDSBS)

Ends the Universal Broker subsystem, **UNVUBR320**.

```
ENDSBS UNVUBR320
```

Figure 6.2 Universal Broker for OS/400 - Subsystem End Command

### Work With Subsystem Command (WRKSBS)

Allows users to work with all active subsystems. Choose the **UNVUBR320** subsystem from the list of subsystems displayed.

```
WRKSBS
```

Figure 6.3 Universal Broker for OS/400 - Subsystem Work With Command

## 6.2 Configuration

This section describes the Universal Broker for OS/400 configuration options.

See Section 2.2.1 Configuration Methods for details on Universal Products configuration methods.

### 6.2.1 Configuration File

The Universal Broker configuration file is named **UNVPRD320/UNVCONF(UBROKER)**. File **UNVCONF** is a physical source file located in the **UNVPRD320** library. File member **UBROKER** contains the configuration options for the Universal Broker. File **UNVCONF** contains configuration members for the Universal family of products. This file can be edited manually with any text editor.

See Section 2.2.2 Configuration Files for details on the location of Universal Product configuration files.

### 6.2.2 Configuration Options

Table 6.1, below, identifies all of the Universal Broker for OS/400 configuration options. Each **Option Name** is a link to detailed information about that option in the Universal Broker 3.2.0 Reference Guide.

| Option Name | Description |
|---|---|
| ACTIVITY_MONITORING | Specification for whether or not product activity monitoring events are generated. |
| CA_CERTIFICATES | Path to PEM formatted trusted CA X.509 certificates. |
| CERTIFICATE | Path to Broker's PEM formatted X.509 certificate. |
| CERTIFICATE_REVOCATION_LIST | Path to PEM formatted CRL. |
| CODE_PAGE | Text translation code page. |
| CTL_SSL_CIPHER_LIST | SSL cipher list for the control sessions. |
| DNS_CACHE_TIMEOUT | Time-out for DNS cache. |
| EVENT_GENERATION | Events to be generated as persistent events. |
| MESSAGE_DESTINATION | Location where messages are written. |
| MESSAGE_LANGUAGE | Language of written messages. |
| MESSAGE_LEVEL | Level of messages written. |
| MONITOR_EVENT_EXPIRATION | Duration of a monitoring event record in the Universal Broker local UES database. |
| PERSISTENT_EVENT_EXPIRATION | Duration of a persistent event record in the Universal Broker local UES database. |
| PRIVATE_KEY | Path to Broker's PEM formatted RSA private key. |

| Option Name | Description |
|---|---|
| PRIVATE_KEY_PWD | Password for the Broker’s PRIVATE_KEY. |
| RUNNING_MAX | Maximum number of simultaneous components. |
| SERVICE_BACKLOG | Service interface backlog size for pending connection requests. |
| SERVICE_IP_ADDRESS | TCP/IP address on which the Broker listens. |
| SERVICE_PORT | TCP/IP port number on which the Broker listens. |
| TRACE_FILE_LINES | Maximum number of lines written to the trace file. |
| TRACE_TABLE | Memory trace table specification. |

Table 6.1 Universal Broker for OS/400 - Configuration Options

## 6.2.3 Configuration Refresh

Refreshing Universal Broker configuration directs Universal Broker to read its configuration data and update its current configuration settings.

Some configuration settings can be updated by refreshing Universal Broker while it is running. Other settings can be updated only when Universal Broker is refreshed by being recycled (stopped and restarted).

For information on the different methods of refreshing Universal Broker, and a list of options that is updated via each method, see Section 2.3 Universal Broker Configuration Refresh).

### Configuration Refresh via Universal Control

One method of refreshing Universal Broker is via the Universal Control Manager **REFRESH** command.

The OS/400 version of the Universal Control Manager **REFRESH** command is:

```
STRUCT REFRESH(*YES) HOST(HOUSTON) USERID(bill) PWD(zonkers)
```

This example refreshes Universal Broker configuration on host **HOUSTON**.

(For detailed information on Universal Control, see the Universal Products Utilities 3.2.0 User Guide.)

The **REFRESH** command directs Universal Broker to take the following actions:

| Step | Description |
|---|---|
| Step 1 | Read its configuration file **UNVCONF** and member **UBROKER**. |
| Step 2 | Read all component definitions found in the component definition file, **UNVPRD320** / **UNVCOMP**. The Broker replaces all component definitions with the newly read component definitions. New component definitions are added and deleted component definitions are removed. |
| Step 3 | Read the Universal Access Control List configuration file **UNVCONF** and member **UACL**.<br>The Broker replaces its UACL entries with the newly read entries. |

In the above example, leaving option **RFSHCMPNM** (refresh component name) unspecified results in the Universal Broker refreshing itself.

In the following example, Universal Broker refreshes the Universal Data Mover component on the local system.

```
STRUCT REFRESH(*YES) RFSHCMPNM(UDM) HOST(localhost) USERID(james)
PWD(akita)
```

## 6.3 Component Management

Universal Broker is aware only of Universal Products components that have been defined. It is the responsibility of Universal Broker to start, stop, and query these defined components.

One of the steps in the installation of a component is defining it to the local Universal Broker. These component definitions provide Universal Broker with the necessary information that it needs to manage the components.

### 6.3.1 Component Definitions

Component definitions are text files that define Universal Products components to the Universal Broker. All OS/400 component definitions are located in the source physical file **UNVPRD320/UNVCOMP** as individual members.

The syntax of a component definition file is the same as the Universal Broker configuration file. See Section 2.2.3 Configuration Files Syntax for detailed syntax information.

Table 6.2, below, identifies all of the options that comprise Universal Products for OS/400 component definitions.

Each **Option Name** is a link to detailed information about that component definition option in the Universal Broker 3.2.0 Reference Guide.

| Option Name | Description |
|---|---|
| AUTOMATICALLY_START | Specification for whether or not the component automatically starts by the Universal Broker at start-up time or only on demand. |
| COMPONENT_NAME | Name by which clients know the component. |
| COMPONENT_TYPE | Type of component. |
| CONFIGURATION_FILE * | Component's configuration file name. |
| RUNNING_MAXIMUM | Maximum number of this component that can run simultaneously. |
| START_COMMAND * | Component program name. |
| WORKING_DIRECTORY * | Path used as the working directory of the component. |
| * These options are required in the component definitions. | |

Table 6.2 Universal Products for OS/400 - Component Definition Options

## 6.4 Security

Universal Broker is designed to be a secure system. As the level of security rises, so does the administrative complexity of the system. Universal Broker has balanced the two to avoid administrative complexity with a minimum sacrifice to security.

Universal Broker security concerns are:

1. Access to Universal Broker files and directories
2. Access to Universal Broker configuration files
3. User account with which the Universal Broker executes
4. Privacy and integrity of transmitted network data

### 6.4.1 Object Permissions

At a minimum, limit non-trusted user accounts to object authority of use to the Universal Broker product library, **UNVPRD320**; the product temporary library, **UNVTMP320**; the command reference library, **UNVCMDREF**; the universal spool library, **UNVSPL320**; and all objects within these libraries.

For maximum security, only trusted accounts (administrators and the **UNVUBR320** profile) should have management, existence, alter, add, update, or delete authority to these objects. As a reminder, the system value **QCRTAUT** controls public access authority to created objects unless overridden by specific commands.

### 6.4.2 Configuration Files

Only trusted user accounts should have management, existence, alter, add, update, or delete authority to the UNiversal Broker configuration files.

## 6.4.3 User Profile

Universal Broker runs with the **UNVUBR320** user profile, which is created at product installation time. Any component started by Universal Broker inherits this user profile.

By default, the **UNVUBR320** user profile has *ALLOBJ, *JOBCTL, and *SPLCTL authority. Unless the user profile is modified as described in the following section, *ALLOBJ authority is required for a component to switch its user profiles based on the request it is servicing. *JOBCTL authority is required for internal control and should not be removed. The **UNVUBR320** user profile requires *SPLCTL authority to provide Universal Submit Job job logs in specific, limited situations. (See the Universal Products Utilities 3.2.0 User Guide for information on Universal Submit Job.)

Any other product or user should not use the **UNVUBR320** user profile. By default, users cannot access the system with the **UNVUBR320** profile.

### Removing *ALLOBJ Authority from UNVUBR320 User Profile

Given the extensive authority allowed by *ALLOBJ special authority, it is desirable to avoid its use when possible. As of PTF 0UC0126 for V1R2M1, it is possible to remove *ALLOBJ special authority from the **UNVUBR320** user profile. However, by removing *ALLOBJ from the **UNVUBR320** user profile, the administrative complexity is increased.

The following describes the steps that are required to use Universal Command with *ALLOBJ special authority removed from the **UNVUBR320** user profile.

1. If the following objects do not have *USE Public Authority, the **UNVUBR320** user profile must be given *USE authority:
   - QSYS/QSYGETPH
   - QSYS/QWTSETP
   - QSYS/QWCRJBST
   - QSYS/QUSRMBRD

   This can be accomplished with the following command:
   ```
   ===> EDTOBJAUT OBJ(QSYS/object_name) OBJTYPE(*PGM)
   ```
   From the resulting screen, use F6 to add user **UBROKER** and give it *USE authority.
2. **UNVUBR320** user profile must be given *USE authority to the user profile objects of all user profiles that will be using the universal command server on the OS/400.

   This can be accomplished with the following command:
   ```
   ===> EDTOBJAUT OBJ(QSYS/user_profile_name) OBJTYPE(*USRPRF)
   ```
   From the resulting screen, use F6 to add user **UBROKER** and give it *USE authority.
3. Use the following command to remove the **UNVUBR410** user profile *ALLOBJ authority:
   ```
   ===> CHGUSRPRF USRPRF(UNVUBR410) SPCAUT(*JOBCTL *SPLCTL)
   ```

### Removing *SPLCTL Authority from UNVUBR410 User Profile

Use the following command to remove the **`UNVUBR410`** user profile *SPLCTL authority:

```
 ===> CHGUSRPRF USRPRF(UNVUBR410) SPCAUT(*JOBCTL *ALLOBJ)
```

### Removing *ALLOBJ and *SPLCTL Authorities from UNVUBR410 User Profile

Use the following command to remove all special authority from the **`UNVUBR410`** user profile:

```
 ===> CHGUSRPRF USRPRF(UNVUBR410) SPCAUT(*JOBCTL)
```

(Please refer to the previous two sections for additional information.)

## 6.4.4 Universal Access Control List

Universal Broker uses the Universal Access Control List (UACL) file as an extra layer of security. The UACL file contains Universal Broker entries that contain Access Control List (ACL) rules that permit or deny access to the Broker.

See Section 2.8 Universal Access Control List for details on the Universal Access Control List feature.

Universal Broker reads in the UACL entries when the program is started. If the UACL file is changed, the new entries can be activated either by:

- Stopping and starting Universal Broker.
- Sending Universal Broker a Universal Control **`REFRESH`** command, which instructs Universal Broker to reread all its configuration files, including the UACL file.

The OS/400 REFRESH command is: **`STRUCT REFRESH(*YES) HOST(hostname)`**.

(See the Universal Products Utilities 3.2.0 User Guide for complete details on Universal Control.)

## 6.4.5 UACL Entries

The syntax of a UACL entry file is the same as the Universal Broker configuration file. See Section 2.2.3 Configuration Files Syntax for detailed syntax information.

Table 6.3 identifies all Universal Broker for OS/400 UACL entries. Each **UACL Entry Name** is a link to detailed information about that UACL entry in the Universal Broker 3.2.0 Reference Guide.

| UACL Entry Name | Description |
|---|---|
| UBROKER_ACCESS | Allows or denies access to Universal Broker services. |
| CERT_MAP | Maps a client X.509 certificate to a certificate identifier. |
| EVENT_ACCESS | Controls which Universal Enterprise Controller has read and delete access to the Universal Event Subsystem event data maintained by the Universal Broker. |
| REMOTE_CONFIG_ACCESS | Authorizes update access to the product configuration files and setting of the configuration managed mode of the Universal Broker. |

Table 6.3 Universal Broker for OS/400 - UACL Entries

## 6.4.6 UACL Examples

The following set of rules permit connections for the subnet 10.20.30 and denies all other connections.

```
ubroker_access     10.20.30.,allow
ubroker_access     ALL,deny
```

The following set of rules permit connections from host 10.20.30.40 and 10.20.30.50 and denies all other connections.

```
ubroker_access     10.20.30.40,allow
ubroker_access     10.20.30.50,allow
ubroker_access     ALL,deny
```

The following set of rules map X.509 certificates to certificate identifiers.

```
cert_map           id=joe,subject="/C=US/ST=Georgia/O=Acme, Inc./
                                   OU=Sales/CN=Joe Black"
```

# Chapter 7
# Universal Broker for HP NonStop

## 7.1 Overview

This section documents the Universal Broker at a detailed level. The material is specific to the HP NonStop based operating system. The following sections describe how to execute and configure the Universal Broker.

**Currently, HP NonStop runs Universal Broker 2.1.1.**
**This chapter provides information for that version.**

### 7.1.1 Environment

Universal Broker for HP NonStop runs as an Open System Services (OSS) application. It can be executed in two different environments:

- Console Application
- Daemon

Differences between the environments are described in the following sections.

Only one instance of the Universal Broker can execute at any one time. A PID file is used to help ensure only one active instance. The PID file is a locking mechanism that prevents the execution of a second Broker. The PID file, named **`UBRPID`**, is created in subvolume **`$SYSTEM.UNVLOG`** by default. If the PID file is in the PID subvolume, it is assumed that a Universal Broker instance is executing.

## 7.1.2 Console Application

The command **`ubroker`** starts Universal Broker as a console application.

Figure 7.1, below, illustrates the Universal Broker start command.

```
ubroker [OPTIONS...]
```

Figure 7.1 Universal Broker for HP NonStop - Start Command

(Options are described in Section 7.2.2 Configuration Options.)

### Console Security

The Universal Broker runs with the same user ID as the user who starts it. The Universal Broker does not require **`super.super`** rights. It only requires access to its installation subvolume and files.

However, components started by Universal Broker also run with the same user ID as Universal Broker. Some components may require **`super.super`** rights.

(See the security documentation of the components you wish to run for details on their security requirements.)

## 7.1.3 Daemon

Universal Broker can run as a daemon process. This is the preferred method of running the Broker. A daemon startup script is provided to manage the starting and stopping of the Broker daemon. The startup script utilizes the PID file to ensure only one instance of the Broker is executing at any one time. For this reason, the startup script should be used to start and stop the Broker.

Note: The Universal Broker daemon startup script and the Universal Broker daemon program file both are installed within the **$SYSTEM.UNVBIN** subvolume. The Broker daemon startup script name is **ubrokerd** and the Broker daemon program file name is **ubrd**.

```
ubrokerd { start | stop | status | restart }
```

Figure 7.2 Universal Broker for HP NonStop - Daemon Startup Script Syntax

Figure 7.3, below, describes the command line arguments to the Universal Broker daemon startup.

| Command | Description |
|---|---|
| `start` | Starts the Universal Broker daemon. Only one instance of Universal Broker can run at any given time, so if the Broker is already running, the command fails and the script returns. |
| `stop` | Stops the Universal Broker daemon. If the Broker daemon is not running, the script simply returns. |
| `status` | Returns the status of the Universal Broker daemon: either *running* or *stopped*. If the daemon is running, the script displays its process ID. |
| `restart` | Performs a **stop** request followed by a **start** request. |

Figure 7.3 Universal Broker for HP NonStop - Command Line Arguments to Daemon Startup Script

### Daemon Security

When a daemon is started at system initialization, it is started as user **super.super**. The **super.super** user ID provides sufficient authority for the Broker and any component it may start.

If the daemon is started with a non-super user ID, the environment is the same as if it was started as a console application.

(See Console Security for more details.)

## 7.2 Configuration

This section describes the Universal Broker configuration options.

See Section 2.2 Configuration for details on Universal Products configuration methods.

### 7.2.1 Configuration File

The configuration file provides the simplest method of specifying configuration values that will not change with each command invocation.

The Universal Broker configuration file is named **UBRCFG**. This file can be edited manually with the EDIT TACL command.

See Section 2.2.2 Configuration Files for details on the location of Universal Product configuration files.

### 7.2.2 Configuration Options

Table 7.1, below, summarizes all configuration options for Universal Broker for HP NonStop. Each Option Name is a link to detailed information about that option in the Universal Broker 3.2.0 Reference Guide.

| Option Name | Description |
|---|---|
| CODE_PAGE | Text translation code page. |
| INSTALLATION_DIRECTORY | Base directory where product is installed. |
| MESSAGE_DESTINATION | Location where messages are written. |
| MESSAGE_LANGUAGE | Language of written messages. |
| MESSAGE_LEVEL | Level of messages written. |
| RUNNING_MAX | Maximum number of simultaneous components. |
| SERVICE_IP_ADDRESS | TCP/IP address on which the Broker listens. |
| SERVICE_PORT | TCP/IP port number on which the Broker listens. |

Table 7.1 Universal Broker for HP NonStop - Configuration Options

## 7.2.3 Configuration Refresh

Refreshing Universal Broker configuration directs Universal Broker to read its configuration data and update its current configuration settings.

Some configuration settings can be updated by refreshing Universal Broker while it is running. Other settings can be updated only when Universal Broker is refreshed by being recycled (stopped and restarted).

For information on the different methods of refreshing Universal Broker, and a list of options that is updated via each method, see Section 2.3 Universal Broker Configuration Refresh).

### Configuration Refresh via Universal Control

One method of refreshing Universal Broker is via the Universal Control Manager **REFRESH** command.

The HP NonStop version of the **REFRESH** command is:

```
run uctl -refresh -host dallas
```

This example refreshes Universal Broker configuration on host **dallas**.

(For detailed information on Universal Control, see the Universal Products Utilities 3.2.0 User Guide.)

The **REFRESH** command directs Universal Broker to take the following actions:

| Step | Procedure |
|---|---|
| Step 1 | Read its configuration file **UBRCFG**.<br>Universal Broker refreshes the following configuration options:<br>• MESSAGE_LANGUAGE<br>• RUNNING_MAX |
| Step 2 | Read all component definitions found in the component definition subvolume. The Broker replaces all component definitions with the newly read component definitions. New component definitions are added and deleted component definitions are removed. |
| Step 3 | Read the Universal Access Control List configuration file **UACLCFG**.<br>The Broker replaces its UACL entries with the newly read entries. |

## 7.3 Component Management

Universal Broker is aware only of Universal Products components that have been defined. It is the responsibility of Universal Broker to start, stop, and query these defined components.

One of the steps in the installation of a component is defining it to the local Universal Broker. These component definitions provide Universal Broker with the necessary information that it needs to manage the components.

### 7.3.1 Component Definitions

Component definitions are text files that define Universal Products components to the Universal Broker. All HP NonStop component definition files (EDIT files) are located in the component definition subvolume, **`$SYSTEM.UNVCOMP`**.

The syntax of a component definition file is the same as the Universal Broker configuration file. See Section 2.2.3 Configuration Files Syntax for detailed syntax information.

Table 7.2, below, identifies all of the options that comprise Universal Products for HP NonStop component definitions.

Each **Option Name** is a link to detailed information about that component definition in the Universal Broker 3.2.0 Reference Guide.

| Option Name | Description |
|---|---|
| AUTOMATICALLY_START | Specification for whether or not the component automatically starts by the Universal Broker at start-up time or only on demand. |
| COMPONENT_NAME | Name by which clients know the component. |
| CONFIGURATION_FILE * | Component's configuration file name. |
| RUNNING_MAXIMUM | Maximum number of this component that can run simultaneously. |
| START_COMMAND * | Command that starts the component. |
| WORKING_DIRECTORY * | Path used as the working directory of the component. |
| * These options are required in the component definitions. | |

Table 7.2 Universal Productsr for HP NonStop - Component Definition Options

## 7.4 Security

Universal Broker is designed to be a secure system. As the level of security rises, so does the administrative complexity of the system. Universal Broker has balanced the two to avoid the administrative complexity with a minimum sacrifice to security.

Universal Broker security concerns are:

1. Access to Universal Broker files and subvolumes
2. Access to Universal Broker configuration files
3. User account with which the Universal Broker executes
4. Privacy and integrity of transmitted network data

### 7.4.1 File Permissions

At a minimum, only trusted user accounts should have write permission to the Universal Broker installation subdirectories and all files within. This most likely means only the administrators should have write access. For maximum security, only trusted accounts should have read, write or execute permissions to these subvolumes and files.

All files that the Broker creates or updates are located in either **`$SYSTEM.UNVLOG`** or **`$SYSTEM.UNVTRACE`**. The Broker does not need write access to its installation subvolume.

### 7.4.2 Configuration Files

Only trusted user accounts should have write, create or delete access to the Broker configuration files or the subvolume within which the configuration files exist.

### 7.4.3 Broker User ID

Universal Broker itself does not require **`super.super`** privileges. However, a component started by the Broker inherits the same user ID as the Broker, but the component can have different security requirements than the Broker. For example, Universal Command Server may require **`super.super`** authority. Since the component inherits its user ID from the Broker, either the Broker must be running as **`super.super`** or the UCMD Server program must be owned by **`super.super`** and ProdID must be set for the server program file.

If the Broker is started as a daemon at system startup time, it is started with a user ID of super.super. The Broker and all its components will then have sufficient authority.

## 7.4.4 Universal Access Control List

Universal Broker uses the Universal Access Control List (UACL) file as an extra layer of security. The UACL file contains Broker entries that contain Access Control List (ACL) rules that permit or deny access to the Broker. See Section 2.8 Universal Access Control List for details on the Universal Access Control List feature.

The Broker reads in the UACL entries when the program is started. If the UACL file is changed, the new entries can be activated by stopping and starting the Broker or by sending the Broker a Universal Control REFRESH command that will instruct the Broker to reread all its configuration files including the UACL file. The HP NonStop REFRESH command is run `uctl –refresh –host BROKER-IPADDR`.

(See the Universal Products Utilities 3.2.0 User Guide for complete details on Universal Control.)

## 7.4.5 UACL Entries

The syntax of a UACL entry file is the same as the Universal Broker configuration file. See Section 2.2.3 Configuration Files Syntax for detailed syntax information.

Table 7.3 identifies all Universal Broker for HP NonStop UACL entries. Each **UACL Entry Name** is a link to detailed information about that UACL entry in the Universal Broker 3.2.0 Reference Guide.

| UACL Entry Name | Description |
|---|---|
| UBROKER_ACCESS | Allows or denies access to Universal Broker services. |

Table 7.3 Universal Broker for HP NonStop - UACL Entries

## 7.4.6 UACL Examples

The following set of rules permit connections for the subnet 10.20.30 and denies all other connections.

```
ubroker_access    10.20.30.,allow
ubroker_access    ALL,deny
```

The following set of rules permit connections from host 10.20.30.40 and 10.20.30.50 and denies all other connections.

```
ubroker_access    10.20.30.40,allow
ubroker_access    10.20.30.50,allow
ubroker_access    ALL,deny
```

# Chapter 8
# Database Administration

## 8.1 Overview

This chapter provides information about Universal Broker 3.2.0 database administration.

## 8.2 Product Database Recovery

Universal Product databases, on operating system’s other than OS/400, are implemented using Oracle's Berkeley Database product. Recovering from a corrupted database consists of dumping the corrupted database to a file and reloading it into the database file.

Database can become corrupted if the system or address space that is managing the databases ends abnormally. A Universal Product program that utilizes databases should not be terminated abnormally.

Abnormal methods of termination include:

- z/OS CANCEL or FORCE command.
- UNIX SIGKILL signal.
- Windows process termination through the Task Manager.

### 8.2.1 Database Backups

Database recovery is not a replacement for database backups. If the data maintained by the product in the database has long term value, the databases must be periodically backed up.

## 8.3 Universal Broker Databases

Universal Broker uses databases to maintain component information, configuration information and event data. If a database becomes corrupted, it will prevent the Broker from running.

Database recovery procedures depend partly on the operating system on which the Broker is executing. The following sections describe the procedures for each operating system.

### 8.3.1 z/OS

The Universal Broker started task must be down to perform database recovery. A backup of either the database file being recovered or the entire HFS or zFS data set should be created before recovery is attempted.

A sample database recovery job is provided in member **`UBRDBREC`** in the **`SUNVSAMP`** library. The job uses the Universal Database Utilities to dump and reload a database file.

All databases are located in the HFS or zFS product data set **`#HLQ.UNV.UNVDB`**. The HFS or zFS data set must be mounted prior to running **`UBRDBREC`**. Refer to the Universal Products 3.2.0 Installation Guide for information on mounting the HFS or zFS data set, if necessary.

The user ID with which the recovery job runs requires appropriate permissions to the root directory of the HFS or zFS data set and to the database file. Write access is required to the directory and read and write access is required to the database file.

Customize **`UBRDBREC`** to meet local JCL and installation requirements. Specify the database file name to recover on the PARM keyword of the **`EXEC`** statement of both steps (the dump and load steps). When all modifications are complete, submit the job. All steps should end with return code 0.

## 8.3.2 UNIX

The Broker daemon must be down to perform database recovery. A backup of either the database file being recovered or the entire directory should be created before recovery is attempted.

A sample database recovery script is provided in file **`ubrdbrec`** in the **`/opt/universal/ubroker/bin`** directory. The script uses the Universal Database Utilities to dump and reload a database file.

The default location of all Universal Broker databases is the **`/var/opt/universal/spool`** directory.

The user ID with which the recovery script runs requires appropriate permissions to the database directory and to the database file. Write access is required to the directory and read and write access is required to the database file.

The **`ubrdbrec`** script accepts an optional argument: the database file name to recover. If no database file name is specified, the **`ues.db`** database is recovered. The script ends with exit code 0 if successful and a non-zero exit code if it failed.

## 8.3.3 Windows

The Broker service must be stopped to perform database recovery. A backup of either the database file being recovered or the entire directory should be created before recovery is attempted.

A sample database recovery batch file is provided in file **`ubrdbrec.bat`** in the **`"\Program Files\Universal\UBroker\bin"`** directory. The batch file uses the Universal Database Utilities to dump and reload a database file.

The default location of all Universal Broker databases is directory "**`\Program Files\Universal\spool\ubroker"`**.

The user ID with which the recovery script runs requires appropriate permissions to the database directory and to the database file. Write access is required to the directory and read and write access is required to the database file.

The **`ubrdbrec.bat`** batch file accepts an optional argument: the database file name to recover. If no database file name is specified, the **`ues.db`** database is recovered. The batch file ends with exit code 0 if successful and a non-zero exit code if it failed.

### 8.3.4 OS/400

The Universal Broker subsystem, **UNVUBR320** (by default), must be down in order to perform database recovery. Use standard OS/400 database recovery procedures and attempt to restart the Universal Broker subsystem.

If the problem persists, restore the failing database file. The entire Universal Spool file library may be required if restoring individual files fails to correct the problem. As a last resourt, delete all files in the Universal Spool file library and restart **UNVUBR320**.

Deleting the files from the Universal Spool library will result in loss of all data stored in those files, including spooled output for Manager Fault Tolerant jobs. All affected jobs may need to be re-run.

# Appendix A
# Customer Support

Stonebranch, Inc. provides customer support, via telephone and e-mail, for Universal Broker and all Universal Products.

## TELEPHONE

Customer support via telephone is available 24 hours per day, 7 days per week.

**North America**

**(+1) 678 366-7887, extension 6**

**(+1) 877 366-7887, extension 6  [toll-free]**

**Europe**

**+49 (0) 700 5566 7887**

## E-MAIL

**All Locations**

**support@stonebranch.com**

Customer support contact via e-mail also can be made via the Stonebranch website:

**www.stonebranch.com**

950 North Point Parkway, Suite 200

Alpharetta, Georgia 30005

U.S.A.