Universal Command Agent for SOA

User Guide

Universal Products

Version 3.2.0

ucasoa-user-3203

Universal Command Agent for SOA

User Guide

Universal Products 3.2.0

Document Name	Universal Command Agent for SOA 3.2.0 User Guide				
Document ID	ucasoa-user-3203				
Products	z/OS	UNIX	Windows	OS/400	HP NonStop
Universal Command Agent for SOA		√	√		

Stonebranch Documentation Policy

Stonebranch, Inc.
950 North Point Parkway, Suite 200
Alpharetta, GA 30005 USA
Tel: (678) 366-7887
Fax: (678) 366-7717

SAP Certified Integration

Summary of Changes

Changes for Universal Command Agent for SOA 3.2.0 User Guide (ucasoa-user-3203) October 30, 2009

Universal Command Agent for SOA 3.2.0.4

- Added support of MQ protocol:
  - MQ in Section 1.2.1 Supported Protocols.
  - Added MQ Connector in Section 1.3.4 Universal Command Agent for SOA Connector Overview.
  - Added Section 2.3 Outbound MQ Configuration – MQ Client Jar Files.
  - Added the following MQ command options in Chapter 3 Usage:
    - MQ_CHANNEL
    - MQ_HOST
    - MQ_PORT
    - MQ_PROPERTIES_FILE
    - MQ_QUEUE_MANAGER_NAME
    - MQ_QUEUE_NAME
    - MQ_REPLY_TO
  - Added Section 4.6 MQ Connector Operation.

Changes for Universal Command Agent for SOA 3.2.0 User Guide (ucasoa-user-3202) September 8, 2009

- Moved the Command Options chapter to Chapter 2 Command Options in the newly created Universal Command Agent for SOA Reference Guide.

Changes for Universal Command Agent for SOA 3.2.0 User Guide (ucasoa-user-3201) September 5, 2008

- Added toll-free telephone number for North America in Appendix A Customer Support.
- Added information about vendor-specific `properties.xml` file in Section 2.2.1 Using the properties.xml File.

Changes for Universal Command Agent for SOA 3.2.0 User Guide (ucasoa-user-320) May 16, 2008

- This is the first version of the Universal Command Agent for SOA 3.2.0 User Guide.

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.

Conventions

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

Typeface and Fonts

This document provides tables that identify how information is used. These tables identify values and/or rules that are either pre-defined or user-defined:

- *Italics* denotes user-supplied information.
- **Boldface** indicates pre-defined information.

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

Vendor References

References may be made in 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 may be used:

- **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.

These names do not imply software support in any manner.

Command Options Syntax Diagrams

Command Options 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 command options or values that are optional.
{ }	Encloses command 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.

Windows

This text pertains specifically to the Windows line of operating systems.

This text resumes the information pertaining to all operating systems.

Document Organization

The document is organized into the following chapters:

- Universal Command Agent for SOA Overview (Chapter 1)
  Overview of Universal Command Agent for SOA.
- Configuration (Chapter 2)
  Information on configuring Universal Command Agent for SOA.
- Usage (Chapter 3)
  Information on the use of Universal Command Agent for SOA.
- Operations (Chapter 4)
  Detailed information about Universal Command Agent for SOA operations.
- Troubleshooting (Chapter 5)
  Solutions to common problems that could be encountered using Universal Command Agent for SOA.
- Customer Support (Appendix A)
  Customer support contact information for Universal Command Agent for SOA.

Chapter 1 Universal Command Agent for SOA Overview

This chapter provides a general overview of Universal Command Agent for SOA 3.2.0. It is intended for the first time Stonebranch user or the current Stonebranch user new to Internet and Message-based workload.

The following topics are discussed:

- Introduction
- Architecture
- Components
- Defined Ports

1.1 Introduction

Universal Command Agent for SOA extends the workload execution and management features of the Universal Command (UCMD) product set to Internet and message-based workload. The Internet and message-based protocols are supported by the HTTP Connector, the SOAP Connector, the JMS Connector, and the MQ Connector.

In addition, you can execute compute or batch workload in the WebSphere XD environment using the XD Connector.

Universal Command Agent for SOA enables you to:

1. Consolidate your Internet and message-based workload within your current Enterprise Scheduling environment.
2. Use your existing scheduler, or other workload management applications, along with your new or existing Universal Products.
3. Use your existing development, test, and production business processes.
4. Use a single point of workload execution that is not tied to specific vendor hardware or software platforms.

The following sections summarize the architecture, supported protocols, and components that make up Universal Command Agent for SOA.

1.2 Architecture

Universal Command Agent for SOA is based on a Light Weight Container Architecture (LWCA). This architecture, combined with the Federated architecture of the current Universal Products line, provide your enterprise with a loosely coupled, scalable, and secure solution to your enterprise workload management tasks.

1.2.1 Supported Protocols

Universal Command Agent for SOA supports synchronous and asynchronous communication for workload execution via the following four protocols: HTTP, SOAP, JMS, and MQ.

Synchronous communication requires that the calling party wait for a response from the target application before beginning the next task.

Asynchronous communication allows the calling party to move on the next task without waiting for a response (if there is one) from the target application. If there are responses to asynchronous requests, more effort is required to correlate the request to the reply, as they are two separate events. Most middleware and integration software operate in this manner.

HTTP

HTTP (**H**yper**T**ext **T**ransfer **P**rotocol) is the underlying protocol used by the World Wide Web. It is a synchronous (blocking) protocol, which means that the requestor waits for the response before executing another task.

HTTP uses the Request / Reply message pattern.

HTTP is one of the ways that you can execute remote workload such as CGI, servlet, or web service-based applications.

SOAP

SOAP (**S**imple **O**bject **A**ccess **P**rotocol) is a synchronous protocol for exchanging XML-based messages over computer networks, normally using HTTP / HTTPS. However, you can send SOAP messages over JMS, as well.

SOAP forms the foundation layer of the Web services stack, providing a basic messaging framework upon which abstract layers can be built.

There are several different types of messaging patterns in SOAP, but by far the most common is the Remote Procedure Call (RPC) pattern. In RPC, one network node (the client) sends a request message to another node (the server). The server immediately sends a response message to the client; that is, request / reply. SOAP is used predominantly to provide an interface to web service-based workload or legacy workload with a web service interface.

JMS

JMS (**J**ava **M**essage **S**ervice) defines the standard for reliable Enterprise Messaging and provides a reliable, flexible service for the asynchronous exchange of critical business data and events throughout an enterprise.

JMS uses both point-to-point (queue-based) and publish / subscribe (topic-based) messaging patterns. It is used extensively in middleware implementations and large J2EE application deployments.

MQ

IBM WebSphere MQ (**M**essage **Q**ueue) is a family of network communication software products launched by IBM in March, 1992.

It was previously known as MQSeries, a trademark that IBM rebranded in 2002 to join the suite of WebSphere products. WebSphere MQ, which users often refer to simply as "MQ," is IBM's Message Oriented Middleware offering. It allows independent and potentially non-concurrent applications on a distributed system to communicate with each other. MQ is available on a large number of platforms, both IBM and non-IBM

MQ uses both point-to-point (queue-based) and publish / subscribe (topic-based) messaging patterns. It is used extensively in IBM-based and legacy middleware implementations in mid-size and enterprise environments.

1.2.2 Supported Message Exchange Patterns

A message exchange pattern (MEP) describes the pattern of messages required by a communications protocol to establish or use a communication channel.

There are two major types of message exchange patterns:

- One-way: Publish or Listen (asynchronous)
- Request / Reply pattern (synchronous)

Universal Command Agent for SOA supports the Publish MEP and the Request / Reply MEP, as described in the following sections.

Publish MEP

The Publish MEP represents an asynchronous outbound workload execution event. This means that you can request execution of a workload using the JMS protocol to a target JMS provider.

Since JMS is queue-based, this outbound operation puts a message on the queue of the JMS provider. A process within the target application environment, such as a WebSphere container or middleware application, will read the message from the queue and execute the appropriate workload.

Technically, you can initiate a publish operation using the SOAP protocol, but it is still just a request / reply where the reply is treated as an acknowledgement similar to that of the TCP protocol.

Figure 1.1, below, illustrates a logical view of the Publish MEP.

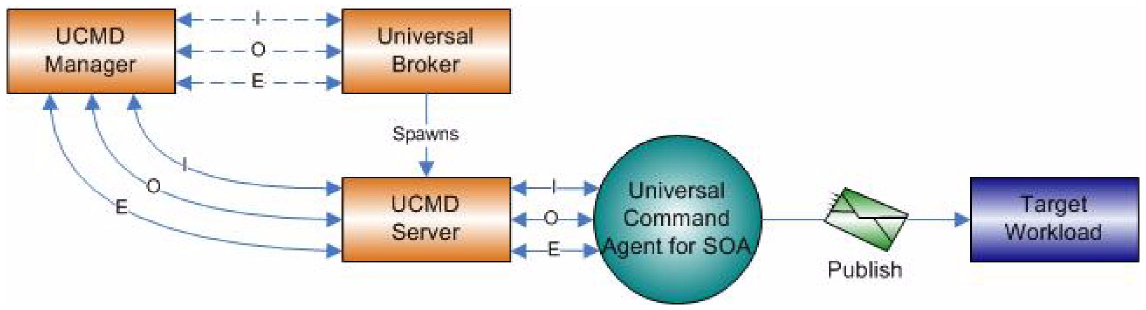


Figure 1.1 Publish MEP - Logical View

Request / Reply MEP

The Request / Reply MEP represents an outbound request to a target workload followed by an inbound reply from a target workload. This is a synchronous operation, as the calling party blocks, or waits, for the reply to come back before releasing its resources and moving on to the next task.

This is one of the most common message exchange patterns used. Every time you use a web browser to go to a website, you are initiating a request / reply operation where you request a page and the server replies with the page (or an error if it cannot find the page).

You can execute workload via the Request / Reply MEP using the HTTP, SOAP, JMS, or MQ protocols.

Figure 1.2, below, illustrates a logical view of the Request / Reply MEP.

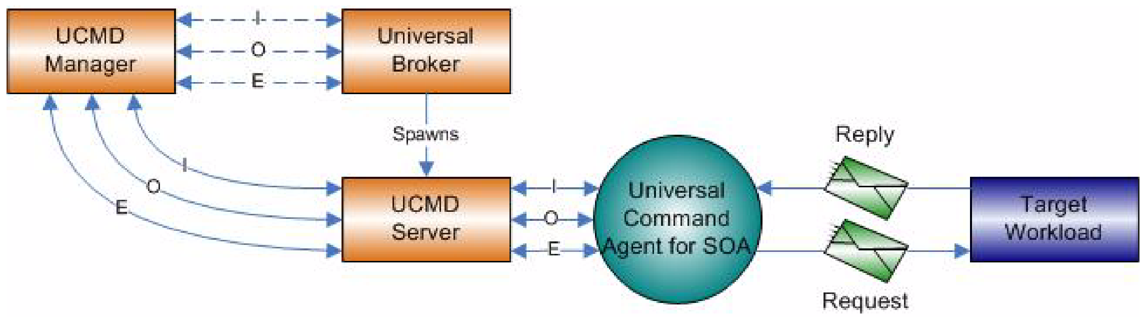


Figure 1.2 Request / Reply MEP - Logical View

1.3 Components

Universal Command Agent for SOA is made up of three major components:

- UAI (Universal Application Interface)
- UAC Server
- UAC (Universal Application Container)

These three components combine to create a powerful solution that spans domain boundaries and further enhances your ability to leverage your current assets.

Figure 1.3, below, illustrates the basic component flow for Universal Command Agent for SOA.

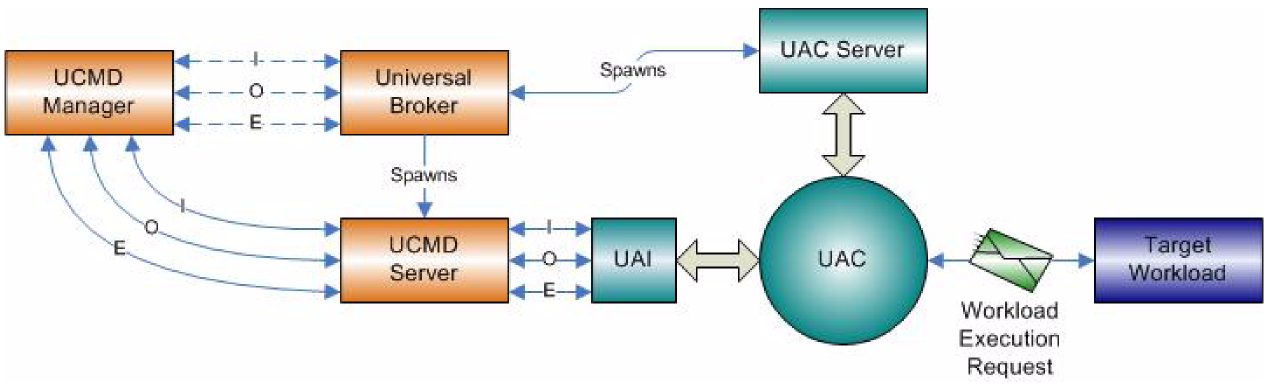


Figure 1.3 Base Component Flow - Logical View

As you can see, Universal Command Agent for SOA gets its input from Universal Command through STDIN. When the parameters and data are passed in, the workload execution request is processed and any return data is passed back to Universal Command.

The following sections provided information on each component of Universal Command Agent for SOA.

1.3.1 UAI - Universal Application Interface

The Universal Application Interface (UAI) component is the interface into Universal Command Agent for SOA and is considered to be the client to UAC.

UAI responsibilities include:

- Accept input parameters through STDIN or the command line interface and payload via STDIN.
- Validate the parameters and payload format.
- Build and send a workload request to UAC for execution.
- Return any application, payload, and error messages via STDOUT or STDERR.

UAI is a non-resident process that is invoked by the UCMD Server. UAI terminates itself once the workload request is complete. SInce the UCMD Server treats UAI as a user job, UAI is subject to all the rights and benefits to which any user job executed by UCMD would be entitled.

(See the Universal Command 3.2.0 User Guide for more details.)

The communication between UAI and UAC is via SOAP messaging over HTTPS with UAI blocking until UAC responds with a reply from the workload.

Figure 1.4, below, summarizes the basic process flow for UAI.

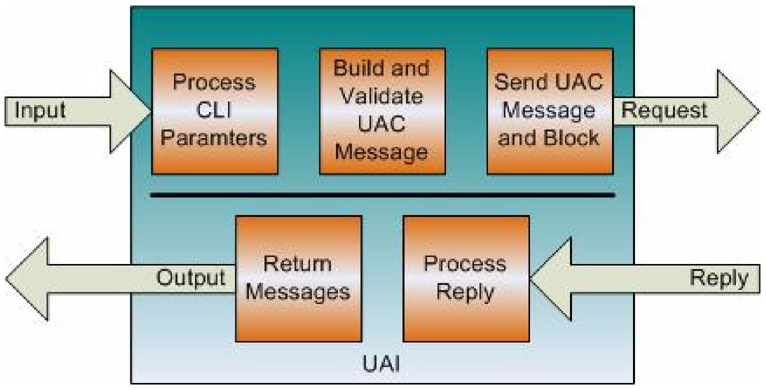


Figure 1.4 UAI Process Flow - Logical View

1.3.2 UAC Server

The UAC Server component is the interface between the Universal Broker and UAC. It provides operation and configuration control of UAC, as well as an interface to the Brokers message mechanisms.

Specifically, the UAC Server lets you:

- Start UAC.
- Stop UAC.
- Manage configuration.

1.3.3 UAC - Universal Application Container

The Universal Application Container (UAC) component executes the workload request and provides the server functions associated with a container environment.

Its responsibilities include the following:

- Provide a scalable and secure platform foundation for Universal Command Agent for SOA.
- Provide Publish, Listen, and Request / Reply functionality.
- Provide a deployment environment for the connectors.
- Provide persistence and fault tolerance mechanisms.
- Provide auditing, logging, and error handling functionality.
- Provide an interface for remote operations for Universal Broker.
- Can process multiple UAI requests.

UAC is a resident process that is started by Universal Broker and stays resident until stopped by Universal Broker. UAC receives and processes the message from UAI. It passes the data in the message to the appropriate connector, which then builds the message and initiates the requested workload operation.

The reply may consist of a simple acknowledgement that the workload started or completed, or it may contain application messages or workload output. In either case, UAC passes the reply message back to UAI unaltered. The exceptions to this are the SOAP faults, which are mapped to error messages, which then are passed to UAI. The requests and replies are persisted as well for fault tolerant operations.

Figure 1.5, below, summarizes the basic process flow for UAC.

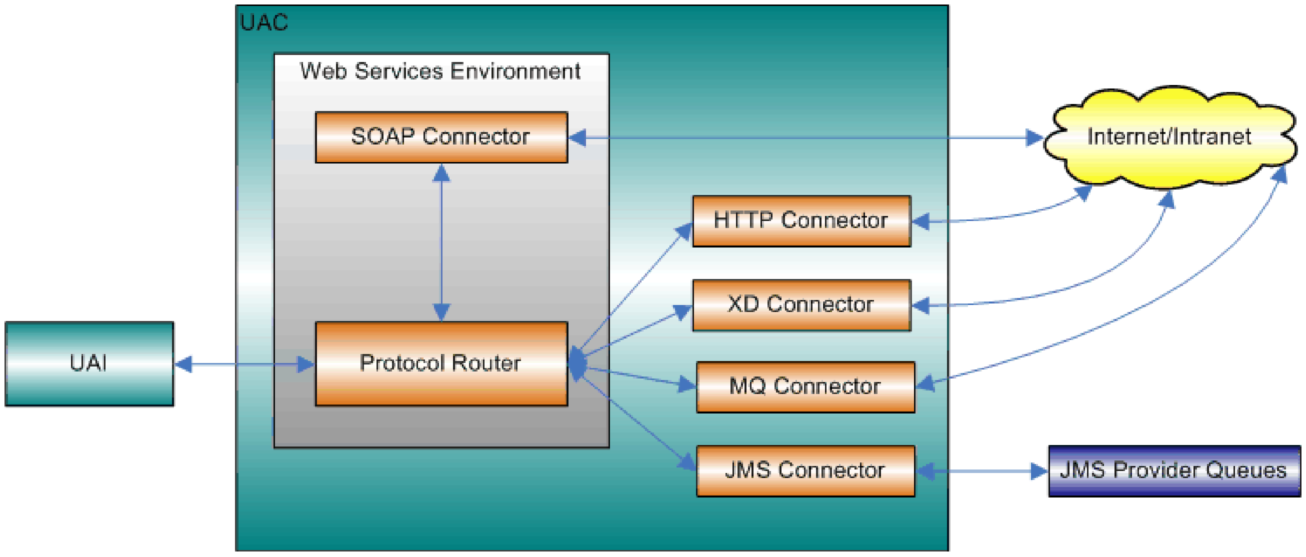


Figure 1.5 UAC Process Flow - Logical View

1.3.4 Universal Command Agent for SOA Connector Overview

The work of transforming the command line and STDIN input to the appropriate protocol message falls to the connectors that are deployed in the UAC environment. The Universal Command Agent for SOA platform allows for the addition of connectors to support future business requirements.

A summary of the current connectors follows.

HTTP Connector

The HTTP connector supports workload execution via the HTTP protocol.

It is a synchronous request / reply component that supports the following features:

- Supports HTTP 1.0 and 1.1 specifications.
- Supports Secure Socket Layer (SSL) and Transport Layer Security (TLS) protocols.
- Supports authentication via Basic, Digest, and NTLM.
- Supports GET and POST operations with Form data.

SOAP Connector

The SOAP connector supports workload execution via the SOAP protocol.

It is a synchronous request / reply component that supports the following features:

- Supports the SOAP 1.1 specification
- Supports Publish, Request / Reply Inbound, and Request / Reply message exchange patterns

JMS Connector

The JMS connector supports workload execution via the JMS protocol using synchronous and asynchronous communication.

It supports the following features:

- Supports the JMS 1.1 specification.
- Supports Publish, Subscribe, and Request / Reply message exchange patterns.
- Supports Queue- and Topic-based operations.

XD Connector

The XD Connector supports workload execution within the WebSphere Extended Deployment environment using synchronous communication via the SOAP protocol.

It supports the following features:

- Submit jobs to WebSphere XD.
- Restart jobs to WebSphere XD.
- Cancelling jobs.
- Pass back return code, job output, and application messages.
- Supports Request / Reply message exchange pattern.

MQ Connector

The MQ connector supports workload execution via the MQ messaging protocol using synchronous and asynchronous communication.

It supports the following features:

- Supports Publish, Subscribe, and Request / Reply message exchange patterns.
- Supports Queue-based operations.

1.4 Defined Ports

Universal Command Agent for SOA uses a specific set of ports (see Table 1.1, below).

Keep in mind that these ports are used by Universal Command Agent for SOA internally and to the target workload.

Port Number	Component	Description
7843	UAI to UAC	Default SSL port – used for secure communication between UAI and UAC.
7880	Target Workload to UAC	Default HTTP port – used for SOAP inbound operations initiated from external workload.
7899	UAC Server to UAC	Default RMI port – used for remote configuration of UAC.

Table 1.1 Universal Command Agent for SOA - Defined Ports

Chapter 2 Configuration

2.1 Overview

Depending on which transaction scenario (MEP) you are using (and, in the case of JMS, what JMS Provider you are using), there are several operations that may need to be configured before you can use Universal Command Agent for SOA.

2.2 Outbound JMS Configuration using WAS

There is no outbound configuration needed for HTTP and SOAP outbound operations.

However, some configuration may be needed for JMS operations, depending on which JMS Provider you are using. This section explains what configuration is required if you are using IBM's WebSphere Application Server as your JMS provider.

Note: Each JMS Provider that currently is available has a different implementation of JMS. Check the documentation that comes with the product to understand what additional configuration may be needed.

Currently, Universal Command Agent for SOA: JMS Connector has been tested against Apache's ActiveMQ JMS provider and IBM's WebSphere Application Server, WebSphere Application Server Network Deployment, and WebSphere Application Server Extended Deployment.

2.2.1 Using the properties.xml File

This properties file is not specific to WebSphere; it could, for example, be named **`message.properties.xml`** or **`bob.properties.xml`**. In general, the name should reflect the system to which the properties pertain. If you are using BEA as your JMS Provider, you might want to call it **`bea.properties.xml`**.

Also note that this properties file is an XML file, with a very simple format. Its purpose is to set properties to be passed to the JMS connection or JMS message, depending on whether **`jms.initialcontext`** or **`jms.hearder`** is used. The format of the properties is in name / value pairs.

A vendor-specific **`properties.xml`** file should be located in the **`Universal/UAI`** directory of the UAC install.

Note: If the **`properties.xml`** file is vendor-specific, the JMS_PROPERTIES_FILE option must be used in order for the file to be in effect.

For WebSphere, you must specify the class for the IBM ORB in order for the client jar files to process the message before UAC sends it to the specified WebSphere queue or topic.

Specifically, the values to set are:

- Name - set to **`jms.initialcontext.com.ibm.CORBA.ORBInit`**.
- Value - set to **`com.ibm.ws.sib.client.ORB`**.

Figure 2.1, below, illustrates a sample **`properties.xml`** file. Remember, these values are specific to the JMS Provider that you are using.

Note: You can set additional JMS properties using this same format.

```
<?xml version="1.0" encoding="UTF-8"?>
<sb:JMSProperties xmlns:sb="http://com.stonebranch/uai/JMSProperties"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://com.stonebranch/uac/JMSProperties
JMSProperties.xsd ">
  <sb:Property>
    <sb:Name>jms.initialcontext.com.ibm.CORBA.ORBInit</sb:Name>
    <sb:Value>com.ibm.ws.sib.client.ORB</sb:Value>
  </sb:Property>
</sb:JMSProperties>
```

Figure 2.1 properties.xml File Sample

For the JMS Request / Reply operation, you must specify the reply-to queue name:

- Name - set to **`jms.header.JMSReplyTo`**.
- Value - set to **`jms/IntegrationTestQueue1`** or the appropriate queue name.

Figure 2.2, below, illustrates a sample operation.

Note: Only the property element is shown; it could be included in the sample illustrated in Figure 2.1.

```
<sb:Property>
  <sb:Name>jms.header.JMSReplyTo</sb:Name>
  <sb:Value>jms/IntegrationTestQueue1</sb:Value>
</sb:Property>
```

Figure 2.2 JMS Request/Reply Operation Sample

2.2.2 JMS Provider Client Jar Files for Outbound

As is the case for inbound, you must have the JMS provider client jar files for outbound or request/reply operations as well. Since each JMS provider implementation is vendor-specific, you must acquire the client jar files that allow third-party applications to connect and communicate with your JMS provider.

For example, if you are using the JMS functions in IBM's WebSphere Application Server, you need the **`sibc.jms.jar`**, **`sibc.jndi.jar`**, and **`sibc.orb.jar`** files.

Note: If you are running WebSphere on AIX, you need the **`sibc.jms.jar`** and **`sibc.jndi.jar`** files only.

You would place the JMS provider client jar files in the following location:

Linux

```
/opt/universal/uac/container/webapps/axis2/WEB-INF/lib
```

Windows

```
\Program Files\Universal\uac\container\webapps\axis2\WEB-INF\lib
```

The names of the jar files differ depending on which JMS provider you are using.

The Universal Command Agent for SOA: JMS Connector does not provide the queue or topic infrastructure. You must have a JMS provider with queues or topics configured to use the JMS outbound or request / reply operations.

2.3 Outbound MQ Configuration – MQ Client Jar Files

As is the case for inbound, you must have the IBM MQ client jar files for outbound or request / reply operations.

You would place the MQ client jar files in the following location:

Linux

```
/opt/universal/uac/container/webapps/axis2/WEB-INF/lib
```

Windows

```
\Program Files\Universal\uac\container\webapps\axis2\WEB-INF\lib
```

You will need the following jar files:

- `com.ibm.mq.commonservices.jar`
- `com.ibm.mq.jar`
- `com.ibm.mq.pcf.jar`
- `com.ibm.mq.headers.jar`
- `com.ib.mq.jmqi.jar`
- `connector.jar`

The Universal Command Agent for SOA: MQ Connector does not provide the queue or topic infrastructure. You must have a WebSphere MQ Message Broker with queues configured to use the MQ outbound or request/reply operations.

2.4 Universal Configuration Manager

The Universal Configuration Manager is a Universal Products graphical user interface application that enables you to configure all of the Universal Products that have been installed on a Windows operating system.

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

2.4.1 Availability

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

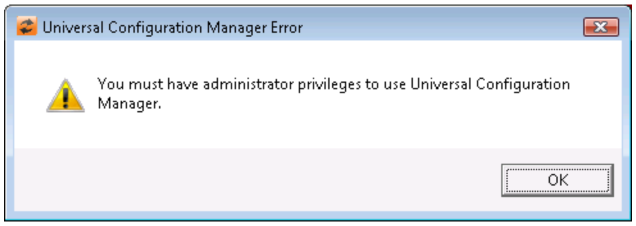
It is available to all user accounts in the Windows Administrator group.

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:

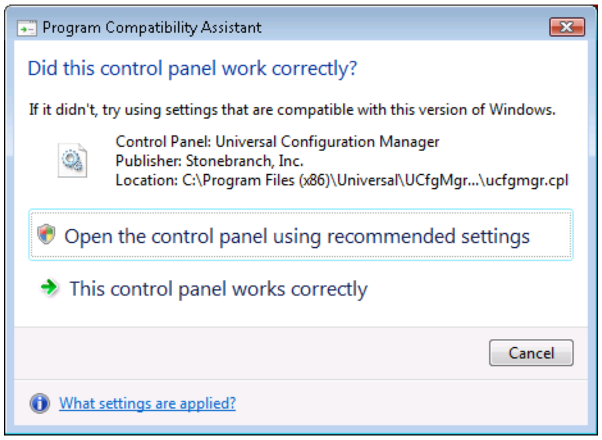


You must have administrator privileges to use Universal Configuration Manager.

Figure 2.3 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 2.4 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.

2.4.2 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 (seeFigure 2.5).

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.

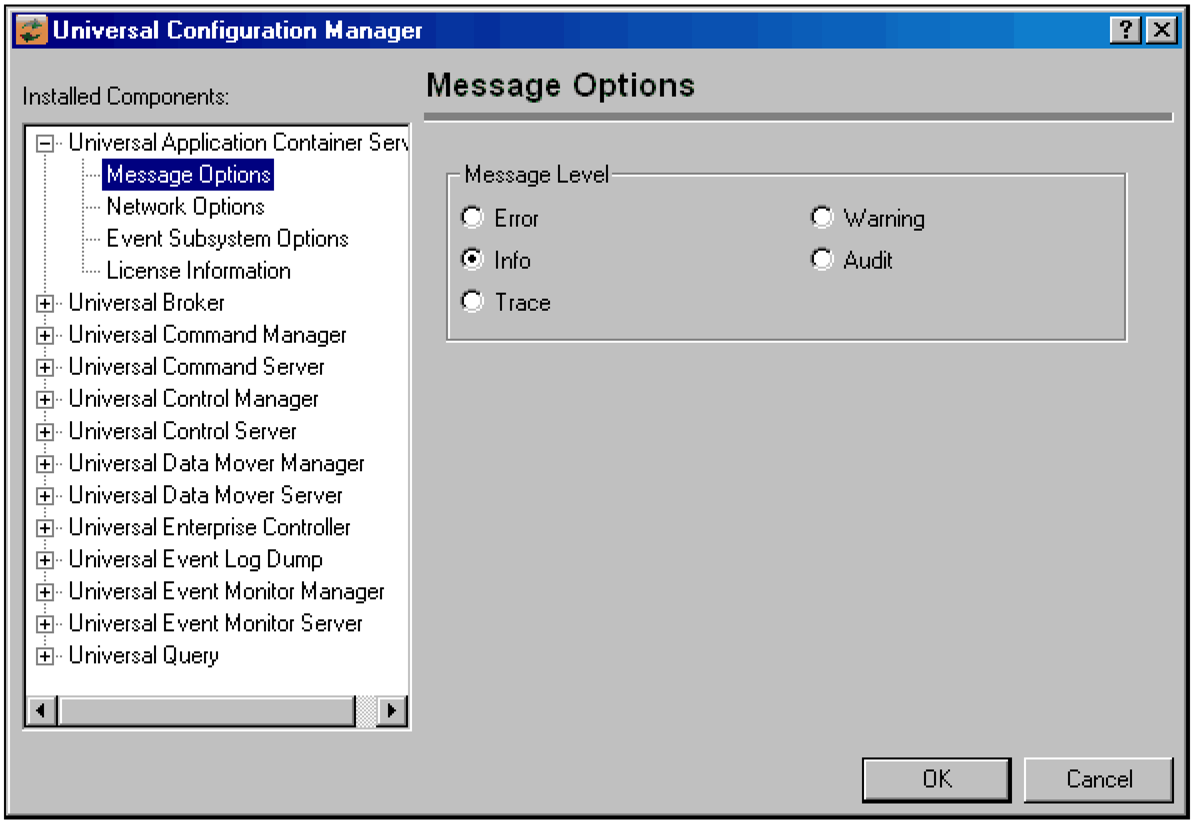
For example, the Windows XP Category View places the Universal Configuration Manager icon under the **`Other Control Panel Options`** link. Windows Vista and Windows Server 2008 place the icon within the **`Additional Options`** category.

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
OK
Cancel

Message Options

Message Level: Error [✓ Info] Trace Warning Audit

Figure 2.5 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.)

2.4.3 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 2.5, for example:

- List of property pages is displayed for Universal Broker.
- 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.

2.4.4 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 2.4.5 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.

2.4.5 Saving Data

To save all of the modifications / 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.

2.4.6 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 and later

The Universal Configuration Manager's context-sensitive help is a WinHelp file, which Windows Vista / Windows Server 2008 and later does 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.

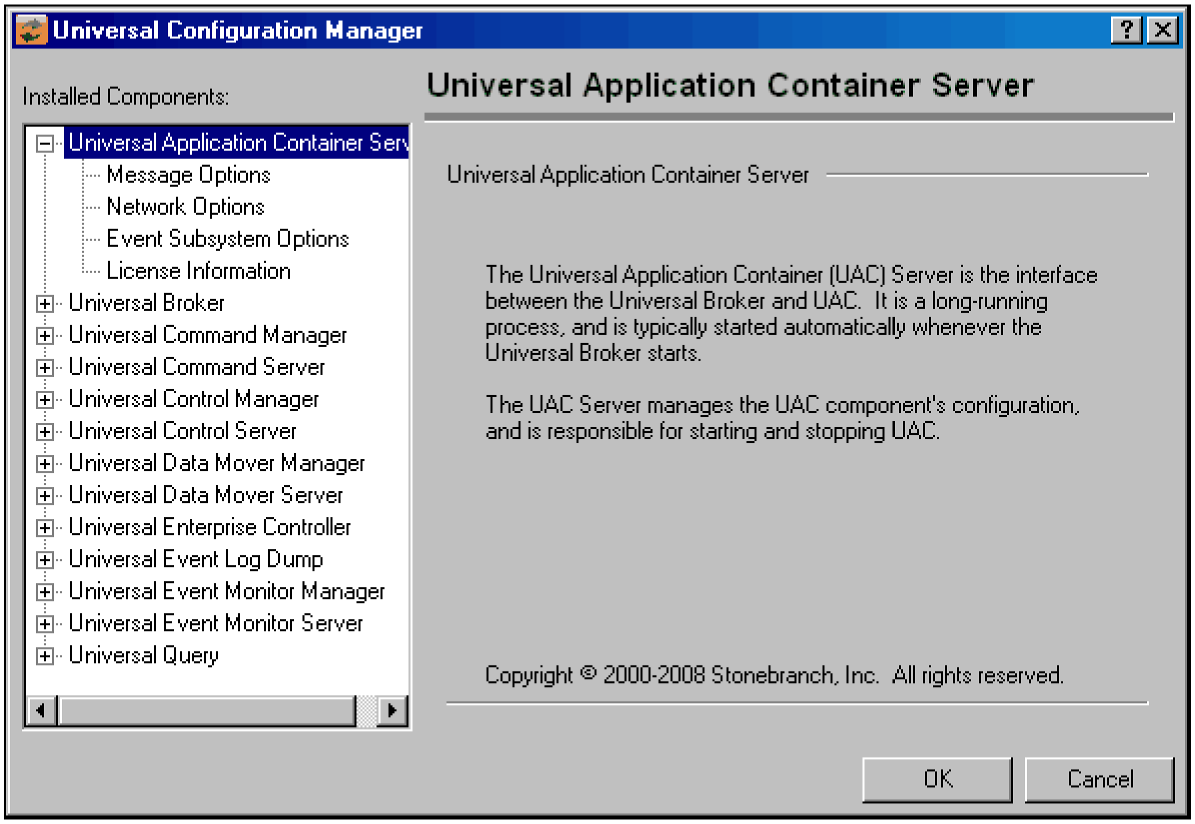
2.4.7 Universal Command Agent for SOA Installed Components

Universal Application Container Server

Figure 2.6 illustrates the Universal Configuration Manager screen for the Universal Application Container Server.

The Installed Components list identifies all of the UAC Server property pages.

The text describes the selected component, Universal Application Container Server.


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
Universal Application Container Server
Universal Application Container Server
OK
Cancel

The Universal Application Container (UAC) Server is the interface between the Universal Broker and UAC. It is a long-running process, and is typically started automatically whenever the Universal Broker starts.

The UAC Server manages the UAC component's configuration, and is responsible for starting and stopping UAC.

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

Figure 2.6 Universal Configuration Manager - UAC Server

Chapter 3
Usage

This chapter contains detailed information on the use of Universal Command Agent for SOA.

It provides the following topics:

- Command Options
- Message Payload
- Starting and Stopping

3.1 Command Options

Universal Command Agent for SOA uses a script file interface to accept the values needed to create the workload execution request.

3.1.1 Command Options Categories

Table 3.1, below, categorizes the command options into logical areas of application.

Category	Description
Required Options	Required for Universal Command Agent for SOA to process the workload execution request.
Dependent Options	Required, depending on the PROTOCOL option value; otherwise, these options are invalid.
Optional Options	Optional usage only; use only as appropriate.

Table 3.1 UEC Load Utility - Command Option Categories

The Universal Command Agent for SOA options for each category are summarized in the following tables. Each **Option Name** is a link to detailed information about that option in the Universal Command Agent for SOA 3.2.0 Reference Guide.

Required Options

Option Name	Description
MEP	Message exchange pattern to be used for the current operation.
PROTOCOL	Message protocol to be used for the current operation.
SERVICE_URL	URL (internet, network, or file-based) of the target workload.

Dependent Options

Option Name	Description
JMS_CONNECTION_FACTORY_NAME	Connection factory to be used to establish a connection to a JMS provider.
JMS_CONTEXT_FACTORY_NAME	Java class name of the JMS providers initial context factory.
JMS_DESTINATION	Name of the target JMS destination queue or topic for the JMS message.
JMS_REPLY_TO	Name of the JMS reply queue for the return JMS message.
MQ_CHANNEL	Name of the MQ channel.
MQ_HOST	Name of the server running MQSeries.
MQ_QUEUE_MANAGER_NAME	Name of the MQ QUEUE Manager.
MQ_QUEUE_NAME	Name of the MQ Queue to use.
MQ_REPLY_TO	Name of the MQ Queue from which to read the reply when MEP is set to request.
XD_CMD	Operation to submit to the WebSphere XD environment.
XD_CMD_ID	Correlates jobs.

Optional Options

Option Name	Description
HELP	Lists the command options and values.
HTTP_AUTH	http authorization scheme to use.
HTTP_FORM_DATA	Specification for whether or not there is HTTP form data, in a name-value format, in the payload file.
HTTP_METHOD	Type of HTTP operation to execute.
HTTP_VERSION	Version of the HTTP protocol to use.
JMS_PROPERTIES_FILE	Name and location of an XML document containing the JMS properties to be included in the JMS message.
MQ_PORT	Name of the port on which the MQ Broker is listening.
MQ_PROPERTIES_FILE	Name of the file containing the MQ name / value pairs.
SERVICE_PASSWORD	Password to be passed to the target workload for authentication.
SERVICE_USER_NAME	User name to be passed to the target workload for authentication.
SOAP_ACTION	soapAction HTTP header value.
SOAP_VERSION	Version of the SOAP protocol to use when making the SOAP request.
TIMEOUT_SEC	Length of time to wait for the request to complete.

3.1.2 Command Options Syntax

Figure 3.1, below, illustrates the syntax of Universal Command Agent for SOA command options.

```
-protocol {HTTP|SOAP|JMS|XDSOAP|MQ}
-mep {Publish|Request}
-serviceurl url
-jmsconnectionfactoryname name
-jmscontextfactoryname name
-jmsdestination name
-jmsreplyto name
-mqchannel channel
-mqhost server
-mqqueuemanagername manager
-mqqueuename queue
-mqreplyto queue
-xdcmd {SUBMIT|RESTART}
-xdcmdid ID *
[-httpauth {BASIC|DIGEST|NTLM}]
[-httpformdata {true|false}]
[-httpmethod {GET|POST}]
[-httpversion {OneDotZero|OneDotOne}]
[-jmspropertiesfile file]
[-mqport port]
[-mqpropertiesfile file]
[-serviceusername name]
[-servicepassword password]
[-soapaction header]
[-soapversion {OneDotOne|OneDotTwo}]
[-timeoutsec time]

-help
```

* The -xdcmdid option is required if -protocol is set to **XDSOAP**.

Figure 3.1 Command Options Syntax

3.2 Message Payload

The message payload contains the data required for the target workload to execute. This can include operation information, input parameters, authentication information, and any other data required by the target workload (application or service) to operate.

The payload file is an XML document that Universal Command Manager reads in through STDIN.

All HTTP and SOAP operations require a payload, while the JMS and MQ operations do not require a payload. The payload format is validated for HTTP, SOAP, and MQ messages because the payload is in **xml** format and must be parsed. There is no validation of the payload format for JMS messages because the payload is in **text** format. If you include **xml**-style elements and attributes in your JMS messages, it will be up to the target application to validate the format.

The content of the payload for all protocols is not validated because the payload represents business data; it is not the responsibility of Universal Command Agent for SOA to know about business details related to the workload that it is executing.

3.2.1 Example

Figure 3.2, below, illustrates an example of a basic message payload.

The first line contains:

- Name of the operation (in this case, **ValidateZip**)
- Location of the web service providing the operation (in this case, **http://webservicemart.com/ws/**.).

The second line contains:

- Tag for the value **ZipCode**.
- Actual value, **30004**, that the web service needs to operate.

The third line is the closing tag for the operation named in the first line (in this case, **ValidateZip**).

The other items, such as **tns** and **xmlns**, are namespace identifiers. In most cases, the application developers will provide you with the message payload.

```
<tns:ValidateZip xmlns:tns="http://webservicemart.com/ws/">
  <tns:ZipCode>30004</tns:ZipCode>
</tns:ValidateZip>
```

Figure 3.2 Message Payload - Basic

3.2.2 Response

Figure 3.3, below, illustrates the response to that the ValidateZip operation returns.

The first line indicates the type of data being returned (in this case, string data).

The second line contains the response from the ValidateZip web service operation. It includes:

- result - root element and indicates the start of the response data
  - code - success or error code from the HTTP transaction. A value of **"200"** indicates success.
- item - Element that defines the attributes returned in response to the ZipCode value submitted.
  - zip - ZIP code that was submitted as part of the request.
  - state - State in which the ZIP code is located.
  - latitude - Latitude of the ZIP code submitted.
  - longitude - Longitude of the ZIP code submitted.

The third line is the closing tag for the response message.

```
<string>
<result code="200"><item zip="30004" state="GA" latitude="34.11917"
  longitude="-84.30292"/></result>
<string>
```

Figure 3.3 Message Payload Response - Basic

3.3 Starting and Stopping

Universal Command Agent for SOA is started and stopped by Universal Broker via the UAC Server component.

There is no user interaction for this operation.

Chapter 4
Operations

4.1 Overview

Universal Command Agent for SOA allows you to execute Internet and message-based workload, in a variety of transaction scenarios, using five types of connectors.

1. HTTP Connector
2. SOAP Connector
3. JMS Connector
4. XD Connector
5. MQ Connector

In this chapter, operations are grouped by connector. Each connector supports both standard and combination message exchange patterns (MEPs). They detail the supported business scenarios and the usage required for each transaction scenario.

Table 4.1, below, identifies the transaction scenarios for each connector:

Connector	**Message Exchange Pattern (MEP)**	
	Request / Reply	**Publish**
HTTP Connector	√	
SOAP Connector	√	√
JMS Connector	√	√
XD Connector	√	
MQ Connector	√	√

Table 4.1 Connectors – Message Exchange Patterns

4.2 HTTP Connector Operation

The HTTP Connector is used for invoking synchronous workload that has, or is exposed via, an HTTP interface.

It supports the following message exchange pattern:

- Request / Reply

The types of workloads that might have an HTTP interface could include, but are not limited to:

Web Services

Your organization may have workload implemented using web services technologies that must be executed as part of a scheduled business process.

Servlets

Your organization may have workload implemented as servlets.

Servlets are objects that contain business logic. Access is via an HTTP URL that specifies the name of the servlet to execute. The servlet could process single transaction or batch records, and it usually has a specific responsibility in a scheduled business process.

CGI (Common Gateway Interface)

CGI provides a common way that application functionality can be accessed by web browsers using HTTP. Your organization may have business logic written, using various technologies with CGI as the interface, and you have to incorporate this into a scheduled business process.

Middleware

Middleware queues or processes often are exposed via an HTTP interface. The HTTP interface can be driven by your enterprise scheduler as part of a scheduled business process.

4.2.1 HTTP Connector Request / Reply Operation

There are two methods of HTTP Connector operation: GET and POST.

When the request is made, the reply from the target workload can be either of two types:

- **Reply with Acknowledgement**
  Request is acknowledged via the reply but no data is sent back. The target workload is executed with no additional feedback.
- **Reply with Payload**
  Request blocks until a reply is received from the target workload containing data, presumably after the target workload has completed or an error was issued during execution. The data can be the results of the workload, the workload status, or an error message.

Figure 4.1, below, illustrates the system flow for an HTTP Connector Request / Reply operation using Universal Command Agent for SOA.

The following list describes the steps (1 - 5) identified in Figure 4.1:

6. Universal Command is executed requesting the HTTP workload. The command options for Universal Command Agent for SOA: HTTP Connector are read in from a script file specified with the SCRIPT_FILE option and the message payload is read in via STDIN. UCMD then sends the workload request to Universal Command Agent for SOA (specifically, the UAI component).
7. Universal Command Agent for SOA receives the request from UCMD Server via STDIN. The UAI component validates the command options and existence of the message payload, sends the request to UAC, and blocks. UAC builds the workload execution message for the target workload.
8. Universal Command Agent for SOA: HTTP Connector sends the workload execution message via HTTP and blocks for the reply.
9. When the target workload completes, or aborts due to an error, it replies to the workload execution request with either return code and data or the error message.
10. UAC replies to UAI which returns the relevant information to UCMD.

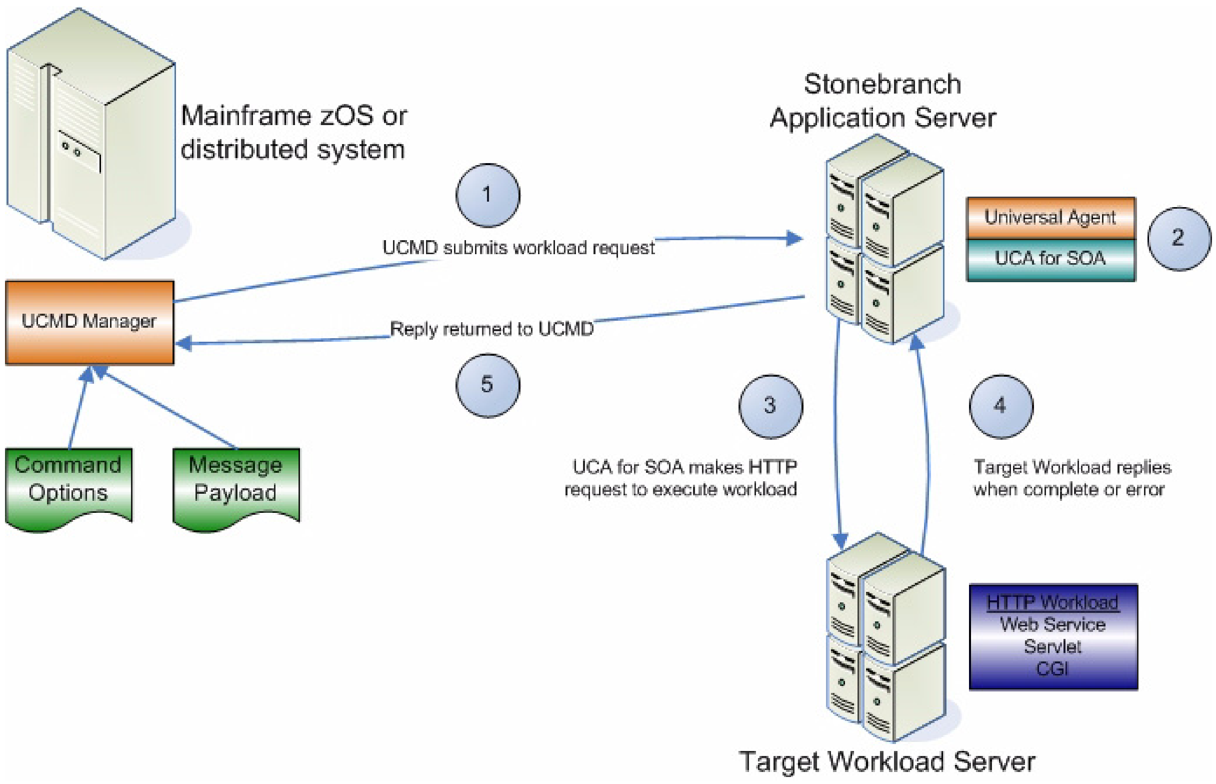

Mainframe zOS or distributed system
UCMD Manager
Command Options
Message Payload
1
UCMD submits workload request
Reply returned to UCMD
5
Stonebranch Application Server
Universal Agent
UCA for SOA
2
3
UCA for SOA makes HTTP request to execute workload
4
Target Workload replies when complete or error
HTTP Workload
Web Service
Servlet
CGI
Target Workload Server

Figure 4.1 HTTP Connector Request / Reply Operation

4.2.2 HTTP Connector Request / Reply Operation – Usage

Usage of Universal Command Agent for SOA is via the Universal Command (UCMD) Manager, with command input coming from a script file specified with the SCRIPT_FILE option.

Figure 4.2, below, illustrates the Universal Command options to execute the HTTP request.

```
-s HTTPPost_RequestReply_Options.txt
-script_type SERVICE
-host server1
-login YES
-userid abc
-pwd 123
```

Figure 4.2 HTTP Request – Universal Command Options

Figure 4.3, below, illustrates the script file to request the HTTP service.

```
-protocol HTTP
-mep Request
-serviceurl http://server1:8889/testService
-timeoutsec 10
```

Figure 4.3 HTTP Script File

Note: The script file illustrated in Figure 4.3 is the argument to the **-s** (SCRIPT_FILE) option for Universal Command shown in Figure 4.2.

The command options shown in Figure 4.2 can be saved in a file and invoked with Universal Command via the **-f** (COMMAND_FILE_PLAIN) option, as illustrated in Figure 4.4, below.

(See the Universal Command 3.2.0 User Guide for more details.)

```
ucmd -f HTTPPostRequestReply_Invoke.txt < HTTPSOAPRequest.xml
```

Figure 4.4 UCMD HTTP Command Line

4.2.3 HTTP Connector Request / Reply Operation – Command Options

Table 4.2, below, identifies the options (and their values) that are required to initiate an HTTP Connector Request/Reply operation.

See Section Optional Options for details on additional HTTP options.

Option	Value	Description
PROTOCOL	HTTP	Connector that UAC will use for the current operation.
MEP	Request	Specification that the operation will be a request/reply operation.
SERVICE_URL	Workload URL	Address of the target workload in the form of: http://machine:port/service_name

Table 4.2 HTTP Request/Reply Options

4.3 SOAP Connector Operation

The SOAP Connector is used for invoking synchronous workload that has, or is exposed via, a SOAP interface.

It supports the following message exchange patterns:

- Request / Reply
- Publish

The SOAP Connector Publish operation is not widely supported. It is dependent on the implementation of the target workload.

The types of workloads that might have a SOAP interface are similar to the HTTP workload and could include, but are not limited to:

Web Services

Your organization may have workload implemented, or wrapped other workload such as legacy or HTTP workload, using web services technologies that need to be executed as part of a scheduled business process.

Middleware

Often times middleware queues or processes are exposed via a SOAP interface, especially in an environment where the web services stack is a major component of the SOA or IT architecture. The SOAP interface can be driven by your enterprise scheduler as part of a scheduled business process.

4.3.1 SOAP Connector Request / Reply Operation

The SOAP Connector operation is, by default, a request/reply operation with the same constraints as the HTTP operation.

When the request is made, the reply from the target workload can be either of two types:

Reply with Acknowledgement

In this type, the request is acknowledged via the reply, but no data is sent back. The target workload is executed with no additional feedback.

Reply with Payload

In this type, the request blocks until a reply is received from the target workload containing data, presumably after either:

- Target workload has completed.
- Error was issued during execution.

The data can be the results of the workload, the workload status, or an error message.

Figure 4.5, below, illustrates the system flow for a SOAP Connector Request / Reply operation using the Universal Command Agent for SOA: SOAP Connector.

The following list describes the steps (1 - 5) identified in Figure 4.5:

1. Universal Command is executed requesting the HTTP workload. The command options for the Universal Command Agent for SOA: SOAP Connector are read in from a script file specified with the SCRIPT_FILE option and the message payload is read in via STDIN. UCMD then sends the workload request to Universal Command Agent for SOA (specifically, the UAI component).
2. Universal Command Agent for SOA receives the request from UCMD Server via STDIN. The UAI component validates the command options and existence of the message payload, sends the request to UAC, and blocks. UAC builds the workload execution message for the target workload.
3. Universal Command Agent for SOA: SOAP Connector sends the workload execution message via SOAP and blocks for the reply.
4. When the target workload completes, or aborts due to an error, it replies to the workload execution request with either return code and data or the error message.
5. UAC replies to UAI, which returns the relevant information to UCMD.

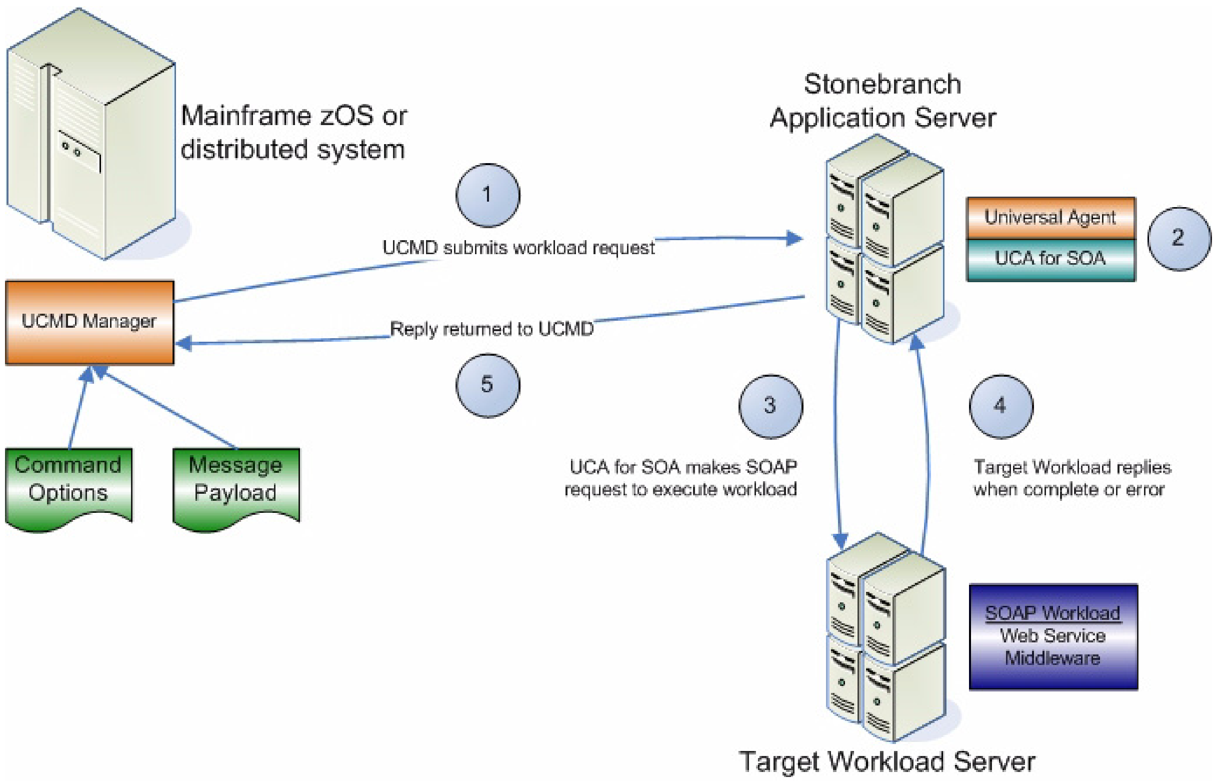

Mainframe zOS or distributed system
Stonebranch Application Server
1
UCMD submits workload request
Universal Agent
UCA for SOA
2
UCMD Manager
Reply returned to UCMD
5
3
4
Command Options
Message Payload
UCA for SOA makes SOAP request to execute workload
Target Workload replies when complete or error
SOAP Workload Web Service Middleware
Target Workload Server

Figure 4.5 SOAP Connector Request / Reply Operation

4.3.2 SOAP Connector Publish Operation

The SOAP Connector Publish operation is an extension of SOAP functionality that allows asynchronous communication using the SOAP protocol. Use of this MEP is highly dependant on the target workload, as target workload must behave in a manner more consistent with asynchronous messaging by not replying to the SOAP request.

Figure 4.6, below, illustrates the system flow for a SOAP Connector Publish operation using the Universal Command Agent for SOA: SOAP Connector.

The following list describes the steps (1 - 4) identified in Figure 4.6:

1. Universal Command is executed requesting the HTTP workload. The command options for the Universal Command Agent for SOA: SOAP Connector are read in from a script file specified with the SCRIPT_FILE option and the message payload is read in via STDIN. UCMD then sends the workload request to Universal Command Agent for SOA (specifically, the UAI component).
2. Universal Command Agent for SOA receives the request from UCMD Server via STDIN. The UAI component validates the command options and existence of the message payload, sends the request to UAC, and blocks. UAC builds the workload execution message for the target workload.
3. Universal Command Agent for SOA: SOAP Connector publishes the workload execution message via SOAP and the operation is complete.
4. The Universal Command Agent for SOA: SOAP Connector returns the acknowledgement to UCMD.

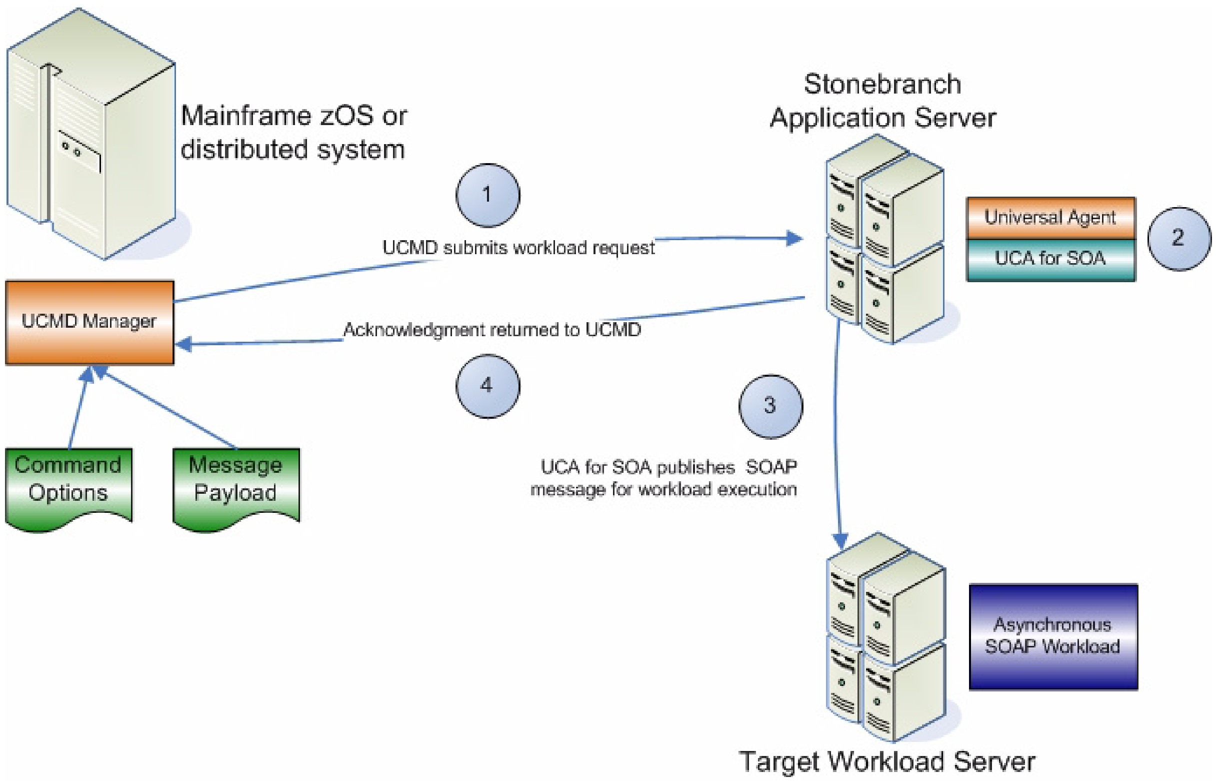

Mainframe zOS or distributed system
Stonebranch Application Server
1
UCMD submits workload request
Universal Agent
UCA for SOA
2
UCMD Manager
Acknowledgment returned to UCMD
4
3
Command Options
Message Payload
UCA for SOA publishes SOAP message for workload execution
Asynchronous SOAP Workload
Target Workload Server

Figure 4.6 SOAP Connector Publish Operation

4.3.3 SOAP Connector (Request / Reply or Publish) Operation – Usage

Usage of Universal Command Agent for SOA is via the Universal Command (UCMD) Manager, with command input coming from a script file specified with the SCRIPT_FILE option.

Figure 4.7, below, illustrates the Universal Command options to execute the SOAP operation.

```
-s REMOTE_SOAP_Options.txt
-script_type SERVICE
-host server1
-login YES
-userid abc
-pwd 123
```

Figure 4.7 SOAP Operation - Universal Command Options

Figure 4.8, below, illustrates the script file.

```
-protocol SOAP
-mep Request
-serviceurl http://www.webservicemart.com/uszip.asmx
-soapaction http:/webservicemart.com/ws/ValidateZip
-timeoutsec 10
```

Figure 4.8 SOAP Script File

Note: The script file illustrated in Figure 4.8 is the argument to the **-s** (SCRIPT_FILE) option for Universal Command shown in Figure 4.7. For the publish operation, the value for **-mep** would be **Publish**.

The command options shown in Figure 4.7 can be saved in a file and invoked with Universal Command via the **-f** (COMMAND_FILE_PLAIN) option, as shown in Figure 4.9, below.

(See the Universal Command 3.2.0 User Guide for more details.)

```
ucmd -f REMOTE_SOAP_Invoke.txt < zipcode.xml
```

Figure 4.9 UCMD SOAP Command Line

4.3.4 SOAP Connector Request / Reply Operation – Command Options

Table 4.3, below, identifies the options (and their values) that are required to initiate a SOAP Request/Reply operation.

See Optional Options for details on additional SOAP options.

Option	Argument	Description
PROTOCOL	SOAP	Connector that UAC will use for the current operation.
MEP	Request	Specification that the operation will be a request/reply operation.
SERVICE_URL	Workload URL	Address of the target workload in the form of: http://machine:port/service_name

Table 4.3 SOAP Connector Request / Reply Options

4.3.5 SOAP Connector Publish Operation – Command Options

Table 4.4, below, identifies the options (and their values) that are required to initiate a SOAP Publish operation.

See Optional Options for details on additional SOAP options.

Option	Value	Description
PROTOCOL	SOAP	Connector that UAC will use for the current operation.
MEP	Publish	Specification that the operation will be a request/reply operation.
SERVICE_URL	Workload URL	Address of the target workload in the form of: http://machine:port/service_name

Table 4.4 SOAP Connector Publish Options

4.4 JMS Connector Operation

The JMS Connector is used for invoking asynchronous workload that has, or is exposed via, a JMS interface

It supports the following message exchange patterns:

- Publish
- Request / Reply

The types of workload that might have a JMS interface are message-based workloads that are associated with enterprise messaging environments.

A JMS workload could include, but is not limited to:

Application Container Interfaces

Your organization may have asynchronous workload deployed to application containers such as WebSphere, BEA, JBoss AS, or Oracle AS (and many others). These containers provide JMS services, such as queues and topics, that allow access to the deployed workload by your enterprise scheduler or other applications. This allows them to be included as part of your scheduled business processes.

Middleware

Middleware workload and processes are often asynchronous and are exposed via JMS queues or topics by the middleware software. They usually are the main interface for messaging operations. Using the JMS interface, the middleware workload, processes, and downstream targets of the middleware can be driven by your enterprise scheduler as part of a scheduled business process.

Universal Command Agent for SOA: JMS Connector does not provide the queue or topic infrastructure. You must have a JMS provider with queues or topics configured to use the JMS Connector operations.

4.4.1 JMS Provider Client Jar Files for Outbound

As mentioned in Section 2.2.2 JMS Provider Client Jar Files for Outbound, setup and use of the JMS Connector is dependant on the JMS provider being used.

Each JMS provider is specific to its vendor implementation; thus, it will have vendor specific setup and configuration that needs to take place before you can run JMS operations. Specifically, the Universal Command Agent for SOA: JMS Connector requires the JMS provider client jar files to connect and communicate with the JMS provider.

When you have acquired these client jar files from your JMS provider vendor, you would place them in the following directory:

Linux

```
opt/universal/uac/container/webapps/axis2/WEB-INF/lib
```

Windows

```
\Program Files\Universal\uac\container\webapps\axis2\WEB-INF\lib
```

For example, if you are using the JMS functions in IBM's WebSphere Application Server, you would need the `sibc.jms.jar`, `sibc.jndi.jar`, and `sibc.orb.jar` files.

If you were using Apache's ActiveMQ JMS provider you would need the `apcache-activemq-4.1.1.jar` file.

4.4.2 JMS Connector Request / Reply Operation

The JMS Connector Request / Reply operation is a synchronous operation that uses a temporary queue to process the reply.

Figure 4.10, below, illustrates the system flow for a JMS request / reply operation using the Universal Command Agent for SOA: JMS Connector.

The following list describes the steps (1 - 5) identified in Figure 4.10:

1. Universal Command is executed requesting the HTTP workload. The command options for Universal Command Agent for SOA: JMS Connector are read in from a script file specified with the SCRIPT_FILE option and the message payload is read in via STDIN. UCMD then sends the workload request to Universal Command Agent for SOA (specifically, the UAI component).
2. Universal Command Agent for SOA receives the request from UCMD Server via STDIN. The UAI component validates the command options and existence of the message payload, sends the request to UAC, and blocks. UAC builds the workload execution message for the target workload.
3. Universal Command Agent for SOA: JMS Connector publishes the workload execution message to the specified destination queue.
4. Universal Command Agent for SOA: JMS Connector then reads the reply message off of the temporary reply queue specified in the properties file.
5. UAC returns the reply message to UCMD (or an error message, if the operation failed).

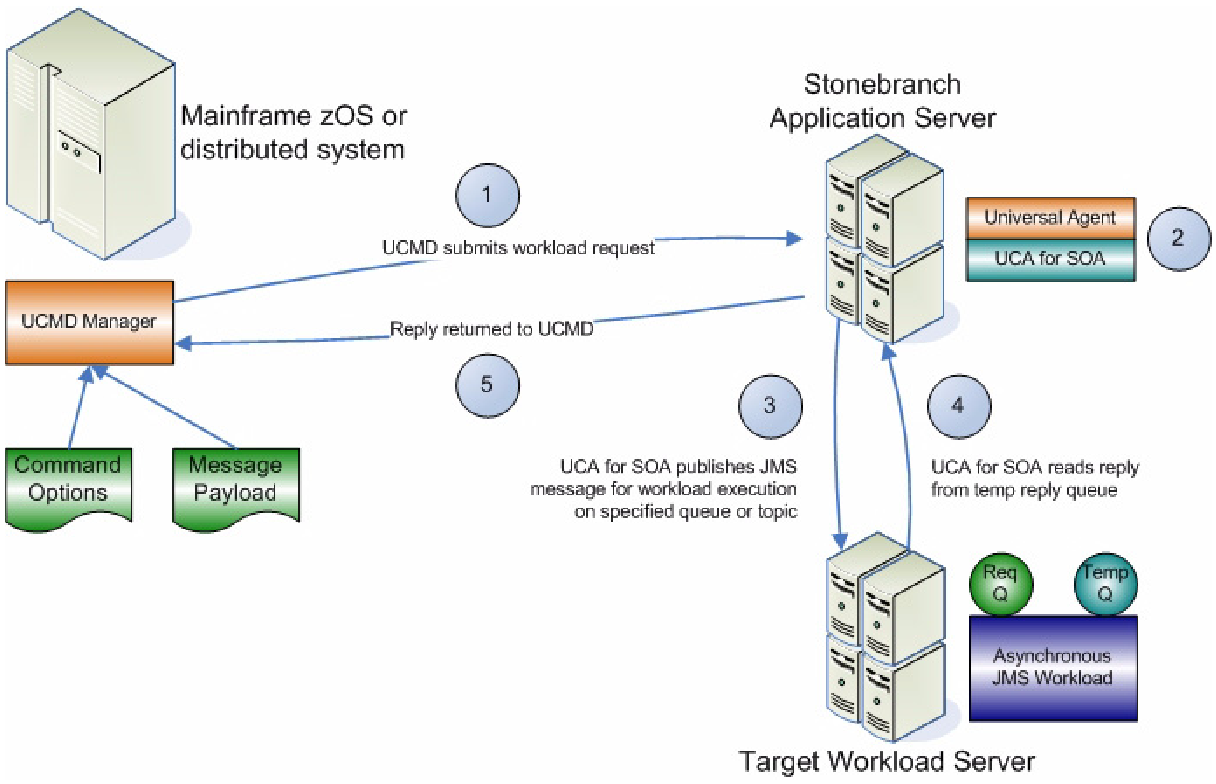

Mainframe zOS or distributed system
UCMD Manager
Command Options
Message Payload
1
UCMD submits workload request
Reply returned to UCMD
5
Stonebranch Application Server
Universal Agent
UCA for SOA
2
3
UCA for SOA publishes JMS message for workload execution on specified queue or topic
4
UCA for SOA reads reply from temp reply queue
Req Q
Temp Q
Asynchronous JMS Workload
Target Workload Server

Figure 4.10 JMS Connector Request / Reply Operation

4.4.3 JMS Connector Publish Operation

The JMS Connector Publish operation is an asynchronous operation that places a JMS message and its payload on the specified destination JMS queue or topic.

UAC returns a message indicating whether the JMS message was successfully placed on the queue or sent on the topic.

Figure 4.11, below, illustrates the system flow for a JMS publish operation using the Universal Command Agent for SOA: JMS Connector.

The following list describes the steps (1 - 4) identified in Figure 4.11:

1. Universal Command is executed requesting the HTTP workload. The command options for Universal Command Agent for SOA: JMS Connector are read in from a script file specified with the SCRIPT_FILE option and the message payload is read in via STDIN. UCMD then sends the workload request to Universal Command Agent for SOA, specifically the UAI component.
2. Universal Command Agent for SOA receives the request from UCMD Server via STDIN. The UAI component validates the command options and existence of the message payload, sends the request to UAC, and blocks. UAC builds the workload execution message for the target workload.
3. Universal Command Agent for SOA: JMS Connector publishes the workload execution message to the specified queue or topic.
4. UAC returns a success message if the message was placed on the queue or topic with no error, or an error message if there was an error. This reply is generated by UAC, not the JMS provider.

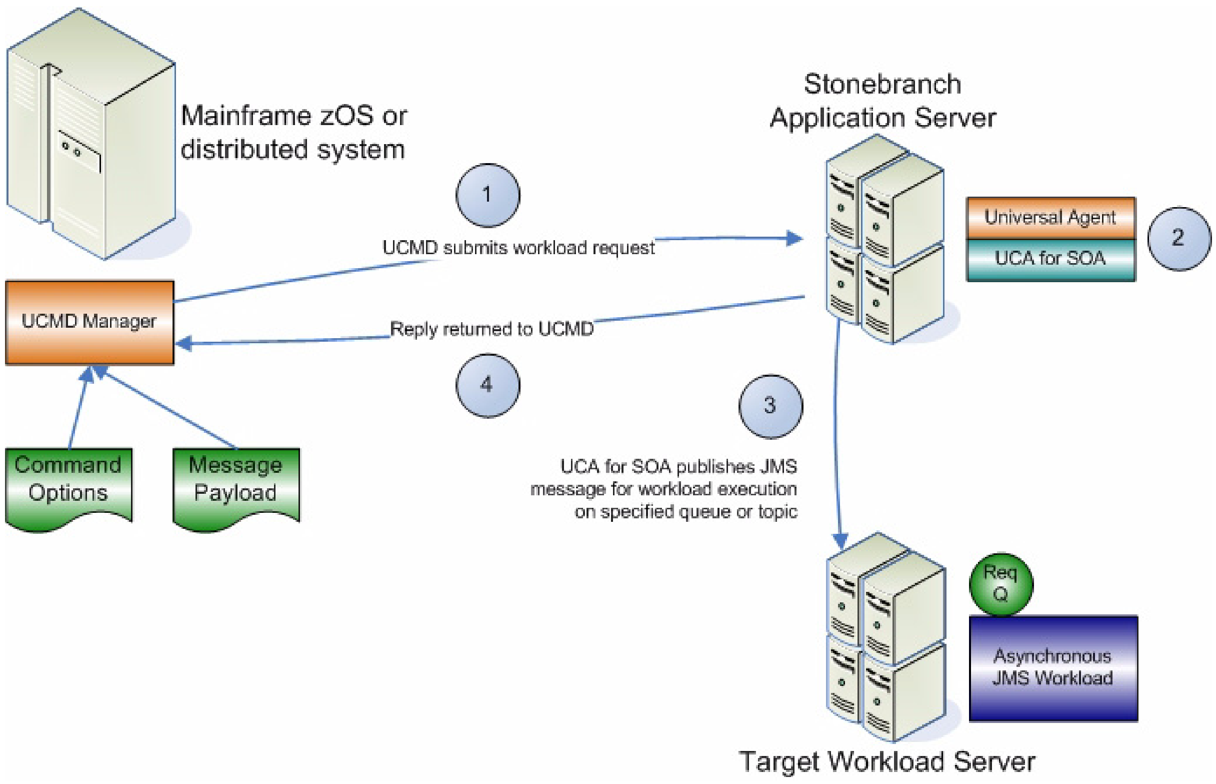

Mainframe zOS or distributed system
Stonebranch Application Server
1
UCMD submits workload request
Universal Agent
UCA for SOA
2
UCMD Manager
Reply returned to UCMD
4
3
Command Options
Message Payload
UCA for SOA publishes JMS message for workload execution on specified queue or topic
Req Q
Asynchronous JMS Workload
Target Workload Server

Figure 4.11 JMS Connector Publish Operation

4.4.4 JMS Connector Request / Reply Operation – Usage

Figure 4.12, below, illustrates the Universal Command options to execute the JMS Connector Request / Reply operation.

```
-s JMSRequestReply_Queues_Options.txt
-script_type SERVICE
-host server1
-login YES
-userid abc
-pwd 123
```

Figure 4.12 JMS Connector Request / Reply – Universal Command Options

Figure 4.13, below, illustrates the script file.

```
-protocol JMS
-mep Request
-serviceurl iiop://server1:2809
-jmsdestination jms/IntegrationTestQueue1
-jmsconnectionfactoryname jms/ConnectionFactory
-jmscontextfactoryname com.ibm.websphere.naming.WsnInitialContextFactory
-jmspropertiesfile xml/websphere.properties.xml
```

Figure 4.13 Script File - JMS Request/Reply

Note: The script file illustrated in Figure 4.13 is the argument to the **-s** (SCRIPT_FILE) option for Universal Command shown in Figure 4.12.

The command options shown in Figure 4.12 can be saved in a file and invoked with UCMD via the **-f** (COMMAND_FILE_PLAIN) option, as shown in Figure 4.14, below.

(See the Universal Command 3.2.0 User Guide for more details.)

```
ucmd -f JMSRequestReply_Queues_Options.txt < JMSPayload.xml
```

Figure 4.14 UCMD JMS Request / Reply Command Line

4.4.5 JMS Connector Publish Operation – Usage

Usage of Universal Command Agent for SOA: JMS Connector is via the Universal Command (UCMD) Manager, with command input coming from a script file specified with the SCRIPT_FILE option.

Note: Because the protocol is JMS, you must use the dependent command options in addition to the standard command options (see Dependent Options).

Figure 4.15, below, illustrates the Universal Command options to execute the JMS Connector Publish operation.

```
-s OutboundJMS_Queues_Options.txt
-script_type SERVICE


-host server1
-login YES
-userid abc
-pwd 123
```

Figure 4.15 JMS Connector Publish – Universal Command Options

Figure 4.16, below, illustrates the script file.

```
-protocol JMS
-mep Publish
-serviceurl iiop://server1:2809
-jmsdestination jms/SBSTestQueue1
-jmsconnectionfactoryname jms/SBSConnectionFactory
-jmscontextfactoryname com.ibm.websphere.naming.WsnInitialContextFactory
-jmspropertiesfile xml/websphere.properties.xml
```

Figure 4.16 Script File - JMS Publish

Note: The script file illustrated in Figure 4.16 is the argument to the **-s** (SCRIPT_FILE) option for Universal Command shown in Figure 4.15.

The command options shown in Figure 4.15 can be saved in a file and invoked with Universal Command via the **-f** (COMMAND_FILE_PLAIN) option, as shown in Figure 4.17, below.

(See the Universal Command 3.2.0 User Guide for more details.)

```
ucmd -f OutboundJMS_Queues_Invoke2.txt < JMSPayLoad.xml
```

Figure 4.17 UCMD JMS Publish Command Line

4.4.6 JMS Connector Request / Reply Operation – Command Options

Table 4.5, below, describes the options (and their values) required to initiate a JMS Request / Reply operation.

(See Optional Options for details on additional options.)

Option	Value	Description
PROTOCOL	JMS	Connector UAC will use for the current operation.
MEP	Request	Specification that the operation will be a request/reply operation.
SERVICE_URL	Workload URL	Address of the JMS provider in the form of: http://machine:port/service_name
JMS_DESTINATION	JMS Destination	Target queue or topic configured in the JMS provider.
JMS_CONNECTION_FACTORY_NAME	JMS Connection Factory	Name of the connection factory configured in the JMS provider including the jndi prefix.
JMS_CONTEXT_FACTORY_NAME	Class Name	Class name of the initial context factory used by the JMS provider.
JMS_PROPERTIES_FILE	Path/Filename	Path and filename of the JMS properties file that contains the values for the JMS properties in the JMS message. JMS providers and your target workload may have different, or no, requirements for additional properties. (Optional)

Table 4.5 JMS Connector Request / Reply - Command Options

4.4.7 JMS Connector Publish Operation – Command Options

Table 4.6, below, describes the options (and their values) required to initiate a JMS Connector Publish operation.

See Optional Options for details on additional options.

Option	Value	Description
PROTOCOL	JMS	Connector that UAC will use for the current operation.
MEP	Publish	Specification that the operation will be a publish operation.
SERVICE_URL	Workload URL	Address of the JMS provider in the form of: http://machine:port/service_name
JMS_DESTINATION	JMS Destination	Target queue or topic configured in the JMS provider.
JMS_CONNECTION_FACTORY_NAME	JMS Connection Factory	Name of the connection factory configured in the JMS provider including the jndi prefix.
JMS_CONTEXT_FACTORY_NAME	Class Name	Class name of the initial context factory used by the JMS provider.
JMS_PROPERTIES_FILE	Path/Filename	Path and filename of the JMS properties file that contains the values for the JMS properties in the JMS message. JMS providers and your target workload may have different, or no, requirements for additional properties. (Optional)

Table 4.6 JMS Connector Publish Operation - Command Options

4.5 XD Connector Operation

The XD Connector is used for invoking batch workload in the WebSphere XD environment.

It is a synchronous component that uses the following message exchange pattern:

- Request / Reply

The types of workload that the XD Connector can invoke are constrained to the WebSphere XD environment and the jobs that are defined in it. This includes both compute intensive and batch workload deployed to WebSphere XD.

Submit, restart, and cancel operations are supported with the submit and restart operations being initiated as arguments to the **`-xdcmd`** option and the cancel operation being initiated when the UCMD Manager process is terminated before the submit or restart operation has completed.

4.5.1 XD Connector Deployment

Deployment of the Universal Command Agent for SOA: XD Connector in a production environment includes host and distributed servers.

Host Server

The host server is where the Universal Command Manager is installed.

Distributed Server

There are two distributed servers in this environment:

1. Universal Agent
   Server where the Universal Broker / Server 3.2.0 and Universal Command Agent for SOA are installed and runs.
2. WebSphere Application Server
   Server where WebSphere Network Deployment v6.1 and WebSphere Extended Deployment (XD) components are installed and runs any operating system that WebSphere supports.

Figure 4.18, below, illustrates this environment.

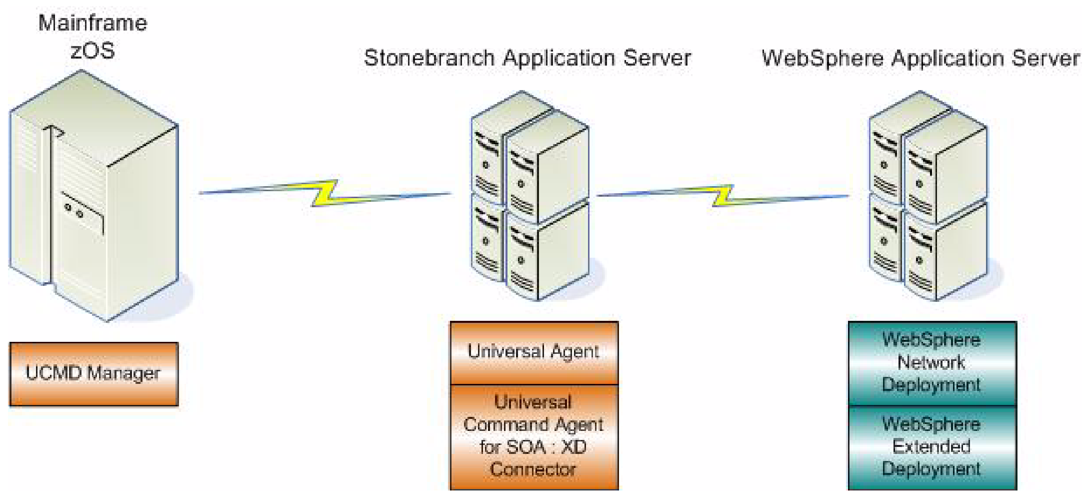

Mainframe
zOS
Stonebranch Application Server
WebSphere Application Server
UCMD Manager
Universal Agent
Universal Command Agent for SOA : XD Connector
WebSphere Network Deployment
WebSphere Extended Deployment

Figure 4.18 Deployment -- XD Connector

4.5.2 XD Connector System Flow

The XD Connector is designed to request workload execution, return status on executing workload, and return the job output and the job log, via the Web Services interface in the IBM WebSphere XD environment.

Figure 4.19, below, illustrate a more detailed logical flow of the XD connector operation.

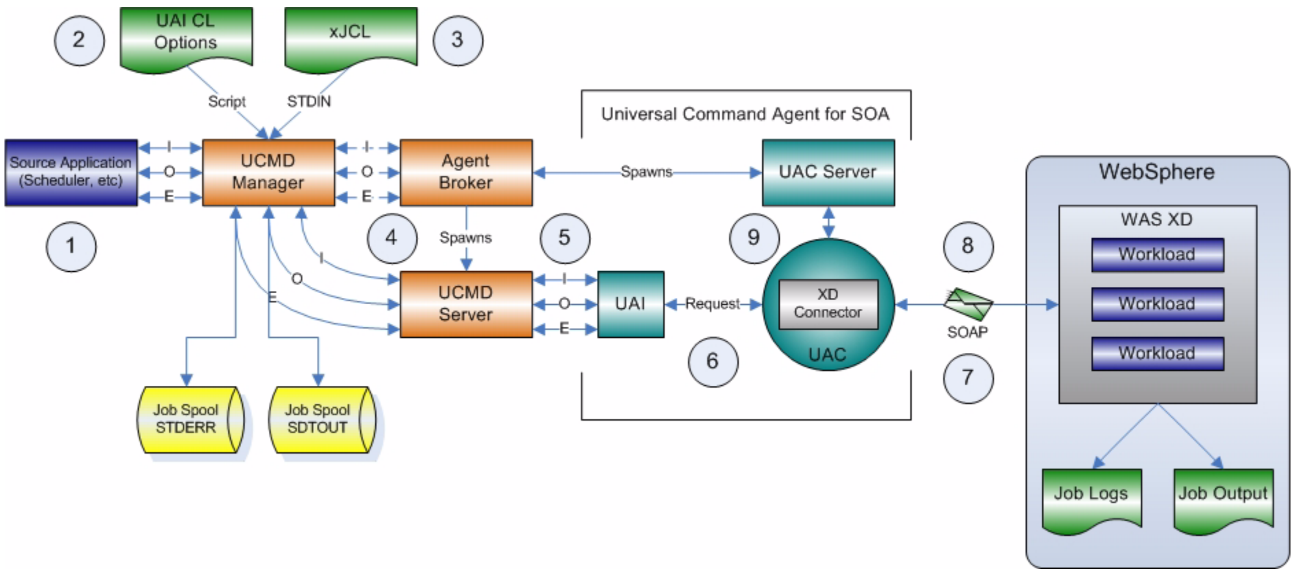


Figure 4.19 XD Connector System Flow – Logical View

The flow is described as follows (from left to right):

1. The scheduler invokes the UCMD Manager with the appropriate command line options.
2. The UCMD Manager reads the script file that contains that contains the XD Connector command options.
3. The UCMD Manager reads the xJCL file from STDIN, as indicated by the UCMD options.
4. The UCMD Manager requests action by the Broker, which spawns the UCMD Server and sets up communication between the UCMD Manager and UCMD Server.
5. The UCMD Server sends the XD Connector command options and xJCL to UAI via STDIN.
6. UAI validates the command options and builds a SOAP message containing the command options and xJCL and sends it to UAC. This is a request / reply operation, so UAI blocks for the reply.
7. UAC, based on the PROTOCOL value, will invoke the XD Connector. The XD Connector creates the XD SOAP message and sends it to WebSphere.
8. WebSphere replies to the XD Connector with the Job ID which will be needed for the status and job log operations for the current transaction.

9. The XD Connector initiates the status operation. When a success or error status is received, the return code, the job output, and the job log are returned to UCMD Manager. At this point, the transaction is considered complete. Please note that the job output is returned on STDOUT and the job log is returned on STDERR.

Figure 4.20, below, illustrates the physical system flow of the XD Connector.

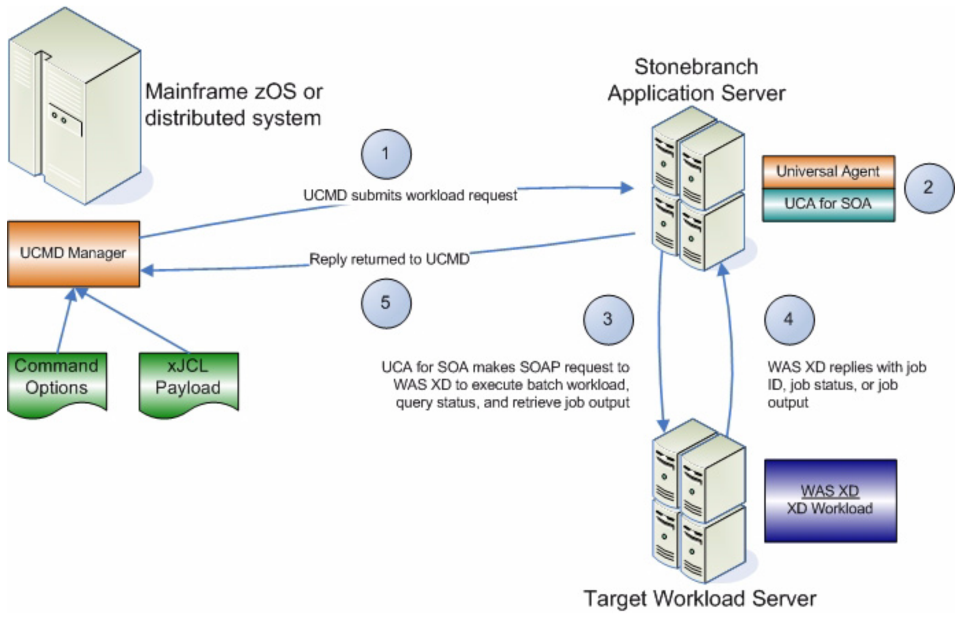

Mainframe zOS or distributed system
Stonebranch Application Server
1
UCMD submits workload request
Universal Agent
UCA for SOA
2
UCMD Manager
Reply returned to UCMD
5
3
4
Command Options
xJCL Payload
UCA for SOA makes SOAP request to WAS XD to execute batch workload, query status, and retrieve job output
WAS XD replies with job ID, job status, or job output
WAS XD
XD Workload
Target Workload Server

Figure 4.20 XD Connector – Request / Reply Operation

4.5.3 XD Connector Request / Reply Operation – Usage

Usage of the Universal Command Agent for SOA: XD Connector is via the Universal Command Manager, with command input coming from a script file specified with the SCRIPT_FILE option.

Figure 4.21, below, illustrates the Universal Command options to execute the XD Connector operation.

```
-s XDSOAP_Options.txt
-script_type SERVICE
-host server1
-login YES
-userid abc
-pwd 123
```

Figure 4.21 XD Connector Operation – Universal Command Options

Figure 4.22, below, illustrates the script file.

```
-protocol XDSOAP
-mep Request
-xdcmd SUBMIT
-xdcmdid 10001
-serviceurl http://wasxd-centos:
9080/LongRunningJobSchedulerWebSvcRouter/services/JobScheduler
-serviceusername abc
-servicepassword 123
-timeoutsec 120
```

Figure 4.22 Script File - XD

Note: The script file illustrated in Figure 4.22 is the argument to the **-s** option for Universal Command shown in Figure 4.21.

The command options shown in Figure 4.21 can be saved in a file and invoked with Universal Command via the **-f** (COMMAND_FILE_PLAIN) option, as shown in Figure 4.23, below.

(See the Universal Command 3.2.0 User Guide for more details.)

```
ucmd -f XDSOAP_Invoke.txt < Your_xJCL_File_Here.xml
```

Figure 4.23 UCMD XDSOAP Command Line

4.5.4 XD Connector Request / Reply Operation – Command Options

Table 4.7, below, describes the options (and their values) required to initiate an XD Request / Reply operation.

(See Optional Options for details on additional SOAP options.)

Option	Value	Description
PROTOCOL	XDSOAP	Connector that UAC will use for the current operation.
MEP	Request	Specification that the operation will be a Request / Reply operation.
SERVICE_URL	Workload URL	Address of the JMS provider in the form of: http://machine:port/service_name For XD operations, this should be the long running scheduler web service.
XD_CMD	SUBMT, RESTART	Specification for whether you are either: • Submitting a job to the XD environment. • Restarting a job in the XD environment.
XD_CMD_ID	Job Identifier	Value passed in from the mainframe request and is used to correlate the mainframe request with the job ID that is passed back from WebSphere XD after the job is submitted and if a restart of the submitted job is required.

Table 4.7 XD Connector Request / Reply Operation – Command Options

4.5.5 Cancelling an XD Operation

The Cancel operation is unique in that it is initiated by a UCMD Manager termination event and not as an argument to the **`-xdcmdid`** option.

To cancel a job that is running, the UCMD Manager process must be terminated, in which case the XD Connector sends a cancel job request to the XD environment. To verify that the XD job has been cancelled, and what the job's status is (cancelled, ended, or restartable), you must log into the XD Job Management Console and select View jobs.

4.6 MQ Connector Operation

The MQ Connector is used for invoking asynchronous workload that has, or is exposed via, an MQ interface.

It supports the following message exchange patterns:

- Publish
- Request / Reply

The types of workload that might have an MQ interface are message-based workloads that are associated with enterprise messaging environments.

An MQ workload could include, but is not limited to:

Application Container Interfaces

Your organization may have asynchronous workload deployed to application containers such as WebSphere or an MQ Series Message Broker. These environments provide MQ services, such as queues and topics, that allow access to the deployed workload by your enterprise scheduler or other applications. This allows them to be included as part of your scheduled business processes.

Middleware

Middleware workload and processes are often asynchronous and are exposed via MQ queues or topics by the middleware software. They usually are the main interface for messaging operations. Using the MQ interface, the middleware workload, processes, and downstream targets of the middleware can be driven by your enterprise scheduler as part of a scheduled business process.

Universal Command Agent for SOA: MQ Connector does not provide the queue or topic infrastructure. You must have an MQ Broker with queues or topics configured to use the MQ Connector.

4.6.1 MQ Connector Request / Reply Operation

The MQ Connector Request / Reply operation is a synchronous operation that uses a temporary queue to process the reply.

Figure 4.24, below, illustrates the system flow for an MQ request / reply operation using the Universal Command Agent for SOA: MQ Connector.

The following list describes the steps (1 - 5) identified in Figure 4.24:

1. Universal Command is executed requesting the MQ workload. The command options for Universal Command Agent for SOA: MQ Connector are read in from a script file specified with the SCRIPT_FILE option and the message payload is read in via STDIN. UCMD then sends the workload request to Universal Command Agent for SOA (specifically, the UAI component).
2. Universal Command Agent for SOA receives the request from UCMD Server via STDIN. The UAI component validates the command options and existence of the message payload, sends the request to UAC, and blocks. UAC builds the workload execution message for the target workload.
3. Universal Command Agent for SOA: MQ Connector publishes the workload execution message to the specified destination queue.
4. Universal Command Agent for SOA: MQ Connector then reads the reply message off of the temporary reply queue specified in supplied options.
5. UAC returns the reply message to UCMD (or an error message, if the operation failed).

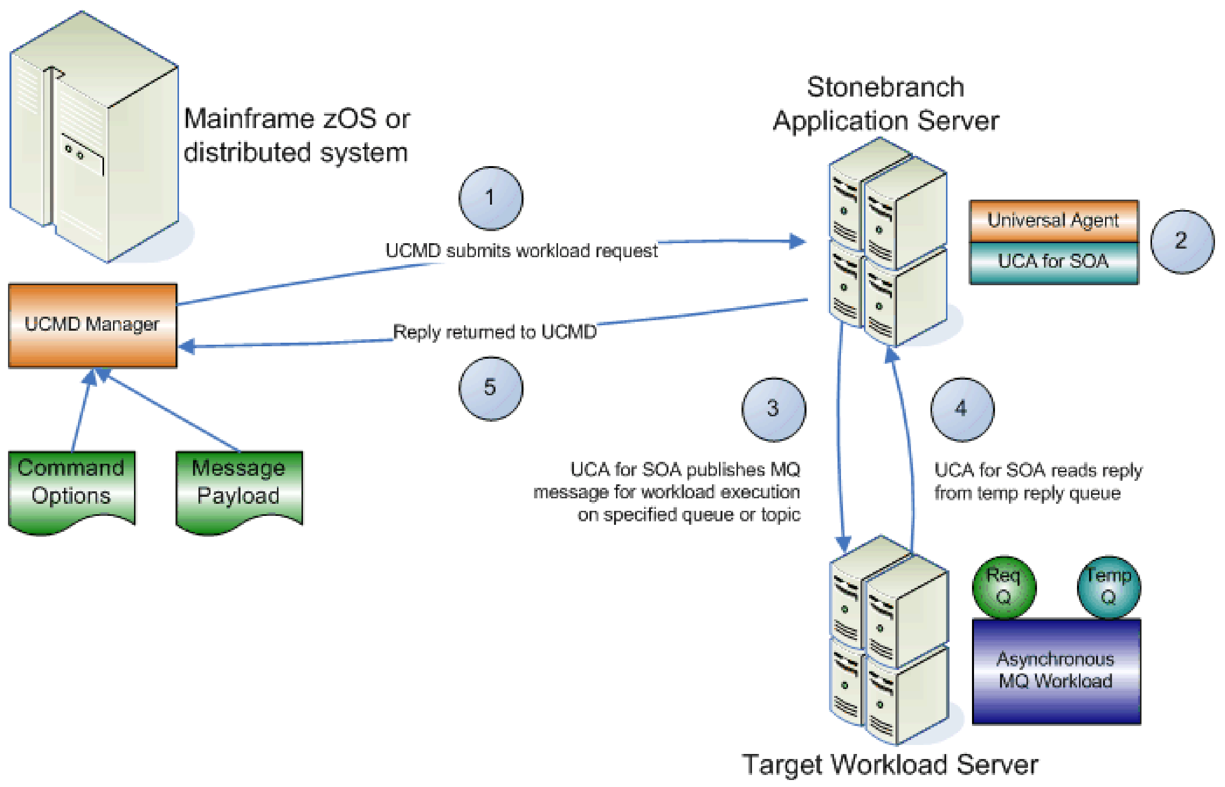

Mainframe zOS or distributed system
Stonebranch Application Server
1
UCMD submits workload request
Universal Agent
UCA for SOA
2
UCMD Manager
Reply returned to UCMD
5
3
4
Command Options
Message Payload
UCA for SOA publishes MQ message for workload execution on specified queue or topic
UCA for SOA reads reply from temp reply queue
Req Q
Temp Q
Asynchronous MQ Workload
Target Workload Server

Figure 4.24 MQ Connector Request / Reply Operation

4.6.2 MQ Connector Publish Operation

The MQ Connector Publish operation is an asynchronous operation that places an MQ message and its payload on the specified destination MQ queue.

UAC returns a message indicating whether the MQ message was successfully placed on the queue.

Figure 4.25, below, illustrates the system flow for an MQ publish operation using the Universal Command Agent for SOA: MQ Connector.

The following list describes the steps (1 - 4) identified in Figure 4.25:

1. Universal Command is executed requesting the MQ workload. The command options for Universal Command Agent for SOA: MQ Connector are read in from a script file specified with the SCRIPT_FILE option and the message payload is read in via STDIN. UCMD then sends the workload request to Universal Command Agent for SOA, specifically the UAI component.
2. Universal Command Agent for SOA receives the request from UCMD Server via STDIN. The UAI component validates the command options and existence of the message payload, sends the request to UAC, and blocks. UAC builds the workload execution message for the target workload.
3. Universal Command Agent for SOA: MQ Connector publishes the workload execution message to the specified queue.
4. UAC returns a success message if the message was placed on the queue with no error, or an error message if there was an error. This reply is generated by UAC, not the MQ Broker.

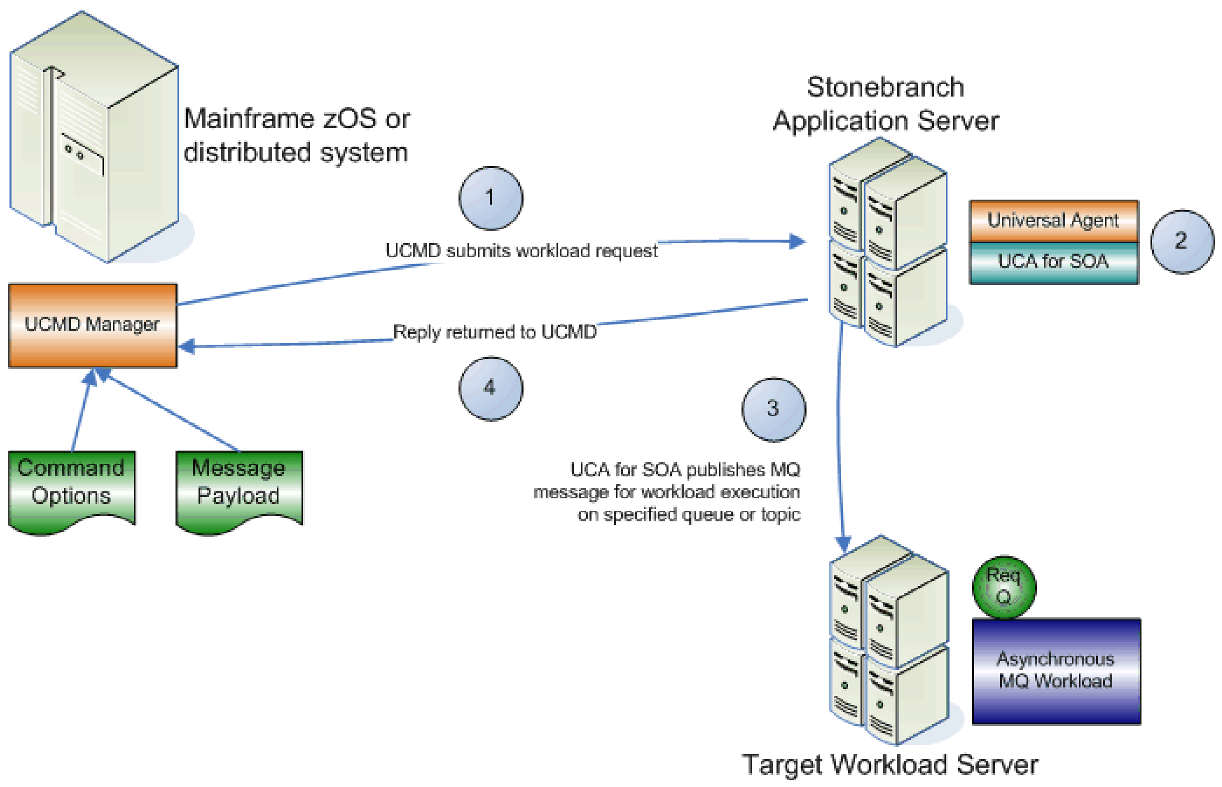

Mainframe zOS or distributed system
UCMD Manager
Command Options
Message Payload
1
UCMD submits workload request
Reply returned to UCMD
4
Stonebranch Application Server
Universal Agent
UCA for SOA
2
3
UCA for SOA publishes MQ message for workload execution on specified queue or topic
Req Q
Asynchronous MQ Workload
Target Workload Server

Figure 4.25 MQ Connector Publish Operation

4.6.3 MQ Connector Request / Reply Operation – Command Options

Table 4.6, below, describes the options (and their values) required to initiate an MQ Connector Request / Reply operation.

See Optional Options for details on additional options.

Option	Value	Description
PROTOCOL	MQ	Connector UAC will use for the current operation.
MEP	Request	Specification that the operation will be a request/reply operation.
SERVICE_URL	Workload URL	Address of the JMS provider in the form of: http://machine:port/service_name
MQ_CHANNEL	MQ channel	Name of the MQ channel.
MQ_HOST	MQ Series server	Name of the server running MQSeries.
MQ_QUEUE_MANAGER_NAME	MQ Queue Mgr.	Name of the MQ QUEUE Manager.
MQ_QUEUE_NAME	MQ Queue	Name of the MQ Queue to use.
MQ_REPLY_TO	MQ Queue	Name of the MQ Queue from which to read the reply.

Table 4.8 MQ Connector Request / Reply Operation - Command Options

4.6.4 MQ Connector Publish Operation – Command Options

Table 4.6, below, describes the options (and their values) required to initiate an MQ Connector Publish operation.

See Optional Options for details on additional options.

Option	Value	Description
PROTOCOL	MQ	Connector that UAC will use for the current operation.
MEP	Publish	Specification that the operation will be a publish operation.
SERVICE_URL	Workload URL	Address of the JMS provider in the form of: http://machine:port/service_name
MQ_CHANNEL	MQ channel	Name of the MQ channel.
MQ_HOST	MQ Series server	Name of the server running MQSeries.
MQ_QUEUE_MANAGER_NAME	MQ Queue Mgr.	Name of the MQ QUEUE Manager.
MQ_QUEUE_NAME	MQ Queue	Name of the MQ Queue to use.
MQ_REPLY_TO	MQ Queue	Name of the MQ Queue from which to read the reply.

Table 4.9 MQ Connector Publish Operation - Command Options

Chapter 5 Troubleshooting

5.1 Overview

This chapter provide solutions to common problems that you might have encountered if the product is not behaving according to the information presented in this document.

5.2 Logging Configuration

At some point, you may want to check the logs for information regarding the operation of Universal Command Agent for SOA.

Configuration of the logging operations is done via the `log4jConfiguration.xml` file for both UAC and UAI.

The logging levels supported by the logging implementation are:

- TRACE
- DEBUG
- INFO
- WARN
- ERROR (default)
- FATAL

Note: The logging level should be changed only at the request of Stonebranch, Inc. Customer Support, as it can have a huge impact on performance.

5.2.1 UAC Logging Configuration

For UAC, the logs are configured to write to a file on Linux and to write to the Event Viewer on Windows with the logging level set to `error`.

The following appenders, or sinks, are available to UAC.

Rolling File Appender

This is the default appender on Linux and logs to a file.

The following attributes can be specified:

- **Name** - name of the appender.
- **File** - path and name of the log file.
- **Max File Size** - maximum size of the log file before rolling over.
- **Max Backup Index** - number of times the log file can be rolled before starting over.
- **Class** - java class that implements the logger.
- **Conversion Pattern** - output format of the log text.

LF5 Appender

Logs to a Java program with a user interface that displays the log in row/column format and enables searches within the log file. Use this for debug only.

The following attributes can be specified:

- **Name** - name of the appender.
- **Class** - java class that implements the logger.
- **Max Number Of Records** - maximum number of records displayed.

NT Event Log Appender

This is the default appender on Windows and logs to the Windows Event Viewer.

The following attributes can be specified:

- **Name** - name of the appender.
- **Class** - java class that implements the logger.
- **Source** - source component that is outputting the log.
- **Conversion Pattern** - output format of the log text.

Console Appender

Logs to the console on STDOUT or STDERR.

The following attributes can be specified:

- **Name** - name of the appender.
- **Class** - java class that implements the logger.
- **Target** - specification to log to STDOUT or STDERR (STDERR is the default).
- **Conversion Pattern** - the output format of the log text.

UAC log4jConfiguration.xml Example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/"
debug="false" threshold="all">
  <appender name="RollingFileAppender"
  class="org.apache.log4j.RollingFileAppender">
    <param name="File" value="/var/opt/universal/log/uac/uac.log"/>
    <param name="MaxFileSize" value="1000KB"/>
    <param name="MaxBackupIndex" value="4"/>
    <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value="%d %-5p %-17c{2} (%30F:%L) %3x -
    %m%n"/>
    </layout>
  </appender>
  <appender name="LF5Appender" class="org.apache.log4j.lf5.LF5Appender">
    <param name="MaxNumberOfRecords" value="1000"/>
  </appender>
    <appender name="NTEventLogAppender"
    class="org.apache.log4j.nt.NTEventLogAppender">
    <param name="Source" value="UAC"/>
    <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value="%c{1} %M - %m%n"/>
    </layout>
  </appender>
  <appender name="ConsoleAppender" class="org.apache.log4j.ConsoleAppender">
    <param name="Target" value="System.err"/>
    <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value="%d %-5p %-17c{2} (%30F:%L) %3x -
    %m%n"/>
    </layout>
  </appender>
  <logger name="com.stonebranch" additivity="true">
    <level value="error"/>
  </logger>
  <root>
    <priority value="error"/>
    <!--<appender-ref ref="LF5Appender"/>-->
    <appender-ref ref="RollingFileAppender"/>
    <!--<appender-ref ref="ConsoleAppender"/>-->
    <!--appender-ref ref="NTEventLogAppender"/-->
  </root>
</log4j:configuration>
```

Figure 5.1 UAC log4jConfiguration.xml Example

5.2.2 UAI Logging Configuration

For UAI, the logs are configured to write to the console on both Linux and Windows with the logging level set to `error`.

The following appenders, or sinks, are available to UAI:

Rolling File Appender

Logs to a file.

The following attributes can be specified:

- **Name** - name of the appender.
- **File** - path and name of the log file.
- **Max File Size** - maximum size of the log file before rolling over.
- **Max Backup Index** - number of times the log file can be rolled before starting over.
- **Class** - java class that implements the logger.
- **Conversion Pattern** - output format of the log text.

LF5 Appender

Logs to a Java program with a user interface that displays the log in row/column format and enables searches within the log file. Use this for debug only.

The following attributes can be specified:

- **Name** - name of the appender.
- **Class** - java class that implements the logger.
- **Max Number Of Records** - maximum number of records displayed.

Console Appender

This is the default appender on Linux and logs to the console on STDOUT or STDERR.

The following attributes can be specified:

- **Name** - name of the appender.
- **Class** - java class that implements the logger.
- **Target** - specification to log to STDOUT or STDERR (STDERR is the default).
- **Conversion Pattern** - output format of the log text.

UAI log4jConfiguration.xml Example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/"
threshold="null" debug="null">
  <appender name="RollingFileAppender"
  class="org.apache.log4j.RollingFileAppender">
    <param name="File" value="/var/opt/universal/log/uai/uai.log"/>
    <param name="MaxFileSize" value="1000KB"/>
    <param name="MaxBackupIndex" value="4"/>
    <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value="%d %-5p %-17c{2} (%30F:%L) %3x -
    %m%n"/>
    </layout>
  </appender>
  <appender name="LF5Appender" class="org.apache.log4j.lf5.LF5Appender">
    <param name="MaxNumberOfRecords" value="1000"/>
  </appender>
  <appender name="ConsoleAppender" class="org.apache.log4j.ConsoleAppender">
    <param name="Target" value="System.err"/>
    <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value="%d %-5p %-17c{2} (%30F:%L) %3x -
    %m%n"/>
    </layout>
  </appender>
  <logger name="com.stonebranch" additivity="true">
    <level value="error"/>
  </logger>
  <root>
    <priority value="error"/>
    <!-- appender-ref ref="LF5Appender" / -->
    <!--<appender-ref ref="RollingFileAppender"/>-->
    <appender-ref ref="ConsoleAppender"/>
  </root>
</log4j:configuration>
```

Figure 5.2 UAI log4jConfiguration Example

Appendix A Customer Support

Stonebranch, Inc. provides customer support, via telephone and e-mail, for Universal Command Agent for SOA 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

INDEX

950 North Point Parkway, Suite 200

Alpharetta, Georgia 30005

U.S.A.