Universal Agent 7.4.x

Universal Command Agent for SOA 7.4.x Reference Guide

Universal Command Agent for SOA 7.4.x Reference Guide

Universal Command Agent for SOA

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 following connectors:

- HTTP Connector
- SOAP Connector
- JMS Connector
- 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 Agent components.
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.

Universal Agent for SOA

Universal Command Agent for SOA is packaged with Universal Event Monitor for SOA and is distributed as part of Universal Agent for SOA.

- Universal Command Agent for SOA comprises the workload execution functionality.
- Universal Event Monitor for SOA comprises the file-based event monitoring functionality.

Detailed Information

The following pages provide detailed information for Universal Command Agent for SOA:

- Universal Command Agent for SOA Architecture
- Universal Command Agent for SOA Components
- Universal Command Agent for SOA Connector Overview
- Universal Command Agent for SOA Defined Ports
- Universal Command Agent for SOA Operations Configuration
- Universal Command Agent for SOA Usage
- Universal Command Agent for SOA Command Options
- Universal Command Agent for SOA Configuration Options
- UAC Server Component Definition Options
- Universal Command Agent for SOA Operations
- Universal Command Agent for SOA Logging Configuration
- Universal Command Agent for SOA Additional Information

Universal Command Agent for SOA Examples

See Universal Agent - Web Services Examples for examples of how to use Universal Command Agent for SOA.

Universal Command Agent for SOA Architecture

Overview

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 Automation Center line, provide your enterprise with a loosely coupled, scalable, and secure solution to your enterprise workload management tasks.

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 (HyperText Transfer Protocol) 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 (Simple Object Access Protocol) 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 (Java Message Service) 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 (Message Queue) 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.

Supported Messages 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.

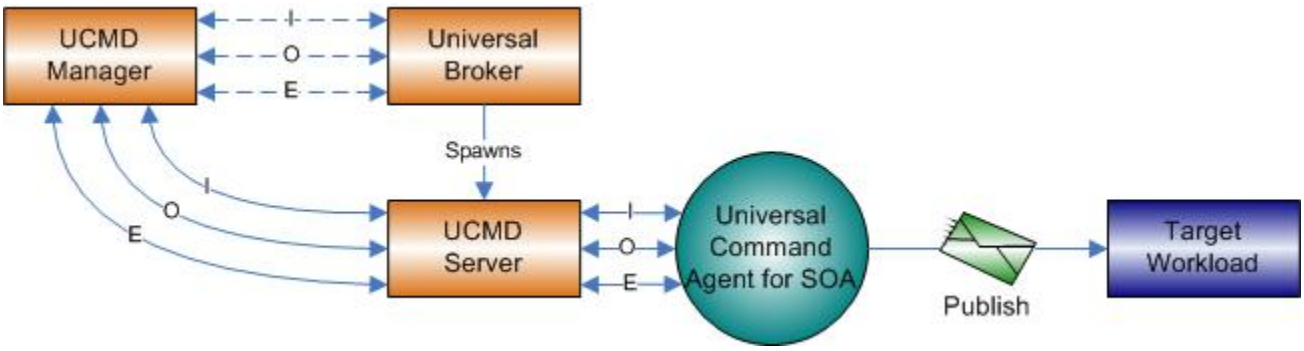
Publish MEP

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.

The following figure illustrates a logical view of the Publish MEP.



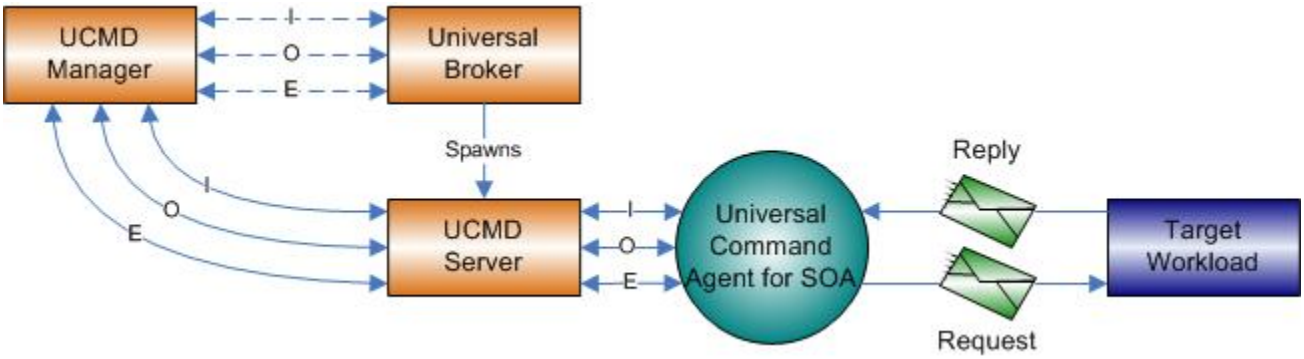
Request / Reply MEP

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.

The following figure illustrates a logical view of the Request / Reply MEP.



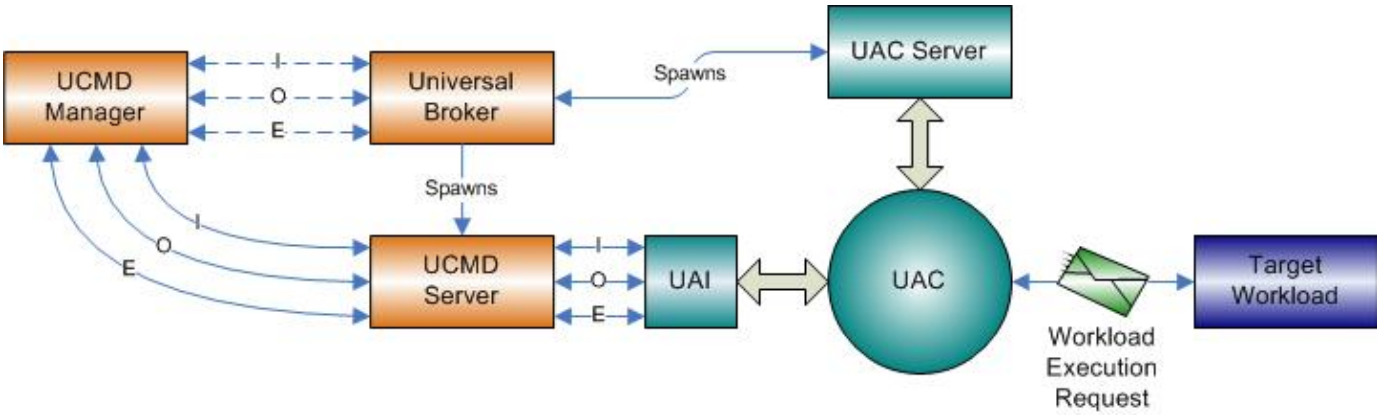
Universal Command Agent for SOA Components

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.

The following figure illustrates the basic component flow for Universal Command Agent for SOA.



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.

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.

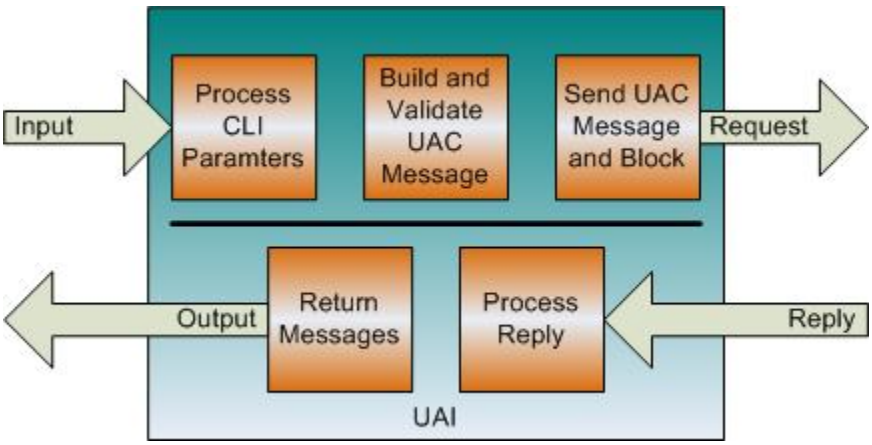
UAI responsibilities include:

- 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 7.3.x Reference 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.

The following figure summarizes the basic process flow for UAI.



UAC Server

The UAC (Universal Application Container) 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.

UAC (Universal Application Container)

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

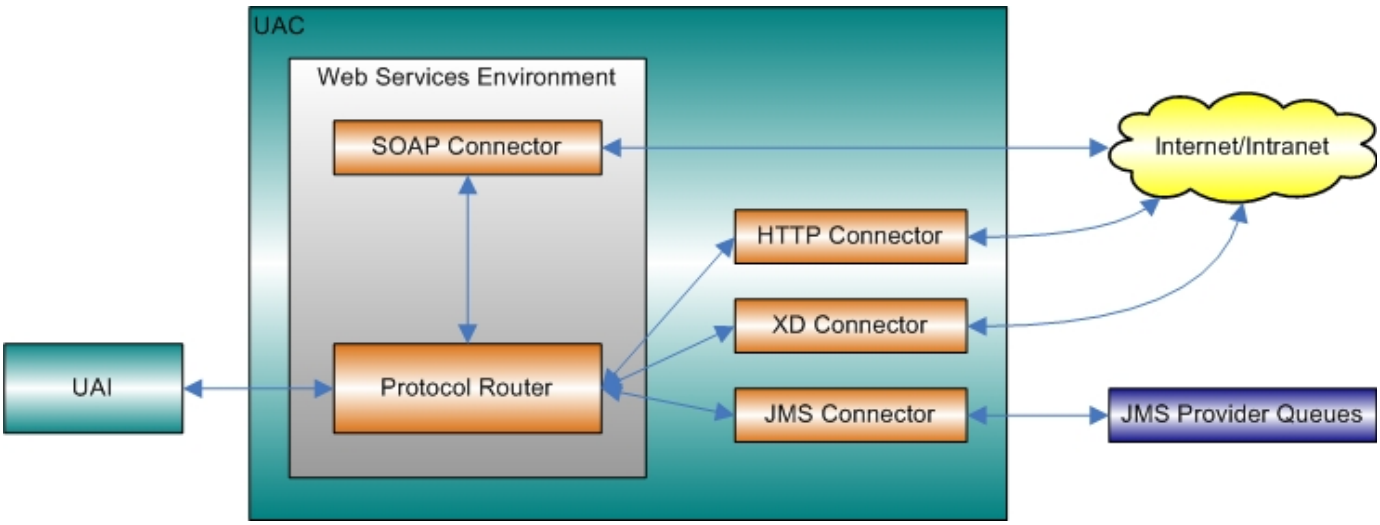
Its responsibilities include the following:

- 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.

The following figure summarizes the basic process flow for UAC.



Universal Command Agent for SOA Connector Overview

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.

This section provides a summary of the current connectors.

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 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.

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.

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.

Universal Command Agent for SOA Defined Ports

Universal Command Agent for SOA uses a specific set of ports (see 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/TLS 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.

Universal Command Agent for SOA Operations Configuration

Overview

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

- Outbound JMS Configuration Using WAS
- Outbound MQ Configuration - MQ Client Jar Files

Outbound JMS Configuration Using WAS

Although, there is no outbound configuration needed for HTTP and SOAP outbound operations, 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 (WAS) 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.

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**.

The following figure 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>
```

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.

The following figure illustrates a sample operation.

> **Note**
> Only the property element is shown; it could be included in the sample illustrated in the figure above.

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

JMS Provider Client Jar Files

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.

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.

Place the MQ client jar files in the following location:

UNIX	**/opt/universal/uac/container/webapps/axis2/WEB-INF/lib**
Windows	**\Program Files\Universal\uac\container\webapps\axis2\WEB-INF\lib**

The following jar files are required:

- **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.

The MQ Client for Java version 7.0 package with the latest fix pack is recommended.

When using a MQ CCDT to establish connections to queue managers, 7.0.1.3 or later is highly recommended.

Universal Command Agent for SOA Usage

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.

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.

For an example of message payload, see Web Services Execution.

Additional Information

The following pages provide additional detailed information for Universal Command Agent for SOA usage:

- Universal Command Agent for SOA - Script File Command Options
- Universal Command Agent for SOA - Component Configuration
- Universal Command Agent for SOA - Component Definition

Universal Command Agent for SOA - Script File Command Options

Command Options Categories

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

The following table categorizes the command options into logical areas of application. Each **Category** name is a link to a table of options in that category. Each **Option Name** in those tables is a link to detailed information about that option.

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.

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.

Command Options Syntax

The following figure 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 | CCDT
-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**.

Universal Command Agent for SOA - Component Configuration

- Configuration File

Configuration File

A configuration file provides the simplest method of specifying configuration option values that will not change with each command invocation. For Universal Command Agent for SOA, a configuration file is the only method of specifying configuration values.

The Universal Command Agent for SOA configuration file is named **`uacs.conf`**. This file can be edited manually with any text editor.

Universal Configuration Manager

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 Universal Command Agent for SOA for Windows configuration options.

Universal Configuration Manager is a Universal Agent graphical user interface application that enables you to configure all of the Universal Agent components 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.

Configuration Options

The following table identifies all of the Universal Command Agent for SOA configuration options. Each **Option Name** is a link to detailed information about that option.

Option Name	Description
ACTIVITY_MONITORING	Specification for whether or not product activity monitoring events are generated.
CODE_PAGE	Character code page used to translate text data received and transmitted over the network.
EVENT_GENERATION	Events to be generated as persistent events.
INSTALLATION_DIRECTORY	Location in which the Universal Application Container Server is installed.
MESSAGE_LEVEL	Level of messages written.
MQ_CCDT_URL	Location of an MQ Client Channel Definition Table (CCDT) that can be used to establish client connections to remote MQ queue managers.
RMI_PORT	Port number or service name on which Universal Application Container will listen for service requests from the Universal Application Container Server.

Universal Command Agent for SOA - Component Definition

Overview

All Universal Agent components managed by Universal Broker have a component definition. The component definition is a text file of options containing component-specific information required by Universal Broker.

The syntax of a component definition file is the same as a configuration file.

The Universal Application Container (UAC) Server component definition is located in the component definition directory of the Universal Broker.

Component Definition Options

The following table identifies all of the options that comprise the UAC Server component definition. Each **Option Name** is a link to detailed information about that option.

Option Name	Description
AUTOMATICALLY_START	Specification for whether or not the UAC Server starts automatically when Universal Broker is started.

COMPONENT_NAME	Name by which the clients know the UAC Server.
CONFIGURATION_FILE*	Name of the UAC Server configuration file.
RUNNING_MAXIMUM	Maximum number of UAC Servers that can run simultaneously.
START_COMMAND*	Program name of the UAC Server.
WORKING_DIRECTORY*	Directory used as the working directory of the UAC Server.
* These options are required in all component definitions.	

Universal Command Agent for SOA Command Options

Overview

This page provides links to detailed information on the command options available for use with Universal Command Agent for SOA.

The options are listed alphabetically, without regard to any specific operating system.

Command Options Information

For each command option, these pages provide the following information.

Description

Describes the command option and how it is used.

Usage

Provides a table of the following information:

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Command Line, Long Form	<Format / Value>					

Method

Identifies the method used to specify Universal Command Agent for SOA command options:

- Command Option, Long Form

Syntax

Identifies the syntax of the method used to specify the option:

- Format: Specific characters that identify the option.
- Value: Type of value(s) to be supplied for this method.

(Operating System)

Identifies the operating systems for which each method of specifying the option is valid:

- IBM i

- HP NonStop
- UNIX
- Windows
- z/OS

Values

Identifies all possible values for the specified value type.

Defaults are identified in **bold type**.

<Additional Information>

Identifies any additional information specific to the option.

Command Options List

The following table identifies all Universal Command Agent for SOA command options.

Option	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_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_PROPERTIES_FILE	Name and location of an XML document containing the JMS properties to be included in the JMS message.
JMS_REPLY_TO	Name of the JMS reply queue for the return JMS message.
MEP	Message exchange pattern to be used for the current operation.
MQ_CHANNEL	Name of the MQ channel.
MQ_HOST	Name of the server running MQSeries.
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.
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.
PROTOCOL	Message protocol to be used for the current operation.
SERVICE_PASSWORD	Password to be passed to the target workload for authentication.
SERVICE_URL	URL (internet, network, or file-based) of the target workload.
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.

XD_CMD	Operation to submit to the WebSphere XD environment.
XD_CMD_ID	Used to correlate jobs.

HELP - UCA for SOA command option

Description

The HELP option displays a description the Universal Command Manager command line options and their format.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Command Line, Long Form	-help			✓	✓	

Values

(There are no values for the HELP option.)

HTTP_AUTH - UCA for SOA command option

Description

The HTTP_AUTH option specifies the HTTP authentication scheme to use.

If the option is not used, UAC defaults to **NONE**.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Command Line, Long Form	-httpauth *scheme*			✓	✓	

Values

scheme is the HTTP authentication scheme to use.

Valid values for *scheme* are:

- **BASIC**
  Method designed to allow a client application such as a web browser, or other client program, to provide credentials in the form of a user name and password when making an authenticated HTTP request.
- **DIGEST**
  Method designed to allow a client application such as a web browser, or other client program, to negotiate credentials with a web server (using the HTTP protocol).
  Digest authentication allows the user identity to be established securely without having to send a password in plaintext over the network. It is basically an application of MD5 cryptographic hashing with usage of nonce values to prevent analysis.
- **NTLM**
  NTLM is the most complex of the authentication protocols. It is a proprietary protocol designed by Microsoft with no publicly available specification.
- **NONE**
  No HTTP authentication scheme is used.

Default is NONE.

HTTP_FORM_DATA - UCA for SOA command option

Description

The HTTP_FORM_DATA option specifies whether or not there is HTTP form data provided as an XML document in the payload file.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Command Line, Long Form	-httpformdata *option*			✓	✓	

Values

option is the specification for whether or not there is form data in the payload file.

Valid values for *option* are:

- **true**
  There is form data in the payload file.
- **false**
  There is not form data in the payload file.

Default is false.

Format of Form Data

Form data is provided as the payload to the service request. The form data is formatted as an XML document as defined by the XML Schema Definition (XSD) below.

```
<?xml version="1.0" encoding="UTF-8"?><xsd:schema xmlns:xsd="http://www.
w3.org/2001/XMLSchema" targetNamespace="http://com.stonebranch/UAI
/HTTPFormData" xmlns="http://com.stonebranch/UAI/HTTPFormData"
elementFormDefault="qualified"><xsd:element name="HTTPFormData" type="
HTTPFormDataType" /><xsd:complexType name="HTTPFormDataType"><xsd:
sequence><xsd:element name="Property" maxOccurs="unbounded" type="
PropertyType" minOccurs="0" /></xsd:sequence></xsd:complexType><xsd:
complexType name="PropertyType"><xsd:sequence><xsd:element name="Name"
type="xsd:string" /><xsd:element name="Value" type="xsd:string" /></xsd:
sequence></xsd:complexType></xsd:schema></pre>
```

The XSD defines the form data as a <HTTPFormData> XML element containing a sequence of property values defined by the <property> XML tag. Each property value consist of a name and value defined by a <name> and <value> XML tag, respectively.

The following illustrates a form data XML document that contains two properties. The first property has a name of "Comments" with a value of "You only live once, but if you work it right, once is enough.". The second property has a name of "box" and a value of "yes".

```
<?xml version="1.0" encoding="UTF-8"?><p:HTTPFormData xmlns:p="
http://com.stonebranch/UAI/HTTPFormData" xmlns:xsi="http://www.w3.org
/2001/XMLSchema-instance" xsi:schemaLocation="http://com.stonebranch/UAI
/HTTPFormData HTTPFormData.xsd "><p:Property><p:Name>Comments</p:
Name><p:Value>You only live once, but if you work it right, once is
enough.</p:Value></p:Property><p:Property><p:Name>box</p:Name><p:
Value>yes</p:Value></p:Property></p:HTTPFormData>
```

HTTP_METHOD - UCA for SOA command option

Description

The HTTP_METHOD option specifies the type of HTTP operation to execute.

If this option is not used, UAC defaults to **POST**.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Command Line, Long Form	-httpmethod *type*			✓	✓	

Values

type is the type of HTTP operation to execute.

Valid values for *type* are:

- **GET**
- **POST**

Default is POST.

HTTP_VERSION - UCA for SOA command option

Description

The HTTP_VERSION option specifies which version of the HTTP protocol to use.

This option is used if the target workload requires a specific version of the HTTP protocol.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Command Line, Long Form	-httpversion *version*			✓	✓	

Values

version is the version of HTTP protocol to use.

Valid values for *version* are:

- **OneDotZero**
- **OneDotOne**

Default is OneDotOne.

JMS_CONNECTION_FACTORY_NAME - UCA for SOA command option

Description

The JMS_CONNECTION_FACTORY_NAME option specifies the connection factory to be used to establish a connection to a JMS provider.

JMS_CONNECTION_FACTORY_NAME is required if the message protocol specified in the PROTOCOL option is **JMS**.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Command Line, Long Form	-jmsconnectionfactoryname *name*			✓	✓	

Values

name is the JNDI name of the connection factory to be used.

JMS_CONTEXT_FACTORY_NAME - UCA for SOA command option

Description

The JMS_CONTEXT_FACTORY_NAME option specifies the java class name of the JMS providers initial context factory.

JMS_CONTEXT_FACTORY_NAME is required if the message protocol specified in the PROTOCOL option is **JMS**.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Command Line, Long Form	-jmscontextfactoryname *name*			✓	✓	

Values

name is the class name of the context factory name.

> **Note**
> *name* is specific to the JMS provider that you are using.

JMS_DESTINATION - UCA for SOA command option

Description

The JMS_DESTINATION option specifies the name of the target JMS destination queue or topic for the JMS message.

JMS_DESTINATION is required if the message protocol specified in the PROTOCOL option is **JMS**.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Command Line, Long Form	-jmsdestination *name*			✓	✓	

Values

name is the JNDI name of the target queue or topic.

JMS_PROPERTIES_FILE - UCA for SOA command option

Description

The JMS_PROPERTIES_FILE option specifies the name and location of an XML document containing the JMS properties to be included in the JMS message.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Command Line, Long Form	-jmspropertiesfile *file*			✓	✓	

Values

file is the path/filename of the properties file.

JMS_REPLY_TO - UCA for SOA command option

Description

The JMS_REPLY_TO option specifies the name of the target JMS reply queue for the return JMS message.

JMS_REPLY is required if the PROTOCOL option is set to **JMS**.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Command Line, Long Form	-jmsreplyto *name*			✓	✓	

Values

name is the JNDI name of the target JMS reply queue.

MEP - UCA for SOA command option

Description

The MEP option specifies the message exchange pattern to use for the current operation.

MEP is required to process the workload execution request.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Command Line, Long Form	-mep *pattern*			✓	✓	

Values

pattern is the message exchange pattern to use for the current operation.

Valid values for *pattern* are:

- **Publish**
  Asynchronous communication using the JMS, SOAP, or MQ protocol.
  Operations using the Publish protocol are one-way and do not block for a reply, as no reply will be returned by the target workload.
- **Request**
  Synchronous communication using the HTTP, SOAP, or MQ protocol. Operations using the Request mep are two-way and blocked until a reply is sent by the target workload.

There is no default; a value must be passed.

MQ_CHANNEL - UCA for SOA command option

Description

The MQ_CHANNEL option specifies the name of the MQ channel.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Command Line, Long Form	-mqchannel *channel* | ccdt			✓	✓	

Values

channel is the name of the MQ channel.

Valid values for *channel* are:

- Any user-defined MQ channel.
- **ccdt**

MQ Client Channel Definition Table (CCDT) is used to find a client channel definition. The Universal Application Container must be configured with a CCDT for this value to be accepted (see MQ_CCDT_URL).

MQ_HOST - UCA for SOA command option

Description

The MQ_HOST option specifies the name of the server running MQSeries.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Command Line, Long Form	-mqhost *server*			✓	✓	

Values

server is the name of the server running MQSeries.

This value is not used if the MQ_CHANNEL value is **ccdt**, in which case the host name of the server comes from the channel definition defined in the CCDT.

MQ_PORT - UCA for SOA command option

Description

The MQ_PORT option specifies the name of the port on which the MQ Broker is listening.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Command Line, Long Form	-mqport *port*			✓	✓	

Values

port is the name of the port.

Default = 1414.

MQ_PROPERTIES_FILE - UCA for SOA command option

Description

The MQ_PROPERTIES_FILE option specifies the name of the file containing MQ name / value pairs.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Command Line, Long Form	-mqpropertiesfile *file*			✓	✓	

Values

file is the name of the file containing MQ name / value pairs.

MQ_QUEUE_MANAGER_NAME - UCA for SOA command option

Description

The MQ_QUEUE_MANAGER_NAME option specifies the name of the MQ Queue Manager.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Command Line, Long Form	-mqqueuemanagername *manager*			✓	✓	

Values

manager is the name of the MQ Queue Manager.

If the MQ_CHANNEL value specifies **ccdt** as the channel name, the client channel definition is selected from the CCDT, based on the MQ_QUEUE_MANAGER_NAME value. Refer to the IBM WebSphere MQ documentation for a description of this selection process.

If a CCDT is used, an asterisk (*) and a blank MQ_QUEUE_MANAGER_NAME value specify the same channel definition selection process. MQ_QUEUE_MANAGER_NAME does not accept a blank value. Instead, specify an asterisk (*) for the same selection process as specified by a blank.

MQ_QUEUE_NAME - UCA for SOA command option

Description

The MQ_QUEUE_NAME option specifies the name of the MQ Queue to use.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Command Line, Long Form	-mqqueuename *queue*			✓	✓	

Values

queue is the name of the MQ Queue.

MQ_REPLY_TO - UCA for SOA command option

Description

The MQ_REPLY_TO option specifies the name of the MQ Queue from which to read the reply when MEP is set to request.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Command Line, Long Form	-mqreplyto *queue*			✓	✓	

Values

queue is the name of the MQ Queue.

PROTOCOL - UCA for SOA command option

Description

The PROTOCOL option specifies the message protocol to use for the current operation.

PROTOCOL is required to process the workload execution request.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Command Line, Long Form	-protocol *protocol*			✓	✓	

Values

protocol is the protocol to use for the current operation.

Valid values for *option* are:

- **HTTP**
  Specifies an HTTP operation for executing a workload with an HTTP interface.
- **JMS**
  Specifies a JMS operation for executing a workload with a JMS interface.
- **MQ**
  Specifies an MQ operation for executing a workload with an MQ interface.
- **SOAP**
  Specifies a SOAP operation for executing a workload with a SOAP interface.
- **XDSOAP**
  Specifies an XD operation for executing a workload within the XD environment.

There is no default; a value must be passed.

SERVICE_PASSWORD - UCA for SOA command option

Description

The SERVICE_PASSWORD option specifies the password to be passed to the target workload for authentication.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Command Line, Long Form	-servicepassword *password*			✓	✓	

Values

password is the password to be passed to the target workload.

SERVICE_URL - UCA for SOA command option

Description

The SERVICE_URL option specifies the URL address (internet, network, or file-based) of the target workload.

SERVICE_URL is required to process the workload execution request.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Command Line, Long Form	-serviceurl *url*			✓	✓	

Values

url is the address of the target workload.

Valid values for *url* are:

- hostname
- IP address

SERVICE_USER_NAME - UCA for SOA command option

Description

The SERVICE_USER_NAME option specifies the user name to be passed to the target workload for authentication.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Command Line, Long Form	-serviceusername *name*			✓	✓	

Values

name is the user name to be passed to the target workload.

SOAP_ACTION - UCA for SOA command option

Description

The SOAP_ACTION option specifies soapAction HTTP header value.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Command Line, Long Form	-soapaction *header*			✓	✓	

Values

header is the SOAP action name.

SOAP_VERSION - UCA for SOA command option

Description

The SOAP_VERSION option specifies the version of the SOAP protocol to use when making the SOAP request.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Command Line, Long Form	-soapversion *version*			✓	✓	

Values

version is the version of the SOAP protocol to use.

Valid values for *version* are:

- **OneDotOne**
- **OneDotTwo**

Default is OneDotTwo.

TIMEOUT_SEC - UCA for SOA command option

Description

The TIMEOUT_SEC option specifies the length of time - in seconds - to wait for the request to complete.

If this option is not used, UAC defaults to 10 seconds.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS

Command Line, Long Form	-timeoutsec *time*			✓	✓	

Values

time is the number of seconds to wait for the request to complete.

XD_CMD - UCA for SOA command option

Description

The XD_CMD option specifies the operation to submit to the WebSphere XD environment.

XD_CMD is required if the PROTOCOL option is set to **XDSOAP**.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Command Line, Long Form	-xdcmd *operation*			✓	✓	

Values

operation specifies the operation to submit to the WebSphere XD environment.

Valid values for *operation* are:

- **SUBMIT**
  Used, along with the xJCL read from STDIN, to start workload within the XD environment. The **xdcmd** and **xdcmdid** values are returned on STDOUT after job submission for reference and correlation purposes.
  For example:
  `UNV: 5050 - xdcmdid=0ca1af1d-0db6-41bf-9ce4-b91099797503 xdcmd=SUBMIT`
  Note that the value shown for **`xdcmdid`** is the auto-generated value which is returned if **-xdcmdid** is not included, or is commented out, in the command options script.
- **RESTART**
  Used to restart workload within the XD environment. You will need the command ID value passed in with the submit operation using the **-xdcmdid** option to restart the workload. Only workload that is indicated as "Restartable" in the XD environment can be restarted. The xJCL does not need to be read from STDIN on a restart as XD saves the xJCL with the job.

There is no default; a value must be passed.

See Cancelling an XD Operation for information on how to cancel a submit or restart operation.

XD_CMD_ID - UCA for SOA command option

Description

The XD_CMD_ID option is used to correlate jobs.

XD_CMD_ID is required if the PROTOCOL option is set to **XDSOAP**. In this case, it is used to correlate jobs between the Universal Command Manager host (mainframe) request and the WebSphere XD environment as you are not able to submit your own JobID to the WebSphere XD environment.

Universal Command Agent for SOA will generate a unique xdcmdid value or you can use your own value for the xdcmdid option. If you use your own value, make sure it is unique; otherwise, XD will pick the first workload with a value that matches if there are duplicates.

> **Note**
> You will need to comment out (using the # character) or remove the XD_CMD_ID option from the command option script if you want Universal Command Agent for SOA to generate a unique **xdcmdid** value.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Command Line, Long Form	-xdcmdid *ID*			✓	✓	

Values

ID is the current job ID.

Universal Command Agent for SOA Configuration Options

Overview

This page provides links to detailed information on the configuration options available for use with the Universal Command Agent for SOA.

The options are listed alphabetically, without regard to any specific operating system.

Configuration Options Information

For each configuration option, these pages provide the following information.

Description

Describes the configuration option and how it is used.

Usage

Provides a table of the following information:

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Configuration File Keyword	<Format / Value>					

Method

Identifies the method used to specify Universal Command Agent for SOA configuration options:

- Configuration File Keyword

Syntax

Identifies the syntax of each method that can be used to specify the option:

- Format: Specific characters that identify the option.
- Value: Type of value(s) to be supplied for this method.

> **Note**
> If a Method is not valid for specifying the option, the Syntax field contains **n/a**.

(Operating System)

Identifies the operating systems for which each method of specifying the option is valid:

- IBM i
- HP NonStop
- UNIX
- Windows
- z/OS

Values

Identifies all possible values for the specified value type.

Defaults are identified in **bold type**.

<Additional Information>

Identifies any additional information specific to the option.

Configuration Options List

The following table identifies the Universal Command Agent for SOA configuration options.

Option Name	Description
ACTIVITY_MONITORING	Specification for whether or not product activity monitoring events are generated.
CODE_PAGE	Character code page used to translate text data received and transmitted over the network.
EVENT_GENERATION	Events to be generated as persistent events.
INSTALLATION_DIRECTORY	Location in which the Universal Application Container Server is installed.
MESSAGE_LEVEL	Level of messages written.
MQ_CCDT_URL	Location of an MQ Client Channel Definition Table (CCDT) that can be used to establish client connections to remote MQ queue managers.
RMI_PORT	Port number or service name on which Universal Application Container will listen for service requests from the Universal Application Container Server.

ACTIVITY_MONITORING - UCA for SOA configuration option

Description

The ACTIVITY_MONITORING option specifies whether or not product activity monitoring events are generated.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Configuration File Keyword	activity_monitoring *option*			✓	✓	

Values

option is the specification for whether or not product activity monitoring events are generated.

Valid values for *option* are:

- **yes**
  Activate product activity monitoring events
- **no**
  Deactivate product activity monitoring events

Default is yes.

CODE_PAGE - UCA for SOA configuration option

Description

The CODE_PAGE option specifies the character code page that is used to translate text data received and transmitted over the network.

The Universal Translate Table (UTT) files are used to translate between Unicode and the local single-byte code page.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Configuration File Keyword	codepage *codepage*			✓	✓	

Values

codepage is the character code page that is used to translate data.

codepage references a Universal Translate Table (UTT) file provided with the product (see Section 9.3 UTT Files). UTT files are used to translate between Unicode and the local single-byte code page. (All UTT files end with an extension of **.utt**.)

> **Note**
> UTF-8 is not a supported *codepage* value for CODE_PAGE. UTF-8 codepage is valid only for standard I/O text file translation.

See Character Code Pages for a complete list of character code pages provided by Stonebranch Inc. for use with Universal Automation Center.

Default

Default is different for different operating systems:

- ISO8859-1 (8-bit ASCII): ASCII-based operating systems
- IBM1047 (EBCDIC): EBCDIC-based operating system

EVENT_GENERATION - UCA for SOA configuration option

Description

The EVENT_GENERATION option specifies which types of events are to be generated and processed as persistent events by the Universal Event Subsystem (UES).

A persistent event record is saved in a Universal Enterprise Controller (UEC) database, the UES database (***uec.evm.db***), for long-term storage.

For a list of all event types for all Universal Agent components, see Event Definition Details.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Configuration File Keyword	event_generation *types*			✓	✓	

Values

type specifies a comma-separated list of event types. It allows for all or a subset of all potential event message types to be selected.

Event type ranges can be specified by separating the lower and upper range values with a dash (**-**) character.

Event types can be selected for inclusion or exclusion:

- Exclusion operator is **X** or **x**.
- An asterisk (*) represents all event types.

Examples

100,101,102	Generate event types 100, 101, and 102.
100-102	Generate event types 100 through 102.
100-102,200	Generate event types 100 through 102 and 200.
*	Generate all event types.
*,X100	Generate all event types except for 100.
x*	Generate no event types.

*,X200-250,X300	Generate all event types except for 200 through 250 and 300.

Default is X (no event types).

INSTALLATION_DIRECTORY - UCA for SOA configuration option

Description

The INSTALLATION_DIRECTORY option specifies the location in which Universal Application Container Server is installed.

> **Note**
> This option is required and cannot be overridden.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Configuration File Keyword	installation_directory *directory*			✓	✓	

Values

directory is the location in which the Universal Application Container Server is installed.

The full path name is required.

MESSAGE_LEVEL - UCA for SOA configuration option

Description

The MESSAGE_LEVEL option specifies the level of messages to write.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Configuration File Keyword	message_level *level*			✓	✓	

Values

level is the level of messages to write.

Valid values for *level* are:

- **trace**
  Writes trace messages used for diagnostic purposes.

> **Note**
> Use **trace** only as directed by Stonebranch, Inc. Customer Support.

- **audit**
  Writes audit, informational, warning, and error messages.
- **info**
  Writes informational, warning, and error messages.
- **warn**
  Writes warning and error messages.
- **error**
  Writes error messages only.

Default is info.

MQ_CCDT_URL - UCA for SOA configuration option

Description

The MQ_CCDT_URL option specifies the location of an MQ Client Channel Definition Table (CCDT) file that can be used to establish client connections to remote MQ queue managers.

Usage

Method	Syntax*	IBM i	HP NonStop	UNIX	Windows	z/OS
Configuration File Keyword	mq_ccdt_url *url*			✓	✓	

Values

url is a URL that specifies the location of the CCDT file.

For example, the following URLs specify the location of the CCDT file on a file system:

UNIX	file:///mqm/ccdt/AMQCLCHL.TAB
Windows	file:///e:/path/to/file/AMQCLCHL.TAB The following format also is accepted: file:///e:\path\to\file\AMQCLCHL.TAB
FTP	ftp://userName:password@myServer/definitionPath/AMQCLCHL.TAB

RMI_PORT - UCA for SOA configuration option

Description

The RMI_PORT option specifies the port number or service name on which Universal Application Container will listen for service requests from the Universal Application Container Server.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Configuration File Keyword	rmi_port *port*			✓	✓	

Values

port is the port number or service name on which Universal Application Container will listen for service requests.

Default is 7899.

UAC Server Component Definition Options

Overview

This page provides links to detailed information about the options that comprise Universal Application Container (UAC) Server component definitions.

The options are listed alphabetically, without regard to any specific operating system.

Component Definition Options Information

For each component definition option, these pages provide the following information.

Description

Describes the option and how it is used.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Component Definition Keyword	<Format / Value>					

Method

Identifies the method used for specifying a Universal Command component definition option:

- Component Definition Keyword

Syntax

Identifies the syntax of the method used to specify the option:

- Format: Specific characters that identify the option.
- Value: Type of value(s) to be supplied for this method.

(Operating System)

Identifies the operating systems for which the method of specifying the option is valid:

- IBM i
- HP NonStop
- UNIX
- Windows
- z/OS

Values

Identifies all possible values for the specified value type.

Defaults are identified in **bold type**.

Component Definition Options List

The following table identifies all of the options that can comprise a Universal Application Container component definition.

Component	Description
AUTOMATICALLY_START	Specification for whether or not the UAC Server starts automatically when Universal Broker is started.
COMPONENT_NAME	Name by which the clients know the UAC Server.
CONFIGURATION_FILE*	Name of the UAC Server configuration file.
RUNNING_MAXIMUM	Maximum number of UAC Servers that can run simultaneously.
START_COMMAND*	Program name of the UAC Server.
WORKING_DIRECTORY*	Directory used as the working directory of the UAC Server.
* These options are required in all component definitions.	

AUTOMATICALLY_START - UCA for SOA component definition option

Description

The AUTOMATICALLY_START option specifies whether or not the UAC Server starts automatically when the Universal Broker is started.

Usage

Method	Parameter / Value	IBM i	HP NonStop	UNIX	Windows	z/OS
Component Definition Keyword	auto_start *option*			✓	✓	

Values

option is the specification for how the UAC Server is started.

The only valid value for *option* is:

- **yes**
  UAC Server must be started automatically when Universal Broker is started.

COMPONENT_NAME - UCA for SOA component definition option

Description

The COMPONENT_NAME option specifies the name of the UAC Server.

Component start requests refer to UAC Server by this name.

> **Note**
> COMPONENT_NAME is optional in a component definition. If it is not specified, the file name is used as the component name.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Component Definition Keyword	component_name *name*			✓	✓	

Values

name is the name of the UAC Server.

There is only one valid value for *name*:

- **uac** (This is the name of the UAC Server component definitions file.)

> **Note**
> This name should not be changed.

CONFIGURATION_FILE - UCA for SOA component definition option

Description

The CONFIGURATION_FILE option specifies the name of the UAC Server configuration file.

> **Note**
> CONFIGURATION_FILE is required in a component definition.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Component Definition Keyword	configuration_file *filename*			✓	✓	

Values

member / *filename* is the name of the configuration member / file.

UNIX	Full path name of the configuration file. The file name can be any valid file name. **The installation default is `/etc/universal/uacs.conf`.**
Windows	Full path name of the configuration file. The file name can be any valid file name. **The installation default is `c:\Documents and Settings\All Users\Application Data\Universal\conf\uacs.conf`.**

RUNNING_MAXIMUM - UCA for SOA component definition option

Description

The RUNNING_MAXIMUM option specifies the maximum number of UAC Servers that can run simultaneously.

If this maximum number is reached, any command received to start a UAC Server is rejected.

> **Note**
> RUNNING_MAXIMUM is optional in a component definition.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Component Definition Keyword	running_max *maximum*			✓	✓	

Values

maximum is the maximum number of UAC Servers that can run simultaneously.

Default is 100.

> **Note**
> If you specify 0 for *maximum*, the default (100) will be used. To use 0 for the maximum number of servers, specify -1 or less for *maximum*.

START_COMMAND - UCA for SOA component definition option

Description

The START_COMMAND option specifies the full path name of the UAC Server program.

Optionally, START_COMMAND also can specify command line options.

> **Note**
> START_COMMAND is required in a component definition.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Component Definition Keyword	start_command *name[II:options]*			✓	✓	

Values

name is the full path name of the UAC Server program.

options is the optional list of command line options (enclosed in single or double quotation marks).

> **Windows**
> *name* is the full path name of the UAC Server program. This name is defined at installation; it is not modifiable from the Universal Configuration Manager.

WORKING_DIRECTORY - UCA for SOA component definition option

Description

The WORKING_DIRECTORY option specifies the full path name used as the working directory of UAC Server.

> **Note**
> WORKING_DIRECTORY is required in a component definition.

Usage

Method	Syntax	IBM i	HP NonStop	UNIX	Windows	z/OS
Component Definition Keyword	working_directory *directory*			✓	✓	

Values

directory is the full path name of the working directory.

> **Note**
> Do not change this directory.

Universal Command Agent for SOA Operations

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.

- HTTP Connector
- SOAP Connector
- JMS Connector
- XD Connector
- MQ Connector

In these pages, 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.

The following table identifies the transaction scenarios for each connector.

Connector	Message Exchange Pattern (MEP): Request / Reply	Message Exchange Pattern (MEP): Publish
HTTP Connector	✓	
SOAP Connector	✓	✓
JMS Connector	✓	✓

XD Connector	✓	
MQ Connector	✓	✓

HTTP Connector Operation

HTTP Connector

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.

HTTP Connector Request-Reply Operation

Methods

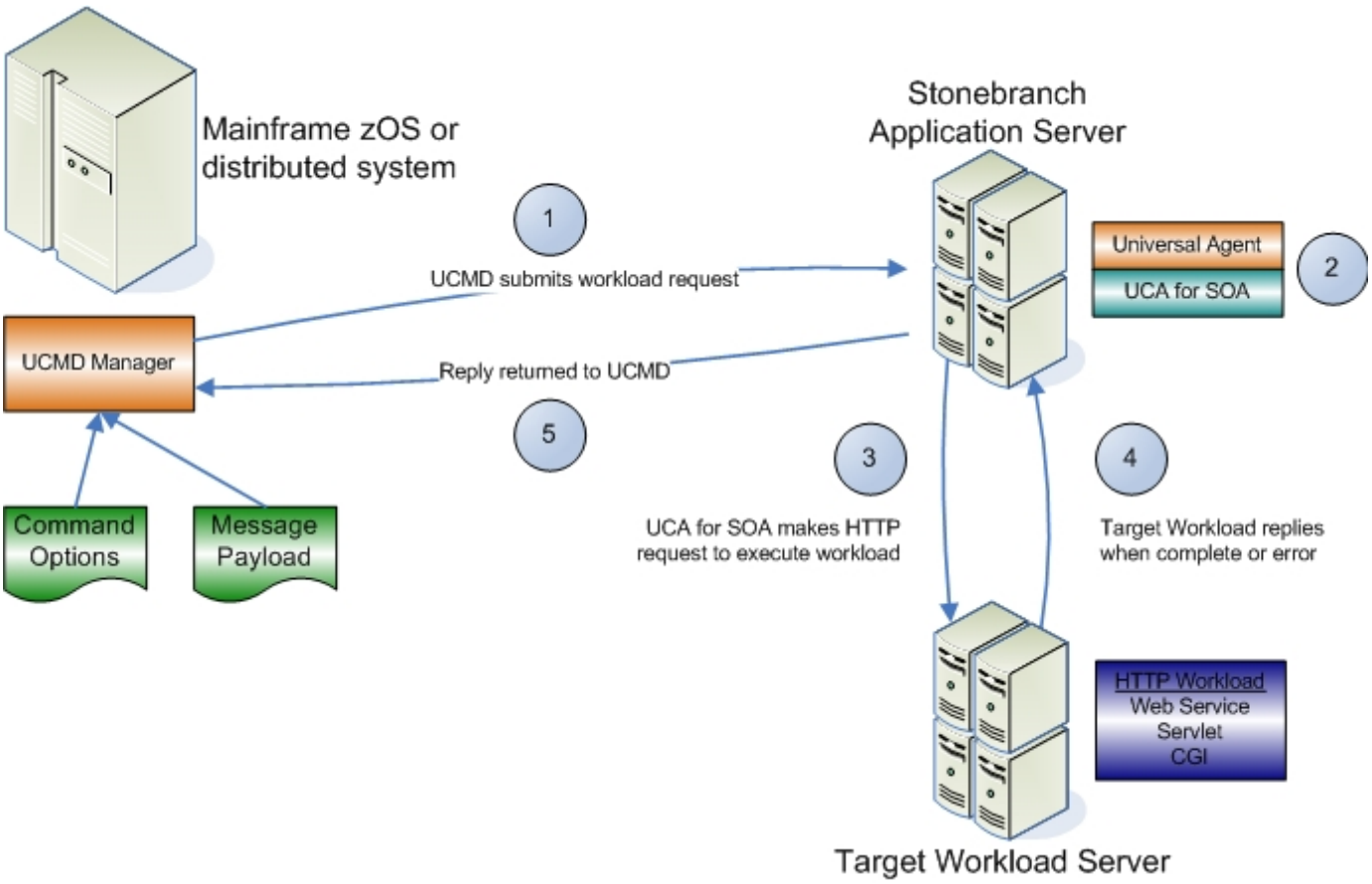
There are two methods of HTTP Connector operation: GET and POST.

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.

System Flow

The following figure illustrates the system flow for an HTTP Connector Request / Reply operation using Universal Command Agent for SOA.



System Flow Description

The following list describes the steps (1 - 5) illustrated above:

Step 1	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).
Step 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.
Step 3	Universal Command Agent for SOA: HTTP Connector sends the workload execution message via HTTP and blocks for the reply.
Step 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.
	UAC replies to UAI which returns the relevant information to UCMD.

Step 5	

HTTP Connector Request-Reply Operation - Usage

Overview

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 (SCRIPT_FILE) option.

Universal Command Options

The following figure illustrates the Universal Command options to execute the HTTP request.

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

Script File

The following figure illustrates the script file to request the HTTP service.

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

Note
The script file illustrated above is the argument to the -script (SCRIPT_FILE) option for Universal Command shown in the first figure.

Command File

The command options shown in the first figure can be saved in a file and invoked with Universal Command via the -file (COMMAND_FILE_PLAIN) option, as illustrated in the following figure.

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

HTTP Connector Request-Reply Operation - Required Command Options

The following table identifies the options (and their values) that are required to initiate an HTTP Connector Request/Reply operation.

For detailed information on these required command options, and all command options, see Universal Command Agent for SOA Command 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

SOAP Connector Operation

SOAP Connector

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.

SOAP Connector Request-Reply Operation

Overview

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

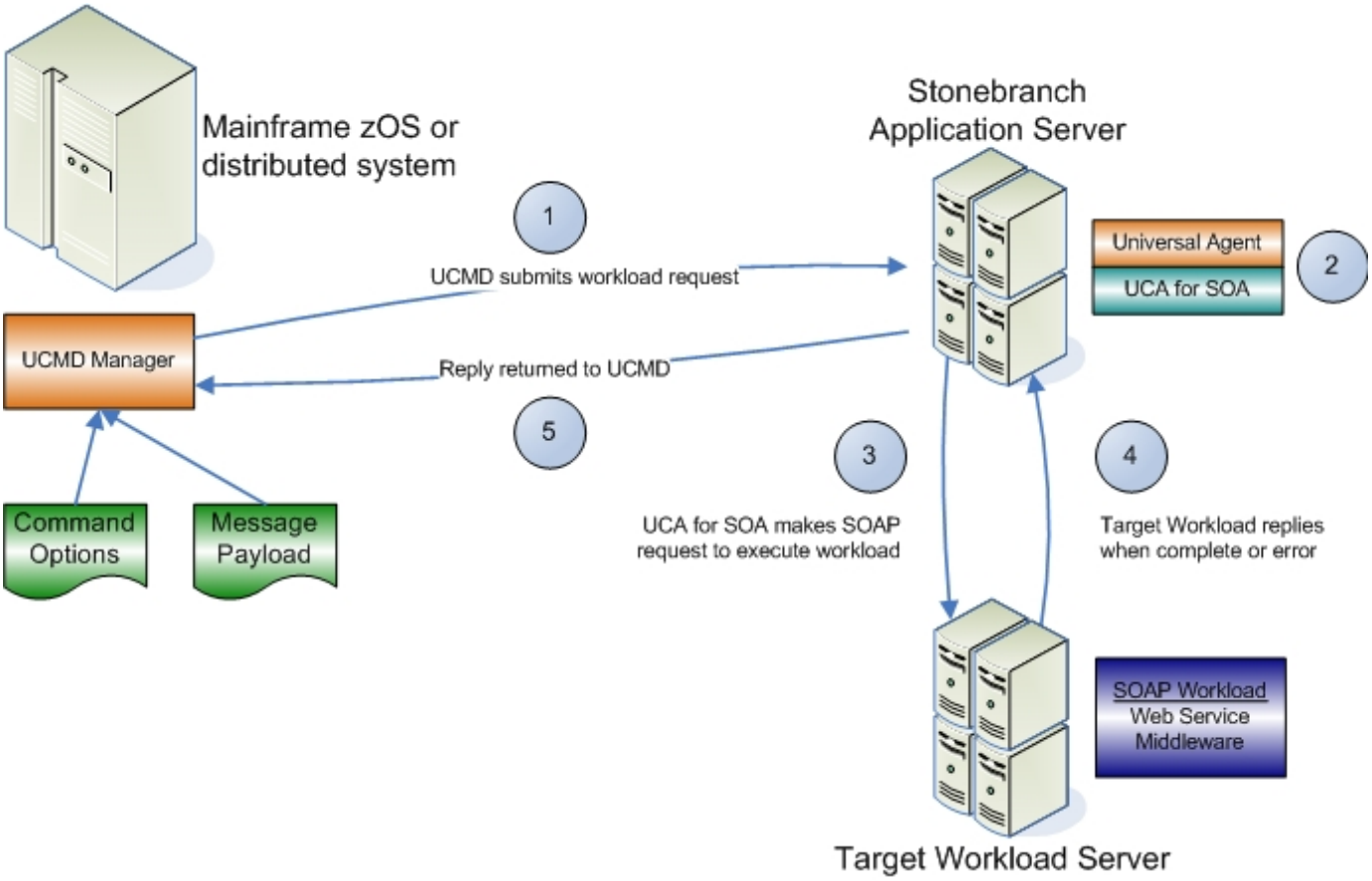
In this type, the request blocks until a reply is received from the target workload containing data, presumably after either:

- 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.

System Flow

The following figure illustrates the system flow for a SOAP Connector Request / Reply operation using the Universal Command Agent for SOA: SOAP Connector.



System Flow Description

The following list describes the steps (1 - 5) identified above:

Step 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).
Step 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.
Step 3	Universal Command Agent for SOA: SOAP Connector sends the workload execution message via SOAP and blocks for the reply.
Step 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.
Step	UAC replies to UAI, which returns the relevant information to UCMD.

5	

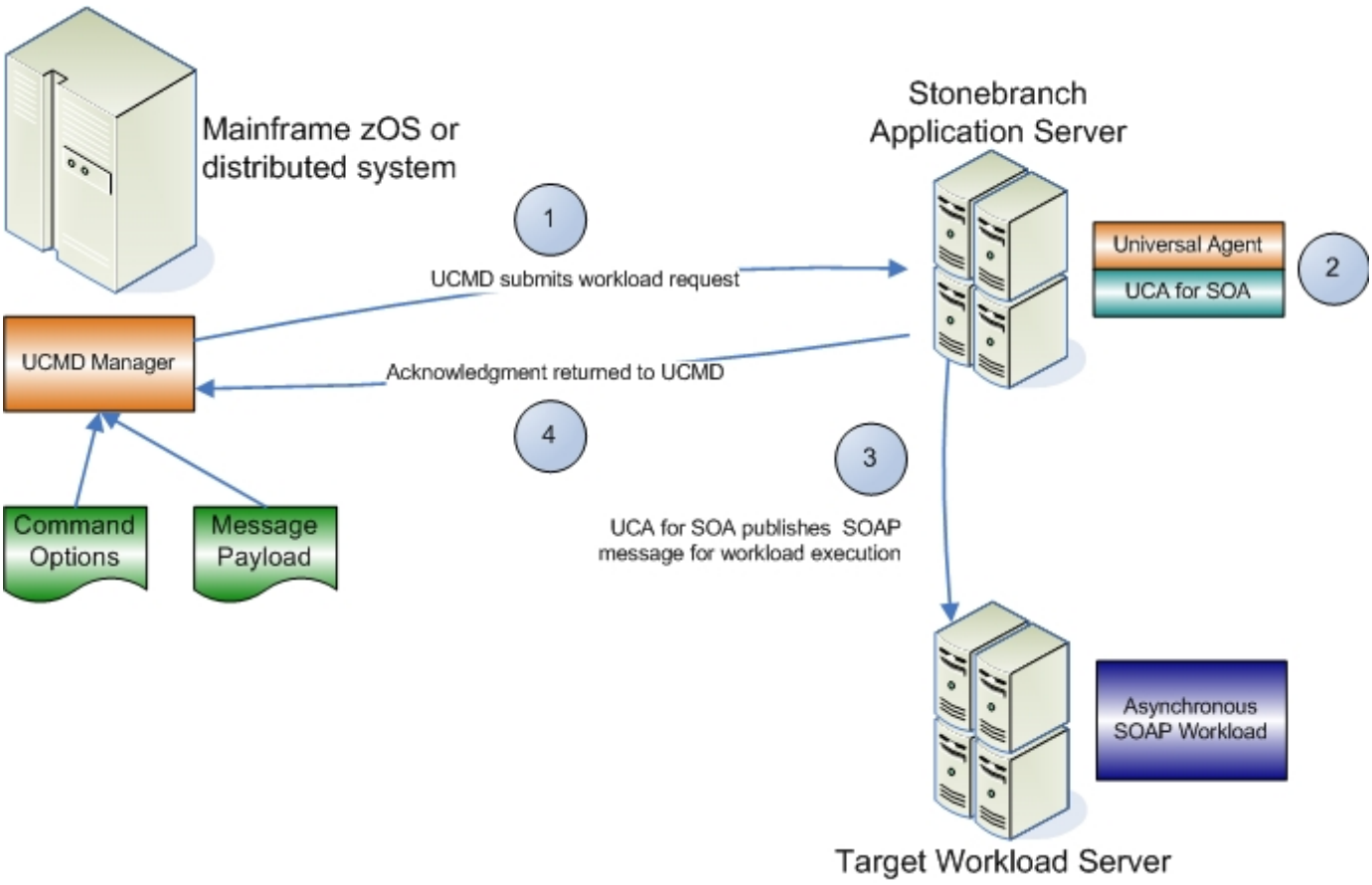
SOAP Connector Publish Operation

Overview

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.

System Flow

The following figure illustrates the system flow for a SOAP Connector Publish operation using the Universal Command Agent for SOA: SOAP Connector.



System Flow Description

The following list describes the steps (1 - 4) identified above:

Step 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).

Step 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.
Step 3	Universal Command Agent for SOA: SOAP Connector publishes the workload execution message via SOAP and the operation is complete.
Step 4	The Universal Command Agent for SOA: SOAP Connector returns the acknowledgement to UCMD.

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.

The following figure illustrates the Universal Command options to execute the SOAP operation.

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

The following figure illustrates the script file.

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

Note
The script file illustrated above is the argument to the -script (SCRIPT_FILE) option for Universal Command shown in the first figure. For the publish operation, the value for -mep would be **Publish**.

The command options shown in the first figure can be saved in a file and invoked with Universal Command via the -file (COMMAND_FILE_PLAIN) option, as shown in the following figure.

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

SOAP Connector Request-Reply Operation - Required Command Options

The following table identifies the options (and their values) that are required to initiate a SOAP Request/Reply operation.

For detailed information on these required command options, and all command options, see Universal Command Agent for SOA Command Options.

Option	Value	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

SOAP Connector Publish Operation - Required Command Options

The following table identifies the options (and their values) that are required to initiate a SOAP Publish operation.

For detailed information on these required command options, and all command options, see Universal Command Agent for SOA Command Options.

Option	Value	Description
PROTOCOL	SOAP	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 target workload in the form of: http://machine:port/service_name

JMS Connector Operation

JMS Connector

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.

JMS Provider Client Jar Files for Outbound

As mentioned in JMS Provider Client Jar Files, set-up 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.

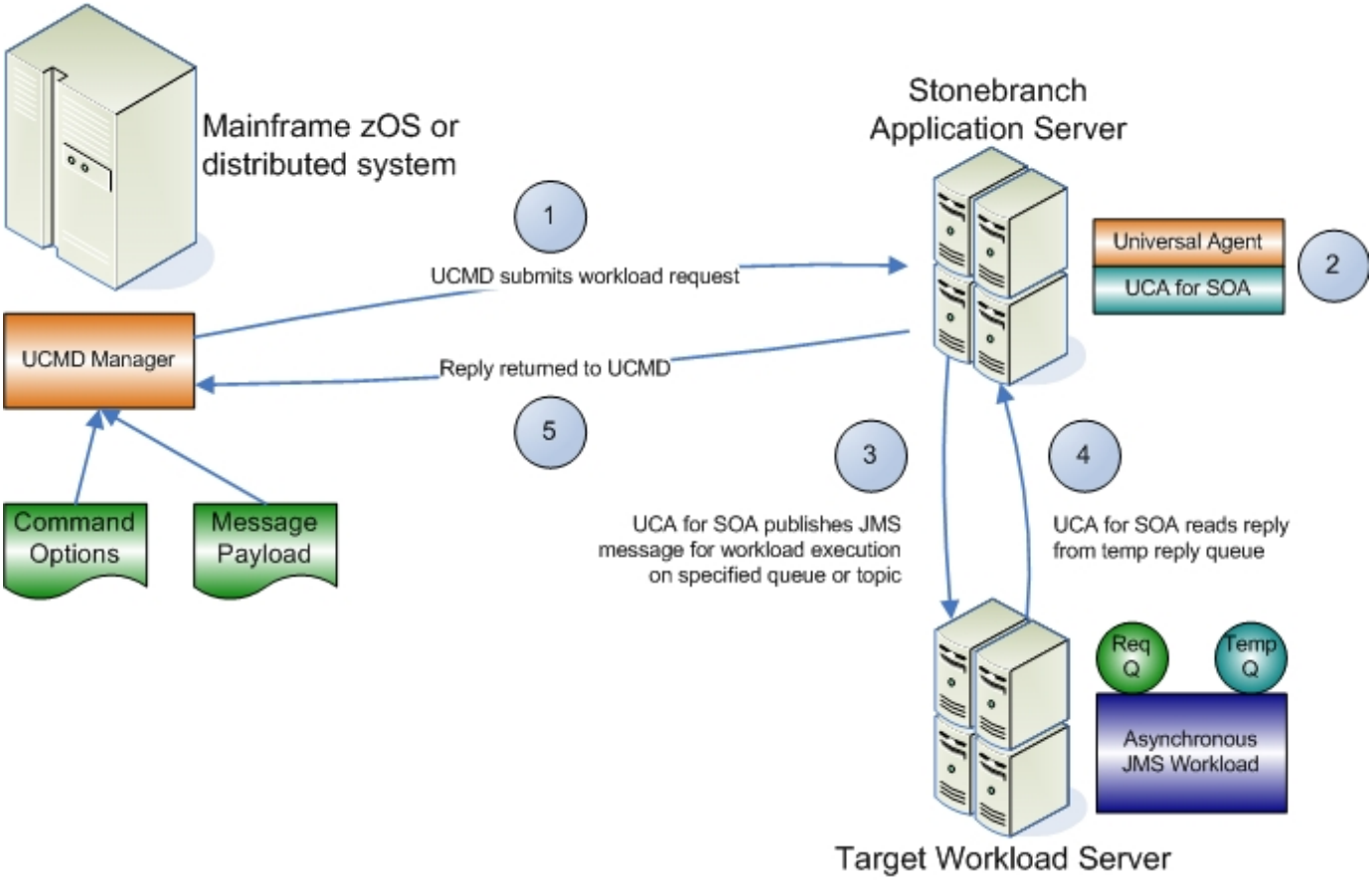
JMS Connector Request-Reply Operation

Overview

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

System Flow

The following figure illustrates the system flow for a JMS request / reply operation using the Universal Command Agent for SOA: JMS Connector.



System Flow Description

The following list describes the steps (1 - 5) identified above:

Step 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).
Step 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.
Step 3	Universal Command Agent for SOA: JMS Connector publishes the workload execution message to the specified destination queue.
Step 4	Universal Command Agent for SOA: JMS Connector then reads the reply message off of the temporary reply queue specified in the properties file.
Ste	UAC returns the reply message to UCMD (or an error message, if the operation failed).

Step
5

JMS Connector Publish Operation

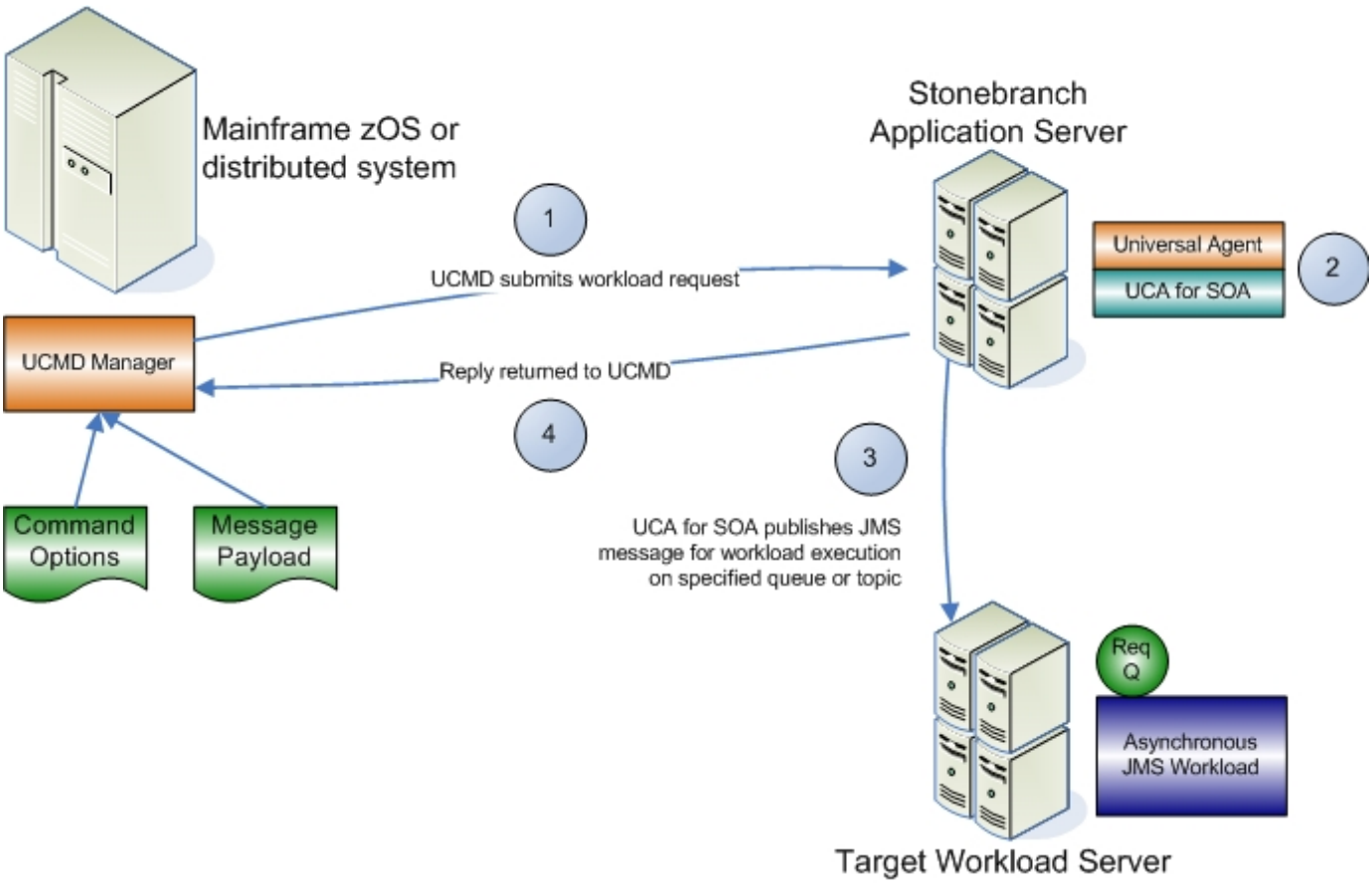
Overview

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.

System Flow

The following figure illustrates the system flow for a JMS publish operation using the Universal Command Agent for SOA: JMS Connector.



System Flow Description

The following list describes the steps (1 - 4) identified above:

Step	
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.

1	
Step 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.
Step 3	Universal Command Agent for SOA: JMS Connector publishes the workload execution message to the specified queue or topic.
Step 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.

JMS Connector Request-Reply Operation - Usage

Universal Command Options

The following figure illustrates the Universal Command options required to execute the JMS Connector Request/Reply operation.

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

Script File

The following figure 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
```

Note
The script file illustrated above is the argument to the -script (SCRIPT_FILE) option for Universal Command shown in the first figure.

Command File

The command options shown in the first figure can be saved in a file and invoked with UCMD via the -file (COMMAND_FILE_PLAIN) option, as shown in the following figure.

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

JMS Connector Publish Operation - Usage

Overview

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).

Universal Command Options

The following figure illustrates the Universal Command options to execute the JMS Connector Publish operation.

```
-script OutboundJMS_Queues_Options.txt
-script_type SERVICE
-host server1
-login YES
-userid abc
-pwd 123
```

Script File

The following figure illustrates the script file.

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

Note
The script file illustrated above is the argument to the -script (SCRIPT_FILE) option for Universal Command shown in the first figure.

Command File

The command options shown in the first figure can be saved in a file and invoked with Universal Command via the -file (COMMAND_FILE_PLAIN) option, as shown in the following figure.

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

JMS Connector Request-Reply Operation - Required Command Options

The following table describes the options (and their values) required to initiate a JMS Request / Reply operation.

For detailed information on these required command options, and all command options, see Universal Command Agent for SOA Command 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)

JMS Connector Publish Operation - Required Command Options

The following table describes the options (and their values) required to initiate a JMS Connector Publish operation.

For detailed information on these required command options, and all command options, see Universal Command Agent for SOA Command Options.

Option	Value	Description
PROTOCOL	JMS	Connector 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)

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.

XD Connector Deployment

Overview

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.

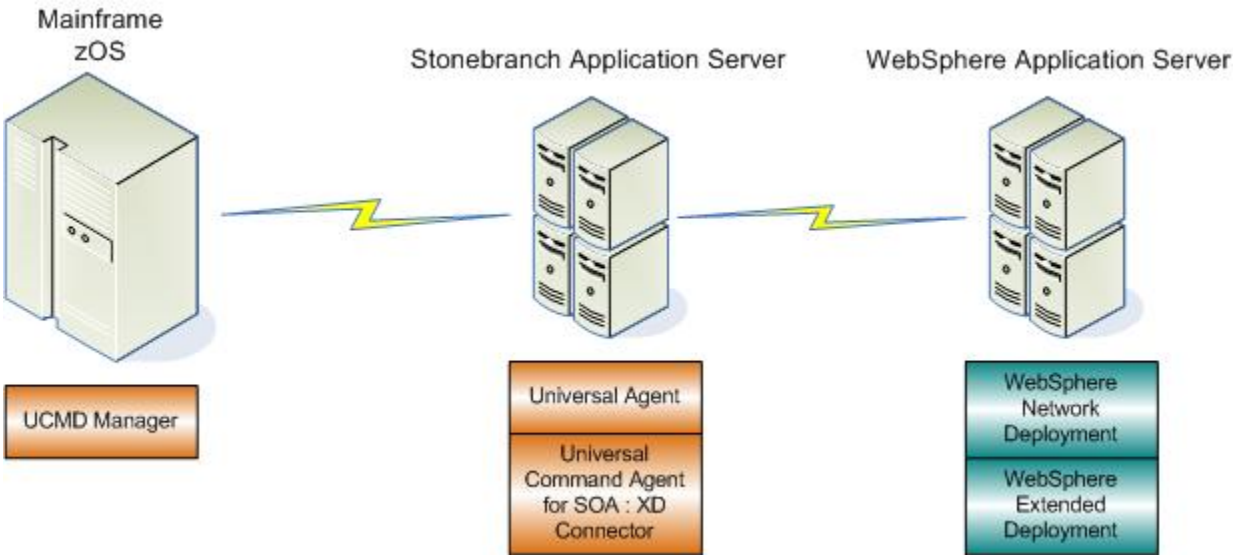
Distributed Server

There are two distributed servers in this environment:

1. Agent
   Server where the Universal Broker / Server 6.7.x 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.

Illustrated Environment

The following figure illustrates this environment.



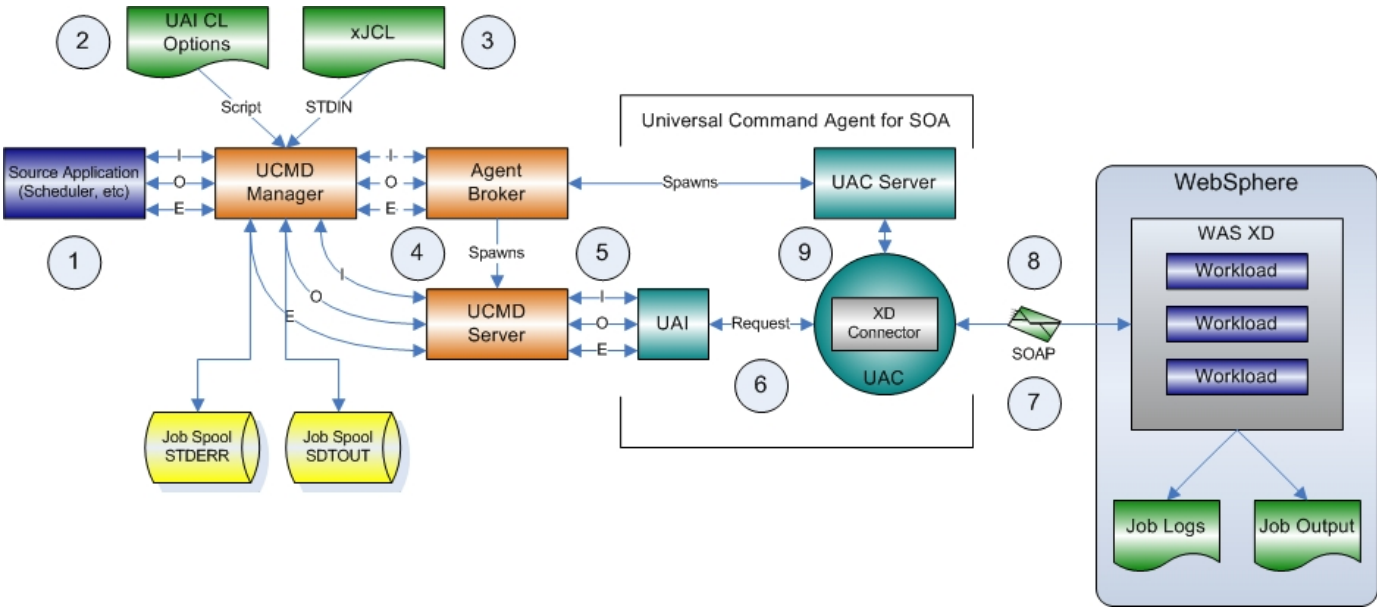
XD Connector Request-Reply Operation

Overview

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.

Logical System Flow

The following figure illustrates a more detailed logical flow of the XD connector operation.



Logical System Flow Description

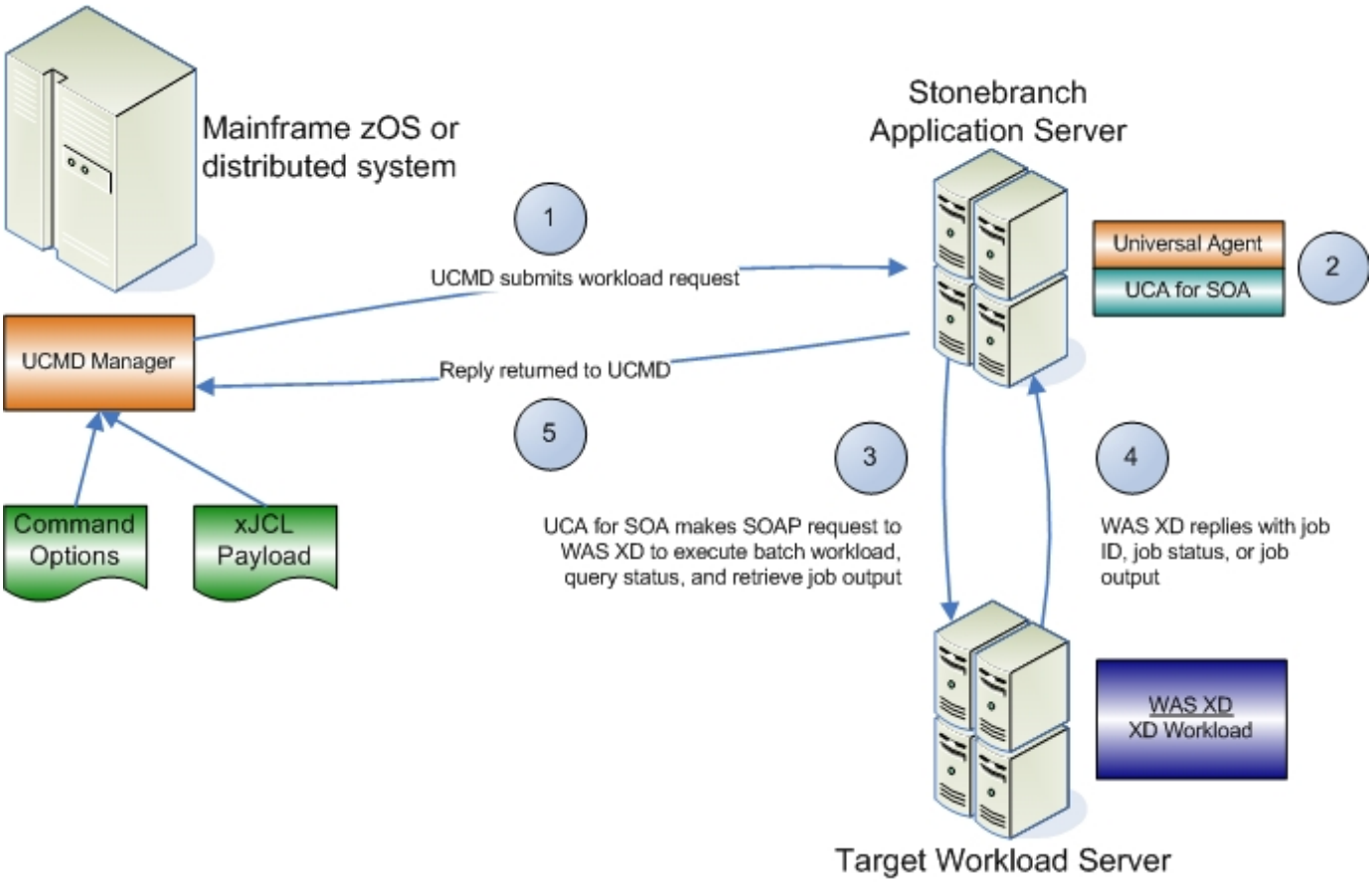
The flow is described as follows (from left to right):

Step 1	Scheduler invokes the UCMD Manager with the appropriate command line options.
Step 2	UCMD Manager reads the script file that contains that contains the XD Connector command options.
Step 3	UCMD Manager reads the xJCL file from STDIN, as indicated by the UCMD options.
Step	UCMD Manager requests action by the Broker, which spawns the UCMD Server and sets up communication between the UCMD Manager and UCMD Server.

4	
Step 5	UCMD Server sends the XD Connector command options and xJCL to UAI via STDIN.
Step 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.
Step 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.
Step 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.
Step 9	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.

Physical System Flow

The following figure illustrates the physical system flow of the XD Connector.



XD Connector Request-Reply Operation - Usage

Overview

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.

Universal Command Options

The following figure illustrates the Universal Command options to execute the XD Connector operation.

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

Script File

The following figure 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
```

Note
The script file illustrated above is the argument to the -script option for Universal Command shown in the first figure.

Command File

The command options shown in the first figure can be saved in a file and invoked with Universal Command via the -file (COMMAND_FILE_PLAIN) option, as shown in the following figure.

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

XD Connector Request-Reply Operation - Required Command Options

The following table describes the options (and their values) required to initiate an XD Request / Reply operation.

For detailed information on these required command options, and all command options, see Universal Command Agent for SOA Command 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	SUBMIT, 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.

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.

MQ Connector Operation

- MQ Connector

MQ Connector

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.

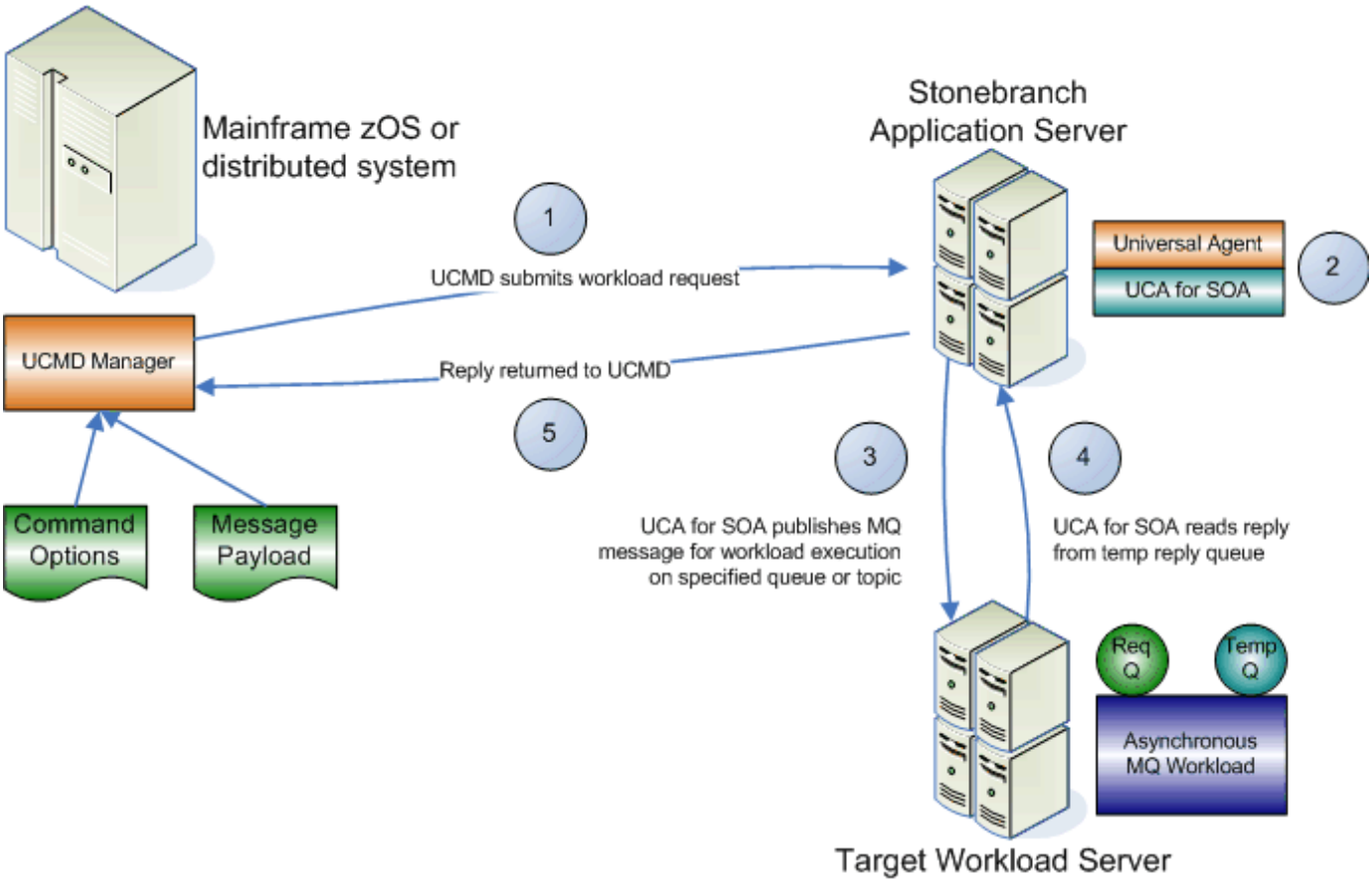
MQ Connector Request-Reply Operation

Overview

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

System Flow

The following figure illustrates the system flow for an MQ request / reply operation using the Universal Command Agent for SOA: MQ Connector.



System Flow Description

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

Step 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).
Step 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.
Step 3	Universal Command Agent for SOA: MQ Connector publishes the workload execution message to the specified destination queue.
Step 4	Universal Command Agent for SOA: MQ Connector then reads the reply message off of the temporary reply queue specified in supplied options.
Ste	UAC returns the reply message to UCMD (or an error message, if the operation failed).

p **5**	

MQ Connector Publish Operation

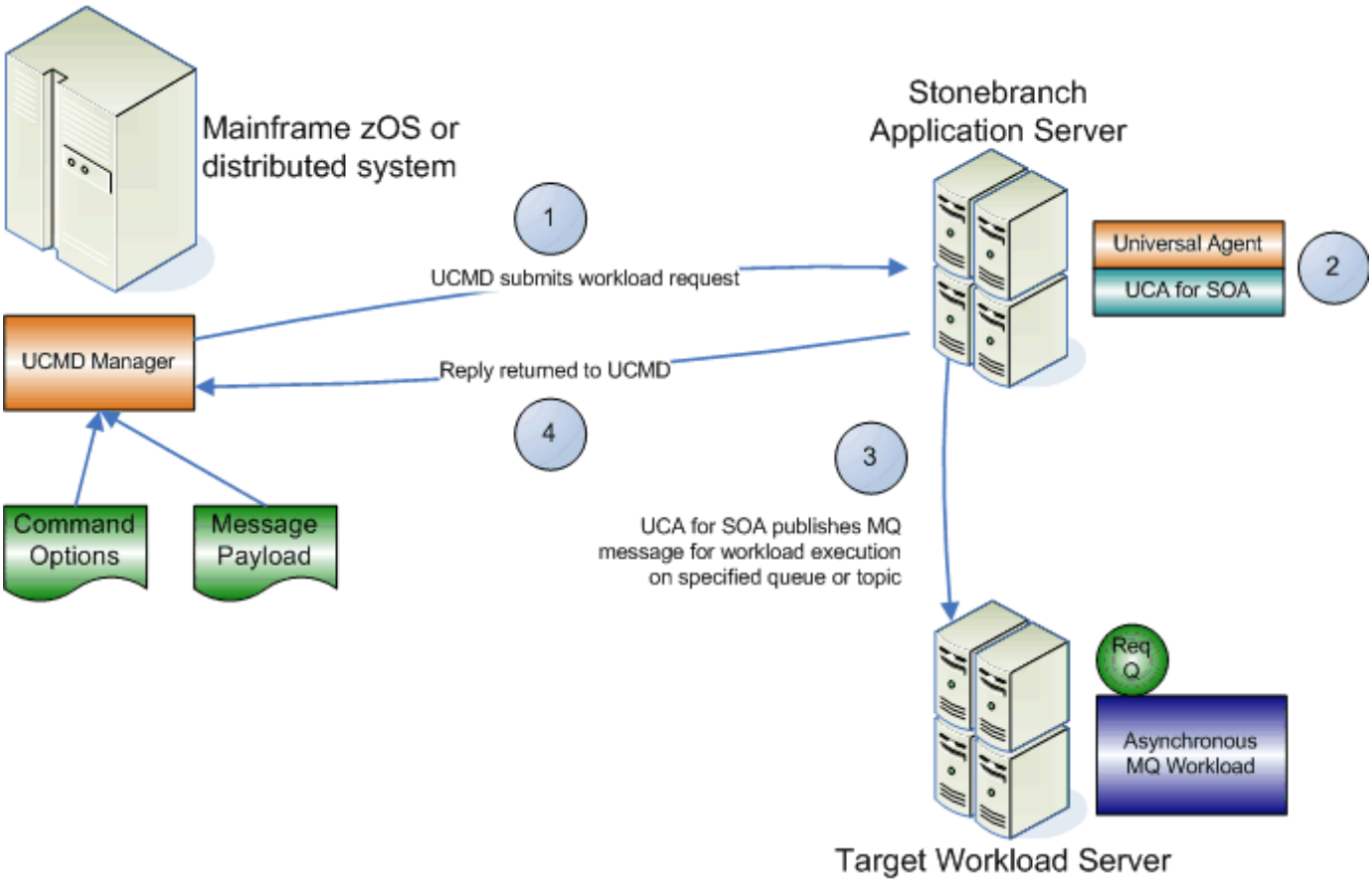
Overview

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.

System Flow

The following figure illustrates the system flow for an MQ publish operation using the Universal Command Agent for SOA: MQ Connector.



System Flow Description

The following list describes the steps (1 - 4) identified in the illustration above:

Step 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.

Step 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.
Step 3	Universal Command Agent for SOA: MQ Connector publishes the workload execution message to the specified queue.
Step 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.

MQ Connector Request-Reply Operation - Required Command Options

The following table describes the options (and their values) required to initiate an MQ Connector Request / Reply operation.

For detailed information on these required command options, and all command options, see Universal Command Agent for SOA Command 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.
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.

MQ Connector Publish Operation - Required Command Options

The following table describes the command options (and their values) required to initiate an MQ Connector Publish operation.

For detailed information on these required command options, and all command options, see Universal Command Agent for SOA Command Options.

Option	Value	Description
PROTOCOL	MQ	Connector UAC will use for the current operation.
MEP	Publish	Specification that the operation will be a publish operation.
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.

Universal Command Agent for SOA Logging Configuration

Overview

These pages provide information of how 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 Universal Application Container (UAC) and Universal Application Interface (UAI).

Logging Levels

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.

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**.

Appenders (Sinks) Available to UAC

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.

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**.

Appenders (Sinks) Available to UAI

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.

Universal Command Agent for SOA Additional Information

The following table identifies and provides links to additional information related to Universal Command Agent for SOA.

Information	Description
Character Code Pages	Character Code pages for use with Universal Command.
UTT Files	Universal Translate Table (UTT) files are used to translate between Unicode and the local single-byte code page.

Character Code Pages - UCA for SOA

The following table identifies the character code pages provided by Stonebranch Inc. for use with Universal Agent on each supported operating system.

Code Page	CCSID	z/OS	UNIX	Windows	IBM i / HFS	IBM i / LIB	HP NonStop
IBM037	037	✓			✓	✓	
IBM273	273	✓			✓	✓	
IBM277	277	✓			✓	✓	
IBM278	278	✓			✓	✓	
IBM280	280	✓			✓	✓	
IBM284	284	✓			✓	✓	
IBM500	500	✓			✓	✓	
IBM875	875	✓					
IBM1025		✓					
IBM1047							
IBM1140	1140	✓			✓	✓	
IBM1141	1141	✓			✓	✓	
IBM1142	1142	✓			✓	✓	
IBM1142	1143	✓			✓	✓	
IBM1144	1144	✓			✓	✓	
IBM1145	1145	✓			✓	✓	
IBM1146	1146	✓			✓	✓	
IBM1147	1147	✓			✓	✓	
IBM1148	1148	✓			✓	✓	
IBM4971	4971	✓			✓	✓	
ISO8859-1	819		✓	✓	✓		✓
ISO8859-2	912		✓	✓	✓		✓
ISO8859-3	913		✓	✓	✓		✓
ISO8859-4	914		✓	✓	✓		✓
ISO8859-5	915		✓	✓	✓		✓
ISO8859-6	1089		✓	✓	✓		✓
ISO8859-7	813		✓	✓	✓		✓
ISO8859-8	916		✓	✓	✓		✓
ISO8859-9	920		✓	✓	✓		✓
ISO8859-10			✓	✓	✓		✓
ISO8859-13	921		✓	✓	✓		✓
ISO8859-14			✓	✓	✓		✓
ISO8859-15	923		✓	✓	✓		✓
PC437	437			✓	✓		

PC737	737			✓	✓		
PC775	775			✓	✓		
PC850	850			✓	✓		
PC852	852			✓	✓		
PC855	855			✓	✓		
PC857	857			✓	✓		
PC860	860			✓	✓		
PC861	861			✓	✓		
PC862	862			✓	✓		
PC863	863			✓	✓		
PC864	864			✓	✓		
PC865	864			✓	✓		
PC866	866			✓	✓		
PC869	869			✓	✓		
PC874	874			✓	✓		
WIN1250	1250			✓	✓		
WIN1251	1251			✓	✓		
WIN1252	1252			✓	✓		
WIN1253	1253			✓	✓		
WIN1254	1254			✓	✓		
WIN1255	1255			✓	✓		
WIN1256	1256			✓	✓		
WIN1257	1257			✓	✓		
WIN1258	1258			✓	✓		

UTT Files - UCA for SOA

The following table identifies the Universal Translate Table (UTT) files that are used to translate between Unicode and the local single-byte code page.

Operating System	UTT File Location
IBM i	UTT files are located in the source physical file **UNVPRD520/UNVNLS**. *codepage* is the member name of the UTT file.
z/OS	UTT files are located in the library allocated to the **UNVNLS** ddname. *codepage* is the member name of the UTT file.
UNIX	UTT files are located in the **nls** subdirectory of the installation directory. *codepage* is the base file name of the UTT file. All UTT files end with an extension of **.utt**.
Windows	UTT files are located in the NLS subdirectory of the installation directory. *codepage* is the base file name of the UTT file. All UTT files end with an extension of **.utt**.
HP NonStop	UTT files are located in the **$SYSTEM.UNVNLS** subvolume. *codepage* is the base file name of the UTT file.