# HP Operations Orchestration Software

Software Version: 9.00

## *Software Development Kit Guide*

Document Release Date: June 2010

Software Release Date: June 2010

# Legal Notices

## Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

## Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

## Copyright Notices



## Trademark Notices

For information on open-source and third-party software acknowledgements, see in the documentation set for this release, Open-Source and Third-Party Software Acknowledgements (3rdPartyOpenNotices.pdf).

# On the Web: Finding OO support and documentation

There are two Web sites where you can find support and documentation, including updates to OO Help systems, guides, and tutorials:

- The OO Support site
- BSA Essentials Network

## Support

Documentation enhancements are a continual project at Hewlett-Packard Software. You can obtain or update the HP OO documentation set and tutorials at any time from the HP Software Product Manuals Web site. You will need an HP Passport to log in to the Web site.

**To obtain HP OO documentation and tutorials**

1. Go to the HP Software Product Manuals Web site (*http://support.openview.hp.com/selfsolve/manuals*).
2. Log in with your HP Passport user name and password.

   OR

   If you do not have an HP Passport, click **New users – please register** to create an HP Passport, then return to this page and log in.

   If you need help getting an HP Passport, see your HP OO contact.
3. In the **Product** list box, scroll down to and select **Operations Orchestration**.
4. In the **Product Version** list, click the version of the manuals that you're interested in.
5. In the **Operating System** list, click the relevant operating system.
6. Click the **Search** button.
7. In the **Results** list, click the link for the file that you want.

## BSA Essentials Network

For support information, including patches, troubleshooting aids, support contract management, product manuals and more, visit the following site: *http://www.hp.com/go/bsaessentialsnetwork*

This is the **BSA Essentials Network** Web page. To sign in:

1. Click **Login Now**.
2. On the **HP Passport sign-in** page, enter your HP Passport user ID and password and then click **Sign-in**.
3. If you do not already have an HP Passport account, do the following:
   a. On the **HP Passport sign-in** page, click **New user registration**.
   b. On the **HP Passport new user registration** page, *enter the required information and then click **Continue***.
   c. On the confirmation page that opens, check your information and then click **Register**.
   d. On the **Terms of Service** page, read the Terms of use and legal restrictions, select the **Agree** button, and then click **Submit**.
4. On the **BSA Essentials Network** page, click **Operations Orchestration Community.**

   **The Operations Orchestration Community** page contains links to announcements, discussions, downloads, documentation, help, and support.

**Note:** Contact your OO contact if you have any difficulties with this process.

# In OO: How to find Help, PDFs, and tutorials

The HP Operations Orchestration software (HP OO) documentation set is made up of the following:

- Help for Central

  Central Help provides information to the following:

  - Finding and running flows
  - For HP OO administrators, configuring the functioning of HP OO
  - Generating and viewing the information available from the outcomes of flow runs

  The Central Help system is also available as a PDF document in the HP OO home directory, in the Central\docs\ subdirectory.
- Help for Studio

  Studio Help instructs flow authors at varying levels of programming ability.

  The Studio Help system is also available as a PDF document in the HP OO home directory, in the Studio\docs\ subdirectory.
- Animated tutorials for Central and Studio

  HP OO tutorials can each be completed in less than half an hour and provide basic instruction on the following:

  - In Central, finding, running, and viewing information from flows
  - In Studio, modifying flows

  The tutorials are available in the Central and Studio subdirectories of the HP OO home directory.
- Self-documentation for operations and flows in the Accelerator Packs and ITIL folders

  Self-documentation is available in the descriptions of the operations and steps that are included in the flows.

# Table of Contents

# Welcome to the Operations Orchestration SDK

The Operations Orchestration Software Development Kit (SDK) contains documentation, tools, libraries, and code samples for developers and IT professionals who want to:

- Learn best practices for designing flows, steps, and operations.
- Create IActions to run Operations Orchestration (OO) operations through a Remote Action Service (RAS).
- Find and run flows from outside Central.
- Run repository functions from outside Studio.
- Package new and updated content for distribution on Central and RAS servers.
- Inspect a repository.
- Automate flow testing.
- Debug OO client/server problems.

## SDK contents

To install the SDK, you just unzip the OO 9.00 SDK.zip file which is available on the BSA Essentials Web site (*www.www2.hp.com*). You can install the SDK in any location on a Central or RAS server. The code samples in this guide can be placed anywhere and you can use most development tools to point to the code and import and use it as if it was on Central.

In this guide, the folder where you install the SDK is referred to as the OO SDK home directory. The basic folder structure of the OO SDK home directory looks like this:

- docs\ folder
  - javadocs\ folder

  SDKGuide.pdf
- lib\ folder

  ContentCommons-9.00.jar

  dharma-commons-9.00.jar

  JRAS-sdk-9.00.jar

  wscentral.dll

  WSCentralService.jar
- samples\ folder

AutoTest.jar

ContentPackager.jar

JRSFlowInvoke.jar

RepoInspector.jar

RepoTool.jar

RSFlowInvoke.exe

sdk_contents.txt

The SDK contains the following components:

**SDKGuide.pdf**

The documentation for the entire SDK. It includes conceptual information, descriptions of and step-by-step instructions for using tools, command syntaxes, class and method syntaxes, code examples, and code samples. The SDKGuide.pdf file is located in the OO SDK home directory, in the docs\ folder.

**IAction interface, methods, and classes**

IAction interface, methods, and classes that allow you to author Java and .NET IActions—code that implements OO operations through a Remote Action Service (RAS). The IAction interface, methods, and classes are located in the OO SDK home directory, in the lib\ folder.

**API documents**

javadocs for both the JRAS and Central. The javadocs are located in the OO SDK home directory, in the docs\ folder.

**WSCentralService SOAP API**

The WSCentralService API Java and .NET classes and interfaces are located in the OO SDK home directory in the lib\ folder. The certificates, keystore, WSDL, and sample code are located in the OO SDK home directory, in the samples\ folder.

**Samples**

IAction Java sample code and WS Central Service SOAP API sample code.

**AutoTest.jar**

A utility allows you to run automated tests. AutoTest.jar is located in the OO SDK home folder.

**ContentPackager.jar**

Tools and commands that allow you to package and install OO content updates. ContentPackager.jar is located in the OO SDK home directory.

**RepoInspector.jar**

A utility that allows you to check the repository. RepoInspector.jar is located in the OO SDK home directory.

**RepoTool.jar**

A utility that allows you to perform a number of repository functions from outside Studio. The RepoTool.jar utility is located in the OO SDK home directory.

**RSFlowInvoke.exe** and **JRSFlowInvoke.jar**

The Windows and Java Versions of a utility with which you find and run OO flows outside of Central from a command line, an application that uses a command line, a script, or a batch file. RSFlowInvoke.exe and JRSFlowInvoke.jar are located in the OO SDK home directory.

## About the SDK Guide

The *SDK Guide* is composed of the following chapters.

**Welcome to the OO SDK**

This chapter shows you the folder structure of the installed SDK, explains the SDK contents, and describes the chapters of the *OO SDK Guide*.

**Content style guide and best practices**

This chapter explains how OO content is organized in Studio, provides guidelines for flow layout and naming conventions, and best practices for creating flows, steps, and operations.

**Authoring IActions**

This chapter explains how to use the IAction interface, methods, and classes to create Java and .NET IActions—OO operations that are implemented through a RAS. It also explains how to load your IActions into Studio and debug them.

**Finding and running flows from outside Central**

This chapter details the ways in which you can manage flows outside of Central using:

- URLS created in Central.
- Command-line tools that access the REST service—Wget.exe, RSFlowInvoke.exe, and JRSFlowInvoke.jar.

- The WSCentralService SOAP API.

**Working with repositories from outside Studio**

This chapter describes how to use the RepoTool.jar utility to perform repository functions from outside Studio.

**Packaging content**

This chapter explains how to use the Content Packager utility to package updated content and publish it to Central and RAS servers in your network.

**Inspecting a repository**

This chapter explains how to use the RepoInspector utility to check a repository.

**Automating flow testing**

This chapter explains how to use the AutoTest utility to stress test your flows.

**Debugging OO client/servers problems**

This chapter explains how to allow HTTP connections to Central and RAS for debugging purposes.

# Content style guide and best practices

This chapter provides style guidelines and best practices for creating content—operations and flows—in Hewlett-Packard Operations Orchestration (HP OO).

These guidelines and best practices are intended for use by content and QA engineers, field engineers, and customers who want to create flows more quickly and efficiently. You should follow these guidelines for any content that you create and submit to the OO content community.

## How default OO content is organized in Studio

Default OO content consists of all the flows and operations that come with your installation of OO. These flows and operations are contained in folders in the Studio Library.

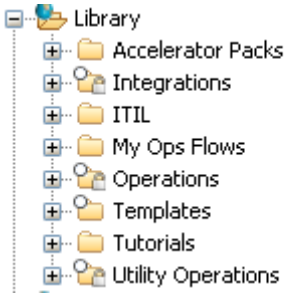


**Figure 1 - Folders in Studio Library**

The following describes the Studio Library folders that contain default content.

| Folder | Folder Contents |
|---|---|
| **Accelerator Packs** | Flows organized into subfolders by technologies, and are designed to solve common IT problems. For most networks, these flows:<br>• Perform complex health checks, triage, diagnosis, or remediation.<br>• Gather one or more pieces of data and display it to the user, or simply acknowledge alerts, gather data, and place it into a ticket.<br>The flows at the top level of an Accelerator Pack are usually full health check, triage, diagnosis, and remediation flows. |
| **Integrations** | Operations that can be used to integrate OO with other enterprise management software products, such as Hewlett-Packard Network Node Manager and BMC Remedy.<br>Because the enterprise software products used in your datacenter may be highly customized, you may need to create custom flows to use these operations. |
| **ITIL** | Flows that automate integrations with other enterprise-level software in accordance with Information Technology Infrastructure Library (ITIL) specifications, such as Change Management. |
| **Operations** | General-purpose operations and flows that work with common technologies. These operations are sealed and cannot be changed once you have installed OO Central.<br>The flows in the **Operations** folder and its subfolders are meant to be used as subflows. Flows that are meant to be run on their own are in the Accelerator Packs folder. |
| **Templates** | Templates that provide steps for flows that perform certain frequently used tasks. For example, the **Restart Service** template restarts a service, so you could use it in a flow that includes this task. |
| **Utility Operations** | Operations and subflows that gather and display data, replace simple command-line operations, manipulate and analyze data, provide structure to flows, and perform other tasks that are not specific to a technology. |

**Note:** The **My Ops Flows** folder is empty when you install OO. When you create flows from templates, OO automatically stores them here. You can also store flows that you create in this folder.

## Guidelines for flow layout

The layout of a flow should be as clear and uncomplicated as possible. This makes it easier for customers to understand what a flow does and how to use it as a subflow. When arranging your flow, consider the following guidelines:

- Lay out flows so that the Start step is in the upper-left corner of the flow and the flow reads down and to the right. However, you may occasionally need to bend or break this rule. For example, if:
  - The Start step of a flow has many responses, each of which leads to another step.
  - Placing the Start step in the upper-left corner would cause excessive visual complexity, such as the crossing of transitions.
- As much as possible, do not cross transition lines.

  See how much easier it is to read the following flow graph once you uncross the transitions.

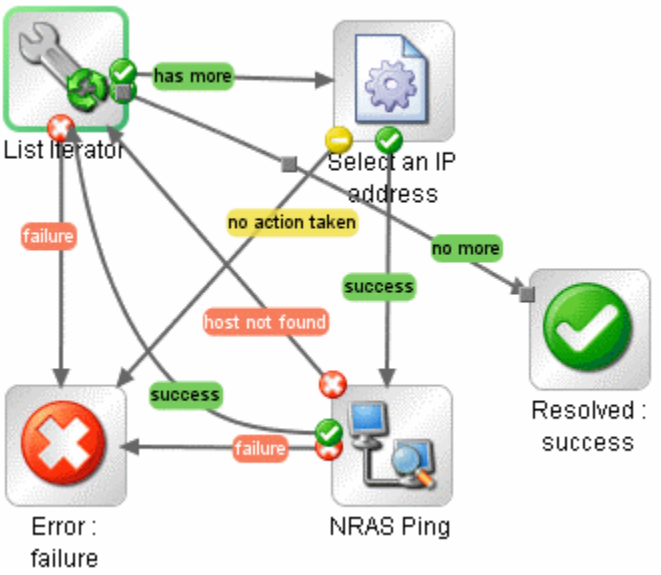


**Figure 2 - Flow with crossed transitions**

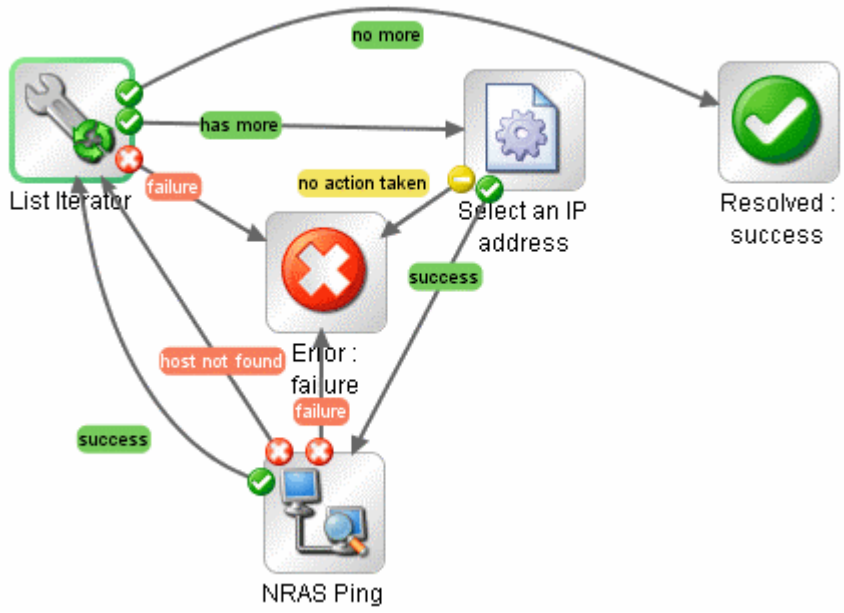


**Figure 3 - Transitions uncrossed**

- Position transition labels so that they are not superimposed on the step labels. The preceding illustration also demonstrates this principle.
- Place transition labels toward the outside of the flow when possible. For example, labels for transitions between steps at the top of the flow canvas should be above the transition lines. Labels for transitions between steps at the bottom of the canvas should be below the transition lines.
- If possible, and without doing violence to the other principles mentioned above, fit your flow to the **Authoring** pane on a 1024x768 screen with Studio maximized and a 1:1 view magnification.

  If a flow is larger than that, see whether you can break out some of its sequences of steps into subflows.

## Best practices for flows

The following best practices will make it easier for customers to use the flows you create.

**For flow inputs**

- Ideally, input values used by flow steps are supplied by flow inputs and passed to the steps by flow variables.

  This may not always be practical. For instance, a user might need to enter an input in response to a prompt somewhere in the flow run. In general, though, flow authors should assume that a user will begin a flow and then start another task while the flow is running. Assigning as much data as possible to flow inputs also simplifies making changes to the flow.

**For flow descriptions**

To help Central users who will use your flows and authors who will create other flows using them as subflows, add the following information to the flow's **Description** tab. (If you create multiple flows or operations that interact with the same technology, group them into a single folder and provide this information in the folder's **Description** tab. This is the practice for default HP OO content.) Note that putting this information on the **Description** tab makes it available to authors and Central users through the **Generate Documentation** feature. For more information on **Generate Documentation**, see the *Guide to Authoring Operations Orchestration Flows*.

- A **description** of what the flow does, including the following:
  - Any special requirements or changes that are necessary for the flow to run automatically (on a schedule or started from outside Central).
  - Limitations to the flow's usage. For example:

    ```
    Limitations:
    This flow only works:
    -- On Windows 2003 or later.
    -- If the Windows Telnet Service is enabled.
    -- If RAS is installed on a host running Microsoft Operations Manager.
    ```
- **Inputs** that the flow requires, including where authors can find the data that the inputs require and the required format for the data

  For instance, to enable schedules in Central to define different specific values for the input, in the **Input Type** box of the Studio input editor, you can do one of the following:
  - Select **Single Value**.
  - Select **Use Constant** and then enter `${<flowvariablename>}` in the **Constant Value** box, where `<flowvariablename>` is the name of a flow variable that exists and has had a value assigned to it by the time the flow arrives at this point.
- **Responses**, including the meaning of each response
- **Result fields**, including a description of the data supplied in each result field
- Any additional implementation **notes**, such as:
  - Supported platforms or applications, including version information

- Application or Web service APIs that the flow interacts with (this can be particularly important for flows that require an RAS to run, because the RAS operation can hide this information from the author or user of the flow.
- Other environmental or usage requirements

### Other best practices for flows

- A flow that performs triage, diagnosis, or remediation should first verify that a problem exists.
- A flow that sends a notification to the user should use notification subflows, which enable the flow author to choose from several means of notifying the user.

  For instance, the **Web site Health Check** flow uses the **Notify** subflow. Once the user configures the Web site Health Check flow to his or her e-mail and ticketing systems, all flows that use this flow will send notifications correctly.
- Annotate (that is, supply a description for) all transitions in a top-level parent flow.

  These transition descriptions should describe what happened in the step that preceded the transition. In Central, the Results Summary for the run displays the description for each transition, and so provides a running, high-level account of what took place in the flow run. You need not annotate transitions in a subflow unless the data is critical to see during a run in Central.

## Best practices for steps

To streamline the steps in a flow, consider using the following best practices for steps.

- Steps do not generally require descriptions, because (as described above) the transition description of the step's response tells what happened in the step.
- An operation may provide many results. Only results that the flow needs should be assigned to flow variables by the step.
- If a step or transition needs the exact error that came back from an operation, create a step result that captures the error code, and assign the error code to a flow variable.
- To assign information to a flow variable, use the step's **Results** tab. Filters on the results greatly enhance your flexibility in obtaining data from step results.
- Any time a step makes a modification to the IT environment, consider recording the data for Dashboard reporting in Central. If a change is made, reporting information should be recorded on the next step following the success transition. This often means that reporting information is recorded on flow return steps.

## Best practices for operations

To help authors who will create flows using the operations you create, use the following best practices for operations.

- Add the following information to the operation's **Description** tab. (If you create multiple flows or operations that interact with the same technology, group them into a single folder and provide this information in the folder's **Description** tab. This is the practice for default HP OO content.) Note that putting this information on the **Description** tab makes it available to authors and Central users through the **Generate Documentation** feature. For more information on **Generate Documentation**, see the *Guide to Authoring Operations Orchestration Flows*.
  - A description of what the operation does
  - **Inputs** that the operation requires, including where authors can find the data that the inputs require and the required format for the data
  - **Responses**, including the meaning of each response
  - **Result fields**, including a description of the data supplied in each result field

    - Any additional implementation notes, such as:
        - Supported platforms or applications, including version information
        - Application or Web service APIs that the flow interacts with (this can be particularly important for flows that require an RAS to run, because the RAS operation can hide this information from the author or user of the flow.
        - Other environmental or usage requirements
- Base your operation descriptions on the following template.

**Description template**

```
Description of what the operation does.

Inputs:

Input1 - info about this input
Input2 - info about this input
Input3 - info about this input

Responses:

Response1 - info about this response
Response2 - info about this response

Result:

The primary result of the flow/operation

Extra Results:

Result1 - The first additional result
Result2 - The second additional result
```

- Do not make copies of sealed operations, such as those in the **Operations** folder. Instead, make changes to the steps that you have created from sealed operations.
- By default, operations should use and set flow variables for inputs that are used repeatedly in a particular flow**.** For example, multiple operations in a flow might need the host, username, and password inputs to get information from a server or the port of a mail server. Assigning those values to flow variables that are used in the various steps that require such data simplifies maintenance of the flow and makes it easier to adapt to different situations.

  In contrast, the subject line of an e-mail is probably different for each step that requires an e-mail subject line. Therefore, the subject line is probably not a good candidate for being provided from a flow variable.
- Avoid creating multiple operations that run the same command**.** For example, you can get both packet loss and maximum latency from a ping operation. Rather than create multiple operations that use the ping command, a better practice is to capture both pieces of information in one step by using multiple outputs of one ping operation.

  Exceptions to this principle are operations that are extremely generic, such as an operation that runs a WMI command. It is better to create WMI command operations that are specific to particular functions, instead of a single operation that has a very generic input for the WMI command and very generic outputs.
- For capturing data from the output stream of a command, using result filters is better than using a scriptlet. There are several reasons:
    - Result filters are accessible and immediately visible on the **Results** tab editor rather than residing separately, as scriptlets do on the **Scriptlets** tab.
    - Scriptlets are more difficult for non-programmers to maintain.

- If one of the operation's results is removed, the result filters are automatically invalidated. Any scriptlets that the author fails to remove after deleting the result that the scriptlet manipulates remain and can cause bugs in the flow.

  If you need a scriptlet for the desired processing of the result data, you can use a scriptlet filter.
- Most operations should have only two responses—**success** and **failure**. Using a small number of responses makes flows easier to create and understand. Multiple responses based on different types of failures should only be used when there are obvious distinct paths to follow or there are circumstances where an outcome may only be a failure because of the situation (such as a redirect response to an HTTP Get).

  However, don't force this principle when it doesn't make sense. For example, an operation that gets data and checks a threshold may require three responses (none of which is a **success** response)—**failure**, **over threshold**, and **under threshold**.
- The default response for an operation should be **failure**. This way an incomplete operation shows as a failure during flow debugging and points the author to the problem before the flow goes into production.

## Naming convention guidelines

Using the following naming conventions will significantly help authors debug or modify flows and operations as necessary:

- Use Title Case (first letter capitalized for all except helper words like 'a', 'the', 'and', 'by', 'for') for:
  - Items in the Library (flows, folders, and operations, and items in the **Configuration** folder).

    For example: "Reboot a Server", "Check the Log Files", and, in the **Configuration\Domain Terms** folder, "CI Minor Type".
  - Step names.
- Use lower case for responses in an operation (spaces are permissible). For example—**failure**, **success**, **over threshold**.
- Use camel case (first letter of the name is lower-case; subsequent first letters of words contained in the name are upper-case) for:
  - Input names, such as **protocol** and **messageNumber**
  - Output names, such as **hopCountThreshold**
  - Result names, such as **aclData**
  - Flow variable names, such as **aclData** and **userId**

  No spaces or other non-alphanumeric characters are allowed in camel case names.
- Some common input names occur across many operations and steps. To make it as easy as possible to author using operations that are available immediately upon installing OO, the following input names are used in OO content:

  **host**

  For Windows, the host is the machine on which the operation works (for example, the host from which you are getting a performance counter or on which you are restarting a service). For secure shell (SSH) operations, the host is the machine on which the command is running.

  **username**

  The name of the account to use for logging on to the machine.

  **password**

  The password to use to log on to the machine.

  Use the following boilerplate to list these inputs in an operation description:

**Inputs boilerplate**

```
Inputs:

   host: The host to run the command against
   username: The user name to use when logging on to this machine
   password: The password to use when logging on to this machine
```

Other common input names include:

**mailHost**
The host machine from which an e-mail is sent.

**target**
When the host affects another system, the system that is affected by the host should be called the target. For example, if you SSH to server1 to run a ping against server2, then the host is server1 and the target is server2.

# Authoring IActions

This chapter defines IActions and Remote Action Service (RAS), and explains how to:

- Use the IAction interface and methods to author Java and .NET IActions.
- Load your IActions into OO Studio.
- Debug your IActions.

It also provides code examples, and useful Java and .NET Commons Library classes that may help you develop IActions.

## What is an IAction?

Within a Remote Action Service (RAS) operation], an IAction is the code that implements an operation through a RAS—a service that executes operations on machines that are remote from the Central server. You can use RAS operations to implement functionality that interacts with systems throughout your network or over the Internet. A good example of this is using RAS operations to integrate OO with other applications, platforms, and services.

Using a RAS operation instead of a scriptlet or command-line operation, allows the operation to run hosted on a RAS. The advantage of this is that you can have multiple RASes running in different network segments and run operations on any of them.

RAS operations are written in either Java or .NET. The RAS operations for Java are packaged in .jar files and those for .NET are packaged in .dll files.

**Important:** RAS installed on a Windows server supports both Java and .NET RAS operations. However, RAS installed on a Linux server only supports Java RAS operations—it does not support .NET RAS operations.

## About RAS

OO Central is installed with a default RAS named RAS_Operator_Path. You can also deploy OO RAS standalone—that is, on machines that are physically separate from the Central server. This process is explained in the *Installing HP Operations Orchestration* guide (InstallGuide.pdf).

The OO RAS contains the IAction interface which specifies how your action classes must be built. For an operation to be accessible to the RAS for execution, the class that holds that operation must implement the IAction interface.

The OO server uses HTTP over the Secure Socket Layer (HTTPS) protocol to initiate communications between itself and the RAS, so you can deploy a RAS on the other side of a firewall or domain boundary and have it execute code for OO.

Currently the OO RAS supports the platform and language combinations shown in the following table.

| Platform | Java | .NET |
|---|---|---|
| Windows 32-bit | X | X |
| Windows 64-bit | X | X |
| Linux 32-bit | X | |
| Linux 64-bit | X | |

## Creating IActions

To create an IAction you implement the IAction interface. This section explains how to use the IAction interface and provides you with guidelines and important points for creating IActions.

### About the IAction interface

The IAction interface specifies how to build your Action classes. These classes define the RAS operations and are stored in the .jar or .dll files associated with the RAS.

The IAction interface uses the execute method as well as methods that are specific to the Web application, standalone application, platform, or extension service for which you want the actions performed. The IAction interface mediates between OO and systems external to OO.

**Syntax**

Java

```
public interface IAction {

      ActionTemplate getActionTemplate();

   ActionResult execute(ISessionContext sessionContext,
   ActionRequest actionRequest, IActionRegistry actionRegistry)
   throws Exception;
}
```

.NET

```
public interface IAction
{
   ActionTemplate GetActionTemplate();

   ActionResult Execute(ActionRequest req, ISession s,
   IActionRegistry reg);

}
```

As you can see, the IAction interface defines the following public methods that are needed to create IActions:

- getActionTemplate method
- execute method

### getActionTemplate method

The getActionTemplate method returns the ActionTemplate object which describes the properties of the IAction to OO. These properties are shown in the following table.

| Property | Description |
|---|---|
| The description of your operation. | Model the description after the operation descriptions included with the built-in OO RAS operations. The description should include the following information for flow authors who may use the operation:<br>• An explanation of what the operation does.<br>• Definitions of the inputs to the operation.<br>• Definitions of the responses that are returned by the operation.<br>• Definitions of any other results that are returned by the operation. |
| A map of the inputs the operation needs. | Set the key to the name of the input. You can leave the value as a blank string or set it to a RASBinding object. A RASBinding object has properties that correspond to most of the options that are available on the **Inputs** tab of the Inspector in Studio. The order in which you add inputs to the map is the order in which they will appear in the operation once it is imported. |
| A map of the responses the operation returns | Set the key to the text that each response transition will show. The value must be the integer that is returned by the IAction's returnCode which tells Central the transition to follow when the operation has completed. |
| A map of any other results returned by the IAction that are available for use in a flow. | Set the key to the name of the result; the value must be a blank string. |

**Syntax**

Java

```
public class ActionTemplate {

    private String description;
    private String overrideRas = "${overrideJRAS}";
    private Map parameters;
    private Map resultFields;
    private Map responses;

    public ActionTemplate() { }
```

```
    public ActionTemplate(String description, Map parameters,
      Map resultFields, Map responses) {

       this.description = description;
       this.parameters = parameters;
       this.resultFields = resultFields;
       this.responses = responses;
    }

    /**
     * Gets the description value for this ActionTemplate.
     * @return description
     */
    public String getDescription() {
        return description;
    }

    /**
     * Sets the description value for this ActionTemplate.
     * @param description
     */
    public void setDescription(String description) {
        this.description = description;
    }

    /**
     * Gets the parameters value for this ActionTemplate.
     * @return parameters
     */
    public Map getParameters() {
        return parameters;
    }

    /**
     * Sets the parameters value for this ActionTemplate.
     * @param parameters
     */
    public void setParameters(Map parameters) {
        this.parameters = parameters;
    }

    /**
     * Gets the resultFields value for this ActionTemplate.
     * @return resultFields
     */
    public Map getResultFields() {
        return resultFields;
    }
```

```
    /**
     * Sets the resultFields value for this ActionTemplate.
     * @param resultFields
     */
    public void setResultFields(Map resultFields) {
        this.resultFields = resultFields;
    }

    /**
     * Gets the responses value for this ActionTemplate.
     * @return responses
     */
    public Map getResponses() {
        return responses;
    }

    /**
     * Sets the responses value for this ActionTemplate.
     * @param responses
     */
    public void setResponses(Map responses) {
        this.responses = responses;
    }

    /**
     * Sets the overrideRas value for this ActionTemplate.
     * @param overrideRas
     */
    public String getOverrideRas() {
      return overrideRas;
    }

    /**
     * Gets the overrideRas value for this ActionTemplate.
     * @param overrideRas
     */
    public void setOverrideRas(String overrideRas) {
      this.overrideRas = overrideRas;
    }
}
```

.NET

```
public class ActionTemplate {

    private String description;
```

```
    private String overrideRas = "${overrideJRAS}";
    private Map parameters;
    private Map resultFields;
    private Map responses;

    public ActionTemplate () { }

    public ActionTemplate (String description, Map parameters,
      Map resultFields, Map responses) {

       this.description = description;
       this.parameters = parameters;
       this.resultFields = resultFields;
       this.responses = responses;
    }

    /**
     * Gets the description value for this ActionTemplate
     * @return description
     */
    public String getDescription() {
        return description;
    }

    /**
     * Sets the description value for this ActionTemplate
     * @param description
     */
    public void setDescription(String description) {
        this.description = description;
    }

    /**
     * Gets the parameters value for this ActionTemplate.
     * @return parameters
     */
    public Map getParameters() {
        return parameters;
    }

    /**
     * Sets the parameters value for this ActionTemplate.
     * @param parameters
     */
    public void setParameters(Map parameters) {
        this.parameters = parameters;
    }
```

```
    /**
     * Gets the resultFields value for this ActionTemplate.
     * @return resultFields
     */
    public Map getResultFields() {
        return resultFields;
    }

    /**
     * Sets the resultFields value for this ActionTemplate.
     * @param resultFields
     */
    public void setResultFields(Map resultFields) {
        this.resultFields = resultFields;
    }

    /**
     * Gets the responses value for this ActionTemplate.
     * @return responses
     */
    public Map getResponses() {
        return responses;
    }

    /**
     * Sets the responses value for this ActionTemplate
     * @param responses
     */
    public void setResponses(Map responses) {
        this.responses = responses;
    }

    /**
     * Sets the overrideRas value for this ActionTemplate.
     * @param overrideRas
     */
    public String getOverrideRas() {
      return overrideRas;
    }

    /**
     * Gets the overrideRas value for this ActionTemplate.
     * @param overrideRas
     */
    public void setOverrideRas(String overrideRas) {
      this.overrideRas = overrideRas;
    }
}
```

### *RASBinding objects*

RASBinding objects expand the inputs in an ActionTemplate method. RASBindings allow you to identify exactly how the input is to be defined once it is imported into OO. This includes the default settings shown on the **Inputs** tab of the Inspector in Studio as shown in the following figure.

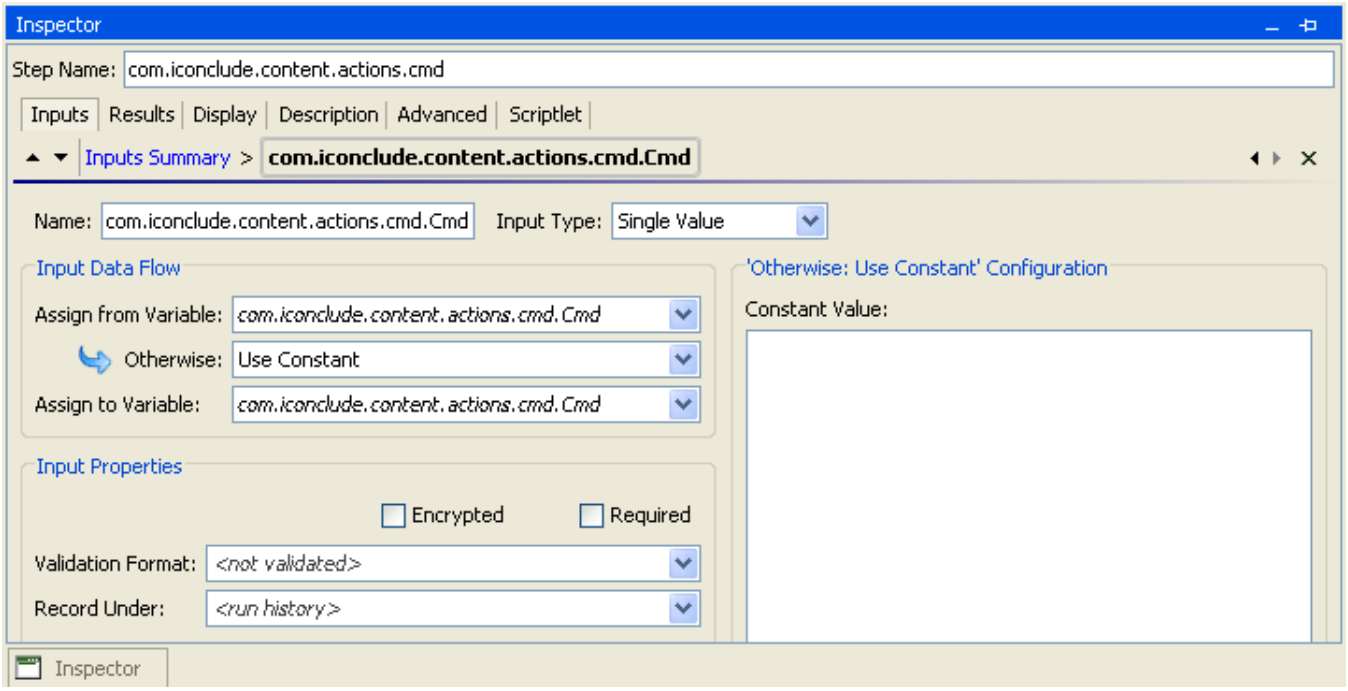


**Figure 4 - Inputs tab of Studio Inspector**

RASBindings have the properties shown in the following table.

| Property | Description |
|---|---|
| encrypted (Boolean, false) | Determines whether the particular input should be encrypted. |
| required (Boolean, false) | Determines whether the particular input should be required. |
| assignTo (Boolean, true) | Determines whether the **Assign to flow variable** check box is checked. |
| assignFrom (Boolean, true) | Determines whether the **Assign from flow variable** check box is checked. |
| assignFromText (String, empty) | Determines the text in the **Assign from flow variable** field. |
| assignToText (String, empty) | Determines the text in the **Assign to flow variable** field. |

| Property | Description |
| --- | --- |
| type (INPUT_TYPE, INPUT_TYPE.Empty) | Determines the type of input:<br>• **Empty**. This is an empty binding that will become a prompt if not changed.<br>• **Static**. This is a static binding. Whatever is entered for the value property will become the value of this input.<br>• **Prompt**. This is a prompt user binding. Whatever is entered in the value property will be the text that is used to prompt the user.<br>• **value (String, empty)**. The value that is assigned to either the static field or the prompt user field depending on the type specified. |

You can also use the RASBindingFactory method whose main purpose is to quickly create RASBindings with default style behaviors.

### Syntax

Java

```
public class RASBindingFactory {

   /*
    * Empty Bindings
    */
   public static RASBinding createEmptyRASBinding(){
      return new RASBinding();
   }

   public static RASBinding createEmptyRASBinding(boolean required,
         boolean encrypted){
      return updateBinding(createEmptyRASBinding(), required,
         encrypted);
   }

   /*
    * Prompts
    */
   public static RASBinding createPromptBinding(String value){
      return createBinding(value, RASBinding.INPUT_TYPE.Prompt);
   }

   public static RASBinding createPromptBinding(String value,
         boolean required){
      return updateBinding(createPromptBinding(value), required,
         false);
   }

   public static RASBinding createPromptBinding(String value,
         boolean required,boolean encrypted){
      return updateBinding(createPromptBinding(value), required,
         encrypted);
   }

   /*
    * Statics
    */
```

```
   public static RASBinding createStaticBinding(String value){
      return createBinding(value, RASBinding.INPUT_TYPE.Static);
   }

   public static RASBinding createStaticBinding(String value,
         boolean required){
      return updateBinding(createStaticBinding(value), required,
         false);
   }

   public static RASBinding createStaticBinding(String value,
         boolean required,boolean encrypted){
      return updateBinding(createStaticBinding(value), required,
encrypted);
   }

   /*
    * Private section
    */
   private static RASBinding createBinding(String value,
         RASBinding.INPUT_TYPE type){
      RASBinding r = new RASBinding();
      r.type = type;
      r.value = value;
      return r;
   }

   private static RASBinding updateBinding(RASBinding b,
         boolean required,boolean encrypted){
      b.required=required;
      b.encrypted=encrypted;
      return b;
   }
}
```

.NET

```
    public class RASBindingFactory
    {
        public static RASBinding createEmptyRASBinding()
        {
            return createGenericRASBindingWithValue(null,
               RASBinding.BindingType.Empty);
        }

        public static RASBinding createEmptyRASBinding
            (Boolean required, Boolean encrypted)
        {
            return createGenericRASBindingWithValue(null, required,
               encrypted, RASBinding.BindingType.Empty);
        }

        public static RASBinding createPromptBinding(String value)
        {
            return RASBindingFactory.createGenericRASBindingWithValue(
               value, RASBinding.BindingType.Prompt);
        }

        public static RASBinding createPromptBinding(String value,
            Boolean required)
```

```
        {
            return RASBindingFactory.createGenericRASBindingWithValue(
               value, required, RASBinding.BindingType.Prompt);
        }

        public static RASBinding createPromptBinding(String value,
            Boolean required, Boolean encrypted)
        {
            return RASBindingFactory.createGenericRASBindingWithValue(
               value, required, encrypted,
               RASBinding.BindingType.Prompt);
        }

        public static RASBinding createStaticBinding(String value)
        {
            return RASBindingFactory.createGenericRASBindingWithValue(
               value, RASBinding.BindingType.Static);
        }

        public static RASBinding createStaticBinding(String value,
            Boolean required)
        {
            return RASBindingFactory.createGenericRASBindingWithValue(
               value, required, RASBinding.BindingType.Static);
        }

        public static RASBinding createStaticBinding(String value,
            Boolean required, Boolean encrypted)
        {
            return RASBindingFactory.createGenericRASBindingWithValue(
               value, required, encrypted,
            RASBinding.BindingType.Static);
        }

        protected static RASBinding createGenericRASBindingWithValue(
               String value, RASBinding.BindingType type)
        {
            RASBinding r = new RASBinding();
            r.Binding = type;
            r.Value = (null != value ? value : "");
            return r;
        }

        protected static RASBinding createGenericRASBindingWithValue(
               String value, Boolean required,
               RASBinding.BindingType type)
        {
            RASBinding r = new RASBinding();
            r.Required = required;
            r.Binding = type;
            r.Value = (null != value ? value : "");
            return r;
        }

        protected static RASBinding createGenericRASBindingWithValue(
               String value, Boolean required, Boolean encrypted,
               RASBinding.BindingType type)
        {
            RASBinding r = new RASBinding();
            r.Encrypted = encrypted;
```

```
            r.Required = required;
            r.Binding = type;
            r.Value = (null != value ? value : "");
            return r;
        }
    }
```

### execute method

The execute method returns the **ActionResult** object from an IAction to Central after the code has been executed. The **ActionResult** object can return the results shown in the following table.

| Result | Description |
|---|---|
| exception (String, empty) | This result should be the stack trace for any exception that your operation encounters. This result is most often used inside of the try/catch block for the operation. It should be left blank if no exception is encountered. |
| returnCode (int, 0) | This result indicates which transition Central should follow after the operation has completed. It should be set to values that can be mapped to the responses in the ActionTemplate for this operation. See the *getActionTemplate method* section for more information. |
| Results | This is a map of the results that are sent back to central. These results are available on the **Result** tab in of Studio. They can be filtered or used as flow variables or flow results by other operations or flows. |

The ActionResult object extends the Map object, so any other results that you have defined in the ActionTemplate can also be returned to Central. The key field is the result name and the value field is a string value.

**Syntax**

Java

```
public class ActionResult extends Map {

    private String exception;
    private int returnCode;
    private String sessionId;

    public ActionResult() {
    }

    public ActionResult(String exception, MapEntry[] entries,
           int returnCode, String sessionId) {
      super(entries);
      this.exception = exception;
      this.returnCode = returnCode;
      this.sessionId = sessionId;
    }

    /**
     * Gets the exception value for this ActionResult.
     * @return exception
```

```
     */
    public String getException() {
        return exception;
    }

    /**
     * Sets the exception value for this ActionResult.
     * @param exception
     */
    public void setException(String exception) {
        this.exception = exception;
    }

    /**
     * Gets the returnCode value for this ActionResult.
     * @return returnCode
     */
    public int getReturnCode() {
        return returnCode;
    }

    /**
     * Sets the returnCode value for this ActionResult.
     * @param returnCode
     */
    public void setReturnCode(int returnCode) {
        this.returnCode = returnCode;
    }

    /**
     * Gets the sessionId value for this ActionResult.
     * @return sessionId
     */
    public String getSessionId() {
        return sessionId;
    }

    /**
     * Sets the sessionId value for this ActionResult.
     * @param sessionId
     */
    public void setSessionId(String sessionId) {
        this.sessionId = sessionId;
    }
}
}
```

.NET

```
public class ActionResult : Map
{
   public int code = (int)ResultCode.SUCCESS;
   public string sessionId = "new session";
   public string exception;

   public ActionResult () {}

   public int GetReturnCode() { return code; }
   public void SetReturnCode(int c) { code = c; }
   public string GetException() { return exception; }
```

```
    public void SetException(string ex) { exception = ex; }
    public string GetSessionId() { return sessionId; }
    public void SetSessionId(string sid) { sessionId = sid; }

    }
```

### Guidelines for creating IActions

When you create a Java or .NET IAction, you should adhere to the following guidelines:

- Create any static constants you need.
- Define your ActionTemplate. (This defines the inputs, outputs, and responses for the IAction. If an input, output, or response does not appear when you open the IAction in Studio, then you have made an error in the ActionTemplate.)
- Add an execute method, and make sure that it uses a try/catch block.
- Convert the inputs from the ActionTemplate to variables in your code and run the code.
- Make sure every result has a value defined, even if that value is an empty string. This is necessary in case an exception is thrown.)
- Set a value for the return code. The response (return code) for success is always 0 and for failure it is always -1.
- When possible, write a meaningful error message in case your operation fails.
- Only the inputs, responses, and results defined in the ActionTemplate are created automatically (see the *getActionTemplate method* section for more information).
- Make sure to handle system accounts properly. For Java IActions, use the StringUtils.resolveString method; for .NET IActions, use the Identities methods.

#### *Important points for creating Java IActions*

When creating a Java IAction, keep the following important points in mind:

- Check inputs to make sure that they have non-null values. (If you use the com.opsware.pas.content.commons.util.StringUtils.resolveString method, null inputs are automatically converted into empty strings). See the *Useful Java Commons Library class* section for more information. Optional inputs may be strings of length zero.
- Use the StringUtils.resolveString method, as it will handle system accounts for you. See the *Useful Java Commons Library class* section for more information.
- Any results you want the user to have access to must be included in the ActionTemplate.
- Do not use an instance variable in an IAction if it is unique to a given run of the IAction.
- Do not write to an instance variable.

#### *Important points for creating .NET IActions*

When creating a .NET IAction, keep the following important points in mind:

- Use the .NET Convert.ToString() method for handling all inputs except system accounts.
- Handle system accounts by using the Identity methods in Commons.dll. See the *Identities class* section for more information.
- Any results you want the user to have access to must be included in the ActionTemplate.
- Do not use an instance variable in the IAction if it is unique to a given run of the IAction.
- Do not write to an instance variable.

# Implementing Java IActions

Java IActions are Java classes that can be imported into and used by OO. This section explains:

- *The files you will need to implement your Java IActions*.
- *How to load your Java IActions into OO Studio*.
- *How to use third-party libraries for Java IActions*.
- *How to debug your Java IActions*.

This section also contains a complete *example of the code* needed to create a Java IAction.

## Required development files

The following files are required for developing Java IActions:

- JRAS-sdk-9.00.jar
- ContentCommons-9.00.jar

These files are included in the OO SDK home directory, in the lib\ folder.

## Loading your Java IActions into Studio

### To import your Java IActions into Studio

1. Create a .jar file with your IAction classes in it (you can create more than one IAction .jar file).
2. Stop the RSJRAS service (the Windows service that runs the RAS).
3. Copy your .jar file to the RAS\Java\Default\repository\ folder in the OO home directory.
4. Copy any additional libraries you may be using for the IActions to the following folder in the OO home directory:
   - RAS\Java\Default\repository\lib\<jarName>\ where jarName is the name of the .jar file without the .jar extension.

   For more information on IAction .jar files and third-party libraries, see *Using multiple versions of third-party libraries*.
5. Restart the RSJRAS service.
6. In Studio, create a folder for the IActions.
7. Select the folder and then, from the **File** menu, click **Create Operations from RAS**.
8. In the **RAS import** dialog box, select **RAS_Operator_Path** (assuming you put the .jar file into the default RAS) and then click **OK**. The IActions are imported.

**Note:** Code in one IAction .jar file cannot use code in another IAction jar file. Reimporting the IAction .jar file will generate different UUIDs for the operations.

## Using third-party libraries for Java IActions

You can have multiple IAction .jar files that each reference different versions of third-party libraries. The JRAS service looks for third-party Java libraries in the following folders in the in the OO home directory:

- RAS\Java\Default\repository\lib\<jarName>\ where jarName is the name of the .jar file without the .jar extension.
- RAS\Java\Default\repository\lib\
- RAS\Java\Default\webapp\WEB_INF\lib\

In addition to these paths, if an IAction .jar file has a main manifest attribute named **Custom-Libraries**, the value of this attribute will be processed as a comma-delimited list of additional folders to load. These folders are relative to the RAS\Java\Default\repository\lib\ folder. Libraries referenced

this way will be loaded between the RAS\Java\Default\repository\lib\<jarName>\ and RAS\Java\Default\repository\lib\ folders.

**Important**: The folder RAS\Java\Default\repository\lib\ is shared by all IAction .jar files. Do **NOT** put your own third-party libraries in this folder. Adding libraries to this folder may break out-of-the-box content.

The folder RAS\Java\Default\webapp\WEB_INF\lib\ is intended to be used by the RAS service itself. IAction libraries should **NOT** put their own third-party libraries into this folder. Adding libraries to this folder may break out-of-the-box content, or the RAS service.

The **PREFERRED** method is for each author of an IAction .jar file to add a folder named RAS\Java\Default\repository\lib\<jarName>\ where jarName is the name of the IAction .jar file, without the .jar file extension. Third party libraries should go into this folder.

For example, if you have an integration called TestApp, your IAction will be in a .jar file called TestApp.jar. Assume that you need integration library TestLib.jar and third party libraries TestThirdPartyLib1.jar and TestThirdPartyLib2.jar for TestApp to run.

The layout of the TestApp integration .jar files will be like the following:

- RAS\Java\Default\repository\lib\TestApp\TestLib.jar
- RAS\Java\Default\repository\lib\TestApp\TestThirdPartyLib1.jar
- RAS\Java\Default\repository\lib\TestApp\TestThirdPartyLib2.jar
- RAS\Java\Default\repository\TestApp.jar

If multiple IAction.jar files need to share a custom set of libraries, you can make one or more specifically named subfolders of the \RAS\Java\Default\repository\lib\ folder, and explicitly reference them through the **Custom-Libraries** manifest attribute. However, we do not recommend this unless there is no choice. Please use the preferred method mentioned above so your integration libraries will be in one place instead of all over the place.

For example, for TestApp IActions you also want to use different set of libraries, say AnotherSetLib1.jar, AnotherSetLib2.jar. You must put theses libraries in different directory. Create a directory called RAS\Java\Default\repository\lib\TestApp2\ and put AnotherSetLib1.jar and AnotherSetLib2.jar in it. The manifest file of TestApp.jar must include the Custom-Libraries to point to RAS\Java\Default\repository\lib\TestApp2\AnotherSetLib1.jar and AnotherSetLib2.jar.

Custom-Libraries: RAS\Java\Default\repository\lib\TestApp2\AnotherSetLib1.jar
RAS\Java\Default\repository\lib\TestApp2\AnotherSetLib2.jar

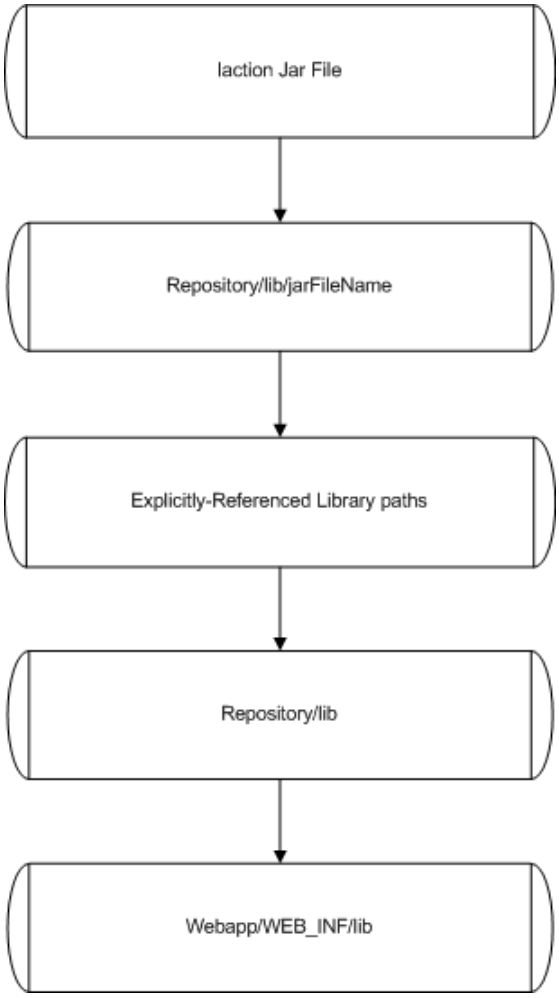


**Figure 5 - How the RAS resolves third-party Java libraries at run time**

## Debugging your Java IActions

### To enable remote RAS debugging for Java IActions

1. Stop the RSJRAS service.
2. Copy your .jar file to the RAS\Java\Default\repository\ folder in the OO home directory.
3. Copy any additional libraries you may be using for the IActions to the RAS\Java\Default\repository\lib\ folder in the OO home directory.
4. Open the RAS\Java\Default\webapp\conf\wrapper.conf file in the OO home directory using your preferred text editor.
5. Uncomment the following debug line:

   ```
   (#wrapper.java.additional.2=-Xdebug -Xnoagent -Djava.compiler=NONE -
   Xrunjdwp:transport=dt_socket,address=8070,server=y,suspend=y
   ```

   This suspends the RSJRAS service startup until a remote debugger is configured to use port 8070.
6. Find the debug line

   ```
   wrapper.java.additional.2
   ```

   and change it to

   ```
   wrapper.java.additional.3
   ```

   or

   ```
   wrapper.java.additional.n
   ```

where `n` is the last number you used plus one.

7. Restart the RSJRAS service.
8. Configure your remote debugger to use the port specified in the wrapper.conf file. Port 8070 is the default, but you can change it to any unused port.
9. Set a breakpoint in your Java source code at which you would like to stop and connect to the remote debug session listening on the port specified in step 8.
10. Execute a flow that uses your operation.
11. Debug the IAction code.

### To disable remote RAS debugging for Java IActions

1. Stop the RSJRAS service.
2. Open the RAS\Java\Default\webapp\conf\wrapper.conf file in the OO home directory.
3. Comment out the following debug line:
   `(#wrapper.java.additional.2=-Xdebug -Xnoagent -Djava.compiler=NONE -Xrunjdwp:transport=dt_socket,address=8070,server=y,suspend=y)`
4. Find the debug line
   `wrapper.java.additional.2`
   and change it to
   `wrapper.java.additional.3`
   or
   `wrapper.java.additional.n`
   where `n` is the last number you used plus one.
5. Start the RSJRAS service.

## Java IAction code example

The following is an example of a Java IAction that reads a file and returns the contents.

**Java IAction code example**

```
public class ReadFile implements IAction {

   private static final String RETURNRESULT = "returnResult";
   public static final int PASSED = 0;
   public static final int FAILED = 1;

   @Override
   public ActionTemplate getActionTemplate() {
      ActionTemplate actionTemplate = new ActionTemplate();

      actionTemplate.setDescription(ReadFile.DESCRIPTION);

      RASBinding arg1 = RASBindingFactory.createPromptBinding(
         "Source File:", true);

      Map parameters = new Map();
      parameters.add("source", arg1);
      actionTemplate.setParameters(parameters);

      Map resultFields = new Map();
      resultFields.add("fileContents", "");
      resultFields.add(RETURNRESULT, "");
      actionTemplate.setResultFields(resultFields);
```

```
      Map responses = new Map();
      responses.add("success", String.valueOf(PASSED));
      responses.add("failure", String.valueOf(FAILED));

      actionTemplate.setResponses(responses);

      return actionTemplate;
   }

   @Override
   public ActionResult execute(ISessionContext session,
   ActionRequest request, IActionRegistry registry)
   throws Exception {
      ActionResult result = new ActionResult();

      String separator = (System.getProperty("line.separator") != null)
                        ? System.getProperty("line.separator") : "\n" ;
      String line = null;
      File file = null;
      FileReader fReader = null;
      BufferedReader bReader = null;

      try {
         file = new File ActionRequestUtils.resolveStringParam(
            request, "source"));

         StringBuilder fileContents = new StringBuilder();
         fReader = new FileReader(file);
         bReader = new BufferedReader(fReader);

         while ((line = bReader.readLine()) != null) {
            fileContents.append(line);
            fileContents.append(separator);
         }

         result.add("fileContents", fileContents.toString());
         result.add(RETURNRESULT, "successfully read file");
         result.setReturnCode(PASSED);

      } catch (Exception e) {
         result.setReturnCode(FAILED);
         result.setException(StringUtils.toString(e));
         result.add(RETURNRESULT, e.getMessage());

      } finally {
         if (bReader != null)
            bReader.close();
         if (fReader != null)
            fReader.close();
      }

      return result;
   }

   private static String DESCRIPTION = ""
      +"<pre>Reads the contents of a file and returns it\n"
      +"Inputs:\n"
      +"source - path to file to read\n"
      +"\n"
      +"Responses:\n"
```

```
        +"success - successfully read file"
        +"failure - failed to read the file\n"
        +"\n"
        +"Extra Results:\n"
        +"fileContents - the contents of the file\n\n</pre>";
}
```

## Implementing .NET IActions

.NET IActions are .NET assemblies that can be imported into and used by OO. This section explains:

- *The files you will need to implement your .NET IActions.*
- *How to load your .NET IActions into OO Studio.*
- *How to debug your .NET IActions.*

This section also contains a complete *example of the code* needed to create a .NET IAction.

### Required development files

The following files are required for developing .NET IActions:

- IAction.dll
- RCAgentLib.dll
- Commons.dll

These files are included in the OO home directory, in the RAS\Java\Default\repository\ folder.

### Loading your .NET IActions into Studio

#### To import your .NET IActions into Studio

1. Create a .dll file with your IAction classes in it.
2. Stop the RSJRAS service (the Windows service that runs the RAS).
3. Copy your .dll file to the RAS\Java\Default\repository\ folder in the OO home directory.
4. Copy any additional .dll libraries you may be using for the IActions to the same folder.
5. Restart the RSJRAS service.
6. In Studio, create a folder for the IActions.
7. Select the folder and then, from the **File** menu, click **Create Operations from RAS**.
8. In the **RAS import** dialog box, select **RAS_Operator_Path** (assuming that you put the .dll file into the default RAS) and then click **OK**. The IActions are imported.

### Debugging your .NET IActions

#### To enable remote RAS debugging for .NET IActions

1. Stop the RSJRAS service.
2. Copy your .dll and .pdb (.NET debug files) files to the RAS\Java\Default\repository\ folder in the OO home directory.
3. Copy any additional libraries you may be using for the IActions to the same folder.
4. Restart the RSJRAS service.
5. Configure your debugger to use the java.exe process that hosts the RSJRAS service.
6. Set a breakpoint in your .NET source code at which you would like to stop.
7. Run a flow that uses your operation.

8. Debug the IAction code.

### To disable remote RAS debugging for .NET IActions

1. Disconnect your debugger from the java.exe process.
2. Stop the RSJRAS service.
3. Remove the .pdb files.
4. Restart the RSJRAS service.

## .NET IAction code example

The following is an example of a .NET IAction that reads a file and returns the contents.

**.NET IAction code example**

```
using System;
using System.IO;
using System.Text;
using System.Collections;
using System.Diagnostics;
using System.Globalization;
using com.iconclude.agent;
using System.Text.RegularExpressions;

using DiskServices;
using dotNET_Commons;

namespace com.hp.oo.content.sdk
{
    public class ReadFile : IAction
    {
        public ActionResult Execute(ActionRequest request,
         ISession session, IActionRegistry registry)
        {
            ActionResult result = new ActionResult();
            StreamReader sReader = null;
            String line = null;

            try
            {
                string strSource = Convert.ToString(
                  request.parameters["source"]);

                StringBuilder fileContents = new StringBuilder();

                Identities.ChangeUserContext(request);

                sReader = File.OpenText(strSource);

                while ((line = sReader.ReadLine()) != null)
                    fileContents.AppendLine(line);

            }
            catch (Exception e)
            {
                ret.SetReturnCode(ReturnCodes.FAILED, e.Message);
                ret.SetException(e.ToString());
            }
            finally
            {
```

```
                Identities.UnchangeUserContext(req);
            }

            return ret.GetActionResult();
        }

      public ActionTemplate GetActionTemplate()
        {
            ActionTemplateEx template = new ActionTemplateEx();

            template.SetDescription("WriteToFile");

            RASBinding arg1 = RASBindingFactory.createPromptBinding(
               "FileName:");
            arg1.AssignFrom(true);
            arg1.AssignTo(true);

            RASBinding arg2 = RASBindingFactory.createPromptBinding(
               "Text To Write:");
            arg2.AssignFrom(true);
            arg2.AssignTo(true);

            RASBinding arg3 = RASBindingFactory.createPromptBinding(
               "Alternate Credentials - UserName:", false, false);
            arg3.AssignFrom(true);
            arg3.AssignTo(true);

            RASBinding arg4 = RASBindingFactory.createPromptBinding(
               "Alternate Credentials - Password:", false, true);
            arg4.AssignFrom(true);
            arg4.AssignTo(true);

            template.AddParameter("File", arg1);
            template.AddParameter("Contents", arg2);
            template.AddParameter("user", arg3);
            template.AddParameter("password", arg4);

            template.AddResponse("success", (int)ReturnCodes.PASSED);
            template.AddResponse("failure", (int)ReturnCodes.FAILED);

            return template.GetActionTemplate();
        }
    }
}
```

## Useful Java Commons Library class

The following class is available for general use and may be helpful as you develop content. It is located in the ContentCommons.jar file in the OO home directory, in the RAS\Java\Default\repository\lib\ folder. This class will be maintained, for backward compatibility.

### com.opsware.pas.content.commons.util StringUtils class

This is a helper class in Java for handling system account inputs and inputs that are null or missing.

**Properties**

| Name | Description |
|---|---|
| none | |

**Constructors**

| Name | Description |
|---|---|
| none | All methods are static. |

**Methods (all are static)**

| Name | Description |
|---|---|
| isNull(String) | Boolean specifying whether the value passed is null. |
| resolveString (ActionRequest, String) | Resolves the value of `String` from the input map in ActionRequest. |

## Useful .NET Commons Library classes

The following classes are available for general use and may be helpful as you develop content. They are located in the Commons.dll file in the OO home directory, in the RAS\Java\Default\repository\ folder. These classes will be maintained to allow for backward compatibility.

### Identities class

The Identities class allows you to deal with user permissions (system accounts) and to perform user impersonation for some operations.

There are two impersonation styles:

- The first (and the one that is attempted first) is an inter-process communication (IPC) connection to the remote machine.
- If this fails or if the authentication is performed against the local machine instead of the RAS, then local thread impersonation is attempted.

In most cases the Identities class uses an ActionRequest method and handles system accounts.

**Properties**

| Name | Description |
|---|---|
| DestinationHost | (Internal) The machine host value when dealing with copying files. |
| File | (Internal) The file path for impersonation purposes. |
| Host | The remote host to connect to. |
| Password | The password to use for the connection. |
| SourceHost | (Internal) The machine host for dealing with copying files. |
| UserName | The username to use for the connection. |

**Constructors**

| Name | Description |
|---|---|
| Identities() | Default. |

**Methods**

| Name | Description |
|---|---|
| ChangeUserContext(ActionRequest) | Assuming that ActionRequest has the proper inputs defined, this method will impersonate the user. If you use this method, you must use the `UnchangeUserContext(ActionRequest)` method when the impersonation is finished. |
| ChangeUserContext(string host, string user, string password) | Attempts to make the connection and impersonate the user specified. If you use this method, you must use the `UnchangeUserContext(string)` when the impersonation is finished. |
| GetUserPass(ActionRequest, string username key, string password key) | This is a useful method if you have inputs that are system accounts but aren't valid input names. ActionRequest must contain the inputs `username key` and `password key` which are passed in to tell which input to use to pull the actual username and password values. These values can then be read back using the UserName and Password properties. |
| GetUserPass(ActionRequest) | This method is the same as `GetUserPass(ActionRequest, string username key, string password key)`, except that the `username key` and `password key` inputs are passed automatically |
| UnchangeUserContext(ActionRequest) | If you used the `ChangeUserContext(ActionRequest)` method, use this method with the ActionRequest used during the call to reverse the impersonation. |
| UnchangeUserContext(string hostname) | If you used the `ChangeUserContext(string)` method, use this method with the hostname used during the call to reverse the impersonation. |

If you use the ChangeUserContext(ActionRequest) method, the names of the inputs mapped in the ActionRequest have to conform to the inputs shown in the following table (the inputs are case sensitive).

| Input Type | Valid Input Names |
|---|---|
| Host | • host<br>• hostname |
| Username | • username<br>• user<br>• User<br>• F5Username<br>• altuser |

| Input Type | Valid Input Names |
|---|---|
| Password | • password<br>• Password<br>• altpass<br>• pass<br>• F5Password<br>• Pass |

If the inputs do not conform to these specifications, you must use the ChangeUserContext(string host, string user, string password) impersonation method.

**Important:** If you use any of the impersonation functions, make sure you have the unimpersonation area wrapped in a finally block. That way you can always roll back your impersonation.

**Remarks**

The most flexible and secure way to communicate between Windows servers is by making authenticated IPC connections. This can be done on the command line by using `net use \\machinename\ipc$` and supplying the necessary credentials. Basically, this is how the Identities class does it as well. However, only one IPC connection can exist between any two machines. Therefore, only one IPC connection can exist between the RAS and a given server at any given time. Because of this the RAS is designed as follows:

1. The RAS receives a request to impersonate a user. To check the request to see whether a remote machine is requested, the RAS examines the hostname, the fully qualified domain name (FQDN), and all of the IP addresses registered on the machine. If the requested machine isn't remote, the RAS attempts thread elevation. If the requested machine is remote, the RAS continues to the next step.
2. The RAS searches on the hostname to see whether an IPC connection with the same username and password is currently mapped. If it is, the RAS adds a flag to indicate that another operation is using the same connection, and the impersonation class returns.
3. If a connection doesn't exist:
   - A new request to map the remote machine's IPC share is sent using the standard Windows API.
   - The RAS adds the first flag for this machine.
   - The impersonation class returns.
4. Once the operation has completed and called the unimpersonation method in this class, the RAS checks to see whether it used a thread impersonation or an IPC connection.
   - If the RAS used a thread impersonation, it restores the thread to its old credential set.
   - If the RAS used an IPC connection, the connection use count is decremented. If the use count is now at zero, the IPC connection with the remote machine is closed and the class returns. If the use count is not zero, the class just returns.

## Password class

This class generates random passwords.

**Properties**

| Name | Description |
|---|---|
| None | |

**Constructors**

| Name | Description |
|---|---|
| None | |

**Methods**

| Name | Description |
|---|---|
| static GenerateRandom(int length, int numberOfNonAlphaNumericCharacters) | A static method to generate random passwords. |

# Finding and running flows from outside Central

In most cases, you use Central to run the flows you create in Studio. There may be situations however, when you want to find or run flows without using Central. For instance, you may want to run a flow from an external application, such as Microsoft System Center Operations Manager, or from a script or batch file.

## About finding and running flows from outside Central

Ways in which you can find and run flows from outside Central include:

- *Creating a URL in Central* that can run a **Guided** or **Run all** flow from a Web browser.
- From a *command line* or from an application that can use a command line.
- Using *tools that access the REST (representational state transfer) service* so you can run flows using the Internet.
- Using the *WSCentralService SOAP API* to access Central features programmatically.

**Important:** Although you do not use it for managing flows externally, Central must be running when you do so.

## Running flows with URLs created in Central

In Central, you can obtain a valid URL and use it to run a **Guided** or **Run all** flow from a Web browser.

### To use a URL created in Central

1. In Central, click the **Flow Library** tab, navigate to the flow, and then click the flow name to open the preview of the flow.
2. Under **Execution Links** in the left pane, select and copy the URL in the box below the desired type of run—**Guided Run** or **Run All**.

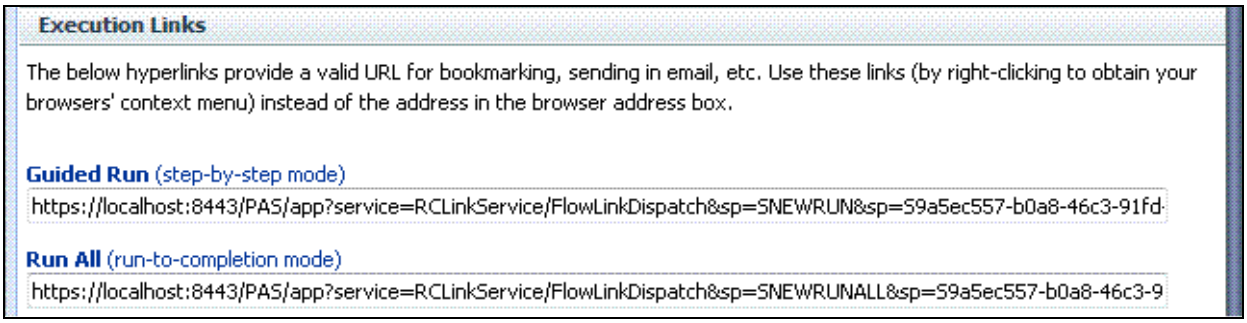


**Figure 6 - Execution Links**

3. Open a new browser window and paste the URL in the address box.

   If the flow has any required inputs, modify the URL by adding input parameters and values for the inputs. See *Specifying the inputs for a flow in a URL* for more information.

**Note:** You can also paste the URL in a document or e-mail, but if you paste it into an e-mail you cannot pass input parameters in the URL.

# Running a flow from a command line

You can run a flow from a command line using a correctly formatted URL.

## Guidelines for running a flow from a command line

When running a flow from a command line:

- The flow must be self-contained—it cannot contain user prompts.
- You can pass input parameters and values to the flow in the URL.
- If an input requires a flow variable, the variable does not have to be defined when you create the flow in Studio. You can create and pass the flow variable to the input by using an input parameter in the URL.

## Creating a URL for running a flow

The format for a URL that runs a flow from a command line is as follows:
`https://<hostname>:<port>/<path>/<flow>`

The main points to be aware of when creating a URL are:

- There are two ways to identify the flow—by its name or by its universally unique ID (UUID).
- The initial inputs (flow input values) that are required for the flow to run must be included in the URL.
- You can modify the URL to provide a result from the flow asynchronously—without waiting for the flow to complete its run.

### *Identifying the flow in the URL*

In a URL that runs a flow, either of the following identifies the flow:

- The name of the flow.
- The flow's Universally Unique Identifier (UUID).

The following examples illustrate these two methods.

**Example 1**

This URL identifies the flow by its name, `TestFlow`.
`https://localhost:8443/PAS/services/rest/run/Library/MyFolder/TestFlow`

**Example 2**

This URL identifies the flow by its UUID (`503c2500-7aae-11dd-a3b5-0002a5d5c51b`).
`https://localhost:8443/PAS/services/rest/run/503c2500-7aae-11dd-a3b5-0002a5d5c51b`

### *Specifying the inputs for a flow in a URL*

You can specify the inputs for a flow by using input parameters (also known as "init params") in the URL. This allows you to run the flow without any user interaction.

Any init params are separated from the flow identifier by a question mark (?). Each init param takes the form `name=value`. If you use more than one init param, separate the init params with an ampersand (&).

**Example**

The URL in this example runs the flow `MyFlow`, passing the input parameter `name0` with a value of `val0` and the init param `input1` with a value of `yes`.
`https://localhost:8443/PAS/services/rest/run/Library/MyFolder/MyFlow?name0=val0?input1=yes`

When naming init params:

- Do not name init params “service” or “sp”, as these are reserved names.
- If a flow uses one of the reserved names for an input, protect your flows from errors that can result from using these reserved names, by defining a prefix for all init param names used in the URL. As long as a required init param prefix is specified, you must use it for all the init params for flows started by means of a URL in Central, including those that do not use the reserved names.

### To define a prefix for init param names

1. Log on to Central with an account that has OO administrative rights.
2. On the **Administration** tab, click the **System Configuration** tab.
3. In the **General Settings** area, in the **Value** box of the **Prefix for init params for flow invocation through URLs using the GUI** row, type the prefix that you want to use for the input parameters in a URL.
4. Click **Save General Settings**.
5. Restart Central.

When defining a prefix for input parameters, avoid using the types of characters shown in the following table.

| Types of characters to avoid using | Examples |
|---|---|
| Characters that are reserved in URLs. | ; / ? : @ = & |
| Characters that can be misunderstood in URLs. | { } \| \ ^ ~ [ ] ` |

If the flow you are starting has a multi-instance step or for any other reason has an input whose value is a list of values, use the separator character defined by the flow’s author in Studio (by default, a comma) to separate the values for the input that has multiple values. You should also do this if your flow has an input that is a list of values.

The following examples show how to use a prefix for initial inputs in a URL and how to specify a list of values for a single input.

**Example**

In the following example:

- The prefix for the init params has been defined as `_xx.`
- The flow specified in the URL has a multi-instance step with the separator character defined as an ampersand (`&`).
- The init param `_xx_input1` has two values—the IP addresses `10.0.0.100` and `10.0.0.101` (with the default value-list separator character of comma [,]).

`https://localhost:8443/PAS/services/rest/run/Library/MyFolder/TestFlow?_xx_input1=10.0.0.100,10.0.0.101&_xx_input2=8443&_xx_input2=8443`

**Notes:**

- You only have to specify values for inputs that get their values from user prompts or that have not been assigned a value (or a way to get a value).
- You do not have to specify a value for an input that has a specific value assigned to it (or a set of values, as in multi-instance steps or steps that get their values from an Iterator operation).

- You do not have to specify a value for an input that gets its value from a system account or from the logged-in user's credentials.

### *Running flows asynchronously using a URL*

You can obtain a result from a flow without waiting for it to finish by running the flow asynchronously. This can be very useful if you run flows with multi-instance steps or multiple input values.

**To run a flow asynchronously**

- In the URL that runs the flow, replace:

  `/run/`

  with

  `/run_async/`

**Example**

The following URL runs the flow `Connectivity Test` asynchronously using the `run_async` parameter. This flow has a multi-instance step with multiple values for the `target` input.

```
https://localhost:8443/PAS/services/http/run_async/Library/MyFolder/Connectivity Test?&host=localhost&target=55.55.0.47,55.55.0.49
```

## Finding and running flows with tools that access the REST service

REST (representational state transfer) is an architectural style that uses existing Internet technology and protocols such as HTTP and XML. This section describes two command-line tools that access the REST service, allowing you to find and run flows using the Internet:

- The Wget tool allows you to send a username and password in the command line, but does not provide feedback as to whether your call worked correctly.
- With RSFlowInvoke.exe or JRSFlowInvoke.jar, you can send encrypted passwords over the Internet. In addition, RSFlowInvoke and JRSFlowInvoke return XML or HTML feedback that verifies whether your call worked.

**Important:** Wget, RSFlowInvoke.exe, and JRSFlowInvoke.jar can take command-line parameters or URLS to specify the flows you want to manage with them. If you use a URL, you must enclose it with quotation marks.

### Running flows using Wget

Wget is a command-line tool that you can use to download and run flows from the Internet. You can download Wget from the GNU Wget Web page.

The basic syntax of Wget is:

```
wget {<options>} {<URL>}
```

Wget downloads and runs flows specified in the URL contained on the command line. It can use the HTTP, HTTPS, and FTP protocols. The Wget options are explained in the *GNU Wget Manual*.

**Important:** Enclose the URL on the command line with quotation marks.

The following examples show how to download and run flows using a URL in a Wget command line.

**Example 1**

The following example downloads the flows in the `MyFolder` folder.

- The Wget `-O` option specifies that all error messages should be logged to the default log file wget.log.

- The Wget `http-user=user` and `http-password=password` options specify a username of `rsadmin` and a password of `iconclude` for the HTTP server.

```
wget --http-user=rsadmin --http-password=iconclude
"https://localhost:8443/PAS/services/rest/list/MyFolder/"
```

### Example 2

This example uses the Wget `no-check-certificate` option to skip Secure Sockets Layer (SSL) checking.

```
wget --no-check-certificate --http-user=rsadmin --http-passwd=iconclude
"https://localhost:8443/PAS/services/rest/list/MyFolder/"
```

### Example 3

This example runs the flow specified in the URL and passes the input variable `name0` with a value of `val0`.

```
wget --http-user=rsadmin --http-passwd=iconclude
"https://localhost:8443/PAS/services/rest/run/Library/MyFolder/MyFlow?name0=val0"
```

The next examples work with an XML file that has the following basic layout.

XML file layout

```
<?xml version="1.0"?>
<run>
   <request>
      <arg name="name1">value1</arg>
      <arg name="name2">value2</arg>

   </request>
</run>
```

### Example 4

This example uses POST as the method to run an XML-encoded flow from the file C:\run-config.xml.

```
wget --http-user=rsadmin --http-passwd=iconclude --post-file="C:\run-config.xml"
--header “Content-Type: text/xml”
"https://localhost:8443/PAS/services/rest/run/Library/MyFolder/MyFlow"
```

### Example 5

This example runs an XML-encoded flow from a command line using the Wget `post-data=string` option.

```
wget --http-user=rsadmin --http-passwd=iconclude --post-data="<?xml version=\"1.0\"
?><run><request><arg name=\"name0\">value0</arg><arg name=\"name1\">value1</arg><arg
name=\"name2\">value2</arg></request></run>" --header "Content-Type: text/xml"
"https://localhost:8443/PAS/services/rest/run/Library/MyFolder/MyFlow"
```

The following shows the XML format that is returned.

Example returned XML format

```
<?xml version="1.0" encoding="UTF-8"?>
<runResponse>
    <runReturn>
   <item>
       <name>runId</name>
```

```
        <value>23</value>
    </item>
    <item>
        <name>runReportUrl</name>

<value>https://localhost:8443/PAS/app?service=RCLinkService/ReportLinkDispatch
&amp;sp=SINDIVIDUAL_REPAIR_LEVEL&amp;sp=Sc2bcb72f-6d6b-4a2d-a678-
de21a1feac81&amp;sp=l0&amp;sp=l23</value>
    </item>
    <item>
        <name>runStartTime</name>
        <value>09/17/08 13:00:54</value>
    </item>
    <item>
        <name>runEndTime</name>
        <value>09/17/08 13:00:54</value>
    </item>
    <item>
        <name>runHistoryId</name>
        <value>23</value>
    </item>
    <item>
        <name>flowResponse</name>
        <value>success</value>
    </item>
    <item>
        <name>flowResult</name>
<value>{Field 1=value0;Field
2=value1;FailureMessage=;TimedOut=;Result=;}</value>
    </item>
    <item>
        <name>flowReturnCode</name>
        <value>Resolved</value>
    </item>
     </runReturn>
```

## Finding and running flows using RSFlowInvoke or JRSFlowInvoke

RSFlowInvoke.exe, or the Java version, JRSFlowInvoke.jar, is a command-line utility that you can use to list, run, and search for flows outside of Central.

You can use RSFlowInvoke or JRSFlowInvoke:

- From a command-line window or an application that can use a command line.
- As part of a script or batch file.

You can run RSFlowInvoke or JRSFlowInvoke on any machine from which you can log on to Central (using HTTPS to the default port 8443). This makes RSFlowInvoke and JRSFlowInvoke useful for starting a flow from an external system or application that can use a command—for example, a monitoring program such as Microsoft System Center Operations Manager.

RSFlowInvoke and JRSFlowInvoke are available in the OO SDK home directory, in the tools\ folder. They are also available in the HP OO home directory, in the Studio\tools\ folder. Run RSFlowInvoke and JRSFlowInvoke from this subfolder or from a path that includes this subfolder.

The basic syntax of RSFlowInvoke.exe is:

```
RSFlowInvoke.exe {–host <hostname>:<port number> –flow <flow name> | <URL>} [-inputs <input name>=<value>] [-u <user>] -p <password>|-ep <encrypted password>] [-async] [-rc <number of retries>] [-rw <number of seconds>] [-t <timeout>] [-v]
```

The basic syntax of JRSFlowInvoke.jar is:

```
java –jar JRSFlowInvoke.jar {–host <hostname>:<port> –flow <flow name>| <URL>} [-inputs <input name>=<value>] [-u <user>] [-p <password>|-ep <encrypted password>] [-async] [-rc <number of retries>] [-rw <number of seconds>]
```

You can reference the flow in RSFlowInvoke or JRSFlowInvoke using:

- The `–host` and `–flow` options.
- A URL. The URL must have the correct format for managing a flow and be enclosed in quotation marks. For information on building a correctly-formatted URL, see *Creating a URL for managing a flow*.

### Option Syntax

Following are the syntax and descriptions of the RSFlowInvoke and JRSFlowInvoke options.

`-async`

Specifies that the flow runs asynchronously—in other words, that it returns a result before completing its run.

`-ep <encrypted password>`

Specifies the encrypted password for the host. If you use the `–p` option to specify a nonencrypted password, do not use this option. For information about creating an encrypted password from within RSFlowInvoke, see *Creating an encrypted password*.

`-flow <flow name>`

Specifies the flow name or UUID.

`-host <hostname>:<port number>`

Specifies the hostname and port number, separated by a colon.

`-inputs <input name>=<value>`

Specifies the inputs for the flow, using the format `name=value&name2=value2`. If any of the inputs or their values contain a space, then the `name=value&name2=value2` argument should be wrapped in quotes (`"name=value&name2=values"`).

**Note:** To be used with RSFlowInvoke or JRSFlowInvoke, an input must have a flow variable assigned to it in the **Assign to Variable** drop-down box on the **Input Summary** tab in Studio. This option is used when you specify host and flow options.

`-p <password>`

Specifies the password for the host. If you use the `–ep` option to specify an encrypted password, do not use this option.

`-rc <number of retries>`

Specifies the number of times to retry a flow that fails. The default is 0; the maximum is 30.

`-rw <number of seconds>`

Specifies the number of seconds to wait between retries. The default is 5 seconds.

`-t <timeout>`

Specifies the timeout, in seconds. The default value is 100 seconds.

`-u <username>`

Specifies the username for the host.

`-v`

`-verbose`

Specifies that all output is to be written to the screen.

## *Using RSFlowInvoke or JRSFlowInvoke from a command line*

The examples in this section demonstrate how to list and run flows using RSFlowInvoke or JRSFlowInvoke from a command line or from any application that can take input from a command line.

**Example 1**

The following example lists the flows specified in the URL.

```
java –jar JRSFlowInvoke.jar “https://localhost:8443/PAS/services/rest/list/MyFolder/”
-u rsadmin -ep BKmIQF6o0dItQkcUYNEeGw==
```

**Example 2**

This example runs the flow with input names and values specified in the URL.

```
java –jar JRSFlowInvoke.jar
“https://localhost:8443/PAS/services/rest/run/Library/MyFolder/TestFlow?inputName1=in
putValue1&inputName2=inputValue2” -u admin -ep BKmIQF6o0dItQkcUYNEeGw==
```

**Example 3**

This example lists the flows specified using the `–host` and `–flow` options.

```
java –jar JRSFlowInvoke.jar –host localhost:8443 –flow
/PAS/services/rest/list/MyFolder/ -u admin -ep BKmIQF6o0dItQkcUYNEeGw==
```

**Example 4**

This example runs the flow with input names and values specified using the `–host` and `–flow` options.

```
java –jar JRSFlowInvoke.jar –host localhost:8443 –flow
/PAS/services/rest/run/Library/MyFolder/TestFlow -u rsadmin -ep
BKmIQF6o0dItQkcUYNEeGw== -inputs “inputName1=inputValue1&inputName2=inputValue2”
```

## *Using RSFlowInvoke or JRSFlowInvoke in a script or batch file*

In a script or batch file, the syntax for using RSFlowInvoke.exe is the same as it is for using it in a command line. The syntax for JRSFlowInvoke.jar is slightly different.

- The syntax of RSFlowInvoke.exe in a script or batch file is:
  ```
  RSFlowInvoke.exe {–host <hostname>:<port number> –flow <flow name>|<URL>} [-inputs
  <input name>=<value>] [-u <user>] -p <password>|-ep <encrypted password>] [-rc
  <number of retries>] [-rw <number of seconds>] [-t <timeout>] [-v]
  ```
- The syntax of JRSFlowInvoke.jar in a script or batch file is:
  ```
  java -jar JRSFlowInvoke.jar {–host <hostname>:<port> –flow <flow name>|<URL>} [-
  inputs <input name>=<value>] [-u <user>] [-p <password>|-ep <encrypted password>]
  [-rc <number of retries>] [-rw <number of seconds>]
  ```

You can type `java –jar JRSFlowInvoke.jar` to find the usage of the tool.

### *Searching for a flow using JRSFlowInvoke.jar*

You can use JRSFlowInvoke.jar to search for a flow outside of Central. The search uses the Apache Lucene search syntax. For more information on this syntax, see the Apache Software Foundation Query Parser Syntax Web page.

Use the following syntax which includes a properly formatted query string:

```
java -jar JRSFlowInvoke.jar “https://{<host>:<port>}/PAS/services/http/search?
queryString = {<sequence_of_term_expressions>}” [-u <user>] [-p <password>]
```

**Option Syntax**

`<host>`

Specifies the Central server on which the search is to be performed.

`<port>`

Specifies the port number on the Central server which Central uses to communicate with the client.

`<sequence_of_term_expressions>`

A search string, which can be one of the following:

- A single term expression of the form:
  `<fieldname>:<term>`
- A sequence of terms of the form:
  `<fieldname>:<term> + (<operator> + <sequence_of_term_expressions>)`

`<fieldname>`

Specifies one of the fields shown in the following table (these are not case-sensitive).

| Field | Description |
|---|---|
| `Category` | The category that has been assigned to the flow. |
| `Description` | The flow’s description. |
| `Domain` | A domain term that has been associated with the flow. |
| `ID` | The flow’s UUID. |
| `Input` | An input to an operation used in the flow. |
| `Name` | The flow’s name. |
| `Type` | The type of an operation used in the flow. The terms that you can match in this field and the operation types that they represent include:<br>• cmd – Command<br>• flow – An operation that is a flow<br>• http – Http (also known as shell)<br>• other - Scriptlet<br>• script.perl – Perl script<br>• ssh – SSH (Secure Shell)<br>• telnet - Telnet<br>• lock = Acquire Lock<br>• unlock = Release Lock |
| `Stepdescription` | The description of one of the flow’s steps. |
| `Stepname` | The name of one of the flow’s steps. |

<term>

Specifies the particular value of the field that may find the desired flow.

<operator>

Is one of the operators supported in the Apache Lucene search syntax:

AND, “+”, OR, NOT, and “-“.

-u <user>

Specifies a user account that has the permissions to view and start a flow.

-p <password>

Specifies the password for the user account.

-ep <encrypted password>

Specifies the encrypted password for the user account. See *Creating an encrypted password* for more information.

**Example**

This example searches for flows that have the field Name with a value of John’s flow.

```
java -jar JRSFlowInvoke.jar “https://{localhost:8443}/PAS/services/http/search?
queryString = Name:John’s Flow” -u admin -ep BKmIQF6o0dItQkcUYNEeGw==
```

### *RSFlowInvoke and JRSFlowInvoke results*

RSFlowInvoke and JRSFlowInvoke use the return codes shown in the following table to tell you what happened when they were run.

| Return Code | Description |
|---|---|
| 0 | The flow was run. This code is not related to the flow's response. |
| 1 | Central responded with HTTP code 503. This usually means that Central lacked the resources needed to run the flow. |
| 2 | An unknown internal server error occurred in Central. |
| 3 | RSFlowInvoke was unable to authenticate against Central. |
| 4 | The specified URL or flow was not found. |
| 5 | A socket timeout occurred. A *socket* is a software object that connects an application to a network protocol. |
| 6 | An unknown socket (communication) error occurred. |
| 7 | An unknown error occurred. |

### *Registering RSFlowInvoke with the Global Assembly Cache*

The Global Assembly Cache (GAC) is a store on a local .NET machine for assemblies of .NET code. If you register RSFlowInvoke.exe with GAC, you can start a flow from within a .NET application, using any .NET-compatible language, such as C#.

**To register or unregister RSFlowInvoke in GAC**

1. On a .NET machine, open a command-line window and type the following command:

   ```
   gacutil.exe [/i|/u] RSFlowInvoke.exe
   ```

   **Option syntax**

   /i

   Registers RSFlowInvoke.exe with GAC.

`/u`

Unregisters RSFlowInvoke.exe with GAC.

2. Once RSFlowInvoke.exe is registered with GAC, type the following to view the assembly (compiled code) information:

   `RSFlowInvoke.exe –s`

The following is an example of the output of the `RSFlowInvoke.exe –s` command:

Example output from RSFlowInvoke.exe –s command

```
Assembly Name:
  RSFlowInvoke, Version=1.0.3098.16154, Culture=neutral, PublicKeyToken=4c0918
1d83b84dbc

Fully Qualified Type Name:
  RepairSystem.RSFlowInvoke

Method Name:
  ExecuteHeadlessFlow(
      System.String url,
      System.String username,
      System.String password,
      System.String authType,
      System.Boolean encryptedPassword)
```

### *Creating an encrypted password*

RSFlowInvoke and JRSFlowInvoke allow you to send encrypted passwords over the Internet.

**To create an encrypted password for use with RSFlowInvoke or JRSFlowInvoke**

1. In a command window, type and run one of the following commands:

   `RSFlowInvoke.exe –cp`

   `java –jar JRSFlowInvoke.jar –cp`

2. At the **Enter password** prompt, type the password.
3. At the **Enter password again** prompt, retype the password.

RSFlowInvoke or JRSFlowInvoke encrypts the password. When you run RSFlowInvoke or JRSFlowInvoke with the encrypted password, use the `-ep` option instead of the usual `–p` option for the password.

## Finding and running flows using the WSCentralService SOAP API

The WSCentralService service basically does two things—search and execution. The WSCentralService service provides a SOAP API with which you can:

- Search for a flow by querying the flow library using criteria provided by a query string. Query strings are based on an Apache Lucene indexed search. For more information on the Lucene search, see the Apache Lucene Web page.
- Control flow execution—this includes running, pausing, resuming, and canceling a flow, and viewing the status of a flow run.

The WSCentralService SOAP API Java and .NET classes and interfaces are located in the OO SDK home directory, in the lib\ folder. The certificates, keystore, WSDL, and sample code are located in the OO SDK home directory, in the samples\ folder.

## Accessing the WSCentralService WSDL

The WSCentralService WSDL describes the WSCentralService service and the operations it can perform. You can access the WSCentralService WSDL at the following URL:

`https://central_server:8443/PAS/services/WSCentralService?wsdl`

where *`central_server`* is the name of the server running Central. Samples included in the OO SDK home directory, in the samples\client\ subfolder, demonstrate using the service with Java and .NET.

## Using the API documentation

The WSCentralService SOAP API reference documentation is included in the OO SDK home directory, in the docs\javadocs\WSCentralService\ subfolder. The reference documentation covers the WSCentralService API and objects plus the wrapper class WSCentralServiceSession that implements the service public interface.

### *How WSCentralService manages security and authentication*

WSCentralService supports HTTP basic authentication. The client must provide the username and password of a user account that can run and manage flows started outside of Central. The message context associated with a client session must include the username and password, or the session will fail and a security violation will be issued to the client. For examples, see the test client code Util.java in the samples\client\java\src\ subfolder.

The service also supports Kerberos single sign-on authentication based on the Oasis Web Services Security Kerberos Token Profile. The format of the **BinarySecurityToken** node in the header security node is shown below.

Format of BinarySecurityToken node

```
<soapenv:Header
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <wsse:Security xmlns:wsse="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
      <wsse:BinarySecurityToken
         EncodingType="wsse:Base64Binary"
         ValueType="wsse:Kerberosv5_AP_REQ"
         Id="CentralKrbSSOToken">YIIJAQYJKoZIhvcSAQICAQBuggjwMIII....
      </wsse:BinarySecurityToken>
   </wsse:Security>
</soapenv:Header>
```

The Id attribute of `<wsse:BinarySecurityToken>` must be included and must be set to `"CentralKrbSSOToken"`. The name and value of the `Id` attribute are case sensitive.

### *Importing the SSL Certificate*

To enable the service to handle the SSL handshake that begins an SSL session, you can import the central.crt security certificate included in the OO SDK home directory in the samples\client\java\resources\ folder, into any keystore you choose, or you can use the sample

keystore provided in samples\client\java\resources\cacerts.sample. A keystore is a file containing keys, certificates, and trusted roots. The root certificates of signing authorities are kept in a file called a cacert.

To import the certificate into the default keystore provided with the Java Runtime Environment (JRE), open a command window and change directories to the lib\security\ subfolder in the Java home directory and run the following command under $(java.home)\lib\security:

```
keytool -import -alias pas -file central.crt –keystore cacerts
```

When prompted for the password for the JRE cacert, type `changeit`.

### *Sample client code*

Sample client code for WSCentralService is included in the OO SDK home directory, in the samples\client\.Net\src\ and samples\client\java\src\ folder. The lib\WSCentralService.jar file contains a wrapper for WSCentralService service named WSCentralServiceSession. You can use this class in developing your client. It wraps the service API stubs and includes additional functionality for Java and .NET.

### *Service stubs sample*

The following sample .cmd file generates the Java service stubs, which you can use if you choose not to use the supplied WSCentralService.jar file.

**Sample .cmd file**

```
@echo off
Rem replace %axis-1_4% and %wsdl_path% with the actual paths on your
machine.
Rem your path statement must include the folder where java.exe exist.

set JLIBS=%axis-1_4%\lib
set SERVICE_ADDRESS=%wsdl_path%\WSCentralService.wsdl
set SERVICE_PACKAGE=com.iconclude.dharma.services.wscentralservice.client


set BUILD_PATH=.
set CLASS_PATH=.
set CLASS_PATH=%CLASS_PATH%;%JLIBS%\axis.jar
set CLASS_PATH=%CLASS_PATH%;%JLIBS%\jaxrpc.jar
set CLASS_PATH=%CLASS_PATH%;%JLIBS%\commons-codec.jar
set CLASS_PATH=%CLASS_PATH%;%JLIBS%\commons-httpclient.jar
set CLASS_PATH=%CLASS_PATH%;%JLIBS%\commons-logging-1.0.4.jar
set CLASS_PATH=%CLASS_PATH%;%JLIBS%\commons-discovery.jar
set CLASS_PATH=%CLASS_PATH%;%JLIBS%\wsdl4j-1.5.1.jar
set CLASS_PATH=%CLASS_PATH%;%JLIBS%\activation.jar
set CLASS_PATH=%CLASS_PATH%;%JLIBS%\saaj.jar
set CLASS_PATH=%CLASS_PATH%;%JLIBS%\mail.jar
set command1=java -classpath %CLASS_PATH% org.apache.axis.wsdl.WSDL2Java
-a -p %SERVICE_PACKAGE% -v -o %BUILD_PATH% %SERVICE_ADDRESS%

%command1%
```

## Resuming runs from the command line

When a flow run has been handed off, you can resume it from a command line. You can resume the run so that it completes either synchronously or asynchronously.

- Synchronous: resumes the run identified by a run ID (not the run history ID) and waits for the run to complete.
- Asynchronous: resumes the run identified by a run iD (not the run history ID) but does not wait for it to complete..

If you don't know what the run's run ID is, you can obtain it from OO Central on the **Current Runs** tab.

**Note:** Although you can resume a run that has not been paused, it is strongly recommended that you not do so.

Suppose you have a flow that contains a couple of transitions that are marked for hand off.

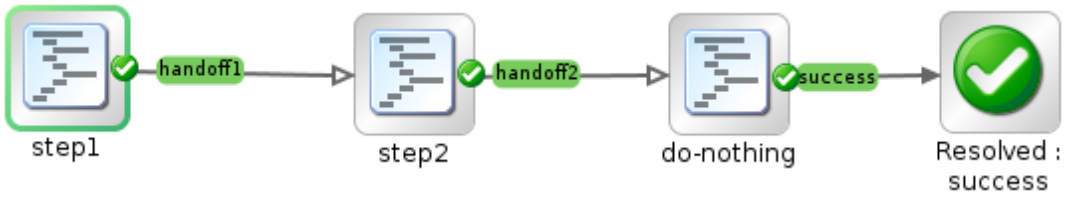


**Figure 7 - Flow with transitions marked for handoff**

The following example is of a command that starts a run of such a flow.

```
$ java -jar JRSFlowInvoke.jar
http://localhost:8080/PAS/services/http/run/Library/test/handoff/flow-that-hands-off
-u myuser -p *****
```

This action returns the following XML, in the block as presented below:

```
<?xml version="1.0" encoding="UTF-8"?>
<runResponse><runReturn><item><name>runId</name><value>718032</value></item><item><na
me>runHistoryId</name><value>718031</value></item><item><name>runReportUrl</name><val
ue>http://localhost:8080/PAS/app?service=RCLinkService/ReportLinkDispatch&amp;sp=SIND
IVIDUAL_REPAIR_LEVEL&amp;sp=S368a860a-a54a-4901-83d7-
11193ce2ca64&amp;sp=l0&amp;sp=l718031</value></item><item><name>displayRunReportUrl</
name><value><![CDATA[http://localhost:8080/PAS/app?service=RCLinkService/ReportLinkDi
spatch&sp=SINDIVIDUAL_REPAIR_LEVEL&sp=S368a860a-a54a-4901-83d7-
11193ce2ca64&sp=l0&sp=l718031]]></value></item><item><name>runStartTime</name><value>
03/01/10 10:11</value></item><item><name>runEndTime</name><value>03/01/10
10:11</value></item><item><name>flowResponse</name><value>HANDOFF</value></item><item
><name>flowResult</name><value>{}</value></item><item><name>flowReturnCode</name><val
ue>Not a Return</value></item></runReturn></runResponse>
```

To clarify its structure and improve its readability, we'll format the above XML conventionally. Remember that the block above is how the XML appears in the command-line window.

```
<?xml version="1.0" encoding="UTF-8"?>
<runResponse>
   <runReturn>
      <item>
         <name>runId</name>
         <value>718032</value>
      </item>
      <item>
         <name>runHistoryId</name>
         <value>718031</value>
      </item>
```

```
      <item>
         <name>runReportUrl</name>
         <value>http://localhost:8080/PAS/app?service=RCLinkService/ReportLinkDispatc
         h&amp;sp=SINDIVIDUAL_REPAIR_LEVEL&amp;sp=S368a860a-a54a-4901-83d7-
         11193ce2ca64&amp;sp=l0&amp;sp=l718031</value>
      </item>
      <item>
         <name>displayRunReportUrl</name>
         <value><![CDATA[http://localhost:8080/PAS/app?service=RCLinkService/ReportLi
         nkDispatch&sp=SINDIVIDUAL_REPAIR_LEVEL&sp=S368a860a-a54a-4901-83d7-
         11193ce2ca64&sp=l0&sp=l718031]]></value>
      </item>
      <item>
         <name>runStartTime</name>
         <value>03/01/10 10:11</value>
      </item>
      <item>
         <name>runEndTime</name>
         <value>03/01/10 10:11</value>
      </item>
      <item>
         <name>flowResponse</name>
         <value>HANDOFF</value>
      </item>
      <item>
         <name>flowResult</name>
         <value>{}</value>
      </item>
      <item>
         <name>flowReturnCode</name>
         <value>Not a Return</value>
      </item>
   </runReturn>
</runResponse>
```

Observations about the preceding XML:

- The run ID is in this case `718032`.
- The `flowResponse` item shows us that the run was stopped in a `HANDOFF` state because a handoff transition was followed.
- The `flowReturnCode` item has the value `Not a Return` because no return step was ever reached and thus the flow run is not complete.

### *Resuming a flow run synchronously*

The following is the command to resume the run headlessly in synchronous mode:

```
$ java -jar JRSFlowInvoke.jar http://localhost:8080/PAS/services/http/resume/718032 -
u myuser -p *****
```

Note that in the above, `resume` is the verb that determines that the flow is resumed in synchronous mode. In the section on resuming a flow asynchronously, you will see the verb that is used to resume it asynchronously.

Following is the XML that is returned (we won't format this, but just highlight the points of interest):

```
<resumeResponse><resumeReturn><item><name>runId</name><value>718032</value></item><it
em><name>runHistoryId</name><value>718031</value></item><item><name>runReportUrl</nam
e><value>http://localhost:8080/PAS/app?service=RCLinkService/ReportLinkDispatch&amp;s
p=SINDIVIDUAL_REPAIR_LEVEL&amp;sp=S368a860a-a54a-4901-83d7-
11193ce2ca64&amp;sp=l0&amp;sp=l718031</value></item><item><name>displayRunReportUrl</
name><value><![CDATA[http://localhost:8080/PAS/app?service=RCLinkService/ReportLinkDi
spatch&sp=SINDIVIDUAL_REPAIR_LEVEL&sp=S368a860a-a54a-4901-83d7-
11193ce2ca64&sp=l0&sp=l718031]]></value></item><item><name>runStartTime</name><value>
03/01/10 10:11</value></item><item><name>runEndTime</name><value>03/01/10
10:12</value></item><item><name>flowResponse</name><value>HANDOFF</value></item><item
><name>flowResult</name><value>{}</value></item><item><name>flowReturnCode</name><val
ue>Not a Return</value></item></resumeReturn></resumeResponse>
```

Note how the XML returned is very similar to that from the initiation of the run. Also:

- The `flowResponse` item shows us that the run was stopped again in a `HANDOFF` state, due to the second handoff transition being followed.

Let's resume the run once again:

```
$ java -jar JRSFlowInvoke.jar http://localhost:8080/PAS/services/http/resume/718032 -
u myuser -p *****
```

Following is the return XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<resumeResponse><resumeReturn><item><name>runId</name><value>718032</value></item><it
em><name>runHistoryId</name><value>718031</value></item><item><name>runReportUrl</nam
e><value>http://localhost:8080/PAS/app?service=RCLinkService/ReportLinkDispatch&amp;s
p=SINDIVIDUAL_REPAIR_LEVEL&amp;sp=S368a860a-a54a-4901-83d7-
11193ce2ca64&amp;sp=l0&amp;sp=l718031</value></item><item><name>displayRunReportUrl</
name><value><![CDATA[http://localhost:8080/PAS/app?service=RCLinkService/ReportLinkDi
spatch&sp=SINDIVIDUAL_REPAIR_LEVEL&sp=S368a860a-a54a-4901-83d7-
11193ce2ca64&sp=l0&sp=l718031]]></value></item><item><name>runStartTime</name><value>
03/01/10 10:11</value></item><item><name>runEndTime</name><value>03/01/10
10:14</value></item><item><name>flowResponse</name><value>success</value></item><item
><name>flowResult</name><value>{FailureMessage=;TimedOut=;Result=;}</value></item><it
em><name>flowReturnCode</name><value>Resolved</value></item></resumeReturn></resumeRe
sponse>
```

This time the flow has finished, the response is `success` and the return code is `Resolved`.

### *Resuming a run asynchronously*

For the asynchronous style, the links are very similar, the only difference being that the `resume` verb is replaced with `resume_async`. Let's conduct the experiment again:

Initial invocation of the run:

```
$ java -jar JRSFlowInvoke.jar
http://localhost:8080/PAS/services/http/run/Library/test/handoff/flow-that-hands-off
-u myuser -p *****
```

The run encounters the first "handoff" transition, and returns the following XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<runResponse><runReturn><item><name>runId</name><value>718067</value></item><item><na
me>runHistoryId</name><value>718066</value></item><item><name>runReportUrl</name><val
ue>http://localhost:8080/PAS/app?service=RCLinkService/ReportLinkDispatch&amp;sp=SIND
IVIDUAL_REPAIR_LEVEL&amp;sp=S368a860a-a54a-4901-83d7-
11193ce2ca64&amp;sp=l0&amp;sp=l718066</value></item><item><name>displayRunReportUrl</
name><value><![CDATA[http://localhost:8080/PAS/app?service=RCLinkService/ReportLinkDi
spatch&sp=SINDIVIDUAL_REPAIR_LEVEL&sp=S368a860a-a54a-4901-83d7-
11193ce2ca64&sp=l0&sp=l718066]]></value></item><item><name>runStartTime</name><value>
```

```
03/01/10 10:16</value></item><item><name>runEndTime</name><value>03/01/10
10:16</value></item><item><name>flowResponse</name><value>HANDOFF</value></item><item
><name>flowResult</name><value>{}</value></item><item><name>flowReturnCode</name><val
ue>Not a Return</value></item></runReturn></runResponse>
```

Here, the run ID is `718067`.

Now we resume the run asynchronously, using the `resume_async` verb:

```
$ java -jar JRSFlowInvoke.jar
http://localhost:8080/PAS/services/http/resume_async/718067 -u myuser -p *****
```

Return XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<resumeResponse><resumeReturn><item><name>runId</name><value>718067</value></item><it
em><name>runHistoryId</name><value>718066</value></item><item><name>runReportUrl</nam
e><value>http://localhost:8080/PAS/app?service=RCLinkService/ReportLinkDispatch&amp;s
p=SINDIVIDUAL_REPAIR_LEVEL&amp;sp=S368a860a-a54a-4901-83d7-
11193ce2ca64&amp;sp=l0&amp;sp=l718066</value></item><item><name>displayRunReportUrl</
name><value><![CDATA[http://localhost:8080/PAS/app?service=RCLinkService/ReportLinkDi
spatch&sp=SINDIVIDUAL_REPAIR_LEVEL&sp=S368a860a-a54a-4901-83d7-
11193ce2ca64&sp=l0&sp=l718066]]></value></item><item><name>runStartTime</name><value>
03/01/10
10:16</value></item><item><name>flowResponse</name><value></value></item><item><name>
flowResult</name><value></value></item><item><name>flowReturnCode</name><value>Not a
Return</value></item></resumeReturn></resumeResponse>
```

The response is very similar to the synchronous resumption of the run, with the following differences:

- There is no `runEndTime` item (because the end time is unknown)
- The `flowResponse` item is empty.

Let's run the asynchronous resumption more time:

```
$ java -jar JRSFlowInvoke.jar
http://localhost:8080/PAS/services/http/resume_async/718067 -u myuser -p *****
```

The return XML is:

```
<?xml version="1.0" encoding="UTF-8"?>
<resumeResponse><resumeReturn><item><name>runId</name><value>718067</value></item><it
em><name>runHistoryId</name><value>718066</value></item><item><name>runReportUrl</nam
e><value>http://localhost:8080/PAS/app?service=RCLinkService/ReportLinkDispatch&amp;s
p=SINDIVIDUAL_REPAIR_LEVEL&amp;sp=S368a860a-a54a-4901-83d7-
11193ce2ca64&amp;sp=l0&amp;sp=l718066</value></item><item><name>displayRunReportUrl</
name><value><![CDATA[http://localhost:8080/PAS/app?service=RCLinkService/ReportLinkDi
spatch&sp=SINDIVIDUAL_REPAIR_LEVEL&sp=S368a860a-a54a-4901-83d7-
11193ce2ca64&sp=l0&sp=l718066]]></value></item><item><name>runStartTime</name><value>
03/01/10
10:16</value></item><item><name>flowResponse</name><value></value></item><item><name>
flowResult</name><value></value></item><item><name>flowReturnCode</name><value>Not a
Return</value></item></resumeReturn></resumeResponse>
```

Again, the same answer.

If we were to resume the flow a third time, the flow should have finished in the background, so running it should produce an error:

```
$ java -jar JRSFlowInvoke.jar
http://localhost:8080/PAS/services/http/resume_async/718067 -u myuser -p *****
```

The error looks like the following:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
```

```
<title>Error 404 request=/PAS/services/http/resume_async/718067, error=path not
found: Run: 718067 not found, either finished or deleted., host=127.0.0.1</title>
</head>
<body><h2>HTTP ERROR: 404</h2><pre>request=/PAS/services/http/resume_async/718067,
error=path not found: Run: 718067 not found, either finished or deleted.,
host=127.0.0.1</pre>
<p>RequestURI=/PAS/services/http/resume_async/718067</p><p><i><small><a
href="http://jetty.mortbay.org/">Powered by LaunchJetty6://</a></small></i></p>
<br/>
</body>
</html>
```

Note that resuming the flow in synchronous mode would obtain the same response:

```
$ java -jar JRSFlowInvoke.jar http://localhost:8080/PAS/services/http/resume/718067 -
u myuser -p *****
```

Here is the error:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
<title>Error 404 request=/PAS/services/http/resume/718067, error=path not found: Run:
718067 not found, either finished or deleted., host=127.0.0.1</title>
</head>
<body><h2>HTTP ERROR: 404</h2><pre>request=/PAS/services/http/resume/718067,
error=path not found: Run: 718067 not found, either finished or deleted.,
host=127.0.0.1</pre>
<p>RequestURI=/PAS/services/http/resume/718067</p><p><i><small><a
href="http://jetty.mortbay.org/">Powered by LaunchJetty6://</a></small></i></p>
<br/>
</body>
</html>
```

# Working with repositories from outside Studio

As someone who works with OO repositories frequently, you may occasionally find it easier and less time-consuming to perform common OO repository functions outside of Studio.

The OO SDK Repository Tool RepoTool is a thin wrapper that includes all java dependencies and configuration files. It is a jar file called RepoTool.jar. A java JRE is required to run it. You can use it to perform the following repository functions outside of Studio:

- *Publish* new or changed OO objects, including flows and operations, from a source repository to a target repository.
- *Update* a source repository with new or changed OO objects from a target repository.
- *Publish and update* in one operation.
- *Export* a repository.
- *Verify* a repository, finding problems with the OO objects in it, and optionally fixing them.
- *Upgrade* a repository to the latest version.
- *Encrypt*, *decrypt*, and *re-encrypt* a repository.
- *Set default permissions* for a repository.
- *Import* a file of localized content.
- *Export* content to be localized to a file.

- *Set the sealed, hidden, and content flags* for objects in a specified path.
- *Delete objects* in a specified path.

**Note:** The target repository is always a Central public repository.

## Using the Repository Tool

You run the Repository Tool, also known as RepoTool, from a command line using one of the RepoTool primary options. RepoTool has ten primary options, each of which tells RepoTool which repository function to perform. RepoTool also has secondary options that provide information that is required by the primary options.

**Syntax**

```
java jar RepoTool.jar [<primary option>] [<secondary options>]
```

The RepoTool.jar file is located in the OO SDK home directory. It is entirely self-contained in a single .jar file, to see the list of parameters, run it with no arguments as follows:

```
java –jar RepoTool.jar
```

### Primary RepoTool options

The primary RepoTool options specify the repository functions to be performed.

| Primary Option | Description |
|---|---|
| `-publish` | Publishes changes from a source repository to a target repository. |
| `-update` | Updates from a target repository to a source repository. |
| `-publishupdate` | Combination of `–publish` and `–update`. |
| `-export` | Exports a source repository, making flows and OO objects available to users who do not share the same public repository as you. |
| `-verify` | Verifies the structural integrity of a repository, then lists and optionally fixes any problems. The repository must be a local repository. |
| `-upgrade` | Upgrades a repository to the latest version. |
| `-encrypt` | Encrypts a repository using an encryption password and saves it to another repository. The repository must be a local repository. |
| `-decrypt` | Decrypts an encrypted repository using an encryption password and saves it to another repository. The repository must be a local repository. |
| `-reencrypt` | Re-encrypts a local repository using an encryption password and encrypts it to another repository using another encryption password. The repository must be a local repository. |

| Primary Option | Description |
|---|---|
| -defaultperms | Sets default permissions for a repository. The repository must be a local repository. |
| -localizeimport | Reads a localization file and places the strings into the content. The repository must be a local repository. |
| -localizeexport | Exports localizable content to a file. |
| -setflags | Sets flags based on the settings of the -seal, -hide, and –content secondary options. The repository must be a local repository. |
| -delete | Deletes objects based on the specified -includepath. The repository must be a local repository. |

### Secondary RepoTool options

Most of the secondary RepoTool options work with more than one primary option. If a secondary option works with only one primary option, it is described in the section that explains how to use the primary option.

The rest of the secondary RepoTool options and the primary options they work with are shown in the following table:

| Secondary Option | Description | Used With |
|---|---|---|
| -1 <repository1> | Specifies the path and name of <repository1>. For most of the primary RepoTool options, this is the source repository.<br>For the following options, <repository1> can only be a local repository:<br>• -verify<br>• -upgrade<br>• -encrypt<br>• -decrypt<br>• –reencrypt<br>• -delete<br>• -localizeimport<br>• -localizeexport<br>For the following options, <repository1> can be a local or Central repository:<br>• –publish<br>• -update<br>• -publishupdate<br>• -export<br>• -defaultperms | -publish<br>-update<br>-publishupdate<br>-export<br>-verify<br>-upgrade<br>-encrypt<br>-decrypt<br>-reencrypt<br>-defaultperms<br>-localizeexport<br>-localizeimport<br>-setflags<br>-delete |

| Secondary Option | Description | Used With |
|---|---|---|
| | A local repository is specified by a path, for example c:\MyFolder\MyRepository.<br>A Central repository is specified with a URL, for example https://central-host2:8443. | |
| -2 <repository2> | Specifies the path and name of <repository2>. For most of the primary RepoTool options, this is the target repository.<br>For the following options, <repository2> can only be a local repository:<br>• -encrypt<br>• -decrypt<br>• –reencrypt<br>For the following options, <repository2> can be a local or Central repository:<br>• -publishupdate<br>• -export<br>A local repository is specified by a path, such as c:\MyFolder\MyRepository.<br>A Central repository is specified with a URL, such as https://central-host2:8443 | -publish<br>-update<br>-publishupdate<br>-export<br>-encrypt<br>-decrypt<br>-reencrypt |
| -c <value> | Specifies how conflicts are resolved. A conflict can occur if changes have been made to the same object in both the target and source repositories. The <value> is one of the following:<br>• 0 Skips the conflicts.<br>• r1 Conforms <repository1> to <repository2>.<br>• r2 Conforms <repository2> to <repository1>.<br>See *Publishing a repository* for more information about using the –c option. | -publish<br>-update<br>-publishupdate |
| -children | Used with the –delete secondary option, -children deletes only the children of the paths specified by -includepath. | -delete |
| -content <*true\|false*> | All objects specified by -includepath will have their **content** flag set to the selected value <true\|false>. Folders are recursive. | -setflags |

| Secondary Option | Description | Used With |
|---|---|---|
| -excludepath <path> | Specifies a path in <repository1> to exclude when publishing or updating (for example, -excludepath "/Library/My Repairs"). The path must be enclosed in quotation marks. The flows and objects in the excluded path will not be published or updated. | -publish<br>-update<br>-publishupdate |
| -hide <*true*\|*false*> | All folders and flows specified by -includepath will have their **hidden** flag set to this value. If set to **true**, the objects are hidden in Central. Folders are not recursive. | -setflags |
| -includepath <path> | Specifies the only path in <repository1> to include when publishing or updating (for example, -includepath "/Library/My Repairs"). Only the path specified in <repository1> will be published or updated. | -publish<br>-update<br>-publishupdate |
| -includereferences | You can only use this option when you use the -includepath option. The -includereferences option specifies that any flows or operations used by the flows in the path specified in the -includepath option will also be published. | -publish<br>-update<br>-publishupdate |
| -localizationfile <*localizationfile*> | Specifies the file to read from or write to for localization. | -localizeimport<br>-localizeexport |
| -loginurl <loginurl> | Specifies the URL of the Central server with which to authenticate if <repository1> or <repository2> is remote. | -publish<br>-update<br>-publishupdate<br>-export<br>-localizeexport<br>-localizeimport |
| -n | Specifies that RepoTool should not perform the –publish, -update, -publishupdate, or -export option with which the –n option is used, but instead should print the results that would occur if the option was performed. | -publish<br>-update<br>-publishupdate<br>-export |
| -p <password> | Specifies the password for the Central or remote server. | -publish<br>-update<br>-publishupdate<br>-export<br>-upgrade<br>-encrypt |

| Secondary Option | Description | Used With |
|---|---|---|
| | | -decrypt<br>-defaultperms<br>-localizeexport<br>-localizeimport |
| -q <repo2password> | Specifies the encryption password for <repository2>. It is ignored if <repository2> is a remote repository or if it is not encrypted. | -publish<br>-encrypt<br>-decrypt<br>-reencrypt |
| -r <repo1password> | Specifies the encryption password for <repository1>. It is ignored if <repository1> is a remote repository or if it is not encrypted. | -publish<br>-update<br>-publishupdate<br>-export<br>-verify<br>-upgrade<br>-decrypt<br>-reencrypt<br>-defaultperms<br>-localizeexport<br>-localizeimport |
| -seal <*true*\|*false*> | All folders specified by -includepath will have their **sealed** flag set to the selected value <true\|false>. If set to **true**, the folders are sealed. When a folder is sealed, it cannot be changed. Folders are recursive. The repository must be a local repository. | -setflags |
| -u <username> | Specifies the username for the Central or remote server.<br>**Note:** The username must belong to an administrator for the options shown in the **Used With** column except for the –publish option, where the username can belong to a member of the PROMOTER group. | -publish<br>-update<br>-publishupdate<br>-export<br>-upgrade<br>-encrypt<br>-decrypt<br>-defaultperms<br>-localizeexport<br>-localizeimport |

## Return codes

0 Success

1 An exception happened during publish

2 Repository changed while publishing

3 Bad command line options were given

4 There are no differences between the source and target repositories

## Publishing a repository

The –publish option copies new or changed objects—such as flows and operations—from a source repository (<repository1>) to a target repository (<repository2>).

**Syntax**

```
java –jar RepoToo.jar -publish -loginurl <loginurl> -u <username> -p <password>
-1 <repository1> [-r <repo1password>] -2 <repository2>
[-q <repo2password>] -c 0|r1|r2 [-n] [-excludepath <path>] [-includepath <path>] [-
includereferences]
```

Both <repository1> and <repository2> stand for paths to local repositories or URLs of Central repositories.
The –c option tells RepoTool what to do if a conflict is reported because changes have been made to an object with the same name in the target and source repositories.

- 0 Skips the conflicts.
- r1 Conforms <repository1> to <repository2>.
- r2 Conforms <repository2> to <repository1>.

The following scenario illustrates how this works:

1. Local repository <repository1> and Central repository <repository2> contain a flow named testflow and are synchronized.
2. You import a new version of testflow to <repository1>.
3. From a second local repository, you publish another version of testflow to <repository2>. Now testflow has changed in both <repository1> and <repository2>. A conflict will be reported for testflow1 when you preview publishing from <repository1> to <repository2>. Use the –c <value> option to specify how you want to resolve a potential conflict.

The values for the –c option and descriptions of the other secondary options used in the -publish syntax, are shown in *Secondary RepoTool options*.

To learn how to publish a repository using Studio, see the material on publishing a repository in the *Guide to Authoring Operations Orchestration Flows*.

The following examples show some of the ways you can use the RepoTool –publish option.

**Example 1**

This example publishes the contents of the exported repository at c:\MyFolder\export to the Central server central-host2. The option –c r2 tells RepoTool that if conflicts occur, it should change central-host2 to resolve them.

```
java –jar RepoTool.jar -publish -loginurl https://central-host2:8443 -u admin
-p iconclude -1 c:\MyFolder\export -2 https://central-host2:8443 -c r2
```

**Note:** This is comparable to connecting Studio to central-host2 and to the repository at c:\MyFolder\export.

**Example 2**

This example publishes the contents of the folder /Library/My Ops Flows/Network Flows/ from the Central host central-host1 to the Central host central-host2.

```
java –jar RepoTool.jar -publish -loginurl https://central-host1:8443 -u admin
-p iconclude -1 https://central-host1:8443 -2 https://central-host2:8443 -c r2 -
includepath "/Library/My Ops Flows/Network Flows"
```

## Updating from a repository

The –update option is the opposite of the –publish option. When you update a source repository (<repository2>) to a target repository (<repository1>), any flows and objects that are new or have changed in the source repository are copied to the target repository.

**Syntax**

```
java –jar RepoTool.jar  -update -loginurl <loginurl> -u <username> -p <password>
-1 <repository1> [-r <repo1password>] -2 <repository2> -c 0|r1|r2 [-n]
[-excludepath <path>] [-includepath <path>] [-includereferences]
```

The parameter <repository1> represents the repository from which you initiate the update. Both <repository1> and <repository2> can be paths to local repositories or URLs of Central repositories.

The secondary options used in the -update syntax are described in *Secondary RepoTool options*.

To learn how to update a repository using Studio, see the material on updating from a repository in the *Guide to Authoring Operations Orchestration Flows*.

**Example**

In this example, RepoTool updates the path /Library/My Ops Flows/TestFlow/ in the central2 repository from the local repository My Repository. The -includereferences option tells RepoTool to include all references to the flows and operations used by TestFlow. The –c option tells RepoTool to change the central2 repository to resolve conflicts if they occur.

```
java –jar RepoTool.jar  -update -1 https://central2:8443 -2 c:\MyFolder\My Repository
-includepath "/Library/My Ops Flows/TestFlow" –includereferences –c r1
```

## Publishing and updating a repository simultaneously

The –publishupdate option copies objects that are new or have changed from a source repository (<repository1>) to a target repository (<repository2>) and copies flows and objects that are new or have changed from a target repository (<repository1>) to a source repository (<repository2>).

**Syntax**

```
java –jar RepoTool.jar  -publishupdate -loginurl <loginurl> -u <username> -p
<password> -1 <repository1> [-r <repo1password>] -2 <repository2> -c 0|r1|r2 [-n] [-
excludepath <path>][-includepath <path>]
[-includereferences]
```

The secondary options used in the -publishupdate syntax, are described in *Secondary RepoTool options*.

## Exporting a repository

To make flows and OO objects available to authors with whom you do not share a public repository, you can use the –export option to export one repository (<repository1>), which can be a local repository or a Central repository, to a target location (<repository2>), which specifies a directory on the local file system. Other Studio authors can then import the repository from that location.

**Note:** The RepoTool –export option creates <repository2>. The <repository2> directory should not exist prior to exporting <repository1>.

**Syntax**

```
java –jar RepoTool.jar  -export -loginurl <loginurl> -u <username> -p <password>
-1 <repository1> [-r <repo1password>] -2 <repository2> [-x <path1>
-x <path2>] [-n]
```

The `-x <path>` option sets the path in the repository or library to be exported (for example, `-x "/Library/My Repairs/"`). The path specified in the `–x` option must be enclosed in quotation marks. You can specify multiple paths using more than one `–x` option. The default is `-x "/Library" -x "/Configuration/"`.

The other secondary options used in the `-export` syntax, are described in *Secondary RepoTool options*.

To learn how to export a repository using Studio, see the material on exporting a repository in the *Guide to Authoring Operations Orchestration Flows*.

**Example**

This example exports the folders /Library/My Ops Flows/Network Flows/ and /Library/My Ops Flows/Database Flows/ from the repository of the Central host central-host1 to a new local repository named My Repository.

```
java –jar RepoTool.jar  -export -loginurl https://central-host1:8443 -u admin
-p iconclude -1 https://central-host1:8443 -2 c:\MyFolder\My Repository
-x "/Library/My Ops Flows/Network Flows/" -x "/Library/My Ops Flows/Database Flows/"
```

**Note:** This is comparable to connecting Studio to central-host1, selecting the folders Library/My Ops Flows/Network Flows/ and Library/My Ops Flows/Database Flows/, and exporting them to c:\MyFolder\My Repository.

## Verifying a repository

The `–verify` option allows you to verify the structural integrity of a local repository.

**Syntax**

```
java –jar RepoTool.jar -verify [-f] -1 <repository1> [-r <repo1password>]
```

The `–f` option specifies that RepoTool will attempt to fix any structural integrity problems it encounters in `<repository1>`.

The other secondary options used in the `-verify` syntax, are described in *Secondary RepoTool options*.

**Example**

This example verifies the repository at C:\MyFolder\MyRepo, lists any problems that exist in the flows and other objects in the repository, and then attempts to fix the problems.

```
java –jar RepoTool.jar  -verify -f -1 C:\MyFolder\MyRepo
```

## Upgrading a repository

The `–upgrade` option upgrades a local repository to the latest version. This option is designed to be used mainly by OO authors, since upgrades are normally done automatically in a production environment.

**Syntax**

```
java –jar RepoTool.jar  -upgrade -u <username> -p <password> -1 <repository1>
[-r <repo1password>]
```

The secondary options used in the `-upgrade` syntax are described in *Secondary RepoTool options*.

**Example**

This example upgrades the repository C:\MyFolder\MyRepo to the latest version that is associated with the RepoTool.jar.

```
java –jar RepoTool.jar -upgrade -1 C:\MyFolder\MyRepo
```

## Encrypting a repository

The `–encrypt` option makes a copy of a local repository (`<repository1>`), encrypts it, and then saves it as `<repository2>`. You can use encryption to protect your repository from unauthorized users.

**Syntax**

```
java –jar RepoTool.jar -encrypt [-u <username> -p <password>] -1 <repository1>
[-r <repo1password>] -2 <repository2> -q <repo2password>
```

If you modify, publish to, update from, import to, or export `<repository2>`, you must use the `<repo2password>` password.

The secondary options used in the `-encrypt` syntax are described in *Secondary RepoTool options*.

To learn how to encrypt a repository using Studio, see the material on encrypting a repository in the *Guide to Authoring Operations Orchestration Flows*.

**Example**

This example copies the repository My Repository, encrypts it, and then saves it as My Encrypted Repository.

```
java –jar RepoTool.jar -encrypt -1 c:\MyFolder\My Repository -r iconclude -2
c:\MyFolder\My Encrypted Repository -q iconclude2
```

## Decrypting a repository

The `–decrypt` option decrypts an encrypted repository (`<repository1>`) and saves it as a decrypted repository (`<repository2>`).

**Syntax**

```
java –jar RepoTool.jar -decrypt [-u <username> -p <password>] -1 <repository1>
-r <repo1password> -2 <repository2>
```

For `<repo1password>`, use the password you set when you encrypted the repository. See *Encrypting a repository* for more information.

The secondary options used in the `-decrypt` syntax are described in *Secondary RepoTool options*.

To learn how to decrypt a repository using Studio, see the material on decrypting a repository in the *Guide to Authoring Operations Orchestration Flows*.

**Example**

This example decrypts My Encrypted Repository using the password `iconclude2` and saves it as the decrypted repository My Decrypted Repository.

```
java –jar RepoTool.jar -decrypt -1 c:\MyFolder\My Encrypted Repository
-q iconclude2 -2 c:\MyFolder\My Decrypted Repository
```

## Re-encrypting a repository

The `–reencrypt` option allows you to create a second encrypted copy of a repository with a different password.

**Syntax**

```
java –jar RepoTool.jar -reencrypt -1 <repository1> -r <repo1password>
-2 <repository2> -q <repo2password>
```

Using `–reencrypt`, RepoTool makes and encrypts a second copy of the encrypted repository `<repository1>` named `<repository2>` and re-encrypts it with a new password specified in `<repo2password>`.

The secondary options used in the `-reencrypt` syntax are described in *Secondary RepoTool options*.

To learn how to re-encrypt a repository using Studio, see the material on creating a second encrypted copy of a repository in the *Guide to Authoring Operations Orchestration Flows*.

**Example**

This example creates a second copy of the encrypted repository My Encrypted Repository named My Second Encrypted Repository with the new password iconclude4.

```
java –jar RepoTool.jar -reencrypt -1 c:\MyFolder\My Encrypted Repository –r
iconclude2 -2 c:\MyFolder\My Second Encrypted Repository
-q  iconclude4
```

## Setting default permissions for a repository

The `–defaultperms` option allows you to set the default access permissions for `<repository1>`. This applies the default permissions to all of the contents of `<repository1>`. The default permissions for a newly-created object are **Read**, **Write**, **Execute**, and **Link To** for the group **EVERYBODY**.

**Syntax**

```
java –jar RepoTool.jar -defaultperms -u <username> -p <password> -1 <repository1>
[-r <repo1password>]
```

To learn more about access permissions in Studio, see the material on setting default access permissions for groups in the *Guide to Authoring Operations Orchestration Flows*.

The secondary options used in the `-defaultperms` syntax are described in *Secondary RepoTool options*.

**Example**

This example sets default permissions for My Repository, so that all of the users in the group EVERYBODY have read, write, execute, and link permissions for it.

```
java –jar RepoTool.jar -defaultperms -1 “My Repository”
```

## Exporting content to be localized

The `–localizeexport` option exports values in the specified repository to the file specified in the `–localization` file secondary output so that it can be localized manually. This localized content can then be imported into the repository using the `–localizeimport` option.

**Syntax**

```
java –jar RepoTool.jar -localizeexport [-loginurl <loginurl>] [-u <username> -p
<password>] -1 <repository1> [-r <repo1password>] -localizationfile
<localizationfile>
```

The secondary options used in the `-localizeexport` syntax are described in *Secondary RepoTool options*.

**Example**

This example exports the localizable content in the central-host1 repository to the localization file Localfile.

```
java –jar RepoTool.jar -localizeexport -loginurl https://central-host1:8443 -u admin
-p iconclude -1 https://central-host1:8443 -localizationfile Localfile
```

## Importing a localization file

The `–localizeimport` option reads a localization file and places the localized strings into the Studio repository specified in the `-1` secondary option. The localization file contains OO content that has been translated into another language. This option replaces the content in the specified repository with the localized content.

**Syntax**

```
java –jar RepoTool.jar  -localizeimport [-loginurl <loginurl>] [-u <username> -p
<password>] -1 <repository1> [-r <repo1password>] -localizationfile
<localizationfile>
```

The secondary options used in the `- localizeimport` syntax are described in *Secondary RepoTool options*.

**Example**

This example reads the localization file Localfile and places the localized strings in the Studio repository central-host1.

```
java –jar RepoTool.jar -localizeimport -loginurl https://central-host1:8443 -u admin
-p iconclude -1 https://central-host1:8443  -localizationfile Localfile
```

## Setting flags

The `–setflags` option sets flags for all objects specified by the `–includepath` secondary option based on the settings of the `-seal`, `-hide`, and `–content` secondary options. The repository must be a local repository.

**Syntax**

```
java –jar RepoTool.jar -setflags -1 <repository1> -includepath <path> [-includepath
<path>] [-seal <true|false>] [-hide <true|false>] [-content <true|false>]
```

The secondary options used in the `-setflags` syntax are described in Secondary RepoTool options.

**Example**

This example sets the flags shown below for the Library/My Ops Flows/Network Flows/ repository.

- `-seal true` specifies that all objects in the path have their **sealed** flag set to **true**.
- `-hide false` specifies that all files and folders in the path have their **hidden** flag set to **false**.
- `-content true` specifies that all objects in the path have their **content** flag set to **true**.

```
Java –jar RepoTool.jar -setflags -1 c:\MyFolder\My Repository -includepath
"/Library/My Ops Flows/Network Flows" -seal true -hide false -content true
```

### Deleting objects

The `–delete` option deletes the objects specified in the `-includepath` secondary option.

**Syntax**

`java –jar RepoTool.jar -delete -1 <repository1> -includepath <path> [-includepath <path>] [-children]`

The secondary options used in the `-delete` syntax are described in Secondary RepoTool options.

**Example**

This example deletes only the children of the Library/My Ops Flows/Network Flows/ path.

```
java –jar RepoTool.jar -delete -1 c:\MyFolder\My Repository -includepath "/Library/My
Ops Flows/Network Flows" -children
```

## Packaging content

The HP OO Content Packager allows you to package content—repositories and RAS libraries—into content modules, then install the packaged content on Central and Remote Action Service (RAS) servers in your network. ContentPackager.jar can be found in the /SDK subfolder.

**Important:** RAS installed on a Windows server supports both Java and .NET RAS operations. However, RAS installed on a Linux server only supports Java RAS operations—it does not support .NET RAS operations.

### Using the Content Packager

The process of packaging content for distribution involves the following steps:

1. Create an XML configuration file that defines:
   - The content to be installed on the target Central or RAS servers.
   - The RAS libraries to be updated on the target RAS servers.
   - The repositories to be updated on the target Central servers.

   See *Creating the XML configuration file* to learn how to create the XML configuration file.
2. Package the content. The Content Packager uses the information in the XML configuration file to incorporate the content into a content module. It then creates a file named <name>-ContentInstaller.jar which contains the content module and the classes needed to install it. See *Packaging the content* for directions. The <name> is specified in <project>.
3. Create the OO home directory folder structure on the target server where you plan to install the content. If the target server has Studio installed on it, you can skip this step. See *Configuring the OO home directory folder structure* to learn what the structure should look like.
4. Install the packaged content. The Content Packager extracts the content into the OO home directory, in the updates\content*module**version*\ folder on the target server and then updates the Central repository, the local repository, or a RAS, depending on the arguments you use when you install the package. See *Installing the content* for instructions.

When you update a Central repository, it is important to let the authors who access that repository know that they should update their local repositories. For information about updating from a Central

repository, see the material on publishing to and updating from the public repository in the *Guide to Authoring Operations Orchestration Flows*.

## Creating the XML configuration file

The first step in packaging content is to create an XML configuration file that contains the information needed to package and install the content. This includes the location of the repository that contains the content on the source server and the path in the repository or library to be published to Central or a RAS.

The XML configuration file requires the following XML elements:

- A *project element* that defines the properties of content module.
- A *ras element* that specifies the location on the source server of the libraries to be updated on the target RAS servers.
- An *archive element* that provides information about how and where to install the updated content on the RAS servers.
- A *repository element* that specifies the location on the source server of the repositories to be updated and where to install them on the target Central servers.

These elements are described in the following sections. For an example, see *XML configuration file example*.

### The project element

The project element contains information about the content module that the Content Packager creates from your content.

**Syntax**

```
<project schema_version="value" name="value" version="value"
fullname="value"></project>
```

Following are the project element attributes and their values:

| Attribute | Value |
|---|---|
| schema_version | Always set this attribute to a value of `2`.<br>This attribute is reserved for future use. |
| name | A short name for the content package. The name cannot contain spaces or special characters such as quotation marks (“), asterisks (*), slash marks (/ and \), colons (:), angle brackets (< or >), question marks (?), vertical bars (\|), apostrophes (☐), ampersands (&), semicolons (;), and number sign (#). |
| fullname | The full display name of the content package. The full name can contain spaces, but not special characters, and should be understandable and reflect the contents of the package. |
| version | The content package version number. The version number is unique to the content package, and does not need to be related to the HP OO version numbering scheme. |

**Example**

```
<project schema_version="2" name="ExampleIntegration-OO" version="1.0.0" fullname="HP OO Example Integration Content Installer"></project>
```

### The ras element

The ras element defines the RAS libraries (.jar or .NET files) to be updated on the target RAS servers. The ras elements must be nested inside project elements.

**Syntax**

```
<ras type="value" dir="path" description="value" ></ras>
```

Following are the ras element attributes and their values:

| Attribute | Value |
|---|---|
| type | The language in which the RAS libraries are written—Java or .NET. Valid values are `java` and `dot_net`. |
| dir | The path where the libraries are located on the source server, relative to the path from which you are running the Content Packager. Subfolders in this path will be packaged if they include at least one file. |
| description | The description of the RAS libraries. |

**Example**

```
<ras type="java" dir="../dist/JRAS" description="Example Integration Java RAS Libraries"> </ras>
```

### The archive element

The archive element provides information needed to install the RAS libraries on the target RAS servers and determine which files should be packaged for installation on the RAS. Each archive element must be nested inside a ras element.

**Syntax**

```
<archive isLib="value" [libFolderName="value"]>path</archive>
```

where `path` is the path to the archive relative to the `dir` established in the `ras element`, using an asterisk (*) as a wildcard character.

The following describes the archive element attribute `isLib` and its value:

| Attribute | Attribute value |
|---|---|
| isLib | Set this attribute to a value of `false` if the library contains IActions.<br>Set it to a value of `true` if the library does not contain libraries IActions. |
| libFolderName | The subfolder of the content lib\ folder where the libraries are installed. If the isLib attribute is set to `true`, the libFolderName attribute must be |

| Attribute | Attribute value |
|---|---|
| | specified. |

**Examples**

```
<archive isLib=”false” >ExampleIntegration.jar</archive>
<archive isLib=”true” libFolderName="ExampleIntegration">lib/*.jar</archive>
```

The second example installs the specified libraries into the HP OO home folder, in the RAS\Java\Default\repository\lib\ExampleIntegration\ folder.

## The repository element

The repository element tells the Content Packager where to find the repository on the source server and which path to publish to Central. The repository element must be nested inside a project element.

**Syntax**

```
<repository path="value" desctiption="value">
<include>path</path>
</repository>
```

where `<include>` `path` is the source repository directory. The `path` attribute should be the path for the whole repository directory. Only relative paths are supported.

Following are the repository element attributes and their values:

| Attribute | Value |
|---|---|
| path | The path of the source content within the source repository, relative to the path from which you are running the Content Packager. It can also be the path of the target content within the target repository. |
| description | The description for the repository |

**Example**

The following example of the repository element instructs the packager to package all of the repository information under “../dist/Repository/ExampleIntegrationRepo ”, and to install or update all of the contents under source /Library to target /Library.

```
<repository path="../dist/Repsotory/ExampleIntegrationRepo
description=”ExampleIntegraton Respository”>
<include>/Library</include>
</repository>
```

## XML configuration file example

The following is an example of the XML configuration file you need to create for use with the Content Packager.

XML configuration file example

```
<?xml version="1.0" ?>
<project schema_version="2" name="ExampleIntegration-OO9.0" version="1.0.0"
fullname="HP OO Example Integration Content Installer">

<ras type="java" dir="../dist/JRAS" description="Example Integration Java
```

```
RAS Libraries">
<archive isLib=”false” >ExampleIntegration.jar</archive>
<archive isLib=”true” libFolderName="ExampleIntegration">lib/*.jar</archive>
</ras>

<ras type="dot_net" dir="../dist/NRAS"description="Example Integration .NET
RAS Libraries">
<archive>*.dll</archive>
</ras>

<repository path="../dist/Repsotory/ExampleIntegrationRepo”
description=”ExampleIntegraton Respository”>
<include>/Library</include>
</repository>

</project>

You can also incorporate the packaging process into automated build process.

Example:

<project name="Sample Ant Buildfile"
         xmlns:oo="antlib:com.hp.oo.content.utilities.packager.ant"
         xmlns="antlib:org.apache.tools.ant">
    <taskdef uri="antlib:com.hp.oo.content.utilities.packager.ant"
             resource="com/hp/oo/content/utilities/packager/ant/antlib.xml"
             classpath="/path/to/ContentPackager.jar" />

    <oo:packager outputFolder="${build.dir}/OO-Packages"
                 property="acme.installer"
                 name="Acme"
                 version="${version}"
                 fullname="My Acme 10.0 Integration"
                 description="Example Content Installer">
        <ras type="java"
             dir="${dist.dir}/"
             description="Blah">
            <archive>ACME.jar</archive>
            <archive libFolderName="ACME">ACME-lib.jar</archive>
            <archive libFolderName="ACME">
                <include name="*-commons.jar" />
                <exclude name="*test*.jar" />
            </archive>
        </ras>

        <repository path="${dist.dir}/repo"
                    description="Acme Repository content for ASGARD OO
release">
```

```
            <include name="/Library/Integrations/Acme/" />
            <exclude name="/Library/Integrations/Acme/Deprecated/" />
        </repository>
    </oo:packager>
</package>
```

## Packaging, depackaging, and repackaging the content

The Content Packager/Depackager uses the XML configuration file to:

- Package your content into a content module.
- Generate the content installer jar file which contains the content module and installation classes.
- Depackage the content installer jar file.
- Repackage the old content and make an installer for a newer version of OO based on the ContentPackager.jar's version. For example, if you have a content installer for OO 7.51, it cannot be used to install the content on OO 9.0. You can use repackage mode to package the content to an installer that works for OO 9.0.

For information about the makeup of the XML configuration file, see *Creating the XML configuration file*.

### To package the content

- In a command window, type:
  `java -jar ContentPackager.jar [-mode package] –configFile <file> [-outputFolder <dir>]`

**Option syntax**

`-mode`

Indicates that the action is to package the contents. The default is to package.

`-outputFolder`

Specifies the directory where the Content Packager will generate the content installer .jar file.

`-configFile`

Specifies the path of the XML configuration file.

**Example**

```
java –jar tools/ContentPakcager.jar –mode package –configFile config/package.xml –outputFolder output
```

### To depackage the content

- In a command window, type:
  `java -jar ContentPackager.jar -mode depackage –installerJar <file> [-outputFolder <dir>]`

**Option syntax**

`-mode`

Indicates that the action is to depackage the contents. For depackaging, use **depackage**.

`-outputFolder`

Specifies the directory where the Content Depackager will generate the content files.

`-installerJar`

Specifies the path of the content installer file.

**Example**

```
java –jar tools/ContentPakcager.jar –mode depackage –installerJar
output/ExampleIntegration-OO9.0-ContentInstaller.jar –outputFolder output/depackage
```

### To repackage the content

- In a command window, type:

  ```
  java  -jar ContentPackager.jar -mode repackage –installerJar <file> [-outputFolder
  <dir>]
  ```

**Option syntax**

`-mode`

Indicates that the action is to depackage the contents. For repackaging, use **repackage**.

`-outputFoler`

Specifies the directory where the Content Depackager will generate the new content installer.

`-installerJar`

Specifies the path of the old content installer file.

**Example**

```
java –jar tools/ContentPakcager.jar –mode repackage –installerJar
output/ExampleIntegration-OO7.51-ContentPackager.jar –outputFolder output/repackage
```

## Configuring the OO home directory structure

The OO home directory is the folder where HP OO is installed. By default, this is C:\Program Files\Hewlett-Packard\Operations Orchestration\.

Before you can install the packaged content, the OO home directory structure must be in place on the target server. If the server has Studio installed on it, the structure is already in place. If not, configure the OO home directory structure on the target server as shown in the following figure. The conf\ and plugins\ folders should contain all of the files from the version of Studio that was used to develop the content.

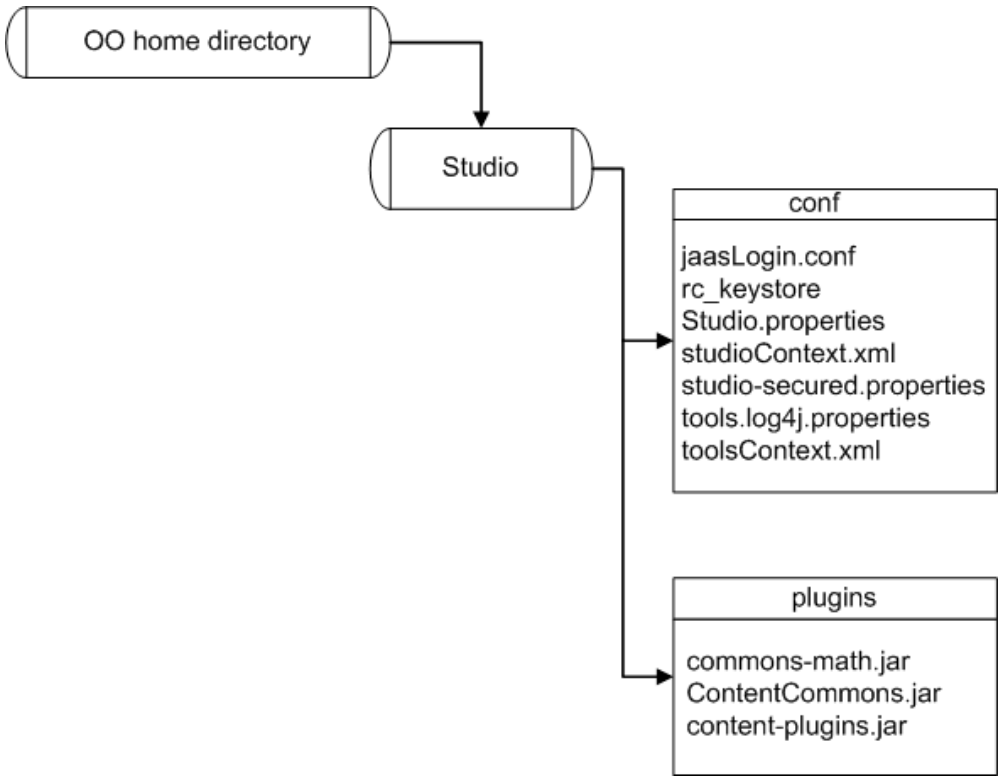


**Figure 8 - OO home directory structure**

After you create the needed folder structure, set the operating system environment variable ICONCLUDE_HOME to the OO home directory.

## Installing the content

When you install the content, the Content Packager extracts the packaged libraries and repositories from the ContentInstaller.jar file into the OO home directory, in the updates\content*module**version*\ folder on the target server. It then updates the Central server repository, local repository, or RAS specified in the arguments passed to it. By default, the Central repository at https://localhost:8443 is updated, as are all RASes referenced by it.

### To install the content

- In a command window, type:

```
java -jar <name>-ontentInstaller.jar [-ep encrypted repository <password>] [-
centralURL url] [-centralUsername <username>]
-centralPassword <password> | -ras RAS URL [-home iconclude_home] [-repo
localRepo]
```

**Option syntax**

`-ep`

Specifies the encrypted password to the target repository.

`-centralURL`

Specifies the URL of the Central repository to be updated. The default URL is `https://localhost:8443`

`-centralUsername`

Specifies the user name for accessing Central. The default is `admin`.

`-centralPassword`

Specifies the password for accessing Central.

`-ras`

Specifies the URL of the RAS to be updated. If you don't specify a RAS URL, the content is installed on all RASes that are registered in the target repository.

`-home`

Specifies the path of your OO installation. This defaults to the value of the ICONCLUDE_HOME environment variable.

`-repo`

Specifies the path of the repository to update. This defaults to the *centralURL* repository.

You can specify a local repository, but it is a best practice to update a Central repository and then have the authors who use that Central repository update their local repositories. For more information, see the material on publishing to and updating from the public repository in the *Guide to Authoring Operations Orchestration Flows*.

**Example**

```
Java –jar ExampleInstalleration-OO9.0-ContentInstaller.jar -centralUrl
https://localhost:8443 –centralUsername admin –centralPassword password
```

# Inspecting a repository

The RepoInspector utility is a tool that inspects a repository. RepoInspector.jar can be found in the SDK home directory.

When an inspection fails, a warning message is issued that contains valuable information including:

- The UUID and path of the repository
- A description of the problem
- Any impacts to users
- Suggested remedies

This information enables content developers of all levels, both OO developers and third-party content developers, to understand, diagnose, and fix problems in their content.

RepoInspector.jar does the following:

- *Perform best practices checks* and reports violations.
- *Perform version compatibility checks* and detects compatibility violations with previously-released snapshots of the same repository
- *Generate release notes*.
- *List repository contents*.

## Checking best practices

RepoInspector checks for violations of best practices, and writes violations to a file. It performs the following best practice checks:

**Deprecated Usage**

Checks for flows that use deprecated operations—any operations in a folder named /Deprecated. Deprecated flows are not subject to this check.

**Description Exists**

Verifies that all objects have descriptions.

**PRE Tags in Description**

Checks that the text on Studio Description tabs are surrounded with <pre> and </pre> tags. These tags are required to make flow descriptions appear correctly in Central.

**Consistent Input Bindings**
Checks for problems in input bindings, such as usernames (or passwords or hostnames) bound to specific values, unencrypted passwords, and empty prompts.

**Obsolete References**
Flags the terms "JRAS" or "NRAS" in descriptions. Only the term "RAS" should be used.

**Description Consistency**
Checks that operation descriptions document all inputs, responses, and results. This check is required because of a deficiency in the IAction interface used to communicate between infrastructure and content.

**Response Type Consistency**
Verifies that response types (**success**, **diagnosed**, **failure**/**failed**) are consistent with their Java enumeration values (**Type.RESOLVED**, **Type.DIAGNOSED**, **Type.ERROR**).

**Questionable Deprecation**
Finds operations that appear to have been deprecated for no reason—the new version of the operation has no new required inputs and does not lack any outputs that the original operation had.

### Syntax

```
Java –jar RepoInspector.jar -repo1 <path to repo1> -bestpractices [-outputFolder
<path>][-includePaths <repo path>][-excludePaths <repo path>][-exemptionFile <path>]
```

### Options

`-repo1 <repository URL>`

The value of `<repository URL>` can be:

- Strings in the following form:
  `<user>:<password>:<encPassword>@host:port`
- Folder paths to local repositories. The path has to be relative to where RepoInspector is run.
- The path to a content installer.
- The path to a local .zip file.
- The URL to a remote .zip file.

`-bestpractices`

Specifies to run a best practices check for the repository.

`-outputFolder`

Specifies the folder where the output of the Best Practice Violations results files.

`-includePaths`

Specifies the repository paths for best practice checking. The repository paths should be in a quoted, comma-delimited list. The default is `includePaths: /Library,/Configuration`.

`-excludePaths`

Specifies which repository paths should be excluded for best practice checking. The repository paths should be in a quoted, comma-delimited list. The default is `excludePaths: /Repository,/Configuration/Groups, /Configuration/System Properties, /Configuration/System Accounts, /Configuration/System Filters`

`-exemptionFile`

The path of the properties file of exemptions to tests. The format of the properties file is:

```
UUID|Library Path = <comma separated list of tests>
```

The following is an example exemption property file:

```
! /Library/..., done because...
```

```
905742c8-2412-4476-a7d3-49d0acfa4b40 = InputTest

! Done because...
/Library/Integrations/My Integration/Add Stuff =
DeprecatedUsageTest,InputBindingTest
```

**Example**

```
Java –jar RepoInspector.jar –repo1 ../ExampleIntegrationRepo –bestPractices
```

## Checking version compatibility

RepoInspector can verify compatibility between a new repository and old (released) repository. The purpose is to identify any changes to content that may break customer's flows. RepoInspector performs the following version compatibility checks:

**Existence**
Every object (UUID) in the old repository must be present in the new repository. If this test fails, all further compatibility checking is skipped.

**Inputs**
Every required input in the new repository must be required in the old repository. Compatibility is broken if you introduce a new required input or change an existing input to be required.

**Version**
If an object in the new repository is present in the old one, then it must be based on the latest version of the old one. In other words, if the new repository object is based on an object in the old repository that has been subsequently changed, then it is flagged as a violation. Conversely, if the embedded version numbers of the object have not changed from the old to the new repository yet the objects really have undergone some changes (normally only possible by manually manipulating the repository XML object outside of Studio), then it is also flagged.

**Syntax**

```
Java –jar RepoInspector.jar -repo1 <path to repo1> -repo2 <path to repo2>  -
compatibility [-outputFolder <path>]  [-includePaths <repo path>] [-excludePaths
<repo path>] [-exemptionFile <path>]
```

**Options**

`-repo1 <repository URL>`

The value of `<repository URL>` can be:

- Strings in the following form:
  `<user>:<password>:<encPassword>@host:port`
- Folder paths to local repositories. The path has to be relative to where RepoInspector is run.
- The path to a content installer.
- The path to a local .zip file.
- The URL to a remote .zip file.

`-repo2 <repository URL>`

The value of `<repository URL>` can be:

- Strings in the following form:
  `<user>:<password>:<encPassword>@host:port`
- Folder paths to local repositories. It has to be relative to where RepoInspector is run.
- The path to a content installer.
- The path to a local .zip file.
- The URL to a remote .zip file.

`-compatibility`

Specifies to run a compatibility check for the repository.

`-outputFolder`

The path to the Compatibility folder that contains the results files.

`-includePaths`

Specifies which repository paths to include for the compatibility checking. The repository paths should be in a quoted, comma-delimited list. The default is `includePaths: /Library,/Configuration`.

`-excludePaths`

Specifies which repository paths to exclude for the compatibility checking. The repository paths should be in a quoted, comma-delimited list. The default is `excludePaths: /Repository,/Configuration/Groups, /Configuration/System Properties, /Configuration/System Accounts, /Configuration/System Filters`

`-exemptionFile`

The path of the properties file for exemptions to tests.

**Example**

```
Java –jar RepoInspector.jar –repo1 ../ExampleIntegrationRepo –repo2
../AnotherExampleIntegraionRepo -compatibility
```

## Generating release notes

RepoInspector can generate release notes for specified repositories.

**Syntax**

```
Java –jar RepoInspector.jar -repo1 <path to repo1>  -releasenotes [-outputFolder
<path>]  [-includePaths <repo path>] [-excludePaths <repo path>]
```

**Options**

`-repo1 <repository URL>`

The value of `<repository URL>` can be:

- Strings in the following form: `<user>:<password>:<encPassword>@host:port`
- Folder paths to local repositories. The path has to be relative to where RepoInspector is run.
- The path to a content installer.
- The path to a local .zip file.
- The URL to a remote .zip file.

`-releasenotes`

Specifies to generate release notes for the repository.

`-outputFolder`

The path to the ReleaseNotes folder that contains the results files.

`-includePaths`

Specifies which repository paths to include for the release notes generation. The repository paths should be in a quoted, comma-delimited list. The default is `includePaths: /Library,/Configuration`.

`-excludePaths`

Specifies which repository paths to exclude for the release notes generation. The repository paths should be in a quoted, comma-delimited list. The default is `excludePaths: /Repository,/Configuration/Groups,/Configuration/System Properties,/Configuration/System Accounts,/Configuration/System Filters`.

**Example**

```
Java –jar RepoInspector.jar –repo1 ../ExampleIntegrationRepo  –releasenotes
```

### Listing repository contents

RepoInspector can list the contents of specified repositories.

**Syntax**

```
Java –jar RepoInspector.jar -repo1 <path to repo1>  -listing [-outputFolder <path>] [-includePaths <repo path>] [-excludePaths <repo path>]
```

**Options**

`-repo1 <repository URL>`

The value of `<repository URL>` can be:

- Strings in the following form: `<user>:<password>:<encPassword>@host:port`
- Folder paths to local repositories. The path has to be relative to where RepoInspector is run.
- The path to a content installer.
- The path to a local .zip file.
- The URL to a remote .zip file.

`-listing`

Specifies to generate a listing of the contents of the repository.

`-outputFolder`

The path to the Listing folder that contains the results files.

`-includePaths`

Specifies which repository paths to include for the repository listing. The repository paths should be in a quoted, comma-delimited list. The default is `includePaths: /Library,/Configuration`.

`-excludePaths`

Specifies which repository paths to exclude for the repository listing. The repository paths should be in a quoted, comma-delimited list. The default is `excludePaths: /Repository,/Configuration/Groups,/Configuration/System Properties,/Configuration/System Accounts,/Configuration/System Filters`.

**Example**

```
Java –jar RepoInspector.jar –repo1 ../ExampleIntegrationRepo  –listing
```

## Automating flow testing

The AutoTest utility is an automated OO content tester with stress testing capabilities. AutoTest.jar is found in the SDK home directory. The tool invokes the flows in OO Central, so the Central service must be available.

**Syntax**

```
java [system properties] -jar AutoTest.jar [parameters] <xmlFilePath>
```

**Example**

```
Java –Dlogfile=output.log –jar AutoTest.jar –dbpass iconclude input.xml
```

## System properties

Use the following syntax to specify system properties for the AutoTest utility:

```
–D<property>=<value>
```

For example:

```
–Dlogfile=output.log
```

`Logfile`

The path to output log file.

`stress.conn.timeout`

The connection timeout in seconds.

`stress.read.timeout`

The read timeout in seconds.

`-threaddelay`

The delay before each thread begins. The default is **0**.

`-threaddelayevery`

Apply ‘threaddelay’ everything n-th trhead to run. The default is **1**.

`-threads`

The maximum number of concurrent threads. The default is **1**.

`-varxml`

The XML file of “variable” overrides.

`-xmlfile`

The XML file of “central” and “variable” overrides.

## Parameters

Use the following syntax to specify parameters for the AutoTest utility:

```
–name <value>
```

For example:

```
–dbpass iconclude
```

`-dbhost`

The OO Central database host. The default is the same host as OO central.

`-dbname`

The OO Central database name. The default is **dharma**.

`-dbpass`

The OO Central database password.

`-dbport`

The OO Central database port.

`-dbtype`

The OO Central database type. The valid values are **oracle**, **mysql**, and **mssql**.

`-dbuser`

The OO Central database user.

`-helpxml`

Displays help information in the format of an XML file.

`-host`

The OO Central host.

`-https`

Specifies whether to use SSL. The valid values are **true** and **false**. The default is **true**.

`-user`

The OO Central username.

`-pass`

The OO Central password.

`-port`

The OO Central port.

`-quiet`

Specifies minimal logging to the console.

`-runcount`

The number of times to repeat each **runnable** element. The default is **1**.

**Note:** Most of the above options can be specified in an XML input file. Values that you specify in a command line take precedence over values contained in XML files, with the exception of `-xmlfile FILENAME`, which loads an entire XML file of overrides.

## Sample XML input files

The following is the example of an all-in-one XML configuration file:

```
<?xml version="1.0"?>
<stress-run>
    <!-- Main config block -->
    <central>
        <!-- Central config -->
        <host>myhost.battleground.ad</host>
        <user>admin</user>
        <pass>secret</pass>
        <https>true</https>
        <port>8443</port>

        <!-- Database config -->
        <dbhost>myhost.battleground.ad</dbhost>
        <dbname>dharma</dbname>
        <dbuser>dharma_user</dbuser>
        <dbpass>secret</dbpass>
        <dbtype>mssql</dbtype>
```

```
        <!-- Stress tool config -->
        <runcount>1</runcount>
        <threads>5</threads>
    </central>

    <!-- Definitions -->
    <variables>
        <myhost>myhost.battleground.ad</myhost>
        <myip>192.168.5.55</myip>
    </variables>

    <!-- Flows -->
    <flow>
        <description>Connectivity Step1 Test</description>
        <name>/Library/Accelerator Packs/Network/ConnectivityTest</name>
        <input name="host">${myhost}</input>
        <input name="lossThreshold">5</input>
        <input name="latencyThreshold">300</input>
        <response>success</response>
        <rule>
            <key>FailureMessage</key>
            <value></value>
            <comparator>contains</comparator>
        </rule>
    </flow>
    <flow>
        <description>JRAS Test 01</description>
        <name>/Library/Stress/JRAScommand</name>
        <input name="hostname">${myhost}</input>
        <response>success</response>
    </flow>

    <!-- Operations -->
    <operation>
        <description></description>
        <name>/Library/Operation/Some Operation</name>
        <input name="myhost" />
        <response>success</response>
    </operation>
</stress-run>
```

The following is an example of a separate Central\variables sample XML configuration file.

Main input XML file:

```
<?xml version="1.0"?>
<anything>
    <central>
        <runcount>1</runcount>
        <threads>1</threads>
        <xmlfile>opsforceenv.xml</xmlfile>
    </central>
    <variables>
        <xmlfile>Input Variables.xml</xmlfile>
    </variables>

    <!--
********************************************************************************
->
    <!--  Operation Name: Library/Integrations/Hewlett-Packard/Universal CMDB/Add
Object -->
```

```
    <!--
********************************************************************************** -
->

    <!-- DL1 -->
    <flow>
        <description>Regression Test 01</description>
        <name>/Library/Havok_Regression/Integrations/Hewlett-
Packard/Universal_CMDB/OP_Add_Object_HVK01</name>
        <input name="cmdbHost">${cmdbHostName}</input>
        <input name="cmdbPort">${cmdbPortValue}</input>
        <input name="username">${cmdbUsername}</input>
        <input name="password">${cmdbPassword}</input>
        <input name="objectType">${AddcmdbObjecttype}</input>
        <input name="prop">${AddcmdbProp}</input>
        <input name="cmdbVersion">${cmdbVersion}</input>
        <response>success</response>
    </flow>

    <!-- DL2a -->
    <flow>
        <description>Regression Test 02</description>
        <name>/Library/Havok_Regression/Integrations/Hewlett-
Packard/Universal_CMDB/OP_Add_Object_HVK01</name>
        <input name="cmdbHost">${badcmdbHostName}</input>
        <input name="cmdbPort">${cmdbPortValue}</input>
        <input name="username">${cmdbUsername}</input>
        <input name="password">${cmdbPassword}</input>
        <input name="objectType">${cmdbObjecttype}</input>
        <input name="prop">${cmdbProp}</input>
        <input name="cmdbVersion">${cmdbVersion}</input>
        <response>Failure</response>
    </flow>

    ...
    ...
</stress-run>


Central Xml file:

<?xml version="1.0"?>
<anything>
    <central>
        <host>16.93.12.44</host>
        <port>8443</port>
        <user>admin</user>
        <pass>admin</pass>
        <https>true</https>

        <dbhost>16.93.12.44</dbhost>
        <dbname>oo</dbname>
        <dbuser>admin</dbuser>
        <dbpass>admin</dbpass>
        <dbtype>mssql</dbtype>
    </central>
</anything>

Variables Xml file;

<?xml version="1.0"?>
```

```
<anything>
    <variables>
        <!-- ****************************************** -->
        <!--               uCMDB Variables              -->
        <!-- ****************************************** -->
        <cmdbHostName>15.23.143.2</cmdbHostName>
        <cmdbPortValue>8080</cmdbPortValue>
        <cmdbUsername>admin</cmdbUsername>
        <cmdbPassword>admin</cmdbPassword>
        <cmdbVersion>8</cmdbVersion>

        <!-- /Library/Havok_Regression_Ros/uCMDB_ros/Add_Object_HVK01 -->
        <AddcmdbObjecttype>unix</AddcmdbObjecttype>
        <AddcmdbProp>host_key=Test</AddcmdbProp>

        ...
        ...
    </variables>
</anything>
```

# Debugging OO client/server problems

Communication between OO components is accomplished using SSL (Secure Sockets Layer), which encrypts data that is transmitted between clients and servers through the Internet. When a client/server problem occurs with OO—such as a call to the Central Web Service not working correctly—SSL does not allow you to capture the data packets transmitted between the client and the server to validate that data is being sent properly.

The solution is to enable HTTP access, which allows you to capture live data packets, and inspect or compare them. This is usually the best way to debug OO client/server problems.

Following are two procedures that you can use for debugging:

- The first procedure allows HTTP access to Central.
- The second procedure does the same thing for RAS.

### To allow HTTP access to Central

1. Stop the RSCentral service.
2. In a text editor, open the file applicationContext.xml, which is located in the Central\WEB-INF\ folder of the OO home directory.
3. Comment out every line in any sections that begin with `<!-- HTTPS_SECTION_BEGIN -->` and end with `<-- HTTPS_SECTION_END -->`, and then save the file.
4. Open the file web.xml, which is located in the Central\WEB-INF\ folder of the OO home directory.
5. Comment out every line in any sections that begin with `<!-- HTTPS_SECTION_BEGIN -->` and end with `<-- HTTPS_SECTION_END -->`, and then save the file.
6. Restart the RSCentral service.
7. Connect to port 8080 using HTTP rather than port 8443 using HTTPS.

### To allow HTTP access to RAS

1. Stop the RSJRAS service.
2. In a text editor, open the file jetty.xml, which is located in the RAS\Java\Default\webapp\conf\ folder of the OO home directory.
3. Comment out the line:
   `<New class="org.mortbay.jetty.security.SslSelectChannelConnector">`
4. Add the following line directly under the line you commented out in the previous step:

```
<New class="org.mortbay.jetty.nio.SelectChannelConnector">
```

5. Comment out the lines starting with:
   - `<Set name="Keystore">`
   - `<Set name="Password">`
   - `<Set name="KeyPassword">`
   - `<Set name="NeedClientAuth">`
6. Save the file.
7. Open the file applicationContext.xml, which is located in the RAS\Java\Default\webapp\WEB-INF\ folder of the OO home directory.
8. Comment out every line in any sections that begin with `<!-- HTTPS_SECTION_BEGIN -->` and end with `<-- HTTPS_SECTION_END -->`, and then save the file.
9. Open the file web.xml, which is located in the RAS\Java\Default\webapp\WEB-INF\ folder in the OO home directory.
10. Comment out every line in any sections that begin with `<!-- HTTPS_SECTION_BEGIN -->` and end with `<-- HTTPS_SECTION_END -->`, and then save the file.
11. Restart the RSJRAS service.
12. Connect to port 9004 using HTTP instead of HTTPS.

To turn off the allowance of HTTP connections for either procedure, just reverse the procedure.

# Index