HP Operations Orchestration Software Central

Software Version: 7.10

Guide to authoring iActions

Document Release Date: March 2008

Software Release Date: March 2008

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

All marks mentioned in this document are the property of their respective owners.

Documentation Updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
  - The number before the period identifies the major release number.
  - The first number after the period identifies the minor release number.
  - The second number after the period represents the minor-minor release number.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software."

You will also receive updated or new editions if you subscribe to the appropriate product support service. If you have additional questions, contact your HP Sales Representative.

Updating documentation

Documentation enhancements are a continual project at Hewlett-Packard Software. You can update the documentation set at any time using the following procedure (which is also available in the HP OO readme file).

To obtain HP OO documentation

1. On The Web site *https://support1.opsware.com/support/index.php*, log in with account name and password that you received when you purchased HP OO.
2. On the **Support** tab, click the **Product Docs** subtab.
3. Under **Quick Jump**, click **Operations Orchestration** (or **Process Automation System**).
4. Under **Operations Orchestration**, click **ZIP** beside **HP OO 7.10 Full Documentation Set**.
5. Extract the files in the .zip file to the appropriate locations on your system:
   - For the tutorials to run, you must store the .swf file and the .html file in the same directory.
   - To obtain the repository that reflects the state of the flow at the start of the tutorial, unzip the file Exportof<preceding_tutorial_name>.zip.
   - To obtain the scriptlet for the tutorial that includes using scriptlets, click the scriptlet .txt file name.
   - To update your Central or Studio Help:
     a. Under **Help Files**, click **Studio Help File Bundle** or **Central Help File Bundle**.
     b. In the **File Download** box appears, click either **Open** or **Save**.
     c. Extract the files to the Hewlett-Packard Software\HP OO home directory, in either the **\Central\docs\help\Central** or **\Studio\docs\help\Studio** subdirectory, overwriting the existing files.

Where to Find Help, Tutorials, and More

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 iConclude folder, and Accelerator Packs

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

Support

For support information, including patches, troubleshooting aids, support contract management, product manuals and more, visit one of the two following sites:

- *https://support1.opsware.com/support/index.php*
- http://www.hp.com/go/hpsoftware/DCA_support

This guide

This Guide is intended for customers, Opsware Systems Engineers (SEs), and Customer Engineers (CEs) who are looking to extend HP OO by building their own RAS Actions (operations). This document assumes that you are creating iActions for use with HP OO 7.10 or later.

What is a RAS?

Inside of HP OO there are several different types of operations that HP OO can execute. There are the built-in operations, such as SSH and TELNET, which you can use with the Create Operation command in Studio.

The list of operations you can create includes Web operation. A Web operation is an operation in HP OO that executed by code within a Remote Action Service (RAS). A RAS server is installed by default on the Central server when you install HP OO, but Web operations can also be executed by a standalone RAS installation on machines that are remote from the Central server.

The HP OO Central server initiates all communications between it and the RAS, and the protocol used for such communications is HTTPS. As a result, a Web operation's code can be executed by a RAS on the other side of a firewall or domain boundary from the Central server (see figure 2 for complete HP OO architecture).

Currently the RAS supports the following platform/language combinations:

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

What is an iAction?

An iAction is an operation whose class implements the iAction interface. Operations that are hosted on a RAS are considered to be iActions because, for an operation to be accessible to the RAS for execution, the operation must be able to implement the iAction interface.

iAction Interface

The iAction interface specifies how to build Action classes. They must implement the iAction interface. The iAction interface states that the following two public methods must be defined:

- Public ActionResult execute(ActionRequest req, Session session, ActionRegistry ar);
- Public ActionTemplate getActionTemplate();

(note: in the JRAS they are a Session and ActionRegistry object, in the NRAS it is ISession and iActionRegistry)

ActionTemplate

The ActionTemplate is an object that describes to HP OO the following about an operation:

- What the operation does
- The inputs it requires
- The responses it returns
- Any results that are available for use in a flow.

While Java and C#/.NET implementations of the ActionTemplate object differ, the objects are functionally the same.

An ActionTemplate object has the following properties that can be sent back to Central:

- A description

  This should be modeled after those operation descriptions you see in the shipping tree. It should include a description of what the operation does, the inputs that it expects and what the developer meant by them, the responses that are returned, and any other results that it will return. By doing this it will help self-document your operation to authors that may use your operation.

- A map representing the inputs

  This map has the key set to the name of the input. The value may be left as a blank string, or a RASBinding object can be used instead. A RASBinding object has properties to correspond to most of the options that are available in the inputs inspector in Studio. The order in which inputs are added to the map are the order in which they will appear in the operation once imported.

- A map representing the responses

  This map has the key value set to the text that the response transition will show, the value is the integer that is returned by the operations “returnCode” to tell Central what transition to follow now that the operation has completed.

- A map representing any results that are returned

  This map has the key set to the name of the result; the value needs to be left as a blank string.

RASBinding

RASBinding objects can be tied with inputs in an ActionTemplate. RASBindings allow the RAS operation author to define exactly how they want the input to be defined once it is imported into HP OO. You can also use the RASBindingFactory to quickly create RASBindings with default style behaviors. RASBindings have the following properties that can be defined:

- encrypted (Boolean, false)

  Sets whether the particular input should be encrypted.

- required (Boolean, false)

  Sets whether the particular input should be required.

- assignTo (Boolean, true)

Sets whether the **Assign to flow variable** checkbox is checked.

- assignFrom (Boolean, true)
  Sets whether the **Assign from flow variable** checkbox is checked.
- assignFromText (String, empty)
  Sets the text in the **Assign from flow variable** field.
- assignToText (String, empty)
  Sets the text in the **Assign to flow variable** field.
- type (INPUT_TYPE, INPUT_TYPE.Empty)
  Specifies that the type of input it should be one of the following:
  - Empty
    An empty binding that will become a prompt if not changed
  - Static
    A static binding, by entering something in the value property, that value will be the value of this input
  - Prompt
    A prompt-user binding, by entering something in the value property, that value will be the text that is prompted to the user
- value (String, empty)
  The value that is assigned to either the static field or the prompt user field depending on the type specified (see above)

ActionResult

The ActionResult object passes results to Central after your code has been executed. There are a few things that you can return back to Central.

- exception (String, empty)
  This should preferably be the stack trace for any exception that your operation encountered. This is most often used inside of the try/catch block for the operation. It should be left blank if no exception was encountered.
- returnCode (int, 0)
  This should be set to the code that represents the transition that Central should follow now that the operation has completed. This should only be set to values that can be mapped to the responses in the ActionTemplate for this operation

The ActionResult objects (both Java and .NET) extend the Map object. Therefore, any other results that you have defined in the ActionTemplate can also be sent back. The key field being the result name and the value field being some string value.

ActionResultEx/ActionTemplateEx

These objects are only available with C#/.NET actions. These are simple wrappers around the ActionResult and ActionTemplate objects. These simple wrappers allow for using enums for responses and also populate a result called returnResult for you based on the SetReturnCode method.

Style Guide for authoring iActions

- Create any static constants you need.
- Define your ActionTemplate.

  The ActionTemplate comprises inputs, outputs, and responses. If an input, output, or response is not available in Studio, it is an ActionTemplate error.
- Add an execute method, and ensure it uses a try/catch block.
- Resolve any inputs you need, and run the code to accomplish your task.
- Set the results.

  If an exception is thrown, you still need to ensure every result has a value set, even if that value is just an empty string.
- Set the return code.
- The response (return) code for success is always 0, and failure is always 1.
- When possible, give a meaningful error message for operation failure.
- Only inputs, responses, and results advertised in the ActionTemplate are created automatically (for more information, see in this guide *ActionTemplate*).
- Make sure to handle System Accounts properly (for more information, see in this guide *.NET iActions*).

Java iActions

Java iActions are Java classes that do the work that the programmer has built into them. HP OO can import and use these operations.

Required development files

- JRAS-sdk.jar
- JRAS-client.jar
- ContentCommons.jar

Things to remember

- Check inputs to ensure they are non-null

  If you use the com.opsware.pas.content.commons.util.StringUtils.resolveString method, then null inputs are automatically converted into empty strings.

  Optional inputs can be strings of length 0.
- It is recommended that you use the StringUtils.resolveString method, because it handles System Account.
- Users have access only to results that are advertised in the Action Template.
- Do not treat/use instance variables in the iAction itself if they are unique to a given run of the iAction (and never write to an instance variable). If you need

instance variables, create a non-iAction wrapper that does the work, and which the execute method calls into. Note, however, that you will rarely need to use this approach.

- Code in one iAction jar cannot use code in another iAction jar.

Loading your iAction into HP OO

Importing your custom operations into Studio

1. Create a jar with your iAction classes in it.
2. Stop the RSJRAS service.
3. Copy your jar to %iconclude_home%\RAS\Java\Default\repository.
4. Copy any added libraries to %iconclude_home%\RAS\Java\Default\repository\lib.
5. Start the RSJRAS service.
6. In Studio, create and select an empty folder.
7. On the **File** menu, choose **Create operations from RAS**.
8. Select RAS_OPERATOR_PATH (or the reference to the RAS that you put the jar into).

Debugging your iAction

Enabling remote RAS debugging for Java Actions

1. Stop the RSJRAS service.
2. Copy your jar to %iconclude_home%\RAS\Java\Default\repository.
3. Copy any added libraries to %iconclude_home%\RAS\Java\Default\repository\lib.
4. Open the %iconclude_home%\RAS\Java\Default\webapp\conf\wrapper.conf file.
5. Uncomment the debug line (#wrapper.java.additional.2=-Xdebug -Xnoagent -Djava.compiler=NONE -Xrunjdwp:transport=dt_socket,address=8070,server=y,suspend=y)
6. Change the debug line from wrapper.java.additional.2 to wrapper.java.additional.3 (or your last one +1).
7. Start the RSJRAS service.
8. Connect your debugger to the port specified in the wrapper.conf file.

   The port is 8070 by default, but that can be changed to any unused port.
9. Set a breakpoint in your Java source code where you want 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.

Disable remote RAS debugging for Java Actions

1. Stop the RSJRAS service.
2. Open the %iconclude_home%\RAS\Java\Default\webapp\conf\wrapper.conf file.
3. Comment out the debug line (#wrapper.java.additional.2=-Xdebug -Xnoagent -Djava.compiler=NONE -Xrunjdwp:transport=dt_socket,address=8070,server=y,suspend=y)

4. Change the debug line from wrapper.java.additional.2 to wrapper.java.additional.3 (or your last one +1).
5. Start the RSJRAS service.

Sample iAction Code (Hello World)

```
import com.opsware.pas.content.commons.util.*;
import com.iconclude.webservices.extensions.java.interfaces.*;
import com.iconclude.webservices.extensions.java.types.*;

public class SampleAction implements iAction {

   private static String RETURNRESULT = "returnResult";
   private static final String RESPONSEPASSED = “success”;
   private static final String RESPONSEFAILED = “failure”;
   private static final int PASSED = 0;
   private static final int FAILED = 1;
   private static final String INPUT_STRING = "text";

   public ActionTemplate getActionTemplate() {

      ActionTemplate actionTemplate = new ActionTemplate();

      actionTemplate.setDescription(“Returns Hello + object”);

      Map parameters = new Map();
      parameters.add(INPUT_STRING,
         RASBindingFactory.createEmptyRASBinding(false, false));
      actionTemplate.setParameters(parameters);

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

      Map responses = new Map();
      responses.add(RESPONSEPASSED, String.valueOf(PASSED));
      responses.add(RESPONSEFAILED, String.valueOf(FAILED));
```

```
      actionTemplate.setResponses(responses);

      return actionTemplate;
   }

   public ActionResult execute(ISessionContext sessionContext,
      ActionRequest actionRequest, iActionRegistry actionRegistry)
      throws Exception {

      ActionResult result = new ActionResult();

      try {
         String object =
            StringUtils.resolveString(actionRequest,
            INPUT_STRING);
         if (object.length()==0) {
            throw new Exception("Hello Oblivion. Please specify
            an object if you want this operation to succeed.");
         }

         result.add(RETURNRESULT, "Hello "+object);
         result.setReturnCode(PASSED);
      }
      catch (Throwable e) {
          result.setException(StringUtils.toString(e));
          result.add(RETURNRESULT, e.getMessage());
          result.setReturnCode(FAILED);
      }

      return result;
   }
}
```

.NET iActions

.NET iActions are .NET assemblies that do the work that the programmer has built into them. HP OO can import and use these operations.

Required development files

- iAction.dll
- RCAgentLib.dll
- ContentCommons.dll

Things to remember

- Use the Convert.ToString() method for handling all inputs other than system accounts.
- Handle system accounts with Identify Methods in ContentCommons.
- Users have access only to results that are advertised in the Action Template.
- Do not treat/use instance variables in the iAction itself if they are unique to a given run of the iAction (and never write to an instance variable). If you need instance variables, create a non-iAction wrapper that does the work, and which the execute method calls into. Note, however, that you will rarely need to use this approach.

Loading your iAction into HP OO

Importing your custom operations into Studio

1. Create one or more DLLs that contain your iAction classes.
2. Stop the RSJRAS service.
3. Copy your DLL(s) to %iconclude_home%\RAS\Java\Default\repository.
4. Copy any added DLL libraries to the same directory.
5. Start the RSJRAS service.
6. On the **File** menu, choose **Create operations from RAS**.
7. Select RAS_OPERATOR_PATH (or the reference to the RAS that you put the jar into).

Debugging your iAction

Enabling remote RAS debugging for .NET Actions

1. Stop the RSJRAS service.
2. Copy your DLLs and PDBs to %iconclude_home%\RAS\Java\Default\repository.
3. Copy any added libraries to the same directory.
4. Start the RSJRAS service.

5. Connect your debugger to the java.exe process that hosts the RSJRAS service.
6. Set a breakpoint in your .NET source code where you want it to stop.
7. Execute a flow that exercises your operation.

Disable remote RAS debugging for .NET Actions

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

Sample iAction Code (Hello World)

```
using System;
using com.iconclude.agent;
using dotNET_Commons;

namespace SimpleActions {
public class SimpleAction : iAction {
   const String INPUT_OBJECT = "Object";

   public ActionResult Execute(ActionRequest req, ISession s,
      iActionRegistry reg) {

      ActionResultEx ret = new ActionResultEx();

      try
      {
         string strObject =
            Convert.ToString(req.parameters[INPUT_OBJECT]));

         if (strObject.Length==0)
            throw new Exception("Hello Oblivion. Please specify an
               object if you want this operation to succeed.");

         ret.SetReturnCode(ReturnCodes.PASSED, "Hello " + strObject);
      }
      catch (Exception e)
      {
         ret.SetReturnCode(ReturnCodes.FAILED, e.Message);
```

```
            ret.SetException(e.ToString());
        }

        return ret.GetActionResult();
    }

    public ActionTemplate GetActionTemplate()
    {
        ActionTemplateEx template = new ActionTemplateEx();
        template.SetDescription(GetType().Name);

        template.AddParameter(INPUT_OBJECT,
            RASBindingFactory.createEmptyRASBinding(false, false));

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

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