Building, debugging and releasing

After specifying the steps and establishing the flow for a new or modified service template, you need to access the Debug tab to generate a build. Once a build is successfully completed, you can use the built-in debugger to run through the steps and correct any problems with the execution flow or property mapping. When the service template is functioning properly, you can release it into the operating environment where it can then be used to create services.

The following sections provide the steps to debug and release service templates and plug-ins.

**Debug and release workflow**

After creating or modifying a service template, you can build and then debug the service according to the following phases:

**Phase 1 - Preparing**

- Prior to building the service template, make sure that the service template details have been specified, that the appropriate plug-in steps are arranged in the order of their execution, and that values for the input and output properties have been properly defined.

**Phase 2 - Building**

- Initiate the building of the service template by clicking on the Debug tab.
- Observe the information returned by the Build/Release dialog to see if any errors are generated.
- If there are errors, hover the cursor over the error message to see details pertaining to the error.
- Make whatever corrections are necessary and continue to rebuild the service template until it completes building successfully.

**Phase 3 - Debugging**

- Once a service template has been successfully built, you can perform any necessary debugging.
- Correct any errors and continue re-building the service template until all errors are resolved.
- After all issues have been resolved, conduct an operation test by adding and executing the service in the development environment.

**Phase 4 - Releasing**

- When the service template is functioning properly and there are no further issues, you can release the service template. A service template must be in Released status in order to submit the service template to the operating environment.
- Create a service from the released service template.
- Check the execution results of the task. If any issues are identified, amend the affected service template or plug-in...
and repeat the debug process.

The following figure shows the typical steps you follow when debugging, testing and releasing a service template:

**Building a service template**

After a service template is created and is in the Developing status, the next step is to build and debug the service template and its related plug-ins.

Here are the steps for initiating the building and debugging of a service template:

**Before you begin**
A service template in the Developing state must exist.

**Procedure**

1. From the Service Builder Edit screen, click the Debug tab. After confirming your intention to perform the debug, the Build/Release Results dialog appears and the service template undergoes the build process. If any errors are generated, error icons and error link buttons are displayed and warning icons appear when you have failed to enter a required property.

2. Make the necessary corrections where indications appear or adjustments to the operation as needed.

3. Repeat the build process until the service template is free of errors.

**Results**

Once you build a service template, the following processing occurs:

- Services added from the debug version of the service template will be deleted, and tasks executed from the service will be archived.
- The debug task executed from the debug version of the service template will be deleted.
- The debug version of the imported service template will be deleted and then re-imported.

Upon successful completion of the build process, the Perform Debugging dialog appears where you can check the flow of execution and make any adjustments that are necessary before releasing the service template.

**Next steps**

- Access the debugger to check the flow of steps and to make any required modifications.
- Create services and tasks based on the debug configuration of the service template. For more information on creating services, see the *Hitachi Command Suite Automation Director User Guide*.
- Provided the service template works properly and has passed the build process, proceed to release the service template. For more information on releasing a service template, see [Releasing a service template](https://knowledge.hitachivantara.com/Documents/Management_Software/Automation_Director/8.5.0/Use_Service_Builder/Cr…).
Once you have successfully built a service template, you can run the built-in debugger by clicking OK from the Perform Debugging dialog box. Before accessing the debug interface, you need to specify the appropriate details for the debug service and tasks from the Perform Debugging dialog box. You can also examine and edit the service properties or make changes from the Create Service or Create Request windows.

You can use the built-in debugger to make sure the plug-ins and flow of a service template are working as intended. During a debugging session, you can:

• Control the execution of steps in the flow of a service template to isolate and correct any issues.
• Execute and manage debug tasks while checking the flow transitions at all hierarchical levels.
• Confirm that property mapping is set correctly and that the conditions for executing subsequent steps flow in the intended way.
• Modify input and output properties for currently running tasks.
• Set breakpoints to have the processing start from or end before a specified step in the flow.
• Skip the processing of specified plug-ins (script/command, repeat, wait for user response) so that it appears to the next step as though the process of the plug-in has executed, thus allowing the flow to continue according to the execution condition of the next step.
• Displays the results of a repeated-execution flow (for each execution time).
• Consult a running log of executing tasks.
• Edit entries in the config and submit windows to simulate processing of the service template.
• If you detect an issue with a plug-in, you can assign an arbitrary property value or return value to the plug-in and execute it again. This allows you to see the effect a given property value or return value has on the plug-in processing and flow transitions.

You can debug a service template any number of times.

When you debug a service template, HAD creates a debug service and debug task.

**Debug service**

A debug service is a service that is generated and executed when debugging a service template. One debug service is generated per service template. When you debug a service template that has already been through a debug process, HAD deletes the existing debug service and creates a new one. Note that debug services appear in the Service Name column in the Debug-Tasks view, but do not appear in the Services window.

**Debug task**

A debug task is a task generated for a debug service when debugging a service template. When you debug a service template that has already been through a debug process, HAD deletes the existing debug task and creates a new one. Debug tasks appear in the service template Debugging view and Debug-Tasks view. Only users assigned the Admin or Develop role can view and work with debug tasks. Note that debug tasks do not appear in the task summary.

Note: You cannot edit the definition of a service template or plug-in from the debugger view. Instead, when a defect is detected during debugging, you need to stop debugging and return to the Editor window to make the necessary correction.

**Limitations on concurrent debugging**
operations and associated tasks

A maximum of one debug service and one debug task can be generated for the same service template, and the same service template cannot be debugged by multiple users at the same time. The same service template cannot be edited by multiple users at the same time (because the last operation to save the template takes precedence, and thus multiple problems cannot be fixed in parallel).

If a debug service and a debug task already exist for a service template that is being debugged, when you execute the build or release operation, the debug service and debug task are deleted automatically, and a new debug service and debug task are created when debugging re-starts.

After the debug task finishes, if you perform debugging again without first closing the Debug view, the created debug service and the finished debug task are deleted automatically and a new debug service and debug task are generated.

Before you begin

Follow these steps to initiate a debugging session:

• A Developing version of the service template to be debugged must have successfully completed the build process.

Procedure

1. Enter the appropriate details from the Perform Debugging dialog.

2. For the Service Name field, accept the original name appended with [DEBUG] or specify some other name of your choosing.

3. Examine, and if necessary, modify the relevant details for the other service and task related fields

4. For the Task Log Level, specify the level of details that are to be stored in the task log file.

5. If necessary, you can supply the values that would normally be specified by the user who is configuring and submitting the service by clicking on the Edit pull-down menu and either selecting From Create Service Window or From Create Request Window.

6. Click OK to access the debugger user interface.

Results

The debug interface is accessed where you can begin the debugging process.

Editing service and request entries while debugging

Besides specifying the service and task details for the debugging from the Perform Debugging dialog box, you can also examine and edit the service properties to supply the config and submit values from the Create Service or Create Request windows.
The properties section shows a listing of service properties implemented for the service template. It shows the Display Name, Key Name, Value, Description and Scope.

Before proceeding to the debugger interface, you often need to first specify the values that would normally be supplied by the Modify and Submit users when running a service. The following messages indicate that you need to supply values for the specified properties before continuing with the debugging session:

**Information is not complete. Please edit properties from [Create Service Window]**

**Information is not complete. Please edit properties form [Create Request Window]**

To edit the properties from either the Create Service or Create Request windows, follow these steps:

1. From the Properties section of the Perform Debugging dialog, click Edit.
2. Choose From Create Service Window or From Create Request Window depending on which interface you want to edit.
3. Click on the category of settings (marked in red and with a warning (!) icon) and enter the missing values for all fields that are marked with a red asterisk (*), until the section turns blue indicating that all required fields have been filled in. Then click OK.
4. After specifying the service and task details for the debug session, and supplying any of the required values for the properties, click OK to access the debugger interface where you can begin debugging the service template.

While editing the service properties, you can also click on the Import button to restore properties from a previous debugging session or the Export button to save the properties from the current session. In some cases, you might want to use this import capability to save time by loading all of the property values at one time, or possibly to restore property values that you previously exported. You can also click Restore Default to restore all default values.

**Working with the debugger**

After successfully building a service template in preparation for its release, and supplying the required service and task details from the Perform Debugging dialog box, you can use the Debugger interface to debug the service template.

The Service Builder Debug screen has the following operational panels:

- **Debugger** -- This area allows you to control debug operations associated with the currently selected step.
- **Flow** -- Shows the placement and flow of steps associated with the service template.
- **Flow Tree** -- Shows the hierarchical flow of steps.
- **Task Log** -- Selecting this tab shows a listing of task entries. You can refresh the listing to see the current state of executed tasks or download the listing to a file for subsequent reference.
- **Service Properties** -- Selecting this tab shows the Display Name of the service properties associated with the service template along with its assigned Key Name and Value.
- **Break Points** -- Selecting this tab shows any currently set breakpoints along with their Flow Hierarchy and Display Name. You can remove all currently set breakpoints by clicking the Remove All Break Points.
The debugger provides a convenient method of controlling the execution of tasks for a debug operation. The following options are provided:

- **Input Response** -- Requests an input response for a waiting task.
- **(Retry Debug)** -- Retries the debugging as follows:
  - Retry Debug -- Retries the debugging process from the beginning of the service template.
  - Retry the Task Starting from the Failed Step -- Retries the task beginning from the last failed step.
  - Retry the Task Starting from the Step after the Failed Step -- Retries the task just after the last failed step.
- **(Resume Debug Operation)** -- Resumes the debugging operation from the last failed step.
- **(Interrupt Debug Operation)** -- Interrupts the debug operation at the current task.
- **(Forcibly Stop Debug Operation)** -- Forcibly stops the debug operation at the current task.
- **(Execution of Step Into)** -- Runs the task to the next interruptible step.
- **(Execution of Step Over)** -- Runs the task to the first interruptible point in the next step.
- **(Execution of Step Return)** -- Runs the task to the first interruptible point in the upper hierarchy.
- **(Set/Unset Breakpoint)** -- Sets (or removes) a breakpoint that causes the execution tasks to pause after the specified step.

**Note:** A system generated breakpoint is set automatically by the system when choosing the step-into, step-over, or step-return option to control the execution of tasks during a debug session.

**Debug Modes:**

- Run plugin in execution mode -- Runs the plug-in in execution mode.
- Run plugin in dry-run mode -- Runs the plug-in in dry-run mode.

**Task Status:**

The status for a debug task can be one of the following:

- **Waiting** -- Indicates that the task is waiting for execution.
• Interrupted -- Indicates that the execution of the task has been interrupted by the step execution feature.

• In Progress -- Indicates that the debug task is currently in progress.

• Waiting for Input -- Indicates that the task is waiting for user input.

• In Progress (with Error) -- Indicates that the task has detected a processing error.

• In Progress (Terminating) -- Indicates that the task has received a stop operation or forcibly stop operation instruction, and is terminating the processing.

• Completed -- Indicates that the execution of the task has completed successfully.

• Failed -- Indicates that the execution of the debug task has failed.

Step Information

Step information provides details for a given step as follows:

• ID -- Shows the name of the currently selected step.
• Display Name -- Shows the display name of the step.
• Status -- Shows the status of the debug operation for the currently selected step.

Step Properties

The Step Properties view shows the input and output properties associated with a given step as well as any variables that may be used along with the current values.

The following icons indicate the type of property or variable:

• Indicates an Output property

• Indicates an Input property

• Indicates a Variable

You can edit the property values directly by clicking on the pencil icon that accesses the Edit Step Property dialog box where you can make any necessary changes.
While examining the step property values, you can click on the Edit button and select either **From Create Service Window** or **From Create Request Window** to specify the values for the properties normally supplied by the Modify and Submit users.

You can also click on the Import or Export buttons to save or retrieve property values from a specified file.

The Task Log, Service Properties, and Break Point tabs provide additionally useful information about the current debugging session. Please see: Examining debug details for complete details.

### Examining debug details

While debugging a service template, you can examine details regarding tasks that have been executed, current service properties, and breakpoints that have been set.

**During a debug session, you can access relevant details from the following tabs:**

#### Task Log

By clicking the Task Log tab, you can access a listing of tasks that have been executed during the current debugging session. To update the task listing with the latest task activity, click Refresh or you can check the Refresh Automatically check box to have the task listing automatically updated.

If you would like to keep a record of the executed tasks for a debug session, click Download and then specify the location for the log file. You specify the level of detail for a log file when starting the debugger.

#### Service Properties

By clicking the Service Properties tab, you can view all of the service properties associated with the service template. This is useful for checking the mapping between the input and output properties and variables associated with a given step and the service properties for a service template. The following fields are provided:

- **Display Name** -- Display name assigned to the service property.
- **Key Name** -- Key name assigned to the service property.
- **Value** -- Value currently assigned to the selected service property.

#### Break Points

From the Break Points tab, you can see all of the breakpoints that have been set for the current debugging session. The following details are shown:

- **Flow Hierarchy** -- Shows the hierarchy of flow usually indicated by a backslash (/).
- **Display Name** -- Shows the display name assigned to the step.

You can set a breakpoint to have the execution flow halted at a particular step, by selecting the step and then clicking on the breakpoint icon control.
When no longer needed, remove all currently set breakpoints by clicking Remove All Break Points.

Managing tasks during debugging

You can control the execution and flow of tasks during the debugging process.

For more information on the options available from the Debugger view, see the following topics:

Controlling the processing flow of debug tasks

When debugging a service template, you can control how the flow of tasks are executed.
The following describes the general procedure for debugging a service template:

Before you begin

A service template in the Developing state must exist with steps added to the Flow view.

Procedure

1. If you do not expect any issues when executing the plug-ins in the service template, the first step of the debug process is to execute the debug task without pausing between steps. In the Debug view or the service template debugging view, make sure that there are no issues with the flow transitions or the processing of the plug-ins. If the service template contains plug-ins that you would prefer not to execute at this time, skip this step and move on directly to step 2.

2. If you identify an issue with a flow transition or the processing of a plug-in, execute the steps in the debug task individually to identify the precise location and nature of the problem. You can also test the behavior of the plug-ins by assigning unexpected values to input and output properties.
   - Click on the Step Into icon to execute the currently selected step.
   - Click on the Step Over icon to execute the next step.
   - Click on the Step Return icon to execute the step in the upper flow after completing the execution of steps in the current flow.
   - Click the debug arrow, and choose Retry the Task Starting from the Failed Step to have the debugging process start again from the failed step. By retrying from a failed step, you can resume the debug task with the same task ID and the original property values. You can use this approach when the cause of the failure has been resolved. For example, a step that fails due to a temporary problem with the network can be retried when the network connection is available again.
   - Click the debug arrow, and choose Retry the Task Starting from the Step after the Failed Step to have the debugging process start again from the step after the failed step. By retrying from the step after the failed step, you can resume the debug task with the same task ID and the original property values. This approach is appropriate in situations where there is no need to execute the failed step. When you retry a task from the step after the failed step, processing of the task continues as if the failed step had ended normally. You can use this approach when you encounter an issue in a step, but want to continue executing the debug task and deal with the issue later.
Handling interruptions of debug tasks

Under certain circumstances, the execution of tasks may be interrupted or terminated in unexpected ways during a debugging session. The steps in the debug task appear in the Flow view of the service template debugging view, in the order in which they are executed. The icon of the step indicates the status of the step. You should be aware of how tasks are affected when the debugging process is interrupted.

Handling of debug tasks when the HAD server stops

If the HAD server stops during debugging, the debug task that is running is forcibly stopped. Therefore, before stopping the HAD server, wait for all debug tasks that are still running (not yet completed) to finish or stop the debug tasks. This is the same as the handling of tasks generated when normal services are executed.

Handling of debug tasks when a failover of the cluster occurs

When a cluster failover (switchover of the system) occurs during debugging, the debug task is forcibly stopped. This is the same as the handling of tasks generated when normal services are executed.

Handling of debug tasks when the user logs out during debugging

When performing an explicit operation for logging out during debugging, a confirmation dialog box is displayed asking for confirmation before logging out. If you choose to log out, the debug task is forcibly stopped.

Handling of debug tasks when the browser is closed

When the browser is closed during debugging, the debug task continues running as-is. If the task is being executed step-by-step, the task remains at the step where it stopped. To stop the debug task, you need to log in again and then stop the debug task from the Debug Task List view.

Controlling the display of tasks in the Task List

While debugging tasks, you can specify how the listing of tasks are managed in the Task List. You can specify the period of time to retain tasks before they are archived and no longer shown in the Task List. Once tasks have been archived (and become history entries), the detailed information about the tasks (input/output parameters and status of AJS in the units of jobs) is deleted, and therefore cannot be returned to the Tasks List view again.

You can have the archival of tasks performed automatically by specifying the appropriate value through one of the
The following properties:

<table>
<thead>
<tr>
<th>Property key</th>
<th>Description</th>
<th>Minimum value</th>
<th>Maximum value</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>tasklist.autoarchive.taskRemainingPeriod</td>
<td>Specifies the period, expressed in days, to retain completed tasks in the task list. When the specified period has passed, the tasks are automatically archived. The automatic archiving of tasks takes place once a day according to the time specified through the tasklist.autoarchive.executeTime property.</td>
<td>1</td>
<td>90</td>
<td>7</td>
</tr>
<tr>
<td>tasklist.debugger.autodelete.taskRemainingPeriod</td>
<td>Specifies the period, expressed in days, before debug tasks are automatically deleted.</td>
<td>1</td>
<td>90</td>
<td>7</td>
</tr>
<tr>
<td>tasklist.autoarchive.maxTasks</td>
<td>Specifies the maximum combined number of tasks and debug tasks to retain in the Tasks List. When the maximum number of tasks in the Tasks List is exceeded, the excess tasks are automatically archived, starting from those with the oldest end date and time. The archived tasks are managed as history entries. Debug tasks are deleted automatically and are not retained in history. Automatic archiving and automatic deletion take place once a day at the time specified by the tasklist.autoarchive.executeTime property. When there are more tasks than the specified value, attempting to execute a new service results in a &quot;maximum exceeded&quot; error, and no task is generated. Periodic tasks that have</td>
<td>100</td>
<td>5000</td>
<td>5000</td>
</tr>
</tbody>
</table>
been executed once are not subject to this limit and can generate new tasks. Therefore, to allow new services to be executed, you must estimate the number of executions that take place per day in order to specify the tasklist.autoarchive.taskRemainingPeriod property.

### Checking the property mapping of a plug-in

When debugging a service template, you can control how the flow of tasks are executed. The following describes the general procedure for checking the property mapping of a plug-in.

**Before you begin**

A service template in the Developing state must exist with steps added to the Flow view.

**Procedure**

1. In the Flow view, select the step whose plug-in property values you want to check. The Debug view displays the input properties and output properties of the step you selected.

2. Click the Service Properties tab at the bottom of the service template debugging view. The values of the service properties are displayed.

3. In the Debugger view, review the contents of the Step Properties for the plug-in property you want to check, and identify the service property to which it is mapped.

4. In the Key Name column of the Service Properties tab, find the service property you identified in step 3.

5. In the Debugger view and the Service Properties view, make sure that the same value appears in the Value columns for the plug-in property and the mapped service property. If a service property is not mapped to the intended plug-in property or the values of the plug-in property and the service property differ, fix the problem in the service template editing view by clicking on the pencil icon and then supplying the correct value through the Edit Step Property dialog box. You can also change the values of the plug-in properties. By doing so, you can test the plug-in processing when property mapping is configured correctly to see how the processing of subsequent steps and the flow transitions change with an assortment of values.
Importing property values

While debugging a service template, you can import property values to a specified property file.

Procedure

1. From the debugger interface, click the Import button. The Import dialog box appears requesting that you specify the name of the file where the property values are to be stored.

2. Enter the name of the property file, or use the browser to search for the specified file then click OK. The property values are imported from the specified file.

When the importing has completed, notifications about the property values that have and have not been imported are temporarily displayed as follows:

- Properties to which the values are applied.
- Properties for which the values are not applied. These are properties for which the attribute values cannot be changed and for which the values are not applied due to the property value definitions.
- Non-existent properties. These are properties that are defined in the file but do not exist in the target service.

When importing a properties file from the debugger, (or the Config and Submit windows), the JSON or key=value format is supported.

When accessing the import capability from the CLI, the additional format key@FILE=file-path is also supported.

The conditions that must be met to apply property values during an import are shown in the following table.

<table>
<thead>
<tr>
<th>Property Group Attribute</th>
<th>Property Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>hidden</td>
<td>paramMode</td>
</tr>
<tr>
<td>True</td>
<td>in</td>
</tr>
<tr>
<td>False</td>
<td>config</td>
</tr>
<tr>
<td>False</td>
<td>exec</td>
</tr>
<tr>
<td>False</td>
<td>false</td>
</tr>
<tr>
<td>False</td>
<td>--</td>
</tr>
<tr>
<td>False</td>
<td>--</td>
</tr>
</tbody>
</table>

If the properties do not meet these conditions, or their are no corresponding properties defined in the service, the values contained in the definition file are not applied. The values are also not applied if the "value field" is not defined or is set to null.

Note: If the length of keyName exceeds the limit, the property is classified as a property that does not exist in the service.

If an error occurs during an import, the error dialog box is displayed, and the import is canceled thus leaving all property values unchanged. An error occurs when the specified file does not exist or the property file definitions.
Exporting property values

While debugging a service template, you can export property files.

While debugging, you can export property values to a property file. This allows you to save multiple property values in a file for subsequent reference.

Exporting property values

Follow these steps to export property values to a specified file.

1. From the debugger interface, click the Export button.
2. Access the browser to locate the property file or specifically enter its name and click OK.

The property values are exported to the specified file. The property values are exported in the Json format and, by default, are saved to the service_properties.json file.

Format

In the property file specified as a command argument, the property key and value used by the executed services can be defined in JSON, key=value, and key@FILE=value formats.

JSON format

```json
{
    "properties": [
        {
            "keyName": "property-key",
            "displayName": "property-display-name",
            "description": "description-of-property",
            "type": "property-type",
            "value": "property-value"
        },
        {
            "keyName": "property-key",
            "displayName": "property-display-name",
            "description": "description-of-property",
            "type": "property-type",
            "value": "property-value"
        },
        ...
    ]
}
```

Following are definition details for the JSON format:

- The displayName, description and type fields are optional.
• When you specify the value field, set an empty value for the property value.
• The value for the password type property can be in plain text or encrypted. Note that the "value" field of the password type property is not exported for security reasons. The defined value is imported as is, and the REST API determines whether it is in plain text or encrypted.
• In the definition file, define only properties for which you want to set values. The values of properties that are not defined in the imported file remain unchanged. When exporting step properties, the type field is only output for the service component.

key=value format:

To specify property values for a key=value property file, use the following format:

```
property-key=property-value [line break]
```

Following are definition details of the key=value format:

• Specify a property name and a property value on each line.
• Lines starting with a hash mark (#) are handled as comment lines.
• Lines that do not contain an equal mark "=" are handled as comment lines.
• A line break needs to be added at the end of each property setting line.
• Do not add line breaks in the middle of the property name and property value lines.
• Characters are case-sensitive.
• Even when a "\" is contained in strings like service and plug-in resource files, you do not need to type "\\".
• "\" is handled as a "\".
• The characters at the beginning of a line up to the first equal sign (=) are treated as a property name. Do not trim lines before and after the property setting line.
• The characters after the equal sign (=) after the property name, until the end of the line are treated as the property value. Do not trim lines before and after the property setting line.
• The end-of-line character at the end of the property file (EOF) is optional.
• Empty lines (lines containing line breaks only) are ignored.
• Both CR+LF and LF can be used as line breaks.
• When using the property-key = [line break] format, set an empty value for the property value.

key@FILE=file-path format

In this format, the property key is stored in the property file, and the property value is stored in the property value file and referenced separately. Note that this format can be used together with the key=value format in the property file:

```
property-key@FILE=absolute-path-of-the-property-value-file or relative-path-from-the-property-file [line break]
```

The definitions in the key@FILE=file-path format are the same as the key=value format. The differences from the key=value format are as follows:

• The absolute path of the property value file or the relative path from the property file must always be specified or an error occurs.
• If the specified file does not contain a property value, an error occurs.
• The property value file can contain end-of-line characters. However, if a property value file that contains an end-of-line character is specified for a property for which end-of-line characters must not be set, an error occurs.

Releasing a service template

The final step to complete a service template is the release process. The release operation changes the configuration type of a service template and its related plug-ins to Released. New services and tasks can be created from a service template in Released status. In Released status, the service template appears in HAD as an available service template from which services can be created.

Note
The release operation can be performed only once.

Before you begin
A service template in the Developing state that has completed the build process with no errors and performs as designed.

Procedure

1. From the Service Builder screen, click Release.
   The service template undergoes the release process and its configuration type changes to Released status.

Results
Upon successful completion of the release process, the service template is available under the Services tab with the Released configuration type.

• The service template is removed from the Developing state and appears under the Released tab when viewing available service templates.
• Any services created while the template was in Developing status are deleted.
• Any tasks executed from the template while in Developing status are archived.
• The service template appears in the Service Template list when creating a new service.
• The related plug-ins appear under the Released tab of the Component view.
• If specified in the service definitions, a service component is created and located under the Service tab of the Component view.

Next steps

• Create a new service using the released service template. For more information on creating services, see the Hitachi Command Suite Automation Director User Guide.