HP Operations Manager

Custom Process Management

Software Version: 9.10

for the UNIX and Linux operating systems

Manufacturing Part Number: None

March, 2013

Legal Notices

Conventions

The following typographical conventions are used in this manual:

Table 1 **Typographical Conventions**

Font	Meaning	Example
Italic	Book titles and manual page names	For more information, see the *HPOM Administrator's Reference* and the *opc(1m)* manual page.
	Emphasis	You *must* follow these steps.
	Variable that you must supply when entering a command (in angle brackets)	At the prompt, enter **`rlogin`** ***`<username>`***.
	Parameters to a function	The *`oper_name`* parameter returns an integer response.
`Computer`	Text and other items on the computer screen	The following system message displays: `Are you sure you want to remove current group?`
	Command names	Use the `grep` command...
	Function names	Use the `opc_connect()` function to connect...
	File and directory names	Edit the `itooprc` file... `/opt/OV/bin/OpC/`
	Process names	Check to see if `opcmona` is running.
`Computer Bold`	Text that you enter	At the prompt, enter **`ls -l`**.

Table 1 Typographical Conventions (Continued)

Font	Meaning	Example
Keycap	Keyboard keys	Press **Return**.
	Menu name followed by a colon (:) means that you select the menu, and then the item. When the item is followed by an arrow (->), a cascading menu follows.	From the menu bar, select **Actions: Filtering -> All Active Messages**.
	Buttons in the user interface	Click **OK**.

In This Document

This document describes how to manage custom processes by adding them to a list of managed components and registering them with the HPOM control daemon. The HPOM process control component (`ovcd`) controls all HPOM management server processes and ensures they are started and stopped in the order that is defined by the `Dependency` element[1]. For details, see the description of `Dependency` on page 12.

Each server process is registered with the process control component with one XML registration file. The default registration files can be found at the following locations:

- ❏ *For server components:*

  `/etc/opt/OV/share/ovc`

- ❏ *For agent components:*

  `/opt/OV/misc/eaagt`

All HPOM processes are automatically registered with the HPOM control daemon, `ovcd`, and can be controlled by using the `ovc` command with the `-start`, `-stop`, and `-status` options respectively. Each HPOM server process has its own XML registration file that defines how the HPOM processes are handled.

The configuration files that define the registration process for each process are stored at the following location on the management server and the managed node:

`<`*`OvDataDir`*`>/conf/ctrl`

On the management server, the `ctrl` directory contains registration files for all HPOM processes—both management server processes and managed node processes if the management server is also configured as a managed node. On the managed node, the directory contains registration files only for the managed node processes.

1. The order in which MSI applications receive messages is not defined by this order.

The information in this document covers the following topics:

- "Sample XML Registration File" on page 10
- "Examples of XML Registration File Configuration" on page 17
- "Adding a Custom Component to HPOM Control" on page 17

For detailed information about HPOM processes, see the *HPOM Administrator's Reference*.

1 Custom Process Management

Custom Process Management

HPOM enables you to manage custom processes by adding them to a list of managed components and registering them with the HPOM control daemon, `ovcd`. In this way, additional custom processes can be managed in the same way as any other HPOM process.

To add a custom component to HPOM control, create an XML registration file for this component. You can use the `opccustproc1.xml` sample file that is provided with HPOM as a template for your XML registration file.

Sample XML Registration File

The `opccustproc1.xml` sample file that is provided with HPOM as a template for your XML registration file can be found at the following location:

`/opt/OV/contrib/OpC/opccustproc`

The syntax of this file is the following (with the default values):

```
<?xml version='1.0' encoding='UTF-8' standalone='yes'?>
<ovc:OvCtrl
xmlns:ovc="http://openview.hp.com/xmlns/ctrl/registration/1.5">

  <ovc:Component>

    <ovc:Name>ComponentName</ovc:Name>

    <ovc:Label>

      <ovc:String>ComponentLabel</ovc:String>

    </ovc:Label>

    <ovc:Category>Category</ovc:Category>

    <ovc:Options>

      <ovc:AllowAttach>false</ovc:AllowAttach>

      <ovc:AutoRestart>false</ovc:AutoRestart>

      <ovc:AutoRestartLimit>5</ovc:AutoRestartLimit>

      <ovc:AutoRestartMinRuntime>60</ovc:AutoRestartMinRuntime>

      <ovc:AutoRestartDelay>5</ovc:AutoRestartDelay>

      <ovc:MentionInStatus>true</ovc:MentionInStatus>
      <ovc:Monitored>true</ovc:Monitored>
      <ovc:StartAtBootTime>true</ovc:StartAtBootTime>
      <ovc:CoreProcess>false</ovc:CoreProcess>
      <ovc:IsContainer>false</ovc:IsContainer>
      <ovc:AutoShutdown>false</ovc:AutoShutdown>
      <ovc:AutoShutdownTimer>1</ovc:AutoShutdownTimer>
      <ovc:PollingInterval>30</ovc:PollingInterval>
    </ovc:Options>
    <ovc:ProcessDescription>ProcessDescription</ovc:
    ProcessDescription>
    <ovc:CommandLine>CommandLine</ovc:CommandLine>
    <ovc:OnHook>
      <ovc:Name>OnHookName</ovc:Name>
      <ovc:Actions>Actions</ovc:Actions>
    </ovc:OnHook>
    <ovc:OnEvent>
      <ovc:Name>OnEventName</ovc:Name>
      <ovc:EventOptions>EventOptions</ovc:EventOptions>
      <ovc:Actions>Actions</ovc:Actions>
    </ovc:OnEvent>
</ovc:Component>
</ovc:OvCtrl>
```

In the context of HPOM control, a component is an entity that can be started, stopped, or notified (performing an action in response to an event). The component consists of the following elements:

Name (*required*): Each component has a unique name that is used to address it. The name is an ASCII identifier and it is not localized.

Label (*required*): Each component has a label that is used when printing the status of a component. For example, the opcle component would have a label “Logfile Encapsulator”. The label can be localized.

Description (*optional*): Each component can have a text description. The description can be localized.

`Dependency` (*optional*): Each component can have dependencies to other components. Dependencies are used when the `start` or `stop` command is issued. For example, when the logfile encapsulator has a dependency on the `opcapm` component, this component must be started before the logfile encapsulator. If a component is stopped, which is defined elsewhere as a dependency, the affected dependant is stopped first.

`Category` (*optional*): Each component can belong to none, one, or more categories. A category is a way to group components together to make operations easier (interfaces of HPOM control allow operations on components directly or on category grouping). The category is not localized.

`Options` (*optional*): Each component can have the following options:

`AllowAttach` (`TRUE/FALSE`): Instructs HPOM control not to kill the component if it is already started, but just to attach to it (meaning that the component does not have to be stopped first). The default is `FALSE`.

`AutoRestart` (`TRUE/FALSE`): Restarts the component if it terminates unexpectedly. The default is `FALSE`.

If you set this option to `TRUE`, you enable the following options:

- `AutoRestartLimit`: Specifies the maximum number of component automatic restarts. The default is 5.
- `AutoRestartMinRuntime`: Specifies how long in seconds the component must run before it can be restarted automatically. The default is 60.
- `AutoRestartDelay`: Specifies after how long in seconds the component is restarted automatically. The default is 5 seconds.

`MentionInStatus` (`TRUE/FALSE`): Specifies whether the status of the component is included in the status report. The default is `TRUE`.

`Monitored` (`TRUE/FALSE`): Specifies whether the errors are reported when a component terminates unexpectedly. The default is `TRUE`.

StartAtBootTime (TRUE/FALSE): Specifies whether the component should be started at a boot time. Evaluated only when -boot is specified. The default is TRUE.

CoreProcess (TRUE/FALSE): Specifies that the component should be stopped only if the -kill option is used. The default is FALSE.

IsContainer (TRUE/FALSE)[1]: Specifies whether the component should be treated as a container for other components. The default is FALSE.

AutoShutdown (TRUE/FALSE): Specifies whether the container should be stopped when all its contained components are stopped. The default is FALSE.

AutoShutdownTimer: Specifies after how long in seconds the container component is stopped when all of its contained components are stopped. The default is 30.

Container: Defines a name of the container for the contained component.

PollingInterval: Defines how often in seconds the container is polled to obtain the run status of the contained component. The default is 30.

WorkingDirectory: Specifies the working directory for the component (that is, the directory in which the component operates by default). Keep in mind, however, that depending on the internal operation of the component, the location of the actual working directory may differ from the specified one.

ProcessDescription (*required*): Each component has a process description that is used as follows: the process name, which is part of the operating system process table, is used to connect to the component in OvCtrl by comparing the string from the process table (that is, the process name) with the string value from the ProcessDescription element.

1. For HP internal use only. Leave the default value.

HP-UX PA-RISC only: Because of system limitation, the length of the process string is limited to 14 characters.

`CommandLine` (*optional*): Allows matching components against their command lines, not just against the process description. It makes it possible to distinguish between different Java virtual machines running on a system. Matching is done by using regular expressions.

`OnHook` (*optional*): A component is usually a process and has a lifetime. Within its lifetime, the component can be in the following states that are known to HPOM control: stopped, starting, initializing, running, or stopping. Several hooks allow the component to register actions that are executed and affect the state changes of a process. A hook is defined by `Name` (*required and predefined*) and `ActionType` (*required*).

The following hooks are available:

NOTE For very simple components, define only the `START` action.

`START_CHECK`: Allows defining a sequence of actions that must complete successfully before starting the component. Can be used to conditionally start the component.

`START`: Specifies the start sequence of the component.

`INITIALIZE`: Allows specifying additional actions that must complete successfully before the component is considered running.

`STOP`: Specifies the way in which the component is stopped. If it is not specified, HPOM control tries to stop the component in the way that is the default for the operating system.

IMPORTANT It is not recommended to use the Execute action in a STOP hook on Windows. On Windows, during the shutdown process, it is not allowed to start a new process and as a result the action will fail.

CHECK_STATUS: Specifies the status check sequence of a contained component.

OnEvent (*optional*): Specifies what must be done when an event is received. The event is defined by the following elements:

- Name (required): A name of the event to register for. It can be also an arbitrary string (the ":" is used to specialize the string into event:subevent).
- EventOptions (optional): Triggers additional processing when the event is received. Currently, the following two options are defined:

  ReevaluateStart: Starts the component if it is not running in the START_CHECK sequence of actions. This is necessary if the event potentially impacts the startup condition of the component.

  ReevaluateStop: Stops the component if it is running and the START_CHECK sequence of actions fails. This is necessary if the event potentially impacts the startup condition of the component.
- ActionType (required): Specifies one or more actions to be executed when an event is received. Different types of actions are supported (not all actions are relevant for each event).

The following predefined events are available:

- CHECK_POLICY: This event is sent when a policy is changed (added or modified).
- REMOVE_POLICY: This event is sent when a policy is removed.

- `FIRST_POLICY: <`*`policy_type`*`>`: The first policy of a given type is installed for the first time.
- `LAST_POLICY: <`*`policy_type`*`>`: The last policy of a given type is removed.
- `ENABLE_POLICY`: A policy is enabled.
- `DISABLE_POLICY`: A policy is disabled.

Example:

```
<ovc:OnEvent>
 <ovc:Name>DISABLE_POLICY:mgrconf</ovc:Name>
 <ovc:EventOptions>
  <ovc:ReevaluateStart>false</ovc:ReevaluateStart>
  <ovc:ReevaluateStop>false</ovc:ReevaluateStop>
 </ovc:EventOptions>
```

`ActionType` (*required*): You can specify more than one `ActionType` for `OnHook` and `OnEvent`. The actions are processed one after another and all must complete successfully for the event or hook to be considered successful. The following actions are available:

`Execute`: Runs a command and waits for it to complete. This action is meant for processes that daemonize themselves. You can also specify environment. On Windows, the `EXE` extension for the command is not required.

`Start`: Runs a command similarly to `Execute` except that it returns as soon as the process is spawned. Use it in actions to start processes that do not daemonize themselves. With the `Start` action, it is possible to put a process in the background. On Windows, the `EXE` extension for the command is not required.

`UXSignal` (Unix only): Sends a signal to the component. The signal name can be specified. Keep in mind that different operating systems use different signals. For details, see the relevant operating system documentation.

`WinEvent` (Windows only): Sends events on Windows. Windows does not have the signal mechanism but uses events for IPC.

Examples of XML Registration File Configuration

The following are examples of XML registration file configuration:

Example 1-1 **OnHook Component**

```
<ovc:OnHook>
   <ovc:Name>START</ovc:Name>
      <ovc:Actions>
         <ovc:Start>
            <ovc:CommandLine>START-CommandLine</ovc:CommandLine>
         </ovc:Start>
      </ovc:Actions>
</ovc:OnHook>
```

Example 1-2 **OnEvent Component**

```
<ovc:OnEvent>
   <ovc:Name>RECONFIGURE</ovc:Name>
   <ovc:EventOptions>
      <ovc:ReevaluateStart>false</ovc:ReevaluateStart>
      <ovc:ReevaluateStop>false</ovc:ReevaluateStop>
   </ovc:EventOptions>
   <ovc:Actions>
      <ovc:UXSignal>
         <ovc:Name>SIGUSR1</ovc:Name>
      </ovc:UXSignal>
   </ovc:Actions>
</ovc:OnEvent>
```

Adding a Custom Component to HPOM Control

To register a custom process with the HPOM control daemon, `ovcd`, follow these steps:

1. Create an XML registration file to register the custom process.

   You can use the `opccustproc1.xml` sample file that is provided with HPOM as a template for your XML registration file, as follows:

   a. Copy and rename the template configuration file `/opt/OV/contrib/OpC/opccustproc/opccustproc1.xml` according to your needs. For example:

      ```
      # cp /opt/OV/contrib/OpC/opccustproc/opccustproc1.xml
      /opt/OV/contrib/OpC/opccustproc/<my_process>.xml
      ```

Note that you must replace *<my_process>* with the name of the process you want to register.

b. Modify the following tags in the *<my_process>*`.xml` file according to your needs. For further help, see the example for the `opccustproc1.xml` file:

```
<ovc:Name>opccustproc1</ovc:Name>
<ovc:Label>
<ovc:String>OMU Custproc 1</ovc:String>
</ovc:Label>
<ovc:AllowAttach>false</ovc:AllowAttach>
<ovc:AutoRestart>true</ovc:AutoRestart>
<ovc:AutoRestartLimit>5</ovc:AutoRestartLimit>
<ovc:AutoRestartMinRuntime>60</ovc:AutoRestartMinRuntime>
<ovc:AutoRestartDelay>5</ovc:AutoRestartDelay>
<ovc:MentionInStatus>true</ovc:MentionInStatus>
<ovc:Monitored>true</ovc:Monitored>
<ovc:StartAtBootTime>false</ovc:StartAtBootTime>
<ovc:WorkingDirectory>/var/opt/OV/share/tmp/OpC/mgmt_sv</ov
c:WorkingDirectory>
<ovc:ProcessDescription>opccustproc1</ovc:ProcessDescriptio
n>
```

Under the `<ovc:Name>`**`START`**`</ovc:Name>` tag of the `OnHook` element, replace the `CommandLine` tag with the program that you want to start. For example:

```
<ovc:OnHook>
   <ovc:Name>START</ovc:Name>
      <ovc:Actions>
         <ovc:Start>
            <ovc:CommandLine>/opt/OV/bin/OpC/opccustproc1</
            ovc:CommandLine>
         </ovc:Start>
      </ovc:Actions>
</ovc:OnHook>
```

You can delete the `<ovc:Name>`**`START_CHECK`**`</ovc:Name>` tag of the `OnHook` element, along with its subtags:

```
<ovc:OnHook>
   <ovc:Name>START_CHECK</ovc:Name>
      <ovc:Actions>
         <ovc:Execute>
           <ovc:CommandLine>/opt/OV/bin/OpC/opcsv \
           -startable</ovc:CommandLine>
         </ovc:Execute>
```

```
            <ovc:Execute>
               <ovc:CommandLine>/opt/OV/bin/OpC/opcsv \
               -available opccustproc1</ovc:CommandLine>
            </ovc:Execute>
         </ovc:Actions>
      </ovc:OnHook>
```

2. Check the syntax of the new *<my_process>*`.xml` file by using the `ovcreg(1)` command with the `-check` parameter. Run the following command:

   ```
   # ovcreg -check \
   /opt/OV/contrib/OpC/opccustproc/<my_process>.xml
   ```

   The location of the `ovcreg` registration tool is the following:

   `<OvInstallDir>/bin`

3. Register the new *<my_process>*`.xml` file by using the `ovcreg` command with the `-add` parameter. Run the following command:

   ```
   # ovcreg -add \
   /opt/OV/contrib/OpC/opccustproc/<my_process>.xml
   ```

4. Start, stop, and check the status of the new custom process by using the `ovc` command with the `-start`, `-stop`, and `-status` parameters respectively.

   For example, to start the custom process, run the following command:

   ```
   # ovc -start <my_process>
   ```

5. To cancel the registration of the custom process, use the `ovcreg` command with the `-del(ete)` parameter.

   For example, run the following command:

   ```
   # ovcreg -del opccustproc1
   ```