Copyright Notice

Copyright © 2014 Wind River Systems, Inc.
All rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means without the prior written permission of Wind River Systems, Inc.

Wind River, Tornado, and VxWorks are registered trademarks of Wind River Systems, Inc. The Wind River logo is a trademark of Wind River Systems, Inc. Any third-party trademarks referenced are the property of their respective owners. For further information regarding Wind River trademarks, please see:
www.windriver.com/company/terms/trademark.html

This product may include software licensed to Wind River by third parties. Relevant notices (if any) are provided in your product installation at one of the following locations:
installDir/product_name/3rd_party_licensor_notice.pdf
installDir/legal-notices/

Wind River may refer to third-party documentation by listing publications or providing links to third-party Web sites for informational purposes. Wind River accepts no responsibility for the information provided in such third-party documentation.

Corporate Headquarters
Wind River
500 Wind River Way
Alameda, CA 94501-1153
U.S.A.
Toll free (U.S.A.): 800-545-WIND
Telephone: 510-748-4100
Facsimile: 510-749-2010

For additional contact information, see the Wind River Web site:
www.windriver.com

For information on how to contact Customer Support, see:
www.windriver.com/support

12 Dec 2014
## Contents

1 IDP EMS Overview ............................................................................................................... 5
   About the EMS Layer ........................................................................................................ 5
   Supported Boards .......................................................................................................... 7
   Supported Development Tools ...................................................................................... 7
   Where to Find Information ........................................................................................... 8
   Accessing Documentation ............................................................................................ 9

2 Configuring and Building the EMS Layer ..................................................................... 11
   Preparing to Build Your Platform Project .................................................................... 11
   About the wrenv.sh Script ........................................................................................... 12
   Building and Deploying Your Platform Project ............................................................ 12

3 Onboarding Devices ....................................................................................................... 15
   Terminology .................................................................................................................. 16
   About Onboarding ......................................................................................................... 16
   Installing Axeda Builder ............................................................................................... 17
   Creating an Agent Project ............................................................................................. 17
   Adding to the Agent Project ......................................................................................... 20
      Creating Data Items ..................................................................................................... 20
      Creating Data Loggers ............................................................................................... 21
      Defining Custom Data Sources ................................................................................... 21
   Preparing an Agent Project Tarball ............................................................................... 22
      Using a Windows Machine ......................................................................................... 22
      Using a Linux Machine ............................................................................................. 25
   Creating Automatic Onboarding Notifications ............................................................ 26
   Creating an Onboarding Package ................................................................................... 26
   Deploying an Onboarding Package ............................................................................... 28
      Deploying an Onboarding Package Manually ............................................................ 28
      Deploying an Onboarding Package Automatically .................................................... 29
   Confirming Successful Onboarding .............................................................................. 30

4 Updating the Components ........................................................................................... 33
   Updating the System ...................................................................................................... 33
      Installing, Updating, or Removing RPMs ................................................................ 34
      Monitoring RPM Updates .......................................................................................... 37
About the EMS Layer

The Wind River Edge Management System (EMS) for the Wind River Intelligent Device Platform XT (IDP XT) delivers a pre-integrated technology stack to support configuring, building, and deploying Wind River Cloud enabled systems to supported boards.

The Edge Management System consists of device-side and cloud-side components:

• On the device side, the rootfs for your BSP contains the IDP XT add-on and the Wind River EMS Layer (*wr-ems*), which includes one or more Event Driven Datasource (EDD) drivers and an EMS Agent.
• On the cloud side, the Edge Management System provides a web-based interface and an API.
Devices are securely connected to the Edge Management System by way of the firewall friendly EMS Agent. The EMS Agent forwards a data stream (such as alarms, telemetry, and events) from a device to the Edge Management System. Web-based applications can access the data stream through a RESTful API, enabling analytics, and monitoring. Administrators can manage, monitor, and update devices through the EMS Console.

**Included in the Wind River Edge Management System**

- Wind River Linux
- Wind River Workbench
- Wind River Intelligent Device Platform XT
- Wind River EMS Layer (**wr-ems**), which includes an EMS Agent, and the Windows installer for Axeda® Builder.
- Mashery API interface to the Edge Management System
- BSPs for the boards supported by the Wind River Intelligent Device Platform XT

**Before You Begin**

Get Wind River EMS Server hostname and login credentials from Wind River.
Supported Boards

The Wind River Edge Management System (EMS) delivers a pre integrated technology stack to support configuring, building, and deploying EMS enabled systems to supported boards.

The Wind River Edge Management System provides BSPs for the following boards.

- Cross Hill
- Clanton Hill
- Galileo
- Advantech UTX-3115 (Bay Trail)

Supported Development Tools

The Wind River Edge Management System (EMS) leverages the Wind River Intelligent Device Platform XT (IDP XT) and adds cloud-based management, monitoring, and support for analytics.

The supported development environments and tools are summarized in the chart below.

<table>
<thead>
<tr>
<th>Task</th>
<th>Environment and Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Building a file system, kernel image, and boot loader for your device</td>
<td>On your development host, you need:</td>
</tr>
<tr>
<td></td>
<td>Ubuntu 12.04 32/64-bit</td>
</tr>
<tr>
<td></td>
<td>Wind River Linux 5.0.1 for Intel IA</td>
</tr>
<tr>
<td></td>
<td>Wind River Workbench</td>
</tr>
<tr>
<td></td>
<td>Wind River Intelligent Device Platform XT 2.0.3</td>
</tr>
<tr>
<td></td>
<td>Wind River Linux EMS Layer 1.0, which includes Axeda Gateway Agent 6.8.0 and Axeda Agent SDK 6.8.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Onboarding your device</th>
<th>On a desktop computer, you need:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Windows 7 SP1 32/64-bit</td>
</tr>
<tr>
<td></td>
<td>Axeda® Builder v6.6.1</td>
</tr>
<tr>
<td></td>
<td>Login credentials on the EMS Console</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Managing devices</th>
<th>Login credentials on the EMS Console</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Login credentials on the EMS Console</td>
</tr>
</tbody>
</table>

For more information, see "Building and Deploying Your Platform Project" on page 12.

For more information, see "About Onboarding" on page 16.
Where to Find Information

The Wind River Edge Management System (EMS) provides documentation for Wind River EMS capabilities. It also utilizes Wind River Linux and Wind River Intelligent Device Platform XT (IDP XT) documentation and third-party hardware and software documentation.

The following documentation is available in the Wind River help system and in the Wind River Knowledge Library.

**Wind River Documentation**

*Wind River Intelligent Device Platform XT Programmer’s Guide*

Provides instructions for installing, configuring the Intelligent Device Platform and modifying it for your specific requirements (this document).

*Wind River Intelligent Device Platform XT Security Guide*

Provides guidance on performing a security analysis and matching IDP XT capabilities with assessed needs.

*Wind River Intelligent Device Platform XT Release Notes*

Provides general product information, changes in this release, usage caveats, and known problems.

*Wind River OPC for IDP Programmer’s Guide*

Provides guidance on using Wind River OPC with IDP XT.

*Wind River Linux Getting Started Guide, 5.0.1*

Provides instructions for creating, modifying, deploying, and debugging platform and application projects using the command-line and Workbench.

*Wind River Linux User’s Guide, 5.0.1*

Provides command-line instructions for configuring, building, and developing platform projects as well as detailed information on the development environment and build system.

*Wind River Workbench by Example Guide (Linux Version), 3.3*

Provides procedures and examples for using Workbench to configure, build, and debug Wind River Linux application, platform, and kernel module projects.

---

**NOTE:** This list represents the primary documents for developing an Intelligent Device Platform device system with an EMS layer, and is not complete. For the full set of documents that come with Wind River Linux, see the *Wind River Linux User’s Guide, 5.0.1*.

**Axeda Documentation**

*Axeda® Builder User’s Guide, 6.6.1*

provides instructions for installing Axeda® Builder and using it to create agent projects.
Axeda® Gateway User’s Guide, 6.8

provides instructions for creating a virtual representation of one or more physical devices, publishing data items from those devices, and remotely managing those devices.

Axeda v2 API/Services Developer’s Reference, 6.8

provides high-level overviews of v2 SDK features, along with detailed descriptions of each v2 bridge object.

Extending Axeda Agents: A Reference for Customizing the Agents, 6.8.0

provides instructions for creating custom components for an EMS Agent.

Axeda Agents: EDD Drivers Reference, 6.6.1

provides instructions for creating custom EDD drivers that form an interface between an EMS Agent and a device.

Using SSL with Axeda Platform Guide, 6.8

provides instructions for configuring secure communications between EMS Agent, EMS Server, and Agent Web service clients.

Axeda Platform Script Developer’s Guide

Accessing Documentation

You can access Wind River Edge Management System (EMS) documentation through the Wind River Knowledge Library.

- Access documentation through the Workbench main menu.
  
  Select Help > Help Contents > Wind River Documentation.

- Access documentation through the independent help browser.
  
  After you install Workbench, run the following command:

  \$ <installDir>/workbench-3.3/x86-linux2/bin/wrhelp.sh

- Access documentation through the file system in the installation directory (installDir).

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wind River PDF Versions</td>
<td>Point your PDF reader to the *.pdf file, for example:</td>
</tr>
<tr>
<td></td>
<td>installDir/docs/extensions/eclipse/plugins/</td>
</tr>
<tr>
<td></td>
<td>com.windriver.ide.doc.wr_ems_1.0/</td>
</tr>
<tr>
<td></td>
<td>wind_river_idp_xt_ems_profile_users_guide/</td>
</tr>
<tr>
<td></td>
<td>wind_river_intelligent_device_platform_xt_ems_profile_users_guide_10.pdf</td>
</tr>
<tr>
<td>Wind River HTML Versions</td>
<td>Point your Web browser to the index.html file, for example:</td>
</tr>
<tr>
<td></td>
<td>installDir/docs/extensions/eclipse/plugins/</td>
</tr>
<tr>
<td></td>
<td>com.windriver.ide.doc.wr_ems/</td>
</tr>
<tr>
<td></td>
<td>wind_river_idp_xt_ems_profile_users_guide/index.html</td>
</tr>
<tr>
<td>Options</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| Axeda PDF Versions | Point your PDF reader to the *.pdf file, for example:  

- Access documentation in the Wind River Knowledge Library.
Configuring and Building the EMS Layer

Preparing to Build Your Platform Project

Before building an image with an EMS layer, you must install Wind River Linux and IDP XT, obtain a supported board and peripherals, and connect the board to the development host.

Step 1 Obtain the required hardware and software.
- A Wind River IDP XT 2.0 installation with Wind River Linux 5.0.1 on a supported host.
- Any IDP XT 2.0 supported board:
  - Cross Hill
  - Clanton Hill
  - Galileo
  - Advantech UTX-3115
- Ability to create a flash image (Quark boards) or a boot medium with 8 GB capacity (Baytrail boards).
- A display device compatible with the display port on the target. For Quark boards, see your BSP documentation for information on the serial console 3.5MM audio to DB9 cable.
- A connection between the development host and the board.

Step 2 Install Wind River Linux 5.0.1 on your development host and update it with the latest RCPL version.

Step 3 Install Wind River Intelligent Device Platform XT 2.0 on your development host.
Step 4  Connect the board to the host.

Postrequisites

Get Wind River EMS Server hostname and login credentials from Wind River.

About the wrenv.sh Script

The `wrenv.sh` script sets all the Wind River Linux related environment variables including the path to the configure command for platform projects.

The command to run the script is:

```
$ installDir/wrenv.sh -p wrlinux-5
```

If you choose not to run `wrenv.sh`, you must explicitly specify the full path to the configure command instead of just specifying `$WIND_LINUX_CONFIGURE` in your configure line. The full path is similar to `installDir/wrlinux-5/wrlinux/configure`.

All the examples in this guide assume that you have executed `wrenv.sh`.

Building and Deploying Your Platform Project

The platform project for your board includes a file system, kernel image, boot loader, and BIOS image.

You build a platform project for your board on your development host. The file system you build contains the IDP XT add-on (`wr-idp`) and the EMS layer (`wr-ems`), which includes the EMS Agent.

Step 1  Set the Wind River Linux environment variables on your development host.

```
$ installDir/wrenv.sh -p wrlinux-5
```

Step 2  Save a copy of the device’s current working bootable image on a separate USB drive.

Booting the device with its current working image, and then updating its firmware with a new image, is a necessary step if:

- You are using a different development environment or keys than previously used.
- You are using the `deploy.sh` script to deploy an image to the board.
- You have a Cross Hill board.

In any of these scenarios, you will need to update the device’s firmware with the flash file from your new build, and to do this, you might need to be able to boot the device with its old image.
For example, to save the current bootable image of a Cross Hill board:

```bash
$ cd old_projDir/export
$ sudo ../deploy.sh -f intel-quark-glibc-idp-standard-dist-srm.tar.bz2 \
   -d /dev/your_device -b cross-hill -u -y
```

For more information about the `deploy.sh` script, see *About the deploy.sh Script* on page 63.

**Step 3** Create a new directory for your platform project.

```bash
$ mkdir -p installDir/workspace/projDir
```

**Step 4** Configure the platform project for your board.

**NOTE:** Add `--enable-addons=wr-idp, --with-layer=wr-ems,wr-ems-demo,wr-ima-appraise` and `--with-template=feature/online_updates` to the `$WIND_LINUX_CONFIGURE` command for your board.

For example, for a Cross Hill board:

```bash
$ cd projDir
$ $WIND_LINUX_CONFIGURE --enable-board=intel-quark \
   --enable-rootfs=glibc-idp --enable-kernel=standard \
   --enable-addons=wr-idp --with-layer=wr-ems,wr-ems-demo,wr-ima-appraise \
   --with-template=feature/online_updates \
   --enable-parallel-pkgbuilds=4 --enable-jobs=4
```

For more information on configuring a platform project, see the *Wind River Intelligent Device Platform XT Programmer’s Guide: Building and Booting*.

**Step 5** Configure your device’s connection to the EMS Server by running the `iot-config.py` script.

`iot-config.py` must be run from your project build directory (`projDir`) after `$WIND_LINUX_CONFIGURE` command, and before the `make fs` command.

For example:

```bash
$ cd projDir
$ layers/wr-idp/wr-ems/scripts/iot-config.py -s "\n"HOST:server_name.com,MODEL:MyModel,SERIAL_NUMBER:MySerialNumber"
```

For more information on the `iot-config.py` script, see *About the iot-config.py Script* on page 63.

**Step 6** Build the platform project for your board:

```bash
$ make fs
```

You now have a new platform project for your board. Your platform project includes a file system, kernel image, boot loader, and BIOS image.

**Step 7** Confirm that `projDir/export/dist/opt/agent/gateway` is populated.

**NOTE:** If there are no files in `projDir/export/dist/opt/agent/gateway`, re-run the `$WIND_LINUX_CONFIGURE` command with the appropriate options (see Step 4 on page 13) and rebuild your platform project.

**Step 8** Update your device’s firmware.
• For new deployments: update the board’s flash memory using the capsule update method in an EFI shell.
• For existing deployments that were based on a different development environment or keys than the new platform project: update the board’s flash memory by booting it with the image you saved in Step 2 on page 12 and then using the capsule update method in Linux; or by using the capsule update method in an EFI shell.

For more information about either capsule update method, see the Wind River Intelligent Device Platform XT Programmer’s Guide: Building and Booting.

Step 9 Deploy your platform project.

a) Create a bootable image on a USB drive or MMC card:

For example, to create a bootable image on a USB drive for a Cross Hill board:

```
$ cd projDir/export
$ sudo ../deploy.sh -f intel-quark-glibc-idp-standard-dist-srm.tar.bz2 -d /dev/your_device -b cross-hill -u -y
```

For more information, see About the deploy.sh Script on page 63 and the Wind River Intelligent Device Platform XT Programmer’s Guide: Building and Booting.

b) Connect your device to a display device compatible with its display port.

c) Connect your device to the network.

d) With the device powered off, insert the bootable USB drive or MMC card, and then power on the device.

The device boots and the login prompt appears.

Step 10 Confirm successful registration of your device on the EMS Server:

a) Log into the EMS Console.

b) Click the Assets tab.

c) Confirm that your device appears in the list of assets. You might need to filter by connection status or model.

If your device does not appear on EMS Server, check the setting for the EMS Server hostname (as defined in Step 5 on page 13). To fix incorrectly set hostnames, go back to Step 5 on page 13.

The device registers with the EMS Server automatically when it boots, and must now be onboarded. For more information about onboarding, see About Onboarding on page 16.

Related Links

About Onboarding on page 16

Onboarding configures the EMS Agent with basic device management content. You must onboard your device’s EMS Agent with a valid agent project containing the data items required for over-the-air updates.
Onboarding Devices

Terminology  16
About Onboarding  16
Installing Axeda Builder  17
Creating an Agent Project  17
Adding to the Agent Project  20
Preparing an Agent Project Tarball  22
Creating Automatic Onboarding Notifications  26
Creating an Onboarding Package  26
Deploying an Onboarding Package  28
Confirming Successful Onboarding  30
**Terminology**

After devices are onboarded onto the Edge Management System, they are referred to as assets.

**About Onboarding**

Onboarding configures the EMS Agent with basic device management content. You must onboard your device’s EMS Agent with a valid agent project containing the data items required for over-the-air updates.

When an EMS-enabled device boots, its EMS Agent automatically registers with the EMS Server. Newly registered devices must be onboarded with the appropriate onboarding package. You must create one onboarding package per device model on the EMS Console. The onboarding package must contain a valid agent project containing the data items required for software update configuration and status tracking.

The onboarding of new devices can be manual or automatic:

- For manual onboarding, you manually deploy the onboarding package to newly registered device(s).
- For automatic onboarding, you create an action that automatically deploys the onboarding package to any newly registered devices of a specific model, and then create an expression rule which will trigger the execution of your action when its condition is met.

**Onboarding Workflow**

1. Install Axeda® Builder. For more information, see *Installing Axeda Builder* on page 17.
2. Create an agent project using Axeda® Builder. For more information, see *Creating an Agent Project* on page 17.
3. Prepare the agent project for upload. For more information, see *Preparing an Agent Project Tarball* on page 22.
4. On the EMS Console, create an expression rule to notify you whenever a device is successfully onboarded. For more information, see *Creating Automatic Onboarding Notifications* on page 26.
5. On the EMS Console, create one onboarding package per device model. For more information, see *Creating an Onboarding Package* on page 26.
6. On the EMS Console, deploy the appropriate onboarding package to selected device(s). For more information, see *Deploying an Onboarding Package* on page 28.
7. Confirm that the onboarding of the selected device(s) was successful. For more information, see *Confirming Successful Onboarding* on page 30.
Installing Axeda Builder

The Axeda® Builder installer is located in the \texttt{wr-ems} layer on your development host.

\textbf{Step 1} Copy the Axeda® Builder installer from your development host to your Windows 7 machine.

The Axeda® Builder installer is located in \texttt{projDir/\ldots/windows}. If you do not see this directory, run the \texttt{\$WIND\_LINUX\_CONFIGURE} command with the appropriate options for your board. For more information, see \textit{Building and Deploying Your Platform Project} on page 12.

\textbf{Step 2} Unzip and run the Axeda® Builder installer on your Windows 7 machine.

Creating an Agent Project

You create an agent project to configure the EMS Agent using Axeda® Builder.

The Wind River EMS Layer on your development host provides a sample agent project that enables basic device management content, such as software update, agent restart, and factory reset. You can use the sample agent project as the basis for your own agent project by copying the contents of \texttt{projDir/\ldots/\ldots/contrib/builder_proj} from your development host to your Windows 7 machine, and opening the .gwsp file with Axeda® Builder.

For more information about Axeda® Builder, see the \textit{Axeda® Builder User’s Guide}, 6.6.1.

\textbf{Step 1} Start Axeda® Builder.

\textbf{Step 2} Create an agent project (skip this step if you are using an existing project).

a) Click File > New Project.

b) In the Project dialog, select New Gateway project, and click OK.

c) In the Gateway Project dialog, type a value with no spaces in the Name text box (hereafter referred to as \texttt{gateway\_project\_name}).

d) Choose a location to save the project in, and click Finish.

Your new project appears in the Project Window pane.

e) In the Project Window pane, expand your project, right-click Gateway Model, and select Rename Project.

f) In the Model Information dialog, type a name (without spaces) for this agent project’s subproject in the Model Number text box, and click OK.

The value you typed for Model Number (hereafter referred to as \texttt{model\_project\_name}) is now the name of this agent project’s subproject.

\textbf{Step 3} Specify settings for the project path.

a) In the Project Window pane, expand your project, and click Configuration.

b) In the Configuration pane, double-click Gateway Settings.
c) In the Gateway Settings dialog, type the subproject name in the **Project Path** text box, and click **OK**.

   The subproject name is *model_project_name*, in other words, the value you used for the **Model Number** text box in Step 2.f on page 17.

d) Click **OK**.

**Step 4** Specify settings for the EMS Server.
- a) In the Project Window pane, expand your project, and click **Configuration**.
- b) In the Configuration pane, double-click **Enterprise Servers**.
- c) In the Enterprise Servers Settings dialog, under **Servers**, click **Add**.
- d) In the Enterprise Server Definition dialog, type a value for **Name**.
- e) In the **Host** text box, type the EMS Server hostname, and click **OK**.
- f) Select the **Enable SSL** checkbox.

**Step 5** Enable server certificate validation.
- a) In the Enterprise Servers Settings dialog, select the **Validate Server Certificate** check box, and click **OK**.
- b) In the Configuration pane, double-click **Gateway Settings**.
- c) In the Gateway Settings dialog, click the **Components** tab.
- d) Select the **xgPKIManager** check box.

   If **xgPKIManager** is not listed, type it in and click **Add**.
- e) Click **OK**.

**Step 6** Define the required data items.

   The table below contains the required data items. For more information on adding data items, see *Creating Data Items* on page 20.

   **NOTE:** Do not remove any of the required data items.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Writeable</th>
<th>Alias</th>
<th>Initialization Value</th>
<th>Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>SwMgrStatus</td>
<td>String</td>
<td>Yes</td>
<td>SwMgrStatus</td>
<td>IDLE</td>
<td>Good</td>
</tr>
<tr>
<td>SwMgrConfig</td>
<td>String</td>
<td>Yes</td>
<td>SwMgrConf</td>
<td>Good</td>
<td></td>
</tr>
<tr>
<td>onboarded</td>
<td>Digital</td>
<td>Yes</td>
<td>onboarded</td>
<td>On</td>
<td>Good</td>
</tr>
</tbody>
</table>

**Step 7** (Optional) Define other entities, such as counters, expressions, alarms, and custom data sources.

   Add any items you need for executing the sample workflows in *About the Alarms Work Flow Example* on page 51.

**Step 8** Add the data items to a new or existing data logger.

   For more information on creating a data logger, see *Creating Data Loggers* on page 21.

**Step 9** Create a new logic schema to set the **onboarded** data item to 1.
- a) In the Project Window pane, expand your project, right-click **Logic Schemas**, and select **Add New Schemas**.
- b) In the Logic Schema dialog, type **onboarding_status** for the data item’s **Name**.
c) On the **Trigger** tab, select **System Registered** from the drop-down menu.

d) Click the **Actions** tab.

e) On the **Action** drop-down menu, select **Write Data Item**.

f) On the **Data** drop-down menu, select **onboarded**.

g) On the **Attribute** drop-down menu, select **Value**.

h) On the **Type** drop-down menu, select **Digital**.

i) In the **Value** text box, type 1, and click **Add**.

j) Click **OK**.

---

**Step 10** Configure run-time settings.

a) In the Project Window pane, expand your project, and click **Settings**.

b) In the Settings pane, double-click **Device Identification**.

c) In the Device Identification dialog, within the **Custom Device Information** section, select the **Enable** check box.

d) In the **DLL Name** text box, type **EDevID**, and click **OK**.

**Step 11** Click **File > Save All**.
You have created or modified an agent project. For information on preparing an agent project for upload to the EMS Server, see *Preparing an Agent Project Tarball* on page 22.

**Postrequisites**

You must validate your agent project before deploying it as part of an onboarding or update package.

**Related Links**

*Preparing an Agent Project Tarball* on page 22
Prepare an agent project for uploading to the EMS Server by removing extraneous files and creating a tarball.

---

**Adding to the Agent Project**

**Creating Data Items**

Using the Axeda® Builder, you can create data items in an agent project.

Create data items in an open agent project. For information on creating an agent project, see *Creating an Agent Project* on page 17.

---

**NOTE:** Defining data items in the Axeda® Builder is independent of defining data sources. For information on defining data sources, see *Defining Custom Data Sources* on page 21.

1. In the Project Window pane, expand your project, right-click **Data Items**, and select **Add New Data Item**.

2. In the Data Item Definition dialog, enter values for the data item's **Name**, **Type**, and **Alias**, and click **Create**.

3. Repeat the previous step for each data item.

4. Click **Close**.

The agent project contains the definition(s) of the data items, but an EMS Agent running this project would not publish their values until they are added to a data logger. For information about adding data items to a data logger, see *Creating Data Loggers* on page 21.

**Related Links**

*Creating Data Loggers* on page 21
Data items must be added to a data logger in order to be published by an EMS Agent running this agent project.

*Defining Custom Data Sources* on page 21

*Creating Data Loggers* on page 21
Data items must be added to a data logger in order to be published by an EMS Agent running this agent project.

*Creating an Agent Project* on page 17
You create an agent project to configure the EMS Agent using Axeda® Builder.

Defining Custom Data Sources on page 21

Creating Data Loggers

Data items must be added to a data logger in order to be published by an EMS Agent running this agent project.

Add new data items to a new or existing data logger.

1. In the Project Window pane, select an existing data logger or right-click **Logger**, select **Add New Data Logger...**, and type a value in the **Name** text box, such as `project_name_logger`.

2. In the Data Logger Configuration dialog, click the **Data** tab, and click **Add...**.

3. Highlight the data items created above, and click **OK**.

4. In the **Data** tab, under **Log Method**, select **Only changed data**.

5. Click **OK**.

Related Links

Creating Data Items on page 20
Using the Axeda® Builder, you can create data items in an agent project.

Creating Data Loggers on page 21

Defining Custom Data Sources

You can write a custom EDD driver that interfaces with your agent project. There is one EDD per event/data type/source. For more information on defining custom data sources, see the Axeda Agents: EDD Drivers Reference, 6.6.1.

Related Links

Creating Data Items on page 20
Using the Axeda® Builder, you can create data items in an agent project.

Creating Data Loggers on page 21
Data items must be added to a data logger in order to be published by an EMS Agent running this agent project.

Preparing an Agent Project Tarball

Prepare an agent project for uploading to the EMS Server by removing extraneous files and creating a tarball.

To prepare an agent project for uploading to the EMS Server, remove extraneous files and create a tarball. You can do this on either a Linux machine or a Windows 7 machine.

Related Links

- Using a Windows Machine on page 22
- Using a Linux Machine on page 25

Using a Windows Machine

Use a Windows 7 machine to prepare an agent project for uploading to the EMS Server.

Step 1  Install `bsdtar`.

`bsdtar` is a command-line tool with a syntax similar to the Linux `tar` command.


b) Run the installer.

c) (Optional) Add `C:\Program Files (x86)\GnuWin32\bin` to your Windows PATH environment variable:

Step 2  Copy your agent project to a temporary directory.

For example, `C:\Axeda\Builder\onboarding_staging`.

Step 3  Navigate to the temporary directory.

Step 4  Delete everything except the `.xml` files and the subproject directory.

The result should be similar to the image below.
Step 5 Create a new empty file named `xgDeployConfig.xml` in the temporary directory (in this example, in `C:\Axeda\Builder\onboarding_staging`).

Step 6 Navigate to the subproject directory (in this example, `sw_mgr_demo`).

The subproject directory looks like the image below.

![Gateway directory contents](image)

Step 7 Delete everything in the subproject except the **Gateway** directory.

Step 8 Move all the files in the **Gateway** directory up one level into the subproject directory.

Using the example above, this means moving the files in `C:\Axeda\Builder\onboarding_staging\sw_mgr_demo\Gateway` to `C:\Axeda\Builder\onboarding_staging\sw_mgr_demo`.

Step 9 Delete the empty **Gateway** directory.
Step 10 Create a tarball.
   a) Open a **Command Prompt**.
   b) Using **bsdtar**, create a tarball of everything in your temporary directory (in this example, C: \Axeda\Builder\onboarding_staging)

   ```
   bsdtar -czvf tarball_name.tar.gz *
   ```

You now have an agent project encapsulated in a tarball that is ready to be uploaded to the EMS Server as part of an onboarding package.

**Related Links**

*Preparing an Agent Project Tarball* on page 22
Prepare an agent project for uploading to the EMS Server by removing extraneous files and creating a tarball.

*Creating an Onboarding Package* on page 26
You must create one onboarding package per device model on the EMS Console. The onboarding package must contain a valid agent project tarball containing the data items required for software update configuration and status tracking.

**Using a Linux Machine**

Use a Linux machine to prepare an agent project for uploading to the EMS Server.

**Step 1** Copy the entire agent project directory from your Windows 7 machine to your Linux machine.

**Step 2** Navigate to the temporary directory.

**Step 3** Delete the **Gateway Model** subdirectory.

```
$ rm -rf Gateway\ Model/
```

**Step 4** Remove all non-XML files.

```
$ rm -f `find . -type f ! -name '*.xml'`
```

**Step 5** Create a new empty file named **xgDeployConfig.xml** in the temporary directory.

```
$ echo "" > xgDeployConfig.xml
```

**Step 6** Move all the files in the **Gateway** directory up one level into the subproject directory.

```
$ mv subproject_name/Gateway/* subproject_name/.
```

**Step 7** Delete all directories within the subproject.

```
$ rm -rf `find subproject_name/* -type d`
```

**Step 8** Create a tarball.

```
$ tar cvfz tarball_name.tar.gz *
```

You now have an agent project encapsulated in a tarball that is ready to be uploaded to the EMS Server as part of an onboarding package.

**Related Links**

* Preparing an Agent Project Tarball* on page 22
  Prepare an agent project for uploading to the EMS Server by removing extraneous files and creating a tarball.

* Creating an Onboarding Package* on page 26
You must create one onboarding package per device model on the EMS Console. The onboarding package must contain a valid agent project tarball containing the data items required for software update configuration and status tracking.

Creating Automatic Onboarding Notifications

You can create an expression rule to notify you whenever a device is successfully onboarded. You can create an expression rule on the EMS Console to notify you whenever any device of a specified model is successfully onboarded. Your notification can be in any form supported by the expression rules action. It is best practice to create the expression rule(s) before deploying any onboarding packages.

When you create the expression rule, use the following expression for the **If** clause:

```
DataItem.onboarded.value == 1 && DataItem.onboarded.changed
```

For more information about expression rules, see the EMS Console online help.

Creating an Onboarding Package

You must create one onboarding package per device model on the EMS Console. The onboarding package must contain a valid agent project tarball containing the data items required for software update configuration and status tracking.

Before you create an onboarding package, your setup must meet the following prerequisites:

- You must have a valid agent project that you have encapsulated in a tarball. For more information about creating an agent project tarball, see *Creating an Agent Project* on page 17 and *Preparing an Agent Project Tarball* on page 22.
- You must have at least one device of the specified model registered and visible on the EMS Server before the content package for that model can be created, otherwise some cloud content for that device may not be available.

**Step 1**

Create a new software package.

a) Log into the EMS Console.
b) Click the **Content** tab.
c) Click **New > Package**.
d) In the **Name** text box, type a name for your package (hereafter referred to as **package_name**).
e) In the **Model** drop-down list, select the model that **package_name** can be deployed to.
f) Type a value in the **Version** text box.
g) Clear **Enable Package Retries**.
NOTE: Package retries must be disabled to avoid the following:

- an infinite loop due to automatic deployment of a bad package, which can cause the EMS Agent to disconnect from the EMS Server
- redeployment of an update package to an asset that is in the middle of an over-the-air update, which results in a failed update

h) Click Next.
i) On the Package Dependencies page, click Next.

Step 2 On the Package Instructions page, add an instruction to download the agent project tarball to the device.

a) In the Select Instruction drop-down list, select Download File from Platform.
b) Click Browse, select update_name.txt, and click Upload.
c) Wait for the "File uploaded successfully" message to be displayed.
d) Leave the Destination text box blank.
e) Select the following check boxes:
   - Path is relative to asset's home directory.
   - This file is a .tar.gz archive; unpack it in the given directory after downloading.
   - Overwrite existing files ...
f) Click Add.

Step 3 On the Package Instructions page, add an instruction to execute the /opt/agent/scripts/register_scripts.sh application.

a) In the Select Instruction drop-down list, select Execute Application.
b) In the Application Name drop-down list, type /opt/agent/scripts/register_scripts.sh.
c) Click Add.

Step 4 On the Package Instructions page, add an instruction to restart the agent (Soft Restart).

a) In the Select Instruction drop-down list, select Restart Agent and choose Soft Restart.
b) Click Add.
c) Click Finish.

Step 5 Confirm the package setup.

On the Package Confirmation page, click Finish.

You have created a new package named package_name on the EMS Server that can be deployed to devices matching the model specified in package_name.

To find package_name, click the Content tab, and select View > Packages.

Related Links

Deploying an Onboarding Package on page 28
The deployment of an onboarding package to new devices must be done manually for the first device of a new model, and can done automatically for subsequent devices of the same model.

**Deploying an Onboarding Package**

The deployment of an onboarding package to new devices must be done manually for the first device of a new model, and can done automatically for subsequent devices of the same model.

The first time a device of a new model registers with the EMS Server, you must manually deploy the onboarding package to it because there are no expression rules in the cloud for this model, at this point. Subsequent devices of the same model can be automatically onboarded, if you create an expression rule to automatically deploy the onboarding package to all new devices of that model.

To summarize:

• You must manually deploy the onboarding package to the first device of each model. For more information, see [Deploying an Onboarding Package Manually](#) on page 28.

• You can set up automatic onboarding for subsequent devices of the same model. For more information, see [Deploying an Onboarding Package Automatically](#) on page 29.

**Related Links**

*Deploying an Onboarding Package Manually* on page 28
The onboarding package must be manually deployed to every device that is the first of its model to register with the EMS Server.

*Deploying an Onboarding Package Automatically* on page 29
For automatic onboarding, you create an action that automatically deploys the onboarding package to devices of a specific model, and then you create an expression rule of type Registration which triggers the execution of your action on newly registered devices.

---

**Deploying an Onboarding Package Manually**

The onboarding package must be manually deployed to every device that is the first of its model to register with the EMS Server.

Locate the onboarding package you created on the EMS Console's Contents tab, and deploy it onto newly registered target(s).

**Step 1** Click the Content tab, and select View > Packages.

**Step 2** Locate the onboarding package you created in the list of packages.

**Step 3** Click its Deploy link.

**Step 4** Select Manual Deployment, and click Next.

**Step 5** On the Deployment Routing Criteria page, click Next.

**Step 6** On the Select Assets - `package_name` page, select all applicable assets and click Add Checked.

**Step 7** Click Finish.
Step 8  Click Deploy.

Postrequisites

You must confirm that onboarding is successful, and fix any onboarding failures. For more information about confirming successful onboarding, see Confirming Successful Onboarding on page 30.

Related Links

Deploying an Onboarding Package on page 28
The deployment of an onboarding package to new devices must be done manually for the first device of a new model, and can done automatically for subsequent devices of the same model.

Confirming Successful Onboarding on page 30
Successful onboarding means that the onboarding package was successfully deployed to the asset. If onboarding is unsuccessful, the asset cannot receive over-the-air updates.

Deploying an Onboarding Package Automatically

For automatic onboarding, you create an action that automatically deploys the onboarding package to devices of a specific model, and then you create an expression rule of type Registration which triggers the execution of your action on newly registered devices.

Do the following steps to deploy an onboarding package automatically.

Step 1  Create an action that automatically deploys the onboarding package to any newly registered devices of a specific model.

a) Log into the EMS Console.
b) Click Manage, and then select New > Action.
c) In the Name text box, type a name for this action (hereafter referred to as action_name).
d) In the Category drop-down list, select Software management actions.
e) Select the Deploy a software package option, and click Next.
f) In the Model drop-down list, select the value of Model that matches your asset (hereafter referred to as model_name) and then click Next.
g) In the Package drop-down list, select a package (hereafter referred to as package_name), and click Next.
h) In the User Groups That Handle Edit Permissions For Action page, click Next.
i) In the User Groups That Handle Execute Permissions For Action page, click Next.
j) In the Confirm changes to action: action_name page, click Finish.

You have created a new action named action_name on the EMS Server. When executed, action_name deploys package_name to all assets matching model_name.

To find action_name, click the Manage tab, and select View > Actions.

Step 2  Create an expression rule of type Registration, which triggers the execution of your action when its condition is met.

a) Click Manage, and select New > Expression Rule.
b) In the Name text box, type a name for the new rule (hereafter referred to as rule_name).
c) In the Type drop-down list, select Registration.
d) Click Apply to asset, add the models to be included, add individual assets to be included, add individual assets to be excluded, and click OK.
e) Select Enabled and Execute action each time rule evaluates to true.
f) In the If text box, type Registration.first.

g) In the Then text box, type ExecuteAction("action_name"), where action_name is the name of the action you created to deploy the onboarding package.

You must enclose the action name in double quotes.

**TIP:** For help with the expression syntax, click in the text box, and then double-click the required object in the Select Functions, Actions and Variables pane on the left side of the page.

h) Click Validate.

i) Click Save and then click OK.

The first registration of any asset matching model_name causes the EMS Server to execute action_name, which deploys package_name to that asset.

**Postrequisites**

You must confirm that onboarding is successful, and fix any onboarding failures. For more information about confirming successful onboarding, see Confirming Successful Onboarding on page 30.

**Related Links**

- *Deploying an Onboarding Package* on page 28
  The deployment of an onboarding package to new devices must be done manually for the first device of a new model, and can done automatically for subsequent devices of the same model.

- *Confirming Successful Onboarding* on page 30
  Successful onboarding means that the onboarding package was successfully deployed to the asset. If onboarding is unsuccessful, the asset cannot receive over-the-air updates.

---

**Confirming Successful Onboarding**

Successful onboarding means that the onboarding package was successfully deployed to the asset. If onboarding is unsuccessful, the asset cannot receive over-the-air updates.

You can confirm that an asset has been onboarded by checking its dashboard, or by receiving automatic notifications. For more information about automatic onboarding notifications, see Creating Automatic Onboarding Notifications on page 26.

**Step 1** Log into the EMS Console.

**Step 2** Open the asset’s dashboard.

**Step 3** Check the status of the deployment action.

In the Recent Actions section, verify that the status of Package Deployed [onboarding_package_name] is Delivered to Agent.

**Step 4** In the Scripts section, verify that the following scripts are present:

- exec_sw_mgr
- register_scripts
- agent-update
Step 5  In the Data section, verify that the required data items are present:

SwMgrConfig
SwMgrStatus
onboarded

Step 6  In the Data section, verify that the value of onboarded is 1.

Postrequisites

If any of the above verification steps fail, see Fixing Onboarding Failures on page 47.
Updating the Components

Updating the System 33
Updating the EMS Agent 38
Creating Update Packages 43
Deploying Update Packages 43
Downloading Files to Assets 44
Uploading Files from Assets 45

Updating the System

Updates to the system are sent to assets over the air from a secure RPM repository.

A system update may contain:

• User-generated RPMs
• Wind River IDP XT RPMs.

The EMS Agent RPM is a special case and is updated with a different workflow. For information about updating the EMS Agent, see Updating the EMS Agent on page 38.

Related Links

Installing, Updating, or Removing RPMs on page 34
You can install, update, or remove RPMs over the air from the EMS Console by creating and deploying a package containing update instructions.

Monitoring RPM Updates on page 37
Verify that an update has been applied to an asset.

**Installing, Updating, or Removing RPMs**

You can install, update, or remove RPMs over the air from the EMS Console by creating and deploying a package containing update instructions.

You must have access to your secure repository of signed RPMs.

An asset must have been successfully onboarded in order to receive over-the-air updates.

**Step 1** Create a `.txt` file with update instructions in JSON format (hereafter referred to as `update_name.txt`).

For more information about the supported JSON file content, see [Supported JSON File Content](#) on page 65.

The example below shows installation instructions for an RPM named `lm-sensors`.

```json
{
  "identifier": "Config to install lm-sensors RPM package",
  "operation": "install",
  "channelMode": "append",
  "channels": [
    {"alias": "Test-rpm-secure", "name": "My Secure RPM Test Channel", "type": "rpm-md", "url": "rpm_repo_URL", "priority": "-100"}
  ],
  "packages": [{"name": "lm-sensors"}],
  "rebootOnCompletion": "no"
}
```

The example below shows update instructions for an RPM named `lm-sensors`.

```json
{
  "identifier": "Config to upgrade lm-sensors RPM package",
  "operation": "upgrade",
  "channelMode": "override",
  "channels": [
    {"alias": "Test-rpm-srm", "name": "My SRM RPM Test Channel", "type": "rpm-md", "url": "rpm_repo_URL", "priority": "1"}
  ],
  "packages": [{"name": "lm-sensors"}],
  "rebootOnCompletion": "no"
}
```

The example below shows removal instructions for an RPM named `lm-sensors`.

```
NOTE: Removing packages does not delete the channel for the install. If your JSON file contains errors on the initial install, use a new channel name the second time.
```

```json
{
  "identifier": "Config to remove lm-sensors package",
  "operation": "remove",
  "packages": [
    {"name": "lm-sensors"},
    {"name": "lmsensors-sensors"},
    {"name": "lmsensors-scripts"}
  ],
  "rebootOnCompletion": "no"
}
```

**Step 2** Create a new software package.
a) Log into the EMS Console.
b) Click the Content tab.
c) Click New > Package.
d) In the Name text box, type a name for your package (hereafter referred to as package_name).
e) In the Model drop-down list, select the model that package_name can be deployed to.
f) Type a value in the Version text box.
g) Clear Enable Package Retries.

NOTE: Package retries must be disabled to avoid the following:
- an infinite loop due to automatic deployment of a bad package, which can cause the EMS Agent to disconnect from the EMS Server
- redeployment of an update package to an asset that is in the middle of an over-the-air update, which results in a failed update

h) Click Next.
i) On the Package Dependencies page, click Next.

Step 3 On the Package Instructions page, add an instruction to set the SwMgrConfig data item to the [planned] location of the update instructions on the asset.
a) On the Package Instructions page, select Set Data Item from the drop-down list.
b) In the Data Item drop-down list, select SwMgrConfig.
c) In the Value text box, type /opt/agent/sw_mgr/config/update_name.txt, and click Add.

Step 4 On the Package Instructions page, add an instruction to download the update instructions to the asset.
a) In the Select Instruction drop-down list, select Download File from Platform.
b) Click Browse, select update_name.txt, and click Upload.
c) Wait for the "File uploaded successfully" message to be displayed.
d) In the Destination text box, type /opt/agent/sw_mgr/config.
e) Select the Overwrite existing files ... check box, and click Add.

Step 5 On the Package Instructions page, add an instruction to set the SwMgrStatus data item to IDLE.
a) In the Select Instruction drop-down list, select Set Data Item.
b) In the Data Item drop-down list, select SwMgrStatus.
c) In the Value text box, type IDLE, and click Add.

Step 6 On the Package Instructions page, add an instruction to run the exec_sw_mgr script.
a) In the Select Instruction drop-down list, select Run Script.
b) In the Script Name drop-down list, select exec_sw_mgr, and click Add.
c) Click Finish.

Step 7 Confirm the package setup.
On the Package Confirmation page, click Finish.
You have created a new package named `package_name` on the EMS Server. To find `package_name`, click the `Content` tab, and select `View > Packages`.

**Step 8** Deploy the package to a specific asset or model.

**NOTE:** Deploy only one update at a time per asset.

For more information on deploying packages, see *Deploying Update Packages* on page 43.

**Postrequisites**

Verify that the update succeeded. For more information on how to verify update status, see *Monitoring RPM Updates* on page 37.

**Related Links**

- *Updating the System* on page 33
- Updates to the system are sent to assets over the air from a secure RPM repository.
- *Supported JSON File Content* on page 65
- Specific attribute-value pairs are supported in the JSON file used for RPM updates.
- *Deploying Update Packages* on page 43
- Packages can be deployed to assets manually or automatically.
- *Monitoring RPM Updates* on page 37
Verify that an update has been applied to an asset.

Monitoring RPM Updates

Verify that an update has been applied to an asset.

Failed updates result in a rollback to your backup tarball. It is not enough for the package deployment to be successful; that only indicates that the package instructions were successful and the `exec_sw_mgr` script was started, but not necessarily that the update was successful.

Step 1  Log into the EMS Console.

Step 2  Open the asset's dashboard.

Step 3  In the Recent Actions section, verify that the status of Package Deployed [package_name] is Delivered to Agent.

Step 4  Verify that there is no `SwMgrFailedAlarm` alarm in the Recent Actions section.

Step 5  In the Data section, verify that the value of the `SwMgrStatus` is SUCCESS.

Step 6  In the Uploaded Files section, check the contents of the newest `wra_sw_mgr_exec.log` file.

The contents of `wra_sw_mgr_exec.log` should be similar to the example below.

| Software Manager Status: STARTING...  |
| Parsing JSON configuration file '/opt/agent/sw_mgr/config/wr_sw_mgr_cfg_demo_lmsensors_inst_srm.txt'... |
| Executing software manager operation using configuration: Config to install lm-sensors RPM package from security-enabled repo |
| Configuring channels... |
| Using channels ['Test-rpm-srm'] |
| Updating the channel info... |
| Executing package install... |
| Software Manager Status: COMPLETED SUCCESSFULLY. (elapsed time = 0 days 0h:3m:40s). |

Step 7  If `wra_sw_mgr_exec.log` indicates "no interesting upgrades available", check the value of the channel's priority attribute in the JSON update instructions. For more information about this attribute, see Supported JSON File Content on page 65.

Step 8  If `wra_sw_mgr_exec.log` indicates a problem, check `wra_sw_mgr_dbg.log` for the details.

An empty JSON file causes a log entry in `wra_sw_mgr_dbg.log` similar to the following:

| Computing transaction... |
| No interesting upgrades available. |

Errors in the JSON file cause log entries in `wra_sw_mgr_dbg.log` similar to the following:

| ERROR: Invalid JSON configuration: Expecting object: line 17 column 2 (char 379) |
| Software Manager Status: STARTING... |
| Parsing JSON configuration file '/opt/agent/sw_mgr/config/wr_sw_mgr_cfg_demo_lmsensors_inst_missing.txt'... |
| Software Manager Status: FAILURE! (elapsed time = 0 days 0h:0m:0s) |

For more information about supported JSON file content, see Creating an Onboarding Package on page 26.

Step 9  Correct the JSON file and redo the update.
Related Links

- Updating the System on page 33
  Updates to the system are sent to assets over the air from a secure RPM repository.

Updating the EMS Agent

Updates to the EMS Agent are sent over the air from the EMS Server to assets. The EMS Agent RPM is a special case and is updated with a workflow that is different from other RPM updates.

An EMS Agent update may contain:

- Agent binaries: the EMS Agent code built into the wr-ems layer
- An agent project and external files: An agent project is the set of files that configure the EMS Agent. The agent project’s external files are files referred to by the agent project running on the EMS Agent, but external to the agent project (in other words, external to the asset’s home directory, /opt/agent/gateway. These include agent DLLs and scripts.

Before You Begin

Before you begin, set up an automated handshake between the EMS Server and the EMS Agent. This handshaking action is triggered whenever the EMS Server detects that the EMS Agent is being updated. For more information, see Monitoring EMS Agent Updates on page 38.

Related Links

- Monitoring EMS Agent Updates on page 38
  Updates to the EMS Agent must be tested and rolled back if necessary.
- Updating Agent Binaries on page 39
  Agent binaries refer to the EMS Agent code built into the wr-ems layer. You can update the agent binaries over the air from the EMS Console.
- Updating the Agent Project on page 41
  An agent project is a set of files that configure the EMS Agent that runs on an asset. You can update the agent project, and any external files that the agent project refers to, over the air from the EMS Console.

Monitoring EMS Agent Updates

Updates to the EMS Agent must be tested and rolled back if necessary.

Set up the following automatic handshaking before you attempt over-the-air updates of the EMS Agent.

Step 1 Create an empty file named .reg-notify.

Step 2 Create an action named reg-notify to download .reg-notify to the asset.

Select the Overwrite existing files ... and Path is relative to asset’s home directory. check boxes in the Download files to an asset option.

For more information on creating actions to download files to assets, see Downloading Files to Assets on page 44.
Step 3 Create an expression rule named `agent-update-notify` of type `Registration`, which triggers the execution of `reg-notify` when its condition is met.

a) In the If clause, type `Registration.time > 0`.
b) In the Then clause, type `ExecuteAction("reg-notify")`.

You must create this expression rule for each model.

Related Links

*Updating the EMS Agent* on page 38

Updates to the EMS Agent are sent over the air from the EMS Server to assets. The EMS Agent RPM is a special case and is updated with a workflow that is different from other RPM updates.

**Updating Agent Binaries**

Agent binaries refer to the EMS Agent code built into the `wr-ems` layer. You can update the agent binaries over the air from the EMS Console.

Updates to the EMS Agent are sent from Wind River as part of an IDP XT RCPL.

**Step 1** On your development host, create a new directory for your platform project.

```
$ mkdir -p installDir/workspace/newProjDir
```

**Step 2** In your new project directory, create a new directory with two subdirectories for staging the update.

```
$ cd newProjDir
$ mkdir -p update-agent/RPMS
$ mkdir -p update-agent/RPM_rollback
```

**Step 3** Copy the EMS Agent RPMs in your old project directory to your staging area.

```
$ cp oldProjDir/export/RPMS/i586/axeda-agent-gateway* newProjDir/update-agent/RPM_rollback
```

**Step 4** Install the RCPL on your development host in `newProjDir`.

**Step 5** Configure your platform project in `newProjDir`, as described in *Building and Deploying Your Platform Project* on page 12.

**Step 6** Build the EMS Agent RPMs.

```
$ cd newProjDir
$ make -C build axeda-agent-gateway
```


**Step 7** Copy the EMS Agent RPMs in your new project directory to your staging area.

```
$ cp export/RPMS/i586/axeda-agent-gateway* update-agent/RPMS
```

**Step 8** Create a single tarball out of the entire contents of your staging area (hereafter referred to as `tarball_name.tar.gz`).

```
$ cd update-agent && tar czf ../tarball_name.tar.gz RPM* && cd..
```
Step 9
Create a `.txt` file with update instructions in JSON format (hereafter referred to as `update_name.txt`).

Use this exact syntax, replacing `version` with the version number that appears in the RPM filename:

```json
{
  "identifier": "Update the axeda agent",
  "operation": "local-install",
  "channels": [
    {"alias": "Test-local", "dir": "/opt/agent/gateway/RPMS" }
  ],
  "packages": [{"name": "axeda-agent-gateway-db-g-version.i586.rpm"}, {"name": "axeda-agent-gateway-version.i586.rpm"}],
  "rebootOnCompletion": "no"
}
```

Step 10
Create a new software package on the EMS Console.

a) Log into the EMS Console.
b) Click the **Content** tab.
c) Click **New > Package**.
d) In the **Name** text box, type a name for your package (hereafter referred to as `package_name`).
e) In the **Model** drop-down list, select the model that `package_name` can be deployed to.
f) Type a value in the **Version** text box.
g) Clear **Enable Package Retries**.

**NOTE:** Package retries must be disabled to avoid the following:

- an infinite loop due to automatic deployment of a bad package, which can cause the EMS Agent to disconnect from the EMS Server
- redeployment of an update package to an asset that is in the middle of an over-the-air update, which results in a failed update

h) Click **Next**.
i) On the Package Dependencies page, click **Next**.

Step 11
On the Package Instructions page, add an instruction to set the `SwMgrConfig` data item to the [planned] location of the update instructions on the asset.

a) On the Package Instructions page, select **Set Data Item** from the drop-down list.
b) In the **Data Item** drop-down list, select `SwMgrConfig`.
c) In the **Value** text box, type `/opt/agent/sw_mgr/config/update_name.txt`, and click **Add**.

Step 12
On the Package Instructions page, add an instruction to download the update instructions to the asset.

a) In the **Select Instruction** drop-down list, select **Download File from Platform**.
b) Click **Browse**, select `update_name.txt`, and click **Upload**.
c) Wait for the "File uploaded successfully" message to be displayed.
d) In the **Destination** text box, type `/opt/agent/sw_mgr/config`, and click **Add**.

Step 13
On the Package Instructions page, add an instruction to set the `SwMgrStatus` data item to IDLE.

a) In the **Select Instruction** drop-down list, select **Set Data Item**.
b) In the **Data Item** drop-down list, select `SwMgrStatus`.
c) In the **Value** text box, type IDLE, and click **Add**.
Step 14 On the Package Instructions page, add an instruction to download the tarball to the asset.
   a) In the Select Instruction drop-down list, select Download File from Platform.
   b) Click Browse, select tarball_name.tar.gz, and click Upload.
   c) Wait for the "File uploaded successfully" message to be displayed.
   d) Leave the Destination text box blank.
   e) Select the following check boxes:
      • This file is a .tar.gz archive; unpack it in the given directory after downloading.
      • Overwrite existing files ...
   f) Click Add.

Step 15 On the Package Instructions page, add an instruction to run the agent-update script.
   a) In the Select Instruction drop-down list, select Run Script.
   b) In the Script Name drop-down list, select agent-update, and click Add.
   c) Click Finish.

Step 16 Confirm the package setup.
   On the Package Confirmation page, click Finish.

   You have created a new package named package_name on the EMS Server. To find package_name, click the Content tab, and select View > Packages.

Step 17 Deploy the package to a specific asset or model.

   NOTE: Deploy only one update at a time per asset.

   For more information on deploying packages, see Deploying Update Packages on page 43.

Postrequisites

   Verify that the update succeeded. Failed updates result in a rollback to your backup tarball. For more information, see Monitoring RPM Updates on page 37.

Related Links

   Updating the EMS Agent on page 38
   Updates to the EMS Agent are sent over the air from the EMS Server to assets. The EMS Agent RPM is a special case and is updated with a workflow that is different from other RPM updates.

Updating the Agent Project

   An agent project is a set of files that configure the EMS Agent that runs on an asset. You can update the agent project, and any external files that the agent project refers to, over the air from the EMS Console.

   Your existing cloud-side events, alarms, scripts, and actions must be compatible with the new agent project to be installed.

   You must validate your agent project before deploying it as part of an onboarding or update package.

   There are agent DLLs and scripts that are external to the agent directory on the device. For information on how to update these files, see Downloading Files to Assets on page 44.
Deploy an updated agent project to selected EMS Agents. This workflow applies updates to previously onboarded devices only.

**Step 1** Prepare the new agent project tarball.

For information on preparing the agent project for upload, see *Preparing an Agent Project Tarball* on page 22.

**Step 2** Create a new software package.

a) Log into the EMS Console.
b) Click the **Content** tab.
c) Click **New > Package**.
d) In the **Name** text box, type a name for your package (hereafter referred to as *package_name*).
e) In the **Model** drop-down list, select the model that *package_name* can be deployed to.
f) Type a value in the **Version** text box.
g) Clear **Enable Package Retries**.

---

**NOTE:** Package retries must be disabled to avoid the following:

- an infinite loop due to automatic deployment of a bad package, which can cause the EMS Agent to disconnect from the EMS Server
- redeployment of an update package to an asset that is in the middle of an over-the-air update, which results in a failed update

h) Click **Next**.
i) On the Package Dependencies page, click **Next**.

**Step 3** On the Package Instructions page, add an instruction to download the tarball to the asset.

a) In the **Select Instruction** drop-down list, select **Download File from Platform**.
b) Click **Browse**, select *tarball_name.tar.gz*, and click **Upload**.
c) Wait for the "File uploaded successfully" message to be displayed.
d) Leave the **Destination** text box blank.
e) Select the following check boxes:

- **This file is a .tar.gz archive; unpack it in the given directory after downloading.**
- **Overwrite existing files** ...
f) Click **Add**.

**Step 4** On the Package Instructions page, add an instruction to run the *agent-update* script.

a) In the **Select Instruction** drop-down list, select **Run Script**.
b) In the **Script Name** drop-down list, select *agent-update*, and click **Add**.
c) Click **Finish**.

**Step 5** Confirm the package setup.

On the Package Confirmation page, click **Finish**.

You have created a new package named *package_name* on the EMS Server. To find *package_name*, click the **Content** tab, and select **View > Packages**.

**Step 6** Deploy the package to a specific asset or model.

---

**NOTE:** Deploy only one update at a time per asset.

For more information on deploying packages, see *Deploying Update Packages* on page 43.
Postrequisites

Since updating the agent project on an asset is very similar to onboarding a new device, follow the instructions in *Confirming Successful Onboarding* on page 30 to verify that the update succeeded.

Related Links

*Updating the EMS Agent* on page 38

Updates to the EMS Agent are sent over the air from the EMS Server to assets. The EMS Agent RPM is a special case and is updated with a workflow that is different from other RPM updates.

---

Creating Update Packages

Packages contain a payload and instructions, and are deployable to assets of a particular model.

**Step 1** Create a package by specifying the model that it can be deployed to, its payload, and its instructions.

a) Log into the EMS Console.
b) Click the **Content** tab.
c) Click **New > Package**.
d) In the **Name** text box, type a name for your package (hereafter referred to as *package_name*).
e) In the **Model** drop-down list, select the model that *package_name* can be deployed to.
f) Type a value in the **Version** text box.
g) Clear **Enable Package Retries**.

**NOTE:** Package retries must be disabled to avoid the following:

- an infinite loop due to automatic deployment of a bad package, which can cause the EMS Agent to disconnect from the EMS Server
- redeployment of an update package to an asset that is in the middle of an over-the-air update, which results in a failed update

h) Click **Next**.
i) On the Package Dependencies page, click **Next**.

---

Deploying Update Packages

Packages can be deployed to assets manually or automatically.

To deploy a package manually, find the package and click its **Deploy** link.

To deploy a package automatically, create an action on the EMS Console, and then create an expression rule to execute the action automatically whenever its conditions are met, as explained below.

**Step 1** Create an action to deploy a package automatically.

a) Log into the EMS Console.
b) Click Manage, and then select New > Action.

c) In the Name text box, type a name for this action (hereafter referred to as action_name).

d) In the Category drop-down list, select Software management actions.

e) Select the Deploy a software package option, and click Next.

f) In the Model drop-down list, select the value of Model that matches your asset (hereafter referred to as model_name) and then click Next.

g) In the Package drop-down list, select a package (hereafter referred to as package_name), and click Next.

h) In the User Groups That Handle Edit Permissions For Action page, click Next.

i) In the User Groups That Handle Execute Permissions For Action page, click Next.

j) In the Confirm changes to action: action_name page, click Finish.

You have created a new action named action_name on the EMS Server. When executed, action_name deploys package_name to all assets matching model_name.

To find action_name, click the Manage tab, and select View > Actions.

**Step 2** Create an expression rule to trigger the execution of action_name.

a) Click the Manage tab.

b) Click New > Expression Rule.

c) In the Name text box, type a name for the new rule (hereafter referred to as rule_name).

 d) In the Type drop-down list, select the rule type.

e) Click Apply to asset, include or exclude certain assets or models, and click OK.

f) Select the Enabled and Execute action each time rule evaluates to true check boxes.

g) Build an if-then-else expression from objects in the Select Functions, Actions and Variables pane on the left.

h) Click Validate.

i) Click Save.

**Downloading Files to Assets**

The EMS Console provides an interface for sending files to an asset of any model.

To send files to an asset manually, create an action on the EMS Console, and then execute that action on an asset.

To send files to an asset automatically, create an action on the EMS Console, and then create an expression rule to execute the action automatically whenever its conditions are met, as explained below.

**Step 1** Create an action to download a file from the EMS Server to an asset.

a) Log into the EMS Console.

b) Click the Manage tab.

c) Click New > Action.

d) In the Name text box, type a name for this action (hereafter referred to as action_name).

e) In the Category drop-down list, select Software management actions.

f) Select the Download files to an asset option, and click Next.

g) Click Browse to select a file, and click Upload.
A message appears above the file pathname stating that the file is successfully uploaded.

h) Select all check boxes that apply:
   For example, you may need to select:
   • This file is a .tar.gz archive; unpack it in the given directory after downloading.
   • Overwrite existing files ...
   • Path is relative to asset’s home directory.
   • Restart the Agent after downloading this file.

   **NOTE:** The asset’s home directory is /opt/agent/gateway.

i) Click Finish.
   The Confirm changes to action: action_name page appears.

j) Click Finish.
   You have created a new action named action_name on the EMS Server. When executed, action_name downloads the specified file to the asset.

**Step 2** Create an expression rule to trigger the execution of action_name.
   a) Click the Manage tab.
   b) Click New > Expression Rule.
   c) In the Name text box, type a name for the new rule (hereafter referred to as rule_name).
   d) In the Type drop-down list, select the rule type.
   e) Click Apply to asset, include or exclude certain assets or models, and click OK.
   f) Select the Enabled and Execute action each time rule evaluates to true check boxes.
   g) Build an if-then-else expression from objects in the Select Functions, Actions and Variables pane on the left.
   h) Click Validate.
   i) Click Save.

**Uploading Files from Assets**

The EMS Console provides an interface for sending files from an asset of any model to the EMS Server.

To upload files from an asset manually, create an action on the EMS Console, and then execute that action on an asset.

To upload files from an asset automatically, create an action on the EMS Console, and then create an expression rule to execute the action automatically whenever its conditions are met, as explained below.

**Step 1** Create an action to upload a file from an asset to the EMS Server.
   a) Log into the EMS Console.
   b) Click the Manage tab.
   c) Click New > Action.
   d) In the Name text box, type a name for this action (hereafter referred to as action_name).
   e) In the Category drop-down list, select Software management actions.
   f) Select the Upload files to an asset option, and click Next.
g) In the File Specification text box, type the name or full pathname of the file on the asset, and select any check boxes that apply.

**NOTE:** The asset's home directory is `/opt/agent/gateway`.

h) Click Add Instruction.

i) Click Finish.

The Confirm changes to action: *action_name* page appears.

j) Click Finish.

You have created a new action named *action_name* on the EMS Server. When executed, *action_name* uploads the specified file from the asset.

To execute *action_name* on a specific asset, go to that asset's dashboard. In its Actions section, click *action_name*.

The uploaded files are listed in the Uploaded Files section of the asset's dashboard.

**Step 2** Create an expression rule to trigger the execution of *action_name*.

a) Click the Manage tab.

b) Click New > Expression Rule.

c) In the Name text box, type a name for the new rule (hereafter referred to as *rule_name*).

d) In the Type drop-down list, select the rule type.

e) Click Apply to asset, include or exclude certain assets or models, and click OK.

f) Select the Enabled and Execute action each time rule evaluates to true check boxes.

g) Build an if-then-else expression from objects in the Select Functions, Actions and Variables pane on the left.

h) Click Validate.

i) Click Save.
Fixing Onboarding Failures

Onboarding failures prevent the remote management of devices. Recovery from an onboarding failure requires access to the device’s console through SSH or some other mechanism.

Follow these steps to fix onboarding failures (such as missing scripts or data items, or assets that disconnect from the EMS Server due to receiving a bad or wrong onboarding package).

Step 1  Delete the asset on the EMS Console.
   a) Log into the EMS Console.
   b) Click the Manage tab.
   c) Select the check box for the asset you want to delete.
   d) Scroll down to the bottom of the list of assets on that page.
   e) Click the Remove Assets link in the bottom right corner.

Step 2  Check (and correct) the package recently deployed to this asset.
   a) Log into the EMS Console.
   b) Check the asset’s dashboard to see which package was recently deployed to it.
   c) Find the package noted in the previous step.
      Click the Content tab, and select View > Packages.
   d) Highlight the package (click the package name).
   e) Scroll down to the bottom of the page, and look at the package details.
   f) Confirm that the package contains the correct instructions and payload, and is deployable to the correct assets or models.

If the package has errors, correct them with any or all of the following actions:
• Disable retries on the package: click its **Edit** link, and clear the **Enable Package Retries** check box on its Package Definition page. Otherwise, a bad onboarding package with retries enabled can continue to cause problems.
• Delete the package: click its **Cancel** link to cancel waiting for any pending asset(s) to respond, and then click its **Purge** link.
• Recreate the package.

**Step 3** Check the device’s settings and log file.

a) Log into the device remotely using SSH.

b) Check the log file, `/opt/agent/gateway/xGate.log`, for a registration message or error message (such as **failed to register**):

A log entry indicating successful registration looks like the sample below.

```
xgEnterpriseProxy: Device registered with server server_URL eMessage: model: model_name, serial number: serial_number
```

A log entry indicating failed registration looks like the sample below.

```
[ 0,  3,  7, 230] 11-08-2014 22:38:47.594 INFO    xgEnterpriseProxy: Setting proxy configuration
[ 0,  3,  7, 642] 11-08-2014 22:38:47.596 INFO    xgEnterpriseProxy: No HTTP proxy server used
[ 0,  3,  7, 644] 11-08-2014 22:38:47.598 INFO    xgEnterpriseProxy: No SOCKS proxy server used
[ 0,  1, 26, 150] 11-08-2014 22:38:55.902 DEBUG   xGate: Preparing to Stop.
[ 0,  1,  1, 422] 11-08-2014 22:38:56.907 DEBUG   xgAlarm: Preparing to stop
```

**c)** Verify that the EMS Server hostname is correct in the log file, `/opt/agent/gateway/xGate.log`.

d) Do a factory reset of the EMS Agent.

```
# /etc/init.d/wr-agent stop
# rm /etc/wr_ems.d/configured.lock
# /etc/init.d/wr-agent factory-reset
```

e) Reconfigure the EMS Agent.

```
# /etc/init.d/wr-agent reconfig
```

f) Restart the EMS Agent.

```
# /etc/init.d/wr-agent start
```

g) Confirm that the server certificate, `/opt/agent/gateway/xgDeployConfig.xml`, contains the correct EMS Server URL, device model and serial number.

```
# cat /opt/agent/gateway/xgDeployConfig.xml
<AgentRequest destination="agent" v="1.0">
  <UpdateConfiguration>
    <Configurations>
      <Config>
        <ProjectSettings>
          <EnterpriseServers>
            <SSL enabled="true" strength="128" certificate="true" cipherSuites=""/>
            <Primary host="EMS_server_hostname" port="443" db="drm-data_source" encryption="none"/>
          </EnterpriseServers>
          <Identification mn="model_name" sn="serial_number"/>
        </ProjectSettings>
      </Config>
    </Configurations>
  </UpdateConfiguration>
</AgentRequest>
```
At the completion of these steps, the device should register with the EMS Server as a new device. You must manually re-deploy the onboarding package to this device. For more information on manual deployment, see *Deploying an Onboarding Package Manually* on page 28.

**Recovering Lost Assets**

Assets can disappear or disconnect from the EMS Server for a number of reasons.

**Step 1** If the device never registered with the EMS Server, check the setting for the EMS Server hostname (as defined in Step 5 on page 13 of *Building and Deploying Your Platform Project* on page 12).

To fix incorrectly set hostnames, start over from this step.

**Step 2** Confirm that the asset is actually disconnected from the EMS Server.

a) Log into the EMS Console.

b) Open the asset’s dashboard.

c) Verify the *last contact* time is greater than the value of the asset’s *ping rate*.

d) (Optional) Create and execute an action to set the asset’s ping rate to a different value.

**Step 3** If the asset became disconnected from the EMS Server after receiving a bad or wrong onboarding package, see *Fixing Onboarding Failures* on page 47.

**Step 4** Check the device’s physical network connections.

a) Check network cable connections.

b) Log into the device remotely using SSH.

c) Confirm that the device responds to pings.

d) Confirm that the device can ping the EMS Server.

**Step 5** Check the device’s log file, `/opt/agent/gateway/xGate.log`, for an error message.

```
DEBUG xgEnterpriseProxy: Enterprise Server address not found - Proxy layer disabled
```

**Step 6** Reconfigure the agent to use its initial EMS Server information.

```
# /etc/init.d/wr-agent stop
# /etc/init.d/wr-agent reconfig
# /etc/init.d/wr-agent start
```
About the Alarms Work Flow Example

The alarms work flow example demonstrates how to configure an asset to send temperature data to the EMS server, how to raise an alarm based on the data values transmitted from the asset, and how to change the temperature threshold on the asset from the server.

Alarms appear at the EMS Console when the condition specified in the alarm definition occurs. At the console, you can acknowledge, clear, and escalate alarms.

This example demonstrates the following:

- manually onboarding the device
- creating expression rules and actions on the EMS Server
- modifying a data item on the asset from the EMS Server
- creating a custom object

The example work flow is based on the sim_sensors demo, which is available in the wr-ems-demo layer. It monitors the CPU temperature on the asset and raises an alarm if the temperature exceeds a specified threshold. It uses a sensor simulator program in case your device does not...
have the required software sensors package. It uses the EDD_TEXT.so module to transmit temperature data to the EMS Server.

When you boot the rootfs built with the wr-ems-demo layer enabled, the device has the following files required to run the demo:

- Shell scripts to stop and start transmitting the temperature data
  - /opt/agent/scripts/watchtempstart.sh
  - /opt/agent/scripts/watchtempstop.sh
- Sequence files that the EMS agent executes to run the shell scripts
  - /opt/agent/scripts/watchtempstart.seq
  - /opt/agent/scripts/watchtempstop.seq
- sensor simulator program
  - /opt/agent/scripts/sim_sensors
- configuration for the EDD_TEXT.so module
  - /opt/agent/gateway/EDD_TEXT.xml

To support onboarding, the wr-ems-demo layer provides the following:

- an onboarding tarball containing preconfigured XML files
- a sample gateway builder project you can modify with the Axeda Builder and use to create the onboarding tarball

This example uses the EMS Console for device management, but you can also create an application using the Mashery APIs to perform many of the management activities on the console. For more information, see the Wind River EMS Device Management User’s Guide, 1.0.

Basic Work Flow Steps

The alarms work flow example uses the configuration files from the sim_sensors sample gateway project, files built into the rootfs, and asset management actions and rules on the EMS Server.

The steps to run the sim_sensors_demo are the following:

Step 1  Configure your platform project to build the rootfs with the wr-ems-demo layer.

Step 2  Run the iot-config script to set the model and server name.

Step 3  Build and deploy the rootfs on your device.

Step 4  Modify the onboarding configuration from the sample gateway project (optional).

Step 5  If you modified the onboarding configuration, create the onboarding tarball from the sample gateway project.

Step 6  Onboard the device.

Step 7  At the EMS Console, create the asset management items for the work flow.

Asset management includes creating actions, expression rules, and alarm definitions.
Configuring and Building with the wr-ems-demo Layer

The wr-ems-demo layer includes the files required to build the rootfs and onboard the device. To run the sim_sensors demo, you need to configure the project to include the demo layer and specify the EMS configuration settings to build rootfs. When you build the rootfs, the build process creates tarballs that you can use to onboard the device or modify the gateway project configuration.

Step 1 When you run the configure command, include wr-ems-demo in the --with-layer as follows:

```
--with-layer=wr-ems,wr-ems-demo,wr-ima-appraise
```

If you already built a rootfs without the demo layer, when you run the configure command to add the wr-ems-demo layer, add the --enable-reconfig option.

For more information, see Building and Deploying Your Platform Project on page 12.

Step 2 Run the iot-config.py script to set the model and server name.

```
$ layers/wr-idp/wr-ems/scripts/iot-config.py -s \
"MODEL:modelName,HOST:yourServerName.axeda.com"
```

For example, use the model name SIM_SENSOR_DEMO.

When you specify the server name, do not include the protocol (https).

Step 3 Build and boot the rootfs on your device.

If you already built the rootfs, only new or changed content is built.

For more information, see Building and Deploying Your Platform Project on page 12.

After you build the rootfs and boot the device, the device has the files you need to run the simulated sensors example. Building the rootfs also creates the following file that contains an example configuration you can use to onboard the device:

```
projDir/demos/sim_sensor.demo.tar.gz
```

Onboarding Using the Alarms Sample Configuration

About the Alarms Sample Gateway Configuration

The sample gateway agent project files contain the EMS agent configuration required for the alarms work flow example.

You can use the sample agent project as the initial package for onboarding, or you can use it to update your device configuration after the initial onboarding process. If you change the
configuration in the project (for example, by renaming a data item), you need to adapt the EDD_TEXT.xml file and the instructions for creating the actions and rules on the EMS Server.

The wr-ems-demo layer provides two options for using the sample configuration:

- use the tarball that contains the preconfigured XML files
- modify the sample gateway agent project file and create your own onboarding tarball

The sample configuration contains the following:

- data items, a logger, and a logic schema for basic onboarding (the same items that are part of the EMSBaseProjectTemplate base builder project)
- default server configuration and device identification
- data items for temperature monitoring and setting the temperature monitoring interval, which are added to the default logger

The definition for the EDD_TEXT data source is the following:

<table>
<thead>
<tr>
<th>Type</th>
<th>Alias</th>
<th>Driver Name</th>
<th>Driver Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDD</td>
<td>EDD_TEXT</td>
<td>EDD_TEXT</td>
<td>Event Driven</td>
</tr>
</tbody>
</table>

In addition to the data items required for simple onboarding, the following data items are defined specifically for this workflow example:

<table>
<thead>
<tr>
<th>Item</th>
<th>Type</th>
<th>Value</th>
<th>Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>temperature</td>
<td>Analog</td>
<td>N/A</td>
<td>Good</td>
</tr>
<tr>
<td>TempMonitorPeriod</td>
<td>Analog</td>
<td>90</td>
<td>Good</td>
</tr>
<tr>
<td>Threshold</td>
<td>Analog</td>
<td>N/A</td>
<td>Good</td>
</tr>
</tbody>
</table>

For more information about data sources and data items, see the following:
Modifying the Sample Gateway Agent Project

The sample gateway agent project contains the configuration required to run the alarms workflow example.

If you want to use the sample builder project to create the onboarding configuration, you need to do the following:

- specify the hostname of your EMS server in the agent builder project
- add an empty `xgDeployConfig.xml` file
- include the `EDD_TEXT.xml` file in the onboarding tarball

To access the sample project files, you first need to build the rootfs with the `wr-ems-demo` layer enabled. For more information, see Building and Deploying Your Platform Project on page 12.

Step 1  Copy the following file to your Windows host:

```
projDir/demos/sim_sensor.demo_builder.tar.gz
```

Step 2  On your Windows host, unpack the tarball.

The following commands assume that you installed the `bsdtar` package and added the path to your Windows PATH environment variable. For more information, see Using a Windows Machine on page 22. At the Windows command-line, type the following:

```
C:\> cd C:\Axeda\Builder
C:\Axeda\Builder> mkdir sim_sensors
C:\Axeda\Builder\sim_sensors> cd sim_sensors
C:\Axeda\Builder\sim_sensors> bsdtar -xzf sim_sensor.demo_builder.tar.gz
```

Step 3  To open the sample project in the Axeda Builder, double-click the `SimSensorDemo.gwsp` file.

The Axeda Builder starts and opens the gateway project file.

Step 4  In the Project Window pane, expand the `SimSensor` project, and click `Configuration`.

Step 5  In the Configuration pane, double-click `Enterprise Servers`.

The Enterprise Servers Settings dialog appears.

Step 6  In the Servers list, double-click `null-sandbox.axeda.com`.

The Enterprise Server Definition dialog appears.

Step 7  In the `Name` box, type a descriptive name for the EMS Server.

Step 8  In the `Host` box, type the EMS Server hostname you use when you build the rootfs, and click `OK`. 

---

The `Default_Logger` data logger includes all the data items in the gateway project. For more information about loggers, see the following:

- Axeda Builder User’s Guide, 6.6.1: Configuring Projects to Collect Data: Configuring Data Loggers
- Creating Data Loggers on page 21
When you build the rootfs, you use the `iot-config.py` script to specify the hostname. You must use the same hostname when you configure the EMS Server.

In the hostname box, do not specify the protocol (https).

**Step 9** If needed, change any other project settings.

**Step 10** Click OK.

**Step 11** Click File > Save All.

You have finished modifying the project.

**Step 12** In the directory where you unpacked the `sim_sensor.demo_builder.tar.gz` tarball (for example C:\Axeda\Builder\sim_sensors), create a new empty file named `xgDeployConfig.xml`.

**Step 13** Copy the `EDD_TEXT.xml` file from your Linux host to the directory on your Windows host where you unpacked the `sim_sensor.demo_builder.tar.gz` tarball.

The `EDD_TEXT.xml` file is located in the following directory on your Linux host:

`projDir/layers/wr-idp/wr-ems-demo/device/lx/demos/sim_sensor.demo/config`

**Step 14** Create a tarball with the configuration files.

For more information, see *Using a Windows Machine* on page 22.

You are now ready to onboard with your gateway configuration.

---

**Onboarding with the Alarms Example Configuration**

Onboarding with the configuration from the sim_sensors demo configures the asset with the data items required to create the temperature alarm actions and rules on the EMS Server.

To configu‌‌‌re the device, you need to onboard the device with the configuration files in the sim_sensors demo. If you already onboarded the device, you need to onboard again with a new configuration to use the items in the demo.

To successfully onboard the device, ensure that you first boot the device with the rootfs that includes the `wr-ems-demo` layer.

**Step 1** On the EMS Server, create a package using one of the following tarballs.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use the tarball created when you build the rootfs.</td>
<td>The tarball is located in the following directory: <code>projDir/demos/sim-sensor.demo.tar.gz</code>&lt;br&gt;When you build the rootfs, ensure that you run the <code>iot-config.py</code> command to specify the EMS Server name. Otherwise, the server configuration file in the tarball uses the default value and onboarding will fail.</td>
</tr>
<tr>
<td>Use the tarball you created from the sample gateway agent project you modified.</td>
<td>For more information, see <em>Modifying the Sample Gateway Agent Project</em> on page 55.</td>
</tr>
</tbody>
</table>
For more information, see Creating an Onboarding Package on page 26.

**Step 2** Deploy the onboarding tarball you chose.
For more information, see Deploying an Onboarding Package Manually on page 28.

When the onboarding process completes, the data items for the sim_sensors demo appear in the Data section of the asset dashboard.

---

**Starting and Stopping Temperature Monitoring**

A script starts the temperature monitoring program on the device and starts the flow of data from the asset to the ESM Server.

To start the flow of temperature data, you need to run the temperature monitoring script. When you build the rootfs with `wr-ems-demo` layer enabled, the build process automatically includes the following sequence file in the rootfs that the EMS agent can run to start temperature monitoring:

```
/opt/agent/scripts/watchtempstart.seq
```

You can trigger the script manually from the asset dashboard on the EMS Console, or you can create an action and an expression rule to run it automatically after the asset finishes the onboarding process.

This example describes how to start and stop temperature monitoring manually.

**Step 1** Log in to the EMS Server.

**Step 2** On your asset dashboard, verify that the `watchtempstart` script appears in the Scripts section on the right side of the page.
You may need to click View All to see the complete list.

**Step 3** Beside the `watchtempstart` script, click Run.

The Recent Actions section of the asset dashboard updates to show the progress of the script. When the script is running, the Scripts section of the dashboard shows the following:

<table>
<thead>
<tr>
<th>Scripts</th>
<th>View all</th>
</tr>
</thead>
<tbody>
<tr>
<td>watchtempstart</td>
<td>Stop</td>
</tr>
<tr>
<td>watchtempstop</td>
<td>Run</td>
</tr>
<tr>
<td>agent-update</td>
<td>Run</td>
</tr>
<tr>
<td>agent-update-status</td>
<td>Run</td>
</tr>
<tr>
<td>exec_sw_mgr</td>
<td>Run</td>
</tr>
</tbody>
</table>

You may need to refresh the page to update the Scripts section.

**Step 4** To stop temperature monitoring, follow the same procedure to run the `watchtempstop` script.
If you click **Stop** beside the **watchtempstart** script, the temperature monitoring script does not clean up properly.

**Postrequisites**

To successfully trigger an alarm, ensure that you start temperature monitoring.

**Creating the Alarm Expression Rule**

The alarm expression rule raises an alarm if the temperature exceeds a specified threshold.

To trigger the alarm, you need to compare the temperature data transmitted to the EMS Server through the **temperature** data item to a threshold value that is defined using the **Threshold** data item, and raise an alarm if the condition is satisfied.

This work flow example uses an expression rule to evaluate the temperature data and uses an action to trigger the alarm.

For more information about the syntax of the expression in expression rules, see the **Help** menu.

---

To trigger the alarm, ensure that you start temperature monitoring.

**Step 1** Log in to the EMS Console.

**Step 2** Click **Manage**.

**Step 3** From the **New** menu, select **Expression Rule**.

The Configure The Expression Rule page appears.

**Step 4** In the **Name** box, type a name for the rule.

Do not use spaces in the name. For example, use the name **Alarm_Trigger_Rule**.

**Step 5** From the **Type** menu, select **Data**.

**Step 6** In the **Description** box, type a short description of the rule.

**Step 7** To select the assets to which to apply this rule, click **Apply to Assets**.

**Step 8** In the dialog box that appears, in the search box, type then name of the model you used when you built the rootfs, for example, **SIM_SENSOR_DEMO**, and then click the search icon.
The model appears in the Model list.

**Step 9** Click **Add** and then click **OK**.

**Step 10** In the **If:** box, type the following:

```
DataItem.temperature.value > DataItem.Threshold.value
```

**Step 11** In the **Then:** box, type the following:

```
CreateAlarm("alarmName", 100, "Alarm for Temperature crossing Threshold", "temperature")
```

The **temperature** parameter is the value of the temperature data item, which appears as the data value in the alarm entry.

**Step 12** To verify syntax of the expression, click **Validate** (optional).

**Step 13** Click **Save** and then click **OK**.

On the asset dashboard for your asset, alarms appear in the Alarms section when the value of the **temperature** data item (appears in the Data section) is above the value of the **Threshold** data item.

---

### Updating the Temperature Alarm Threshold

An action created at the EMS Console can change the value of the temperature threshold for the alarm.

You can create an action at the EMS Console to update the value of a data item on the asset. To change the value of the data item, ensure that you select **Writable** in the Axeda Builder when you create the data item.

This example shows how to change the value of the temperature threshold at which an alarm is triggered.

**Step 1** Log in to the EMS Server.

**Step 2** Verify that the **Threshold** data item is listed in the Data section on the asset dashboard.

You may need to click **Current** to see the complete list.

**Step 3** To create an action, from the **Manage** list, click **New > Action**. The Create Action page appears.

**Step 4** Create the action.

a) In the **Name** box, type a name for the action.

   Do not use spaces in the name. For example, use the name **SetTempThreshold**.

b) In the **Category** list, select **Asset update actions**.

c) Select **Set data item on the asset** and then click **Next**. The Models for Action page appears.
Step 5 Select the check box next to your model from the list and then click Add Checked. To find your model, you may need to use the arrows to advance through the model list. The model appears in the Selected Models list.

Step 6 Click Next. The Configure Set Data Item Action page appears.

Step 7 From the Data Item list, select the data item Threshold.

Step 8 In the Value box, type an integer value for the data item. You can change the value from the default value 90 to raise or lower the alarm threshold.

Step 9 Click Next. The User Groups That Handle Edit Permissions for Action page appears.

Step 10 Click Finish. The Confirm Changes to Action page appears.

Step 11 Click Finish.

Step 12 To verify that the action is available for your device, in the Actions section of the asset dashboard of your device, click View all. The Actions page appears and the action you created appears in the list.

Step 13 To run the action, in the Actions list, click the action you created in step 4.a on page 59. The confirmation dialog appears.

Step 14 Click Execute. The Recent Actions section on the asset dashboard shows the progress of the action. When the action completes, in the Data section of the asset dashboard, the Threshold data item shows the new value.

Step 15 To verify that the new threshold value is in effect, in the Alarms section of the asset dashboard, click Historical. The values in the Data Value list are now above the new threshold. For example, if the threshold value changed from 90 to 75, values above 75 now appear in the list.

Sending an Alarm Notification Email

Creating a Custom Object

You can create a custom object (a Groovy script) to call when the temperature alarm is triggered. For more information about Groovy scripts, see Axeda Platform API Developer’s Reference SDK v1.0 and v2.0, 6.1.6.
Step 1 Log in to the EMS Console.

Step 2 To create a custom object, from the Manage, select New > Custom Object.

The Create Custom Object page appears.

Step 3 In the Name box, type a name for the object.

Do not use spaces in the name. For example, use the name EmailNotifyScript.

Step 4 From the Type list, select Action.

Step 5 In the Description box, type a description for the custom object.

For example, type Send an email when the temperature alarm triggers.

Step 6 In the Source Code box, type the following:

```java
import com.axeda.drm.sdk.contact.Email
import com.axeda.drm.sdk.device.*

String fromaddress = parameters.fromaddress
String toaddress = parameters.toaddress
String alarmName = parameters.alarmName
String alarmDescription = parameters.alarmDescription
String alarmSeverity = parameters.alarmSeverity
String alarmDataItem = parameters.alarmDataItem
String alarmDataValue = parameters.alarmDataValue
String device = parameters.device
String subject = "Axeda Alarm '{alarmName}' for '{device}' device"
String body = "Your IoT '{device}' device has generated a '{alarmName}'
Alarm:
{alarmDescription}
Severity: {alarmSeverity}
{alarmDataItem}: {alarmDataValue}"

Email.send(fromaddress, toaddress, subject, body)
```

You can view the code for the Groovy script in the following file:

`projDir/layers/wr-idp/wr-ems-demo/device/lx/demos/sim_sensor.demo/Readme.txt`

Step 7 To add the parameters, click Configure parameters.

The Custom Object Parameters page appears in a new window.

Step 8 Add the script parameters.

The parameter names correspond to the parameters the Groovy script uses. For each of the following parameters, in the Variable Name box and the Display Name box, type the parameter name and then click Add.

- fromaddress
- toaddress
- alarmName
- alarmDescription
- alarmSeverity
- alarmDataItem
- alarmDataValue
- device

Step 9 Click Save Changes.

Step 10 On the Edit Custom Object page, click Finish.
Step 11 On the Commit Custom Object page, click Finish.

Creating an Expression Rule

An expression rule can use a custom object to send an email when an alarm occurs.

Create an expression rule to execute the custom object you created (see Creating a Custom Object on page 60) to send an email when the temperature alarm is triggered.

Step 1 Log in to the EMS Console.

Step 2 To create an Expression Rule, from the Manage menu, select New > Expression Rule.

The Configure The Expression Rule page appears.

Step 3 In the Name box, type a name for the rule.

For example, use the name AlarmEmailNotify. Do not use spaces in the name.

Step 4 From the Type menu, select Alarm.

Step 5 In the Description box, type a short description of the rule.

Step 6 To select the assets to which to apply this rule, click Apply to Assets.

Step 7 In the dialog box that appears, in the search box, type the name of the model you used when you built the rootfs, for example, SIM_SENSOR_DEMO, and then click the search icon.

The model appears in the Model list.

Step 8 Click Add and then click OK.

Step 9 In the If: box, type the following:

```plaintext
Alarm.name == "alarmName"
```

The value alarmName must match the name of the alarm you created in Creating the Alarm Expression Rule on page 58.

Step 10 In the Then: box, type the following expression.

```plaintext
ExecuteCustomObject("yourCustomObjectName", "fromEmailAddress", "toEmailAddress", 
Alarm.name, Alarm.description, Alarm.severity, 
Alarm.dataItem, Alarm.value, Device.serial)
```

For the value of yourCustomObjectName, use the name of the custom object you created in Creating a Custom Object on page 60.

Step 11 To verify syntax of the expressions, click Validate (optional).

Step 12 Click Save and then click OK.

When the temperature alarm triggers, you receive an email.
About the deploy.sh Script

**NOTE:** The `deploy.sh` script is intended for use only with USB storage devices. The script does not support SD card deployment.

For a detailed list of options for `deploy.sh`, execute the script with the `-h` option:

```
$ ./deploy.sh -h
```

For detailed information on using the `deploy.sh` script, see the README file located at: `projDir/layers/wr-idp/wr-idp-devkit/recipes-devtools/deploy-tool/files`.

About the iot-config.py Script

The `iot-config.py` script must be used to specify the EMS Server hostname, and, optionally, a different model name, serial number, or port number than the default value.

The `iot-config.py` script, which must be run before the `make fs` command, results in a final build image that is fully configured and ready to deploy. This script must be run from your project build directory (`projDir`).
The options on the `iot-config.py` script are:

- **-h, --help**
  Displays a help message and exits.

- **-s "string", --string="string"**
  Inputs a comma-separated string of key:value pairs, where:
  - **key** can be HOST, MODEL, SERIAL_NUMBER, or PORT (in upper case).
  - **value** can be any string but cannot contain commas or line breaks.
  No validation is performed on the key:value pairs.

- **-p password, --password password**
  Specifies a password for root.

- **-r integer, --salt integer**
  Specifies the salt used to encrypt password.
  Must be used in conjunction with the -s "string" or --string="string" option.

- **-d, --display**
  Displays the current key:value pairs.

- **-c, --clean**
  Removes the key:value pairs specified on the command line.
  Must be used in conjunction with the -s "string" or --string="string" option.

- **-t**
  Specifies that the sshd service be enabled at boot time in the built image.

You can run the `iot-config.py` script multiple times within the same build project directory to create different images that have different EMS Server settings. When the `make fs` command is re-run, only packages that reference those settings are rebuilt.

For example, to use your build image for multiple serial numbers:

```
$ cd projDir
$ iot-config.py -s "HOST:my-sandbox.axeda.com,SERIAL_NUMBER:00001"
$ make fs
$ iot-config.py -s "SERIAL_NUMBER:00001" -c
$ iot-config.py -s "HOST:my-other-sandbox.axeda.com,SERIAL_NUMBER:00002"
$ make fs
```

For more information on the `iot-config.py` script, run the following command:

```
$ cd projDir/layers/wr-idp/wr-ems/scripts
$ pydoc iot-config
```
Supported JSON File Content

Specific attribute-value pairs are supported in the JSON file used for RPM updates. The JSON file used for RPM updates has the following attribute-value pairs:

```
{
  "identifier": "",
  "operation": "install | local-install | remove | purge | upgrade | download",
  "channelMode": "append | override",
  "channels": [{
    "alias": "some_alias",
    "name": "some_name",
    "type": "valid_smart_channel_type",
    "url": "repo_url",
    "priority": "valid_priority_value",
    ...
  }, {...}],
  "packages": [{"name": "pkg_name"}, {...}],
  "rebootOnCompletion": "yes | no",
  "rebootDelay": "minutes"
}
```

For more information on JSON syntax, see [http://json.org/](http://json.org/)

Attributes and their supported values are described in the table below.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>operation</td>
<td>install</td>
<td>Download and install packages</td>
</tr>
<tr>
<td></td>
<td>local-install</td>
<td>Install local packages without download. When this value is used, the channelMode attribute is not required. When this value is not used for the operation attribute, then the following attributes must be specified for channels: alias, type, url.</td>
</tr>
<tr>
<td></td>
<td>remove</td>
<td>Uninstall packages</td>
</tr>
<tr>
<td></td>
<td>purge</td>
<td>Purge packages and their configurations/downloaded content</td>
</tr>
<tr>
<td></td>
<td>download</td>
<td>Download packages only; do not install</td>
</tr>
<tr>
<td></td>
<td>upgrade</td>
<td>Upgrade specified/existing packages</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For example, if you are running version 1 of an RPM, and you want to upgrade to version 2, set operation to upgrade.</td>
</tr>
<tr>
<td>channelMode</td>
<td>append</td>
<td>Append specified channel to existing list (default mode)</td>
</tr>
<tr>
<td></td>
<td>override</td>
<td>Use specified channel to replace existing ones</td>
</tr>
<tr>
<td>channels</td>
<td>alias</td>
<td>Alias for the channel name</td>
</tr>
<tr>
<td></td>
<td>name</td>
<td>Name of the channel (repository)</td>
</tr>
<tr>
<td>Attribute</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>type</strong></td>
<td></td>
<td>Other attributes may be mandatory based on the value of this attribute. For example, <strong>components</strong> must be specified if <strong>type</strong> is <strong>apt-rpm</strong>.</td>
</tr>
<tr>
<td><strong>url</strong></td>
<td></td>
<td>URL of the channel (repository)</td>
</tr>
<tr>
<td><strong>priority</strong></td>
<td></td>
<td>Numerical priority of channel. If <strong>operation</strong> is <strong>upgrade</strong>, set <strong>priority</strong> to a value greater than 0, or greater than the system channels, or remove <strong>priority</strong> altogether. If <strong>priority</strong> is lower than the priority of the system RPM, then the response from the EMS Agent is &quot;no interesting upgrades available&quot;.</td>
</tr>
<tr>
<td><strong>packages</strong></td>
<td><strong>name</strong></td>
<td>Name of the RPM</td>
</tr>
<tr>
<td><strong>rebootOnCompletion</strong></td>
<td><strong>yes</strong></td>
<td>Reboot the asset upon completion</td>
</tr>
<tr>
<td></td>
<td><strong>no</strong></td>
<td>Do not reboot the asset upon completion</td>
</tr>
<tr>
<td><strong>rebootDelay</strong></td>
<td></td>
<td>Delay the rebooting of the asset by the specified number of minutes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If <strong>rebootOnCompletion</strong> is set to <strong>yes</strong>, <strong>exec_sw_mgr</strong> reboots the asset <strong>rebootDelay</strong> minutes after the configured operations are completed successfully.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The value of <strong>rebootDelay</strong> should be at least 3 times the ping rate of the asset and/or enough time to allow the log files to be uploaded to the cloud. The default value of this attribute is 10 minutes.</td>
</tr>
</tbody>
</table>