WNDRVR
SUPPORT NETWORK

WIND RIVER HELIX DEVICE CLOUD AGENT PROGRAMMER'S GUIDE, 2.3

Corporate Headquarters

Wind River
500 Wind River Way
Alameda, CA 94501-1153
U.S.A.
Toll free (U.S.A.): +1-800-545-WIND
Telephone: +1-510-748-4100
Facsimile: +1-510-749-2010

For additional contact information, see the Wind River website:

www.windriver.com

For information on how to contact Customer Support, see:

www.windriver.com/support

Wind River Helix Device Cloud Agent Programmer's Guide, 2.3

27 January 2019

1. WIND RIVER HELIX DEVICE CLOUD AGENT PROGRAMMER'S GUIDE, 2.3

2. HELIX DEVICE CLOUD OVERVIEW

1. Introduction to Helix Device Cloud

?The Wind River Helix Device Cloud (HDC) agent enables distributed management of devices from cloud-based applications.

Helix Device Cloud includes components for device-side and cloud-side support. The agent on the device enables cloud connectivity to facilitate data capture, rules-based data analysis and response, and configuration. The cloud supports RESTful APIs to enable you to build end-to-end IoT solutions and a browser-based management console to manage devices.

Devices run the agent to enable applications to securely transmit telemetry to the server and create custom device management actions. The agent provides basic device management and configuration. For supported operating systems, the device also includes the ?MRAA sensor library to simplify sensor integration into your application.

The management console displays the telemetry received from the device and provides an interface to execute the list of custom actions the application implements on the device. The management console provides the following:

- rules and alerts that can be triggered based on telemetry and device properties
- package management and deployment for remote software updates
- remote login to devices
- file transfer between the device and the server

The administration utility provides the following:

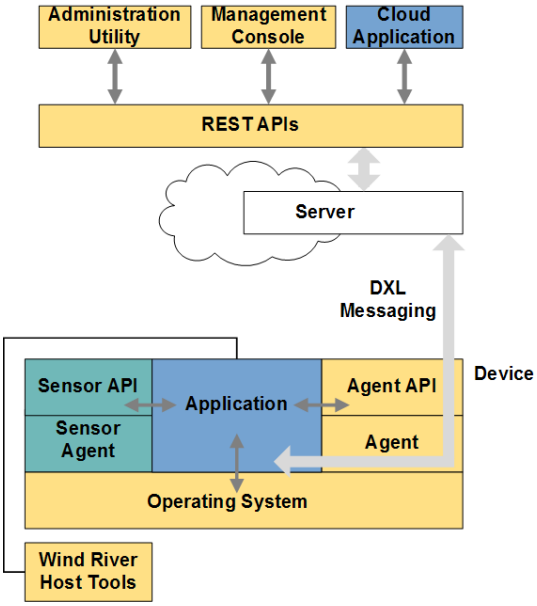
- user account administration to control access and permissions to administrative functions

The Helix Device Cloud agent enables you to build IoT applications using the following operating systems:

- Wind River Linux 7.0 and Wind River Linux 8.0
- Wind River Intelligent Device Platform XT
- Wind River VxWorks 7
- Windows 7 and Windows 10 32-bit
- Ubuntu 16.04

To successfully build applications for the Helix Device Cloud agent on the device, you need to know how to use the supported operating systems. To find the prerequisite information about the Wind River operating environments, see Where to Find Information on page 5.

The following figure shows the components in the end-to-end IoT solution and the basic device-side software architecture. Wind River Host Tools enable you to develop your application for Wind River operating systems on your host computer. The sensor agent is only available on Wind River Linux and IDP XT.



Connectivity and Registration

The device initiates the connection with the server and sends device registration information, capabilities, and configuration to the server. The configuration information enables two-way authentication between the server and the device. The server response provides acknowledgment of the device to a specified tenant account. Devices can reregister to advertise updated device capabilities.

The agent on the device communicates with the server using the DXL (Data Exchange Layer) publish and subscribe message protocol secured with TLS 1.2.

Firewall Exceptions

To enable connectivity between the device and server, ensure that you open the outbound port 443 on your network if you do not user a proxy server.

Proxy Server Support

The agent can connect to the server through a proxy server. You configure the proxy settings after you boot your device or optionally, at installation time on Windows. For more information, see the following:

Connecting Your Wind River Linux and IDP XT Device to the Server
Connecting Your Ubuntu Device to the Server
Installing the HDC Agent on Windows
Connecting Your Windows Device to the Server

Certificates and Credentials

You receive the following credentials for Helix Device Cloud:

- your tenant administrator user name and password

You need this information to log in to the following:

the management console at https://www.helixdevicecloud.com
the administration utility at https://admin.helixdevicecloud.com

For more information, see *Wind River Helix Device Cloud Getting Started*.

2. Helix Device Cloud Agent

The agent services requests from user applications and the server.

Each device runs a single instance of the agent, which can handle requests from multiple applications.

Applications use the agent APIs to do the following:

- create telemetry objects to transmit telemetry data from the device to the server
- create actions to provide a set of custom device management actions that the server can execute on the device

Each application provides its own telemetry and actions.

Agent Services on Linux

The following services on Ubuntu, Wind River Linux, and Wind River IDP XT, which are controlled by the systemd initialization system, provide connectivity and basic device management functionality:

- iot-device-manager
- iot
- iot-ccg
- iot-mux
- mosquitto

Use the iot-control command to start, stop, restart, and query the status of the services. On the command-line, type iot-control --help for more information.

Use the journalctl -u *serviceName* command to view log files for each service.

You may need superuser privileges to run the commands.

For more information about systemd, see https://www.freedesktop.org/wiki/Software/systemd/.

Agent Services on Windows

The following services, which are managed from the Services Management Console (the Services program), provide connectivity and basic device management functionality:

- Internet of Things Core Service
- Internet of Things Device Manager
- Internet of Things Connection Gateway
- Mosquitto Broker

Use the iot-control command from a Command Prompt window to start, stop, restart, and query the status of the Internet of Things services. On the command-line, type iot-control --help for more information.

Log files are available in the **C:\ProgramData\Wind River Systems\Helix Device Cloud** folder.

Agent Tasks on VxWorks 7

The following tasks provide connectivity and basic device management functionality:

- tCcgBroker
- thdcAgent
- thdcDeviceMgr
- tMosquitto

To restart the services, you must reboot the device.

3. Where to Find Information

Documentation is available through the Wind River Knowledge Library.

The following documentation is available:

Wind River Helix Device Cloud Documentation

Wind River Helix Device Cloud Getting Started

Provides instructions to install the HDC agent on a host computer and information about getting started with the platform.

Wind River Helix Device Cloud Release Notes

?Provides general product information about Helix Device Cloud, changes in this release, usage caveats, and known problems.

Wind River Helix Device Cloud Management Console User's Guide

Provides information about performing device and platform management tasks.

Wind River Helix Device Cloud Administration Utility User's Guide

Provides information about performing platform administration tasks.

Wind River Helix Device Cloud Platform Programmer's Guide

Provides information about writing Web-based applications for Helix Device Cloud.

Wind River Helix Device Cloud Interactive REST API Reference

Provides reference information for the platform REST API and an environment to explore the APIs.

Wind River Helix Device Cloud Agent Programmer's Guide

Provides instructions for configuring devices to run the agent and to write Helix Device Cloud applications.

Wind River Helix Device Cloud Agent Configuration and Build Guide

Provides instructions to build images that include the agent for target devices that run Wind River operating systems.

Wind River Helix Device Cloud Troubleshooting Guide

Provides information to help resolve issues with Helix Device Cloud.

Wind River Helix Device Cloud Agent API Reference

Provides reference information for the agent API.

Wind River Linux Documentation

The following documents are available for both Wind River Linux 7.0 and 8.0:

?Wind River Linux Getting Started Guide

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

Wind River Linux Platform Developer's Guide

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

?Wind River Linux Getting Started Workbench Tutorials

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

Wind River Linux User Space Developer's Guide

Provides information about using the Wind River Linux SDK to develop Linux user space applications.

Wind River Intelligent Device Platform XT

The following documents are available for Wind River IDP XT:

Wind River Linux Intelligent Device Platform XT Programmer's Guide, 3.1

?Provides instructions for installing and configuring IDP XT and modifying it for your specific requirements.

?Wind River Intelligent Device Platform XT Security Guide, 3.1

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

Wind River Intelligent Device Platform XT Release Notes, 3.1

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

VxWorks 7

The following documents are available for VxWorks 7:

VxWorks 7 Getting Started Guide

Provides information about getting started with development on VxWorks 7.

VxWorks 7 Release Notes

Provides general product information, changes in the VxWorks 7 components, and known problems.

VxWorks 7 Configuration and Build Guide

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

Accessing Documentation

To access the Helix Device Cloud documentation on the Knowledge Library (http://knowledge.windriver.com), select **Products > Internet of Things > Helix Device Cloud 2**.

To access the interactive REST API Reference, go to https://www.helixdevicecloud.com, click the **Help** button in the upper-right corner, and select **REST API Reference**.

3. APPLICATION DEVELOPMENT

1. Helix Device Cloud Applications

?To transmit and receive information between the device and the server, you need to write an application to run on the device that communicates with the server through the agent.

The agent provides C APIs to enable you to create applications. You can also write applications using languages supported through language bindings. For more information, see Python Bindings on page 24.

On Wind River Linux and IDP XT, the device also provides the open source MRAA I/O library to simplify sensor data collection. For more information about the library, see http://iotdk.intel.com.

API Header Files

On Windows and Ubuntu, the host (development computer) and device may or may not be the same computer.

On Windows, you must select the **Development** component at installation time to install the header and library files.

Header files for each supported operating system are located as follows:

Operating System	Location on Host	Location on Device
Wind River Linux and IDP XT	Agent *projDir* **/build/iot/git/src/api/public/iot.h** MRAA I/O library *projDir* **/bitbake_build/tmp/sysroots/***boardName* **/usr/include/mraa.h** *projDir* **/bitbake_build/tmp/sysroots/***boardName* **/usr/include/mraa** after building the platform project	Agent **/usr/include/iot.h** if the platform project was built with the --with-template=feature/self-hosted template and --with-package=iot-dev package
VxWorks 7	*installDir* **/vxworks-7/pkgs/app/hdc-***releaseNum* **/wr-iot/src/api/public/iot.h**	N/A

Operating System	Location on Host	Location on Device
Windows	*installDir* **\include\iot.h** if you select the **Development** component at installation time	*installDir* **\include\iot.h** if you select the **Development** component at installation time
Ubuntu	**/usr/include/iot.h**	**/usr/include/iot.h**

Libraries

Link your application against the agent library for your operating system and include the library on your device at run time as follows:

Operating System	Compile-time Library	Run-time Library
Wind River Linux and IDP XT	Agent **libiot.so** MRAA I/O library **libmraa.so**	Agent **libiot.so** MRAA I/O library **libmraa.so**
Windows	**iot.lib**	**iot.dll**
Ubuntu	**libiot.so**	**libiot.so**
VxWorks 7	Not applicable.	Not applicable.

Source Code

Source code is provided for a subset of operating systems as follows:

Operating System	Location
Wind River Linux and IDP XT	*projDir* **/build/iot/git** after you build the platform project on your host computer
VxWorks 7	*installDir* **/vxworks-7/pkgs/app/hdc-**_releaseNum_ after you install VxWorks 7

File Transfer Support

Your application can access files sent from the server to the device and can add files to specified directories to enable the server to retrieve them.

By default, the agent searches for files to retrieve in a predefined, operating system-specific directory. You can add to the list of directories the agent searches and configure the agent to leave the files in the directory instead of deleting them after retrieving the files successfully. For more information, see Agent Configuration on page 37.

To enable the server to retrieve files from your application, put the files in the predefined, operating system-specific directory or the additional directories, if configured. When the server runs the **Retrieve Files** action, the agent sends all files (subdirectories are not included) in the directories to the server and then, by default, deletes the files from the directory.

The server is not notified when new files are added to the directory. Your application may need specific permissions to write to the directory, and you must ensure that all files have read permission enabled for all users.

The default directory locations are as follows:

Operating System	Location
Wind River Linux and IDP XT Ubuntu	**/var/lib/iot/upload**
Windows	**C:\ProgramData\Wind River Systems\Helix Device Cloud\upload**

To receive files from the server, check the predefined, operating system-specific directory periodically in your application. Your application must know in advance what files to check for, and the agent does not notify applications when new files are added to the directory. You can write a custom action that the server can run to notify your application that it has sent files to the device. If the server downloads a file with the same name as an existing file in the directory, the existing file is overwritten.

The directory locations are as follows:

Operating System	Location
Wind River Linux and IDP XT Ubuntu	**/var/lib/iot/download**
Windows	**C:\ProgramData\Wind River Systems\Helix Device Cloud\download**

Application Development on Wind River Linux and IDP XT

You need to write an application that runs in Linux user space. You must ensure that your application is compatible with any security features enabled on the device.

To build your application in the platform project directory you created, you need to write a recipe file to compile it, create an RPM, and then add it to the rootfs using a layer. When you build the rootfs, your application is included in the image you use to boot the device. During development, you can include your application in the *projDir* **/layers/local** directory. Once you are ready to release your application, you need to create your own layer. For more information, see the following:

Wind River Linux Developer's Guide, 7.0*: Customizing the Platform*
Wind River Linux Developer's Guide, 8.0*: Customizing the Platform*

You can also build applications using the Wind River Linux SDK you receive from a platform developer.

To install your application on the device, you can do one of the following:

- If you include your application in the rootfs in your own layer, follow the procedure to deploy the images to your boot media and boot your board. For instructions, see the following:
  Deploying Your Wind River Linux Platform Project for Intel Target Devices
  Deploying Your Wind River Linux Platform Project to an ARM Target Device
  Deploying Your Wind River IDP XT Platform Project

- Install the application binary executable or the RPM using the software update process. During the test and modification iterations of developing your application, modify your package to use new versions of your files. For more information, see Software Update on page 31.

  If the underlying operating system is IDP XT and you have the wr-srm layer enabled (the default), you must sign the RPM first.

For more information about Linux applications, recipes, layers, using an SDK, and deploying an application, see the following:

Wind River Linux Platform Developer's Guide, 7.0
Wind River Linux Platform Developer's Guide, 8.0
Wind River Linux User Space Developer's Guide, 8.0
Wind River Linux User Space Developer's Guide, 7.0

For more information about running applications under IDP XT with security features enabled, see the Wind River Intelligent Device Platform XT Programmer's Guide.

Application Development on Windows

You need to write an application that runs in user space. Visual Studio is the recommended development environment. The agent was verified using Visual Studio 2015; however, newer releases should also work.

For applications written in any language, if you did not add the Helix Device Cloud installation folder to the system path either during installation or manually afterward, the **iot.dll** library and the application must be in the same folder to run the application successfully.

When you specify a value for an 64-bit integer parameter, you must cast it as a 64-bit integer as shown in the following example:

```
iot_telemetry_publish(temperature, NULL, 0u, IOT_TYPE_INT64, (iot_int64_t)12 );
```

This restriction applies to any agent API that accepts integer values.

Application Development on VxWorks 7

You need to write an application that runs in kernel space. For information about writing and building applications for VxWorks 7, see the following:

go to VxWorks 7 Getting Started and select **Tutorials**
VxWorks 7 Programmer's Guide

Application Development on Ubuntu

You need to write an application that runs in user space and you must install make, gcc, and glibc on your development computer. Other C compilers should also work.

2. Registering Your Application with the Agent

You must register your application with the agent to use the rest of the agent application functions.

?You must have a valid agent handle to use when you register telemetry and actions.

Optionally, you can register a callback function to receive the agent log messages generated using the **IOT_LOG** macro. You can also use the **IOT_LOG** macro to generate your own application log messages. Your log handler receives all the logs in your log handler, which can help you debug your application.

Procedure

1. Retrieve the agent handle.

   Specify a name for the application in the id parameter. Specify **NULL** for the app_path parameter and zero for the flags parameter.

   ```
   ?iot_t *agentHandle;
   agentHandle = iot_initialize("myApp", NULL, 0);
   ```

2. Optionally, register a callback to receive log messages.
   1. Write the log handler function.

      Use the following function signature:

      ```
      void log_handler( iot_log_level_t log_level, iot_log_source_t *log_source, const char *messa
      ge, void *user_data )
      ```

   2. Register the callback function.

      ```
      iot_log_callback_set(agentHandle, &amp;log_handler, NULL );
      ```

3. Connect the application to the agent.

   Specify the value returned from the iot_initialize function for the handle parameter and the time in milliseconds to wait for a successful connection.

   Specify zero for the max_time_out parameter to wait indefinitely for the agent to connect.

   ```
   result = ?iot_connect(agentHandle, 0);
   ```

   The application blocks until the agent successfully connects to the server at least once.

You are now ready to register actions and telemetry with the agent.

Example: Basic Application Initialization

```
static iot_t *initialize( void )
{
    iot_status_t result = IOT_STATUS_FAILURE;

    iot_t  *agentHandle = iot_initialize("appName", NULL, 0u);

    if (agentHandle)
        {
        result = iot_connect(agentHandle, NULL);

        if (result == IOT_STATUS_SUCCESS) {
            /* Continue with application initialization
               Register actions and metrics as required.
            */
        }
        if (result != IOT_STATUS_SUCCESS)
```

```
        {
            fprintf( stderr, "Error connecting to IoT service: %s\n", iot_error( result ) );
        }
    }
    else
    {
        fprintf( stderr, "Failed to retrieve agent handle\n" );

    return agentHandle;
}
```

Postrequisites

When your application terminates, call the iot_disconnect and iot_terminate functions to disconnect your application from the agent and free the associated memory.

3. About Telemetry

Telemetry is device-specific data that the application collects and publishes to the agent for transmission to the server.

?To transmit telemetry data to the server, you register telemetry objects with the agent. The agent supports 255 or fewer telemetry objects per device, subject to memory constraints on the device. Valid data types for telemetry metrics are Boolean, float32, float64, int32, int64, int16, int8, uint8, uint16, uint32, uint64, string, and raw.

For int64 data types the valid range is 9007199254740991 through -9007199254740991.

For uint64 data types, the maximum value is 9007199254740991 and larger sample values may appear as negative numbers on the server when viewed on the management console or retrieved using the REST APIs.

When the agent transmits the telemetry to the server, it is visible on the management console based on the application name you registered with the agent in the iot_initialize function.

When the application publishes a telemetry sample to the agent, the agent queues the sample for transmission. The agent queue size is fixed, and the queue size is not configurable.

The agent collects a minimum number of telemetry samples before transmitting the samples to the server. The agent also limits the maximum number of samples to send in a single transmission. The agent sends samples at intervals of one second.

By default, the agent specifies the current time on the device as the timestamp of the data sample. Optionally, you can specify a timestamp for the sample. You may want to specify a different time if there is a delay between the time you collect the data and the time you publish it to the agent.

Optionally, the application can call iot_telemetry_attribute_set to specify other information about the telemetry, such as the units. If you specify the telemetry units, the server and the application must share a common interpretation of the units.

The server stores telemetry data for 90 days.

Best Practices for Sending Telemetry

To optimize telemetry processing on the server, observe the following recommendations:

- Send telemetry samples only when information changes or specific events occur.
- Send one sample per minute or fewer.

- For bursts of data, send samples at a maximum rate of once per second, and limit this sample rate to five minutes or less.
- Do not send more than 600 samples within a 15 minute period.

Telemetry APIs

Function	Description
?iot_telemetry_allocate	Create a telemetry object and allocate the required memory.
?iot_telemetry_free	Destroy a telemetry object and free the associated memory.
?iot_telemetry_register	Register a telemetry object with the agent.
?iot_telemetry_deregister	Deregister a telemetry object from the agent.
?iot_telemetry_publish	Publish a telemetry sample to the agent.
?iot_telemetry_publish_raw	Publish a raw data sample to the agent.
iot_telemetry_attribute_set	Specify additional information about the telemetry data.
iot_telemetry_attribute_set_raw	Specify additional information about the raw telemetry data.
iot_telemetry_timestamp_set	Specify the timestamp of a telemetry sample.

4. Sending Telemetry Data to the Server

An application can send telemetry data, such as readings from a sensor, to the server using the telemetry APIs.

The function parameters max_time_out and txn are for future use.

NOTE: If you publish raw telemetry, the maximum length is 1024 bytes minus the header.

Prerequisites

You must have previously initialized the agent and registered the application with the agent (see Registering Your Application with the Agent on page 11).

To send data to the server, the agent must be connected to the server.

To view your telemetry on the management console, you must have an administrator account or an account with the required permissions. For more information, see Permission Sets for the Management Console.

Procedure

1. Allocate the memory for the telemetry object.

Specify the value returned from the iot_initialize function, which you called when you registered your application, for the handle parameter.

Specify a name and a data type for the telemetry object.

```
iot_telemetry_t* temperature;
temperature = iot_telemetry_allocate(agentHandle, "temperature", IOT_TYPE_FLOAT32);
```

2. (Optional) Specify the telemetry units.

```
result = iot_telemetry_attribute_set(temperature, "udmp:units", IOT_TYPE_STRING, "fahrenheit");
```

3. Register the telemetry object with the agent.

   Specify **NULL** for the txn and zero for the max_time_out parameter.

```
result = iot_telemetry_register(temperature, NULL, 0);
```

4. Collect the data.
5. (Optional) Specify the data collection time.

   In the following example, the collection time is the current time, which is the default.

```
iot_timestamp_t tstamp;
?
tstamp = iot_timestamp_now();
result = iot_telemetry_timestamp_set(temperature, tstamp);
```

   You can also specify the timestamp as an offset in milliseconds from the current time to specify an earlier or later collection time. In the following example, the collection time is two minutes earlier than the current time.

```
iot_timestamp_t tstamp;
?
tstamp -= 120000;
result = iot_telemetry_timestamp_set(temperature, tstamp);
```

6. Send the telemetry sample to the server.

   Specify **NULL** for the txn and zero for the max_time_out parameter.

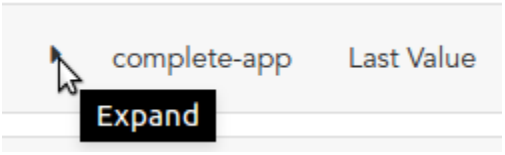
```
result = iot_telemetry_publish(temperature, NULL, 0, IOT_TYPE_FLOAT64, 17.5);
```

   When the application publishes the sample to the agent, the agent transmits the telemetry sample to the server.

7. To view your data on the management console, do the following:
   1. If you have not already done so, sign in to the management console with your user name and password at https://www.helixdevicecloud.com.
   2. Click **DEVICES**.
   3. In the **Device Name, ID, MAC Address** box, begin typing the device name, device ID, MAC address, or any other device attribute until your device appears in the list.
   4. In the Device ID column, click your device.
   5. On the device details page of the device, click **Telemetry Data**.

If the device is transmitting data or has transmitted data in the previous 90 days, the application name appears in the list.

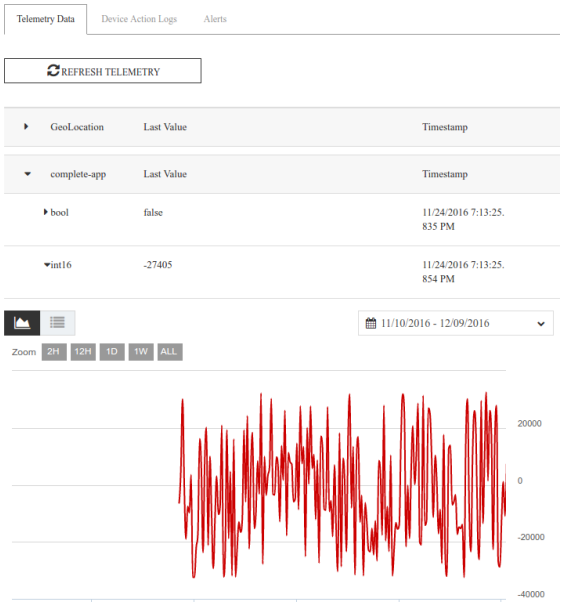
6. Click **Expand** on the left side of the page in the application row.



A snapshot of the current telemetry values appears.

7. To view historical data for individual telemetry metrics, expand the telemetry entry and the data appears in graph or list form.

If historical data is available but does not initially appear on the page, you may need to select a different date range to see the data.



Postrequisites

If you do not need the telemetry object after you send the data, call the iot_telemetry_deregister and iot_telemetry_free functions to free the memory allocated for the telemetry object.

Related information
Viewing Telemetry Data

5. Sending Location Data to the Server

You can send geolocation data to the server to monitor your device location.

Specifying location information for your device is optional.

You can acquire location information using a wireless network adapter or GPS antenna on your device. For example, you might want to track the location of a device in a building or the location of a device in a moving vehicle.

You can specify the following location properties:

Location Property	Description
latitude and longitude (mandatory)	?decimal floating point numbers conforming to the WGS84 specification
source	source of the location data; valid sources are WiFi, GPS, fixed, and unknown
accuracy and altitude accuracy	?accuracy in meters of the source of the location properties
heading	?direction of travel specified as degrees clockwise of true north (range 0 through 359), where north is 0 degrees, east is 90 degrees and west is 270 degrees
altitude	altitude in meters
speed	speed of travel in meters per second
tag	string value to specify additional information about the sample

The agent validates the values of latitude, longitude, and heading properties.

To transmit location data, you create and register a telemetry object, create a location data sample, and specify the location data sample as the telemetry value when you publish the telemetry.

Location information is available as telemetry on the server and location data appears on the management console under the **Telemetry Data** tab for each device. The application and telemetry object names are **GeoLocation** and **Location**, respectively; the data does not appear under the application or the telemetry object names.

Location APIs

Function	Description
?iot_location_accuracy_set	Specify the accuracy of the location sample in meters.
?iot_location_allocate	Create a location sample, allocate the required memory, and specify an initial location. Valid location values: **latitude: -90 through 90** **longitude: -180 through 180**
iot_location_altitude_set	Specify the altitude of the location in meters.

Function	Description
iot_location_altitude_accuracy_set	Specify the accuracy of the altitude in meters.
iot_location_free	Destroy a location sample and free the associated memory.
iot_location_heading_set	Specify the heading of the location in degrees. Valid values are from 0 through 360.
iot_location_set	Update the latitude and longitude of the location. Valid values: **latitude: -90 through 90** **longitude: -180 through 180**
iot_location_source_set	Specify the source of the location data.
iot_location_speed_set	Specify the speed of the location sample in meters per second.
iot_location_tag_set	Specify a descriptive value of the location.

Prerequisites

You need a wireless network adapter or GPS antenna and the associated drivers that are compatible with your board. For more information about the supported boards and hardware, see the information for the supported operating systems.

If you use a wireless network for location data, you need software libraries on your device to access geolocation data from a service provider, such as Google. Depending on your service provider, you may need a commercial subscription.

You must have previously initialized the agent and registered the application with the agent (see Registering Your Application with the Agent on page 11).

Before you can successfully send data to the server, the agent must be connected to the server.

To view location telemetry on the management console, you must have an administrator account or an account with the required permissions. For more information, see Permission Sets for the Management Console.

Procedure

1. Allocate the memory for the location telemetry object.

   Specify the value returned from the iot_initialize function, which you called when you registered your application, for the handle parameter. Specify **IOT_TYPE_LOCATION** for the type parameter.

   The value you specify for the name parameter is only used internally by the agent to ensure that the telemetry object is unique. Specify any valid string.

```
iot_telemetry_t* location_telemetry;
location_telemetry = iot_telemetry_allocate(agentHandle, "location", IOT_TYPE_LOCATION);
```

2. Register the location telemetry object with the agent.

   Specify **NULL** for the txn parameter and zero for the max_time_out parameter.

```
result = iot_telemetry_register(location_telemetry, NULL, 0);
```

3. Allocate the memory for the location data sample and specify the mandatory location information.

   Retrieve the location data from your geolocation service provider, GPS antenna, or use fixed values.

   Specify latitude and longitude as decimal values.

```
struct iot_location_t* location_data;
iot_float64_t latitude;
iot_float64_t longitude;

/* obtain location information - not shown */

location_data = iot_location_allocate( latitude, longitude );
```

4. (Optional) Specify the source of the location data.

   For the sample parameter, specify the value returned from the iot_location_allocate function.

```
result = iot_location_source_set( location_data, IOT_LOCATION_SOURCE_GPS );
```

5. (Optional) Specify optional location attributes as required, either with fixed values or values obtained from your service provider.

```
result = iot_location_accuracy_set( location_data, accuracy_value );
result = iot_location_altitude_set( location_data, altitude_value );
result = iot_location_tag_set( location_data, "Kitchen" );
```

6. Send the location data sample to the server.

   Specify **NULL** for the txn parameter and zero for the max_time_out parameter.

   Specify the location data sample for the data parameter.

```
result = iot_telemetry_publish( location_telemetry, NULL, 0, IOT_TYPE_LOCATION, location_data )
;
```

   The agent sends the location data to the server.

7. To update the location of the device, call the iot_location_set with new latitude and longitude values.

```
result = iot_location_set( location_data, -80.00, 100.9);
result = iot_telemetry_publish( location_telemetry, NULL, 0, IOT_TYPE_LOCATION, location_data )
;
```

   The agent sends the new location data to the server.

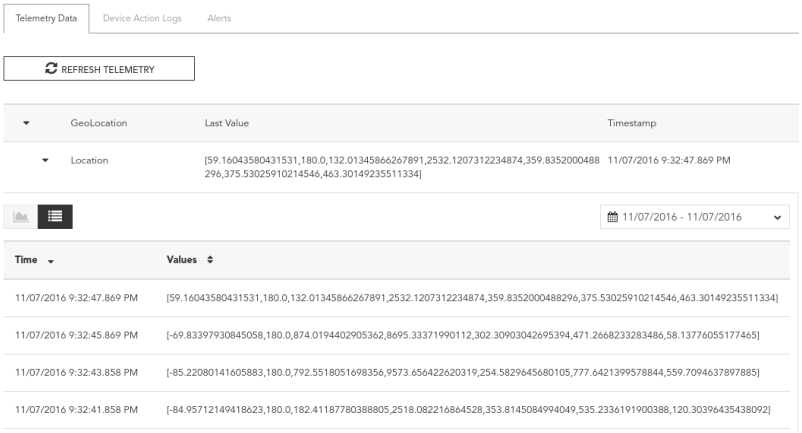
8. To view the location data on the management console, do the following:

1. If you have not already done so, sign in to the management console with your user name and password at https://www.helixdevicecloud.com.
2. Click **DEVICES**.
3. In the **Device Name, ID, MAC Address** box, begin typing the device name, device ID, MAC address, or any other device attribute until your device appears in the list.
4. In the Device ID column, click your device.
5. On the device details page of the device, click **Telemetry Data** and expand the **GeoLocation** entry.

A snapshot of the most recent location data appears.

6. To view historical location data, expand the **Location** entry

The data appears in list form. Graph form is not available for location telemetry.



If historical data is available but does not initially appear on the page, you may need to select a different date range to see the data.

On the map view on the management console, the device position updates if you select **Get position from device** in the basic device details to use the telemetry data from the device. The management console does not update the position dynamically when the device sends new data. For more information, see Showing the Device Position on the Map View.

Examples

The iot-app-complete and iot-app-simple-location example applications provided with the agent show example code for transmitting location data with all attributes specified.

Postrequisites

If you do not need the location data sample and the telemetry object after you send the location data, call the iot_location_free, iot_telemetry_deregister, and iot_telemetry_free functions to free the associated memory.

Related information

Sending Telemetry Data to the Server on page 14
Showing the Device Position on the Map View

6. About Custom Actions

Applications define custom actions to add to the device management capabilities provided by the agent.

You can specify seven parameters or fewer for an action, where the valid data types are Boolean, float32, float64, int32, int64, int16, int8, uint8, uint16, uint32, uint64, string, and raw. An application can register 255 actions or fewer with the agent. An action can have 20 attributes or fewer.

When an application registers an action with the agent, the agent publishes the registered actions to the server where they appear under the **CUSTOM ACTIONS** menu on the management console. The actions on the server can be initiated manually on one or more devices or automatically through a rule when a specified condition occurs. For more information about rules, see Creating Rules.

An application must register an action handler against the action to call when the server sends the action to the device. The action handler can be either a C function or any executable program on the device (a command).

If you use a C function, it can optionally specify a parameter to send a response to the server as part of the command execution. Valid response data types are Boolean, float, integer, string, and raw. The function also can specify the user_data parameter to pass application-specific data to the command. The agent does not use this data or send it to the server. If your application does not require additional data, specify **NULL**.

A command is any executable program on the device, including operating system commands. You might want to register a script as the command if you do not want to write C code or if you want to reuse an existing script. When the agent receives the action from the server, the agent runs the command under the application process, but the application is not notified of the script execution. Parameters are passed to the script through stdin and the script is responsible for parsing the parameter data types and values correctly. Parameter names must contain only alphanumeric characters, dot (**.**), and dash (**-**). Output is sent to stdout; commands cannot return a response to the server.

VxWorks does not support registering a command as a custom action.

Action APIs

Function	Description
?iot_action_allocate	?Create an action and allocate the required memory.
iot_action_attribute_set	Add an attribute to an action.
?iot_action_deregister	Deregister an action from a service.
iot_action_free	Destroy an action and free the associated memory.
iot_action_parameter_add	Add a parameter to an action.
iot_action_parameter_get	Retrieve the value of a parameter.
iot_action_parameter_raw_get	Retrieve the value of a raw parameter.
iot_action_parameter_set	Return a value in response to the action.
iot_action_parameter_set_raw	Return a raw value in response to the action.

Function	Description
iot_action_register	Register an action with the agent.
iot_action_register_callback	Register a callback function to call when the specified action is received.
iot_action_register_command	Register a system command to call when the specified action is received.

7. Receiving Actions from the Server

?Applications on the device can receive and process actions initiated from the server through actions registered with the agent.

When an action is executed from the server, the agent processes the registered callback or command in a worker thread.

The function parameters max_time_out and txn are for future use.

Prerequisites

?You must have previously initialized the agent and registered the application with the agent (see Registering Your Application with the Agent on page 11).

To receive actions, the agent must be connected to the server.

To run your actions from the management console, you must have an administrator account or an account with the required permissions. Sending files, retrieving files, and remote login actions require additional permissions. For more information, see Permission Sets for the Management Console.

Procedure

1. Allocate the memory for the action.

   Specify a name for the action.

   Specify the value returned from the iot_initialize function, which you called when you registered your application, for the handle parameter.

   ```
   iot_action_t* myAction;
   myAction = iot_action_allocate(agentHandle, "myAction" );
   ```

2. Optionally, add one or more parameters to the action.

   Specify the parameter name and data type. Specify zero for the max_time_out parameter.

   Specify **IOT_PARAMETER_IN_REQUIRED** for the param_type.

   To specify that a parameter is both an input parameter and an action response, specify **(IOT_PARAMETER_IN_REQUIRED | IOT_PARAMETER_OUT_REQUIRED)**.

   If you use a script as the action handler, ensure that the parameter name contains only alphanumeric characters, "**.**", and "**-**".

```
result = iot_action_parameter_add( myAction, "parm_int", IOT_PARAMETER_IN_REQUIRED, IOT_TYPE_UI
NT32, 0);
result = iot_action_parameter_add( myAction, "parm_string", IOT_PARAMETER_IN_REQUIRED, IOT_TYPE
_STRING, 0);
```

3. Optionally, add one or more parameters to return as an action response.

   Specify **IOT_PARAMETER_OUT_REQUIRED** for the param_type. Specify zero for the max_time_out parameter.

```
iot_action_parameter_add( myAction, "boolean_response", IOT_PARAMETER_OUT_REQUIRED, IOT_TYPE_BO
OL, 0u );
```

4. If you use a C function as the action handler, write the function as follows:
   1. Use the following function signature:

```
iot_action_status_t myActionCallback( iot_action_request_t *request, void* user_data )
```

   2. If you registered parameters with the action, retrieve the parameter values sent from the server.

      Specify the parameter name and memory to store the returned value.

      Specify **IOT_FALSE** for the convert parameter.

```
iot_status_t result;
const char* param_name_int = "parm_int";
const char* param_name_string = "parm_string";
int32_t value_int = 0;
const char* value_string = NULL;
result = iot_action_parameter_get( request, param_name_int, IOT_FALSE, IOT_TYPE_UINT32, &amp
;value_int );
result = iot_action_parameter_get( request, param_name_string, IOT_FALSE, IOT_TYPE_STRING,
                                        &amp;value_string);
```

   3. Optionally, send a response to the server.

```
result =iot_action_parameter_set( &amp;request, "boolean_response", IOT_TYPE_BOOL, IOT_TRUE)
;
```

5. Register an action handler for the action.

Option	Description

Register a callback function.	Specify the address of the C function you created.
	Specify **NULL** for the txn and zero for the max_time_out parameter.
	`void* user_data;`
	
	`user_data = malloc( 1u );`
	`result = iot_action_register_callback( myAction, &myActionCallbac k, user_data, NULL, 0 );`

Option	Description
	You can also specify **NULL** for the user_data parameter.
Register a script.	Specify the command, including the full path, to execute. Specify **NULL** for the txn and zero for the max_time_out parameter. `result = iot_action_register_command( myAction, "/pathTo/test.sh", NULL, 0 );`

If you register more than one action handler (for example, a callback function and a script), the register function returns an error.

The action appears on the management console as an entry in the **CUSTOM ACTIONS** menu when you select the device in the device list or on the device details page of an individual device.

6. If required, repeat steps 1 on page 22 through 5 on page 23 to create and register additional actions with the server.
7. To execute the action from the server, do the following:
    1. If you have not already done so, sign in to the management console with your user name and password at https://www.helixdevicecloud.com.
    2. Click **DEVICES**.
    3. In the **Device Name, ID, MAC Address** box, begin typing the device name, device ID, MAC address, or any other device attribute until your device appears in the list.
    4. In the Device ID column, click your device.
    5. Select **CUSTOM ACTIONS** and then select the name of your action.

    An entry appears in the list in the upper-right corner to indicate that the action has been sent to the device.

    6. To view the action status and history, from the device details page, click **Device Actions Log**.
    7. Based on the timestamp and the action name, locate the row for your action.

    When the action completes, the Status column shows **Completed**.

    8. Expand the row.

    If the action completed successfully, the Details area shows the message **The device executed the service command successfully and it is complete**.

Postrequisites

When you no longer need the command, call iot_action_deregister to deregister the action from the agent and iot_action_free to free the memory allocated for the action.

8. Python Bindings

The agent API supports language bindings to enable development in languages other than C.

Python bindings for the agent APIs are supported on Wind River Linux, Wind River IDP XT, Windows, and Ubuntu.

Bindings for Linux

Agent-related Python files are located as follows:

Operating System	Location
Wind River Linux and IDP XT	**/usr/lib64/python2.7/site-packages** directory on 64-bit targets **/usr/lib/python2.7/dist-packages** on 32-bit targets
Ubuntu	**/usr/lib/python2.7/site-packages**

In your application, import definitions from the **iot_python** module. To generate the Python API documentation on the device, run the following:

```
# pydoc iot_python > pythondoc.txt
```

The **/usr/bin/iot-py-complete.py** sample application is available on the device.

On your host computer for Wind River Linux and IDP XT, the **complete.py** sample application is available in the following directory:

projDir **/build/iot/git/apps/complete**

Bindings for Windows

You must have Python 2.7*x* 32-bit installed on the device on which you want to run the application and the computer on which you develop your application.

If you select the **Development** component at installation time, the agent-related Python files are located in the *installDir* **\bin** folder, where *installDir* is the folder where you installed the agent. The **iot_python** module is also installed in *pythonInstallDir* **\Lib\site-packages**.

In your application, import definitions from **iot_python**.

To generate the Python API documentation, run the following in a Command Prompt window:

```
C:\> cd installDir\bin
C:\> pythonInstallDir\Lib\pydoc.py iot_python > yourDir\pythondoc.txt
```

If you select the **Examples** component at installation time, the *installDir* **\bin\iot-py-complete.py** sample application is installed with the agent.

API Differences between C and Python

Function	Description
iot_log_callback_set	The user_data parameter is not supported.
iot_log	This function does not support parameters to format the log message. The Python application must format the message.

Function	Description
iot_action_register_callback	The user_data parameter is not supported
iot_action_parameter_get	Returns a tuple of status and parameter instead of a status of type **iot_status_t**, where status is the return code and parameter is the parameter value. Therefore, the API does not include the data_ptr parameter.
iot_action_parameter_get_raw	Returns a tuple of status and parameter instead of a status of type **iot_status_t**, where status is the return code and parameter is the parameter value. Therefore, the API does not include the data parameter. The parameter value is always a string. The API does not include the length parameter because Python strings are automatically null-terminated.
Types	**Description**
iot_log_callback_t	Specify the log_source parameter as a tuple of function_name, file_name, line_number instead of as type **iot_log_source_t**. The user_data parameter is not supported.
iot_action_callback_t	The user_data parameter is not supported.

- In addition to **IOT_TRUE** and **IOT_FALSE**, the Python **True** and **False** definitions are also valid for Boolean parameters and return values.
- Unsigned integer and **IOT_TYPE_FLOAT32** types are not available for functions that need a specified type.
- Integer parameter values greater than the maximum value of a 64-bit integer are rejected.
- The macro **IOT_LOG** is not available because Python does not support macros.

4. SAMPLE APPLICATIONS

1. Sample Applications

Sample applications provide code that you can use as a starting point when you develop your application.

The following C code samples are available on Wind River Linux, Wind River IDP XT, Windows, and Ubuntu:

iot-app-complete

Registers four actions, 13 telemetry metrics, and location telemetry.

iot-app-simple-actions

Registers three actions.

iot-app-simple-telemetry

Registers 13 telemetry metrics and an action to start and stop transmitting telemetry data.

iot-app-simple-location

Registers location telemetry and an action to start and stop transmitting location telemetry data.

The following sample application is available in Python:

iot-py-complete.py

Registers four actions, 13 telemetry metrics, and location telemetry.

NOTE: To view telemetry sent from the sample applications or run device actions on the management console, you need an administrator account or an account with the required permissions. For more information, see Permission Sets for the Management Console.

On Wind River Linux, Wind River IDP XT, and Ubuntu, the executable programs are installed by default on the device. If the device is connected to the server, you can start the application on the device and then sign in to the management console to run the registered actions and view telemetry data.

On Windows, you must select the **Examples** component at installation time to install the sample source and executable programs.

On Windows and Ubuntu, the host (development computer) and device may or may not be the same computer.

Source and executable program files are available as follows:

Operating System	Source Location on Host	Source Location on Device	Executable Location on Device
Wind River Linux and IDP XT	*projDir* **/build/iot/git/apps** after building the platform project	**/usr/share/iot/examples**	**/usr/bin**
Windows	*installDir* **\examples**	*installDir* **\examples**	*installDir* **\bin**
Ubuntu	**/usr/share/iot/examples**	**/usr/share/iot/examples**	**/usr/bin**

NOTE: To modify the source code on the target in the directories in which they are installed, you may need administrator or superuser privileges. You should copy the files to a directory that does not require additional privileges to write to it.

VxWorks 7

The following application is available:

hdc_telemetry_test

Registers an action with four parameters, an action to start and stop transmitting telemetry data, and transmits 13 telemetry metrics. The entry point is hdc_telemetry_test_main.

After you install VxWorks 7 on your host computer, the sample application source is located on your host computer in the following directory:

installDir **/vxworks-7/pkgs/app/hdc-***releaseNum* **/agent/example**

Additional Samples for Wind River Linux and IDP XT

If you include the **wr-iot-apps** layer when you configure your platform project, additional samples are available in source form on your host computer and as executable programs on your target. For details about the examples, see the *projDir* **/layers/wr-iot-apps/README.md** file after you run the configure command.

2. Modifying and Building the Sample Applications on Wind River Linux and IDP XT

If you included development tools in your target image, you can modify and build the sample applications on your target.

The sample applications are located under the **/usr/share** directory on your target, which requires superuser privileges to write to it. You should copy the **/usr/share/iot/examples** directory to a location to which you have write privileges.

Prerequisites

When you built your platform project, you must have enabled the --with-template=feature/self-hosted template and --with-package=iot-dev package, or the application development SDK you received must have been built with these options enabled.

To view telemetry sent from the sample applications or run device actions on the management console, you need an administrator account or an account with the required permissions. For more information, see Permission Sets for the Management Console.

Procedure

1. If you have not already done so, copy the **/usr/share/iot/examples** directory to another directory to which you have write permissions.

   For example, copy the directory to **/home/***yourUsername***/HDC/examples**.

2. In a terminal window, change to the directory that contains the sample you want to modify or build.
3. Make changes to the sample source code as needed.
4. To build the sample, type make.

   After the compile completes, the executable program is created in the same directory.

5. To run the sample, type *./sampleProgram*.

   The application starts and the output appears in the terminal window. Actions appear in the **CUSTOM ACTIONS** menu on the management console.

Postrequisites

You may want to copy your modified source files to your host computer for later use.

3. Modifying and Building the Sample Applications for Windows

Sample applications provide C code and Windows-specific makefiles to create simple actions and telemetry.

Visual Studio is the recommended development environment. The samples were verified using Visual Studio 2015; however, newer releases should also work.

If you installed Helix Device Cloud under **C:\Program Files (x86)** or **C:\Program Files**, you should copy the **examples** folder to a folder that does not require administrator privileges to write to it. Copying the files to a different folder also ensures that the files are not removed if you uninstall the agent.

Prerequisites

You must have installed Visual Studio.

To view telemetry sent from the sample applications or run device actions on the management console, you need an administrator account or an account with the required permissions. For more information, see Permission Sets for the Management Console.

Procedure

1. If you have not already done so, copy the *installDir***\examples** folder to another folder.

   For example, copy the files to **C:\Users***username***\HDC\examples**.

2. Make changes to the sample source code as needed.
3. Open a Visual Studio Developer Command Prompt.

You must perform the following steps in a Visual Studio Developer Command Prompt window and not in a standard Command Prompt window.

4. In the directory of the sample you modified, build the sample.

```
C:\> cd C:\users\username\HDC\examples\sampleDir
C:\> nmake
```

   After the compile completes, the executable file is created in the same directory.

5. To run the sample, double-click the executable file in Windows Explorer.

   A Command Prompt window opens and the sample application output appears. Depending on which sample you built, you can go to the management console to run actions and view telemetry.

4. Modifying and Building the Sample Applications for Ubuntu

Sample applications provide C code and Linux-specific makefiles to create simple actions and telemetry.

The sample applications are located under the **/usr/share** directory, which requires superuser privileges to write to it. You should copy the **/usr/share/iot/examples** directory to a location to which you have write privileges. Copying the files to a different folder also ensures that the files are not removed if you remove the agent package.

Prerequisites

You need make, gcc, and glibc installed.

To view telemetry sent from the sample applications or run device actions on the management console, you need an administrator account or an account with the required permissions. For more information, see Permission Sets for the Management Console.

Procedure

1. If you have not already done so, copy the **/usr/share/iot/examples** directory to another directory to which you have write permissions.

   For example, copy the directory to **/home/***yourUsername***/HDC/examples**.

2. In a terminal window, change to the directory that contains the sample you want to modify or build.
3. Make changes to the sample source code as needed.
4. To build the sample, type make.

   After the compile completes, the executable program is created in the same directory.

5. To run the sample, type *./sampleProgram*.

   The application starts and the output appears in the terminal window. Actions appear in the **CUSTOM ACTIONS** menu on the management console.

5. SOFTWARE UPDATE

1. Software Update

You can create an update to the device software and deploy it from the server to the device where the agent automatically installs the update.

NOTE: This method of creating software updates applies only to devices running release 2.2 of the agent or later.

To create and deploy updates to devices running release 2.1 of the agent or earlier, you must create the update using the steps from previous releases and use the administration utility to deploy the update. For more information, see *Wind River Helix Device Cloud Agent Programmer's Guide, 2.1* and *Wind River Helix Device Cloud Administration Utility User's Guide: Software Update*. To create an update to migrate a device running the 2.1 release of the agent to the 2.2 release, see the *Wind River Helix Device Cloud Release Notes, 2.2*.

You can create an update package that contains update files created from a platform project to a device running the 2.2 release of the agent if you create a new update package from the management console. For an example, see Creating an Update Package on page 33.

Software update is supported for devices that run Wind River Linux, Wind River IDP XT, Windows, and Ubuntu.

To perform updates on devices that run Windows, you must install Python 2.7*x* 32-bit separately.

A package consists of the following:

- the software to update, either in the package itself or on a remote server
- instructions to install the software on the device
- criteria to define the list of devices to which the update can be deployed and to enable the device to verify the update compatibility

When you create the package, you upload the files that are required for the update, including the files that contain instructions for the installation stages, if applicable. The maximum total file size is two GB.

By default, packages are created in the unpublished state. Typically, packages remain in this state until update testing completes. You change the state to published when an update is ready for production.

When you create the package, you can specify the devices that are compatible with the update, based on device properties. If you select the operating system version in compatibility criteria, the agent on the device verifies the specified version against the software on the device before installing the update. If you select multiple versions, the device software is compared against the lowest selected version.

Package Contents

You can update applications on the device, the agent, and the kernel (Wind River Linux and IDP XT only). You can use the process to update your own applications or add applications you receive from third-party vendors.

The update can include any files you want, including individual files and archives in a format that is compatible with the operating system on the device, such as an RPM or a zip file.

You can also create an update package that contains only instructions. For example, an update that retrieves software from an external package repository might consist of only the instructions to retrieve and install the package.

Package Instructions

You create an update package from the management console that specifies the operating system and package specific instructions for each stage.

All instructions are optional. An instruction is any single-line command of 256 characters or fewer. The instruction can be any system command, an existing program on the device, or a program included in the package itself. If your instructions for any stage require more than a single-line command, you must create a file with the instructions, such as a Linux shell script or Windows batch file.

You provide instructions for the following stages of the installation process:

preinstall

Instructions to run before the installation begins, such as instructions to stop services on the device or to back up existing files.

install

Instructions to install the update.

postinstall

Instructions to run after the installation finishes successfully, such as instructions to restart services or remove temporary files.

error handling

Instructions to run if the update fails at any stage, such as instructions to restore the system to its previous state.

On Windows, the update procedure runs under a process with administrator privileges. The update instructions can modify directories and files and run commands that require administrator privileges. However, if you start applications in any of your instructions, you must use the AT command to avoid blocking the update process. See the example scripts provided in the directory listed below.

On Wind River Linux, Wind River IDP XT, and Ubuntu, the update procedure runs as the **iot** user, which can execute the commands specified in the **/etc/sudoers** file. You must run any commands that require superuser privileges or require access to files and directories owned by the **root** user using sudo. If you attempt to run commands that are not in the **/etc/sudoers** file, an error appears in the update log and the update fails.

NOTE: To test your instructions completely, you should create a package on the management console. Testing your instructions from the command line or while running as the **root** user may not expose errors such as insufficient privileges to run commands or applications.

Package Deployment

When you deploy the package to devices, the server downloads the files to the device and the agent runs the instructions you provide. All commands and paths are relative to the predefined directory into which the files are downloaded. If your update contains files in a subdirectory, you must specify the path to the subdirectory in your instructions.

During deployment, the status of the deployment appears under the **Deployments** tab on the Software Updates page.

Detailed progress of each stage of the update is available on the device details page under the telemetry item **telemetry_sw_update**. For information about viewing telemetry, see Viewing Telemetry Data.

If the deployment fails, the software update process does not revert to the previous software running on the device in any situation. You must implement any recovery actions in your error handling instructions.

Update Examples

Samples of scripts for the update stages and instructions to use them are available as follows:

Operating System	Location
Wind River Linux and IDP XT	After you run the make command in your platform project, examples of an application update and kernel update are available in the following directory on your host computer: *projDir* **/build/iot/git/share/ota-examples/linux** The examples are also available on your target in the following directory: **/usr/share/iot/examples**
Windows	If you select the **Examples** component when you install the Helix Device Cloud agent, an example of an application update is available in the following folder: *installDir* **\examples\ota**
Ubuntu	After you install the Helix Device Cloud agent, an example of an application update is available in the following directory: **/usr/share/iot/examples/app-update**

Basic Workflow

1. Create the files to include in the update using any method you want.
2. Create the update package using the management console.
3. Deploy the package using the management console to one or more devices.

2. Creating an Update Package

An update package includes the instructions to install software on the device and if applicable, contains the files to install on the device.

Prerequisites

You must have an administrator account or an account with the required permissions for software packages. For more information, see Permission Sets for the Management Console.

Procedure

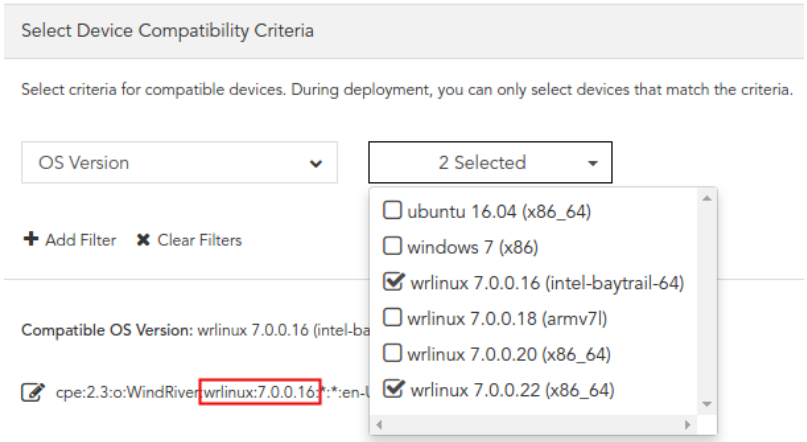
1. If you have not already done so, sign in to the management console at https://www.helixdevicecloud.com with your user name and password.
2. On the Software Updates page, click **Available Packages** and then click **CREATE NEW PACKAGE**.
3. In the **Package Name** box, type a unique name that is seven alphanumeric characters or fewer.
4. In the **Version** box, type a version that is five alphanumeric characters or fewer.
5. (Optional) In the **Description** box, type a detailed description of the package that is 500 characters or fewer.
6. (Optional) Specify the filter criteria to select compatible devices.
    1. In the **Select criteria** list, select the device property you want to use to determine the compatible devices.

The **Select** list contains the values based on currently registered devices.

2. In the **Select** list, select one or more values.

Devices matching **any** of the selected values are compatible with the update.

If you select **OS Version**, the value the agent will use to check compatibility when installing the update appears in the **cpe** string below.



3. (Optional) To add additional device properties, click **Add Filter** and repeat the steps above as needed.

You can select the same property multiple times.

Devices that match **all** the properties in each filter are compatible with the update.

4. (Optional) To confirm the devices that are compatible with the selected properties, click **PREVIEW COMPATIBLE DEVICES**.

7. (Optional) In the Package Files area, drag the required files and directories into the window.

   You can add or remove files from the list at any time during package creation.

   The maximum total file size is two GB.

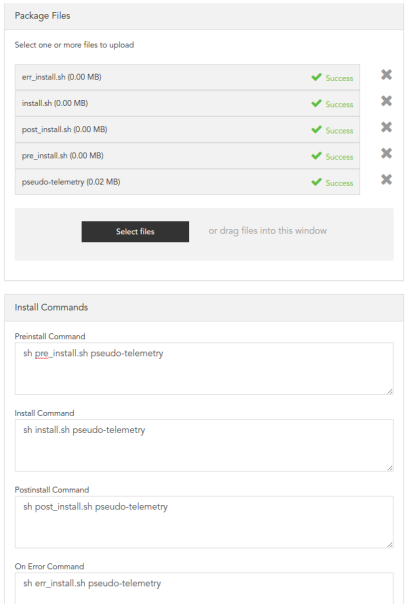
   When the file upload completes successfully, the progress bar shows **Success**.

8. (Optional) In the Install Commands area, type the required commands in each box, where each command is 256 characters or fewer.

   The **Key/Value Command** is for future use and any values you enter are ignored.

   If you specify special characters in your instructions, you must use an escape character, which is operating system-specific.

   The following example shows an update that contains a single, executable file called **pseudo-telemetry** and shell scripts for each stage of the installation.



9. Choose the archive format that is compatible with the operating system running on the device.

   Choose **.zip** for Windows and **.tar.gz** for Linux-based operating systems.

10. Choose whether to reboot the device after the update completes successfully.

    NOTE: If you update any agent-related files, such as configuration files, you must reboot the device.

11. When you finish, click **SAVE**.

The package appears in the list of available packages in the unpublished state and is ready to deploy. The package name is a combination of the values specified in the **Package Name** and **Version** boxes.

Example: Update Package Created from a Platform Project

You can create and deploy an update package to a device running release 2.2 of the agent on Wind River Linux or IDP XT that contains an update generated from a platform project using the instructions from releases prior to 2.2. You need the **system-update.tar.gz** file you created.

Fill in the instructions in the Install Commands section as follows:

Preinstall Command

```
cd /var/lib/iot/update/download && tar -xvf system-update.tar.gz
```

Install Command

```
sudo rpm -ivh --force /var/lib/iot/update/download/system-update-RPMVersion.rpm
```

On Error Command

```
rm /var/lib/iot/update/download/system-update.tar.gz && rm /var/lib/iot/update/download/system
-update-RPMVersion.rpm
```

In the Package Files area, upload the **system-update.tar.gz** file.

Follow the remaining steps above to finish creating the update package.

Postrequisites

You can click the package name to view the details and edit, delete, or deploy the package. Only one user at a time can edit a package.

For information about deploying a package, see Deploying a Software Update Package.

6. AGENT CONFIGURATION

1. Agent Configuration

Agent properties and behavior can be configured in the **iot.cfg** file.

The **iot.cfg** file contains attribute-value pairs in JavaScript Object Notation (JSON) format.

Sample files are available as follows:

Operating System	Location
Wind River Linux and IDP XT	*projDir* **/build/iot/git/cmake_build/bin/iot.cfg.example**
Windows	*installDir* **\etc\iot.cfg.example**
VxWorks 7	*installDir* **/vxworks-7/pkgs/app/hdc-***releaseNum* **/agent/sample_cfg/etc/iot.cfg**
Ubuntu	**/etc/iot/iot.cfg.example**

On Wind River Linux, Wind River IDP XT, Ubuntu, and Windows, you can use software update to download the **iot.cfg** file to the device and restart the device to apply the new values.

On Wind River Linux and IDP XT, you can also write a Bitbake recipe to add the file to the rootfs when you build your platform project.

On all Linux targets, you must ensure that the **iot** user has read permission for the file. If you edit the file on the target, check the permissions after you finish.

To change the values for a device running VxWorks 7, change the values in the file you provide on your persistent storage or in your kernel image. The **HDC_AGENT_CONFIG_FILE** VIP parameter specifies the file location. Only the agent property and log level attributes apply to VxWorks 7.

Agent Properties

Agent properties appear on the device list and device details page on the management console. They appear in the list of criteria to select in the filter list on the device details page and in the list of criteria to select compatible devices when you create a software package. The values of these properties are user-defined; the agent does not use them.

Agent properties have default values for Wind River Linux, Wind River IDP XT, Ubuntu, and Windows.

The following attributes represent agent properties:

- **model_name**
- **vendor_id**
- **device_id**
- **serial_number**

Agent Log Level

By default, all logs are enabled in the agent log files. You can change the value of the **log_level** attribute to specify the log types you want to include.

Disable and Enable Device Actions

The **actions_enabled** object contains a list of the device actions the agent supports. By default, all actions are enabled and appear in the **STANDARD ACTIONS** and **CUSTOM ACTIONS** menus on the management console. You can change the Boolean value for each action to prevent users from running the action. Disabled actions do not appear in the menus or appear but are not available to select.

NOTE: Although the action is shown in the file, disabling and enabling remote login is not currently supported.

For a description of the individual actions, see Agent Actions.

File Upload Directory Configuration

You can add to the list of directories the agent searches for files to include when the **Retrieve Files** action is run from the management console and you can specify whether to remove files from the directories when the agent completes the action successfully. The **upload_additional_dirs** array contains a list of one or more directories to search. The Boolean value of the **upload_remove_on_success** attribute indicates whether the agent deletes the uploaded files in the directories if the files are retrieved successfully.

NOTE: On Linux devices, ensure that the **iot** user owns the directories listed in the **upload_additional_dirs** array.

Ensure that you use escape sequence **** for the **** character in Windows path names.

For details about the attributes and valid values, see iot.cfg on page 41.

2. Updating the Agent Configuration Using Software Update

You can use software update to download the agent configuration file, **iot.cfg**, to the device and restart the device to apply the new values.

The following steps are for Wind River Linux, Wind River IDP XT, Ubuntu, and Windows.

NOTE: Ensure that the **iot.cfg** file has valid JSON syntax. If there are errors, such as missing commas, an error appears in the **iot** service log file and the agent registers successfully using the default values; the contents of the file are ignored.

On Linux targets, ensure that the **iot** user owns the file.

Prerequisites

You must be familiar with the procedures to create and deploy a software package from the management console. For information, see Creating an Update Package on page 33 and Deploying a Software Update Package.

You must have an administrator account or an account with the required permissions for creating and deploying software packages. For more information, see Permission Sets for the Management Console.

Procedure

1. Copy the **iot.cfg.example** file to a suitable location on your host computer and rename it to **iot.cfg**.
2. Change the values of the fields as needed.
3. Create a software update package that contains the file and the operating system-specific instructions to add or replace the file.

   NOTE: You must choose **Reboot on Completion** to apply the update.

4. Deploy the update to compatible devices.

After the update completes and the device reboots, changes to the agent properties appear on the device list page, the device details page, and in the list of criteria you can select when you create an update package or a filter on the device list page.

Disabled device actions do not appear in the actions menus or appear but are not available to select.

Example: Update iot.cfg on Wind River Linux and IDP XT

Copy the *projDir* **/build/iot/git/cmake_build/bin/iot.cfg.example** file to a local directory to which you have write privileges and rename it to **iot.cfg**. Change the file as necessary.

The **iot.cfg** file can exist in three directories. The example shows the commands to remove the file if it exists in any of the directories and to remove the example file.

The following shows the example instructions in the update package:

Preinstall Command

```
sudo rm -f /etc/iot/iog.cfg && sudo rm -f /etc/iot/iog.cfg.example && sudo rm -f /var/lib/iot/
iot.cfg && sudo rm -f /opt/intel/ccg/bin/iot.cfg
```

Install Command

```
sudo cp iot.cfg /etc/iot
```

Ensure that you choose **Reboot on Completion**.

Upload the **iot.cfg** file you created.

Example: Update iot.cfg on Windows

Copy the *installDir* **\etc\iot.cfg.example** file to a local directory to which you have write privileges and rename it to **iot.cfg**. Change the file as necessary.

The **iot.cfg** file can exist in three folders. The example shows the commands in the batch file **cleanup_iot_cfg_files.cmd** used in the preinstall instruction to remove existing copies of the file if it exists in any of the folders and to remove the example file.

```
set BASEPATH=c:\Program Files (x86)\Helix Device Cloud

rem Remove existing copies of the file from all possible folders

if EXIST "%BASEPATH%\etc\iot.cfg" del /F "%BASEPATH%\etc\iot.cfg"
if "%ERRORLEVEL%" neq "0"     goto EOF
if EXIST "%BASEPATH%\etc\iot.cfg.example" del /F "%BASEPATH%\etc\iot.cfg.example"
if "%ERRORLEVEL%" neq "0"     goto EOF
if EXIST "%BASEPATH%\bin\iot.cfg" del /F "%BASEPATH$\bin\iot.cfg"
```

```
if "%ERRORLEVEL%" neq "0"     goto EOF
if EXIST "C:\ProgramData\Wind River Systems\Helix Device Cloud\iot.cfg" del /F "C:\ProgramData\Win
d River Systems\Helix Device Cloud\iot.cfg"
if "%ERRORLEVEL%" neq "0"     goto EOF

echo success executed cleanup_iot_cfg_files.cmd > "c:\ProgramData\1.txt"

exit /B 0
:EOF

echo ERROR > "c:\ProgramData\1.txt"
exit /B 1
```

The following shows example instructions in the update package:

Prenstall Command

```
cleanup_iot_cfg_files.cmd
```

Install Command

```
COPY /y iot.cfg C:\"Program Files (x86)\Helix Device Cloud\etc\iot.cfg"
```

Ensure that you choose **Reboot on Completion**.

Upload the **iot.cfg** file you created and the **cleanup_iot_cfg_files.cmd** file.

Example: Update iot.cfg on Ubuntu

Copy the **/etc/iot/iot.cfg.example** file to a local directory to which you have write privileges and rename it to **iot.cfg**. Change the file as necessary.

The **iot.cfg** file can exist in multiple directories. The example shows the commands to remove the file if it exists in two of the directories and to remove the example file.

It is not necessary to run the commands with superuser privileges because the files and directories modified in the update are owned by the **iot** user.

The following shows the example instructions in the update package:

Preinstall Command

```
rm -f /etc/iot/iog.cfg && rm -f /etc/iot/iog.cfg.example && rm -f /var/lib/iot/iot.cfg
```

Install Command

```
cp iot.cfg /etc/iot
```

Ensure that you choose **Reboot on Completion**.

Upload the **iot.cfg** file you created.

7. REFERENCES

1. iot.cfg

The **iot.cfg** file contains attribute-value pairs in JavaScript Object Notation (JSON) format that you can change to override the default agent properties and agent behavior.

The location of the file on the target is operating-system specific.

Operating System	Valid Locations
Wind River Linux and IDP XT Ubuntu	any of the following locations: **/etc/iot (the default)** **/var/lib/iot**
Windows	any of the following locations: *installDir* **\etc (the default)** *installDir* **\bin** **C:\ProgramData\Wind River Systems\ Helix Device Cloud**
VxWorks 7	the location specified by the **HDC_AGENT_CONFIG_FILE** VIP parameter

On Linux devices, ensure that the **iot** user owns the files.

The following table lists the attributes in the file and the valid values.

Attribute	Valid Values
model_name	any valid string only the first 100 characters or fewer appear on the management console
vendor_id	any valid string only the first 100 characters or fewer appear on the management console
device_id	any valid string only the first 100 characters or fewer appear on the management console
serial_number	any valid string only the first 100 characters or fewer appear on the management console

Attribute	Valid Values	
log_level	ERROR, WARNING, INFO, TRACE, FATAL, ALERT, CRITICAL, NOTICE, DEBUG, ALL. Default: ALL	
actions_enabled	shutdown_device	true, false Default: true
	reboot_device	true, false Default: true
	reset_agent	true, false Default: true
	file_transfers	true, false Default: true
	software_update	true, false Default: true
	restore_factory_images	true, false Default: true
	decommission_device	true, false Default: true
	dump_log_files	true, false Default: true
	remote_login	Not supported
upload_remove_on_success	true, false Default: true	
upload_additional_dirs	array of comma-separated strings that specify valid directories on the device Default: empty string	

Example: Linux

```
{
        "model_name": "model abc",
        "vendor_id": "iot_vendor",
        "device_id": "device 123",
        "serial_number": "123456",
        "fw_version": "2.3",
        "build_version": "c8bbc71550fbf175fb9e7baa6349013f39130947",
        "runtime_dir": "/var/lib/iot",
        "log_level": "ALL",
        "ssl_validation": true,
```

```
        "cert_path": "",
        "actions_enabled": {
                "shutdown_device": false,
                "reboot_device": true,
                "reset_agent": true,
                "file_transfers": true,
                "software_update": true,
                "restore_factory_images": true,
                "decommission_device": true,
                "dump_log_files": true
        },
        "upload_remove_on_success": true,
        "upload_additional_dirs": [
                "/var/log","/home/user"
        ]
}
```

Example: Windows

```
{
        "model_name": "model abc",
        "vendor_id": "vendor xyz",
        "device_id": "device 123",
        "serial_number": "123456",
        "fw_version": "2.3",
        "build_version": "b9ee69959cf15e63dc0383ef53ccf107751ad5cb",
        "runtime_dir": "%PROGRAMDATA%/Wind River Systems/Helix Device Cloud",
        "log_level": "ALL",
        "ssl_validation": true,
        "cert_path": "etc\\ca-certificates.crt",
        "actions_enabled": {
                "shutdown_device": false,
                "reboot_device": true,
                "reset_agent": true,
                "file_transfers": true,
                "software_update": true,
                "restore_factory_images": false,
                "decommission_device": true,
                "dump_log_files": true
        },
        "upload_remove_on_success": true,
        "upload_additional_dirs": [
                "C:\\Users\\test"
        ]
}
```

NOTE: Ensure that you use escape sequence **** for the **** character in Windows path names.