BEA WebLogic RFID Edge Server™

Edgeflow Reference

Version 3.0
August, 2007

Contents

5. Workflow Module Reference for Versions Prior to 3.0

Introduction and Roadmap

The following sections describe the contents, audience for, and organization of this guide—*WebLogic RFID Edge Server Edge Flow Reference*.

- "Document Scope and Audience" on page 1-1
- "Guide to This Document" on page 1-2
- "Related Information" on page 1-2

Document Scope and Audience

Note: In releases prior to the RFID Edge Server 3.0 release, the concepts described in this guide were referred to as workflows. The earlier releases also provided sample workflows.

In this release, workflows are replaced by small, reusable modules that provide finer granularity functions and can be used as building blocks to satisfy use cases. These building blocks are called Edge Flow modules and can be used to create simple or complex Edge Flows. This new feature provides a flexibility that minimizes the need to write custom modules.

Starting with version 3.0 of the Edge Server, limited support is provided for the workflow modules. New workflows cannot be created using the Adminitration Console and messaging between the workflow modules and the new Edge Flow modules is not supported.

For more information about the workflow modules, see "Workflow Module Reference for Versions Prior to 3.0" on page 5-1.

This document describes the basic concepts of Edge Flows, as well as how to configure and use the Edge Flow modules included with the WebLogic RFID Edge Server. It does not explain other Edge Server functionality such as the ECSpec Editor. Refer to "Related Information" on page 1-2 to determine which other documents you may need to consult.

This document is intended for the following audiences:

- System administrators—IT professionals who configure and deploy Edge Flows. Related responsibilities may include installing, testing, maintaining, and upgrading RFID Edge Server. System administrators understand networking and Web protocols, as well as Java and XML.
- Application developers—Java programmers who focus on developing client applications using the Application Level Events (ALE) API, and working with other engineering, quality assurance (QA), and database teams. Application developers have in-depth, working knowledge of Java.

Guide to This Document

This document is organized as follows:

- This chapter, Introduction and Roadmap, describes the scope of this guide and lists related documentation.
- Introduction and Roadmap is an overview of RFID Edge Server Edge Flow functionality and describes how to configure Edge Flow modules.
- Edgeflow Module Reference provides field-by-field descriptions of the Edge Flow modules.
- Writing Custom Modules describes how to write custom Edge Flow modules to satisfy specific use cases not handled by the standard set of modules.
- Workflow Module Reference for Versions Prior to 3.0 provides field-by-field descriptions of workflow module configuration options.

Related Information

The following BEA documents contain information that is relevant to the *WebLogic RFID Edge Server Edge Flow Reference*:

- *Product Overview* provides descriptive information about WebLogic RFID Edge Server, including use cases, key benefits, component architecture, functional overview, RFID concepts and terminology, and developer and RFID resources.
- *Using the RFID Edge Server Administration Console* describes how to use the RFID Administration Console to configure ECSpecs, ECReports, RFID devices, filters, and Edge Flows.
- *RFID Reader Reference* describes how to configure the RFID devices supported by the RFID Edge Server.
- *Programming with the ALE and ALEPC APIs* describes the Application Level Events (ALE) and Application Level Events Programming Cycle (ALEPC) programming interfaces (APIs) that you use to create applications that interact with WebLogic RFID Edge Server by reading and writing electronic product code (EPC) tags.
- *Using the Reader Simulator* describes how to use the Reader Simulator software included with RFID Edge Server. The Reader Simulator minimally simulates a ThingMagic Mercury4 RFID reader.
- *WebLogic RFID Enterprise Server Administration Guide* provides further information on the EPC Information Service (EPCIS), and EPCIS events. Consult this document if you want to configure Edge Flows to send EPCIS events.
- *Edge Flow and Driver Javadocs* provides reference documentation for the Edge Flow and Driver development packages that are provided with WebLogic RFID Edge Server.

Configuring and Using Edgeflows

The following sections describe how to configure and use edgeflows:

Overview of RFID Edgeflow Technology

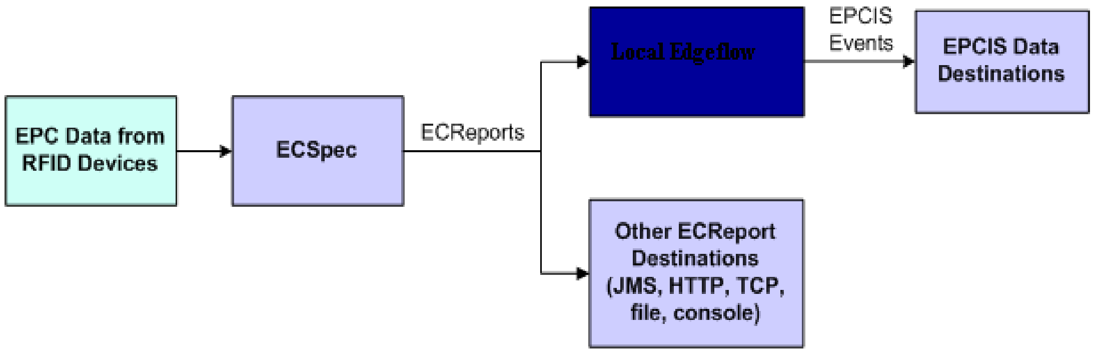
The ALE engine of WebLogic RFID Edge Server collects and filters data from RFID-labeled items moving through the supply chain. This process involves no business logic. However, data gathered during this process can be sent through local **edge flows** on the Edge Server, which apply business logic to the RFID data at the edge. Data that results from this workflow processing may be sent to the organization's enterprise data center or other destinations.

You construct edge flows to satisfy use cases. The edge flows are constructed from **Edgeflow modules**, which take input messages (event or programming cycle reports, or output from other modules), and emit edge flow messages according to embedded business logic.

Edge flows can be viewed as a series of actions triggered by a related event. For example, the observation of an EPC on an RFID-tagged case of consumer goods travelling down a conveyor belt could increment an LED counter, change a stack light to green, send an EPCIS event, or send a notification to another system.

Figure 2-1 shows edge flows in the context of EPC data.

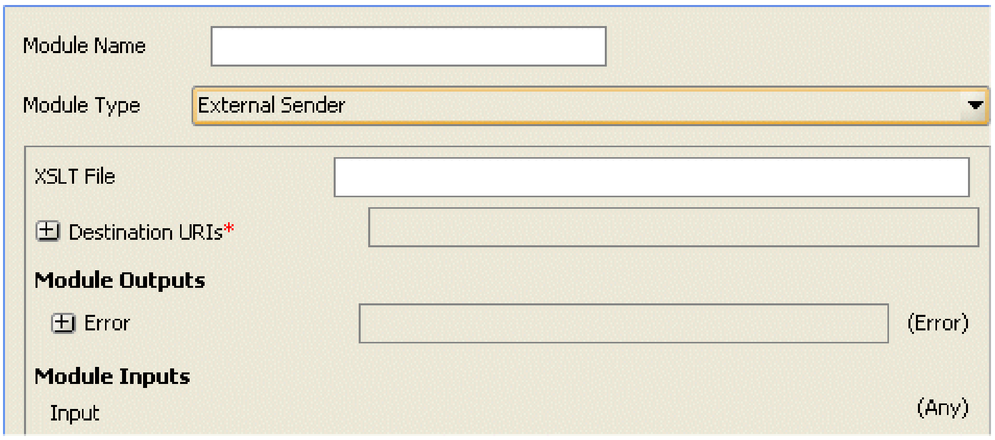
Figure 2-1 Edgeflows in Context


EPC Data from RFID Devices
ECSpec
ECReports
Local Edgeflow
EPCIS Events
EPCIS Data Destinations
Other ECReport Destinations (JMS, HTTP, TCP, file, console)

Structure of Edgeflow Modules

Every edgeflow module has configuration parameters that determine the exact behavior of each module instance. For example, the External Sender module has parameters that let the user specify an XSLT file, destination URI(s), input data connections, and error output connections for a given instance of that module, as shown in Figure 2-2.

Figure 2-2 External Sender Module



Module Name

Module Type External Sender

XSLT File

⊞ Destination URIs*

Module Outputs

⊞ Error (Error)

Module Inputs

Input (Any)

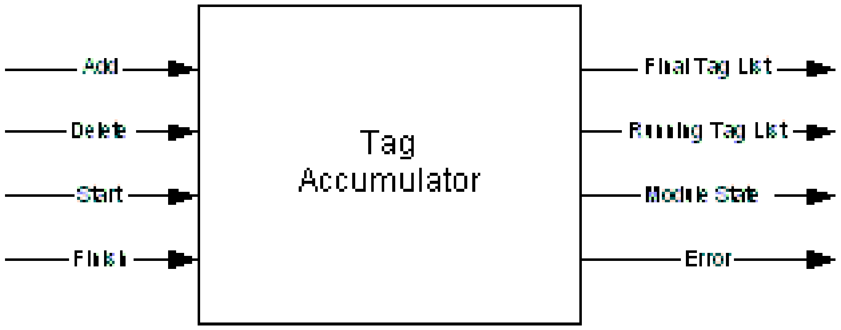
An edgeflow module instance can receive data from, and send data to, other module instances. It sends data to other module instances through **output ports** and receives data from other module instances through **input ports**.

A module can have multiple input ports and multiple output ports. The different ports are typically based on the different functions of the ports. For example, the Tag Accumulator module adds the tags it receives on its Add port to its internally-accumulated list and deletes the tags it receives on its Delete port from its internal list. Similarly, it sends the running list of tags on the Running Tag List output port and the final list of tags on the Final Tag List output port.

The input and output ports of edgeflow modules are typed, which means that they accept or send only specific data types defined by the port. For example, the Tag List Filter module accepts only ECReports in its input port and sends only Tag Lists through its tag list output port.

The structure of the edgeflow modules is illustrated in Figure 2-3.

Figure 2-3 Edgeflow Module Structure



Each module can contain the following:

- A unique name
- Configuration parameters that control the behavior of the module
- Input ports: Data and Control (Start, Stop)
- Output Ports: Data, State and Error

The number of input and output ports is based on the function of the module. All ports are typed.

The WebLogic RFID Edge Server provides the edgeflow modules shown inTable 2-1.

Table 2-1 Available Edgeflow Modules

Aggregation Reconciler	Bar Code Decoder	Counter
Directional Filter	Directional Filter (Auto Configuration)	ECSpec Subscriber
EPC Cache Manager	EPCIS Aggregation Event Generator	EPCIS Manifest Query
EPCIS Object Event Generator	External Sender	HTTP Receiver (One Way)
HTTP Receiver (Request Response)	Numeric Display (EDI111 LED)	Preset Message (Dynamic)
Preset Message (Static)	Serial Number Replenisher (BEA Service)	Serial Number Replenisher (Static)
Stack Light	State Filter	Tag Accumulator
Tag Counter	Tag List Filter	Tag Verifier
Trigger Receiver	Wait For Time	Wait For Trigger
Write Tag	XML Generator	XML Parser
XML Transformer		

Differences Between Legacy Workflow Modules and Edgeflow Modules

Prior to the version 3.0 release of the Edge Server, workflow modules were large single-focused components that addressed specific use cases. These modules included a Bidirectional Portal Module, an Observe Portal Module, and Numeric Display Module, Stack Light Module and Message Notifier Module helper modules

Starting with version 3.0 of the Edge Server, limited support is provided for the legacy modules. New legacy module instances cannot be created using the Admin Console and messaging between the legacy modules and the new edgeflow modules is not supported. For more information about the legacy modules, see "Workflow Module Reference for Versions Prior to 3.0" on page 5-1.

The legacy modules are replaced by a number of small, reusable edgeflow modules that provide finer granularity functions and can be used as building blocks to satisfy use cases. The building blocks can be used to create simple or complex edgeflows and provide a flexibility that minimizes the need to write custom modules.

Prerequisites to Configuring Edgeflows

Before you configure an edgeflow, you may need to configure the ECSpec whose reports will provide input to the edgeflow. See *Using the RFID Edge Server Administration Console* for instructions on configuring an ECSpec.

You must also have a working installation of the WebLogic RFID Edge Server, including one or more hardware devices (stack lights, RFID devices, and LED displays).

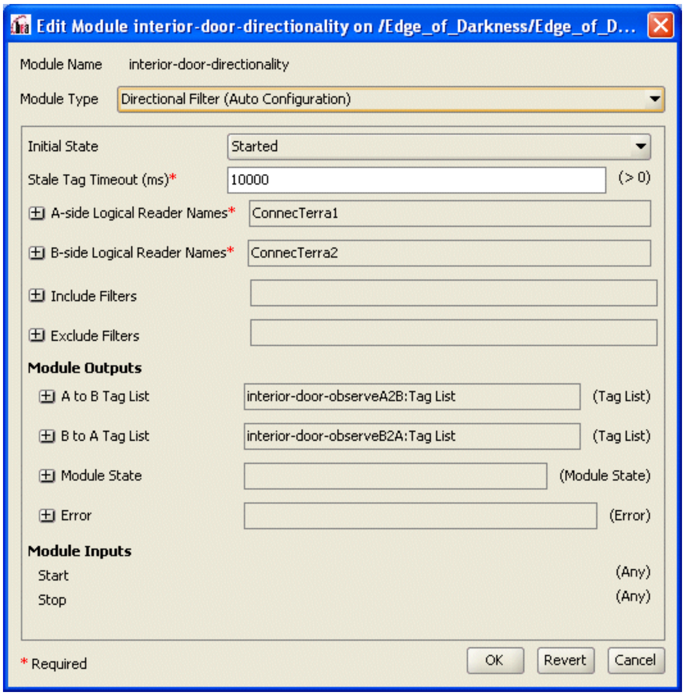
If your edgeflow sends EPCIS events to the RFID Enterprise Server, you will also need a working installation of the WebLogic RFID Enterprise Server.

Configuring Edgeflows

You configure edgeflows by configuring edgeflow modules and connecting them together, using the RFID Edge Server Administration Console. Edgeflow modules are connected to one another by referring to the name of the module and the name of the input port that is receiving output from another module.

Consider the example shown in Figure 2-4. The B to A Tag List output of this Directional Filter (Auto Configuration) module is sent to the Tag List input port of the interior-door-observeB2A EPCIS Object Event Generator module.

Figure 2-4 Connecting Edgeflow Modules


Edit Module interior-door-directionality on /Edge_of_Darkness/Edge_of_D...

Module Name	interior-door-directionality	
Module Type	Directional Filter (Auto Configuration)	
Initial State	Started	
Stale Tag Timeout (ms)*	10000	(> 0)
A-side Logical Reader Names*	ConnecTerra1	
B-side Logical Reader Names*	ConnecTerra2	
Include Filters		
Exclude Filters		
Module Outputs		
A to B Tag List	interior-door-observeA2B:Tag List	(Tag List)
B to A Tag List	interior-door-observeB2A:Tag List	(Tag List)
Module State		(Module State)
Error		(Error)
Module Inputs		
Start		(Any)
Stop		(Any)

* Required

Creating an Edgeflow Module: Main Steps

To define an edgeflow module:

1. Start the RFID Edge Server Administration Console if it is not already running.
2. Select the **Edgeflow Modules** node under the Site and Edge Server you are working with.
3. Choose the type of operation to perform:
   a. To define a new module, click **New**.
   b. To create a copy of an existing module, select the module and click **Clone**.
4. Give the module you are creating a unique **Module Name**.
5. Choose the **Module Type** from the drop-down list.

6. Configure the edgeflow module by filling in values for the fields according to the information provided in "Edge Flow Module Reference" on page 3-1.

7. Click **OK** to save your changes.

Defining an Edgeflow: Main Steps

To define an edgeflow:

1. Start the RFID Edge Server Administration Console if it is not already running.

2. Check that notification drivers are set up for all types of notifications you will be using.

   These drivers are defined in the `RFID_EDGE_HOME/etc/edge.props` file, where `RFID_EDGE_HOME` is the directory where you installed the RFID Edge Server software.

3. Configure any RFID devices you will be using.

   For testing purposes, as an RFID device, you can configure the Reader Simulator software included with the RFID Edge Server.

   For more information on RFID device configuration, see the *RFID Reader Reference* and any manufacturer documentation provided with your hardware.

4. Create the ECSpecs and associated ECReports required by this edgeflow.

   For more information on creating ECSpecs, see *Using the RFID Edge Server Administration Console*.

5. Define edgeflow modules as described in steps 1-7, in "Creating an Edgeflow Module: Main Steps" on page 2-6.

6. Create edge flow subscriptions for ECSpecs, as necessary.

   a. Select `EdgeFlow Module` for the **Type** field.

   b. The **Module** field should contain `<edgeflow module name>:port-name`. For example, `door-taglist:Start`.

7. When you click **Subscribe**, the ECSpec will send ECReports to the edgeflow destination you specified.

   Active subscriptions allow you to test that the edgeflow is functioning correctly, and that the Edge Server is sending messages to the configured locations and devices.

8. Test the edgeflow by simulating a tag observation using the Reader Simulator, or by holding a tag near the antennas on your RFID reader.

You should see evidence of edgeflow activity (Edgeflow messages arriving at notification destinations and delivery of ECReports).

Edgeflow Examples

The RFID Edge Server 3.0 software provides a number of sample edgeflows, available in the `RFID_EDGE_HOME/samples/EdgeFlows` directory. Each of the edgeflow samples can be imported into a RFID Edge Server 3.0 system and modified to meet specific customer needs.

Table 2-2 Sample Edgeflows

ShippingReceivingDoor	Demonstrates a simple observe use case, where EPCIS object events are generated.
StoreInteriorDoor	Demonstrates determining the direction in which tags are moving and generating related events.
PalletBuildingStation	Demonstrates a pallet building use case, in which EPCIS Aggregation events are generated.
SmartShelf	Demonstrates a smart shelf use case.
TagCommisioning	Demonstrates a tag printing use case.
PalletReconciliation	Demonstrates how to use edgeflow modules to perform a pallet reconciliation; that is, shipment reconcilliation against a manifest.

Use the following steps to import the sample edgeflows:

1. Start the following components:
   a. Ensure that your RFID hardware is connected and powered on or start the Reader Simulator.
   b. Start the Edge Server.
   c. Start the Administration Console
   d. Start the Reader Simulator
2. In the Administration Console, import the edgeflow example of you choice:
   Select **File** > **Import**
   EDGE_SERVER_HOME\samples\EdgeFlows\ExampleDirectory\ExampleName.xml

This imports the ECSspec (if applicable), the edgeflow modules, and starts the edgeflow. You can view or modify the module configurations through the Administration Console.

The following sections describe the sample edgeflows. You can load each of the edgeflow samples and use the Administration Console to view the configuration properties of each module.

Note: The first example, Example: Simple Observe Edgeflow, uses screen shots to provide some context about how edgeflow modules connect via the user interface. Having established this context, subsequent examples do not require or include the screen shots.

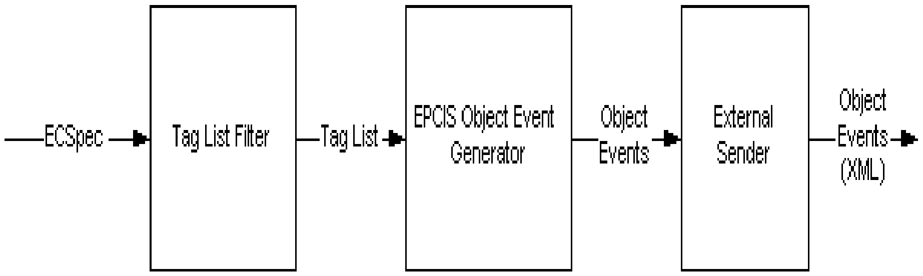
Example: Simple Observe Edgeflow

This example (shipping-receiving-door.xml) creates an edgeflow that reads tags, creates an `OBSERVE` Object Event for each tag, and sends the event to the Edge Server console. This example represents a shipping or receiving door that reads all tags and reports `OBSERVE` events for the tags that pass through.

The example uses the reader simulator and reports events to the Edge Server console. In a production environment, the reader simulator would be replaced by a real reader and the events would be reported to an external system using JMS or HTTP.

A diagram of this edgeflow is shown in Figure 2-5.

Figure 2-5 Example Observe Portal

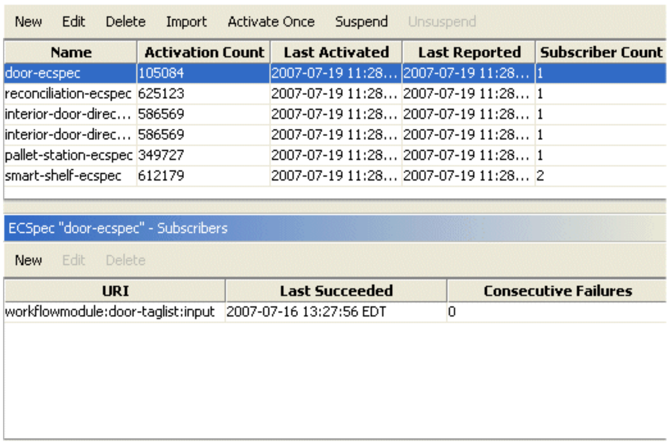


The following sequence summarizes the functions of each component of this edgeflow:

1. An ECSpec (door-ecspec) is configured to always read from antenna 1 (ConnecTerra1) of the reader simulator. This ECSpec has one ECReport, report_0, which contains an EPC List.

2. A subscription to the door-ecspec ECSpec is made by the door-taglist Tag List Filter module, as shown in Figure 2-6. The Tag List Filter module extracts tag information from the EPC List in an ECReport and creates a tag list.

Figure 2-6 Tag List Filter Subscription to door-ecspec



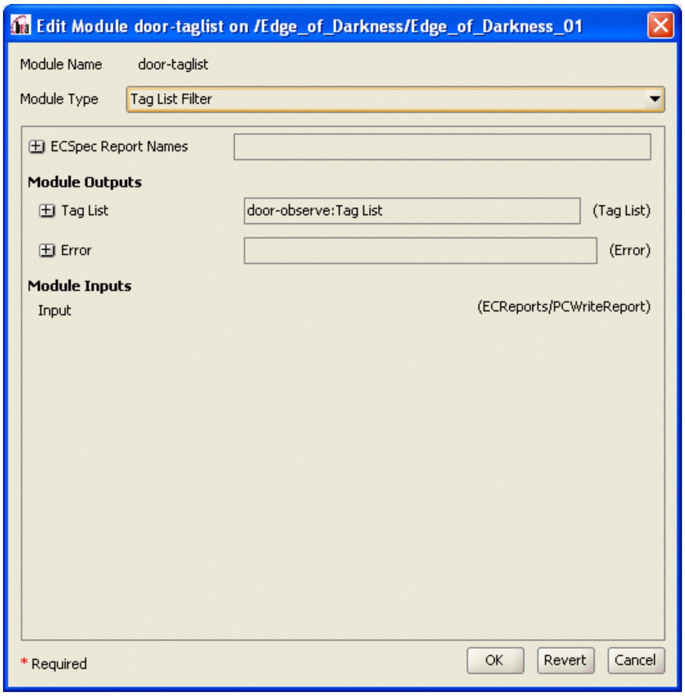
Name	Activation Count	Last Activated	Last Reported	Subscriber Count
door-ecspec	105084	2007-07-19 11:28...	2007-07-19 11:28...	1
reconciliation-ecspec	625123	2007-07-19 11:28...	2007-07-19 11:28...	1
interior-door-direc...	586569	2007-07-19 11:28...	2007-07-19 11:28...	1
interior-door-direc...	586569	2007-07-19 11:28...	2007-07-19 11:28...	1
pallet-station-ecspec	349727	2007-07-19 11:28...	2007-07-19 11:28...	1
smart-shelf-ecspec	612179	2007-07-19 11:28...	2007-07-19 11:28...	2

ECSpec "door-ecspec" - Subscribers

URI	Last Succeeded	Consecutive Failures
workflowmodule:door-taglist:input	2007-07-16 13:27:56 EDT	0

3. The Tag List Filter module extracts the tag information from the ECReports created by the ECSpec, and sends the resulting tag list to the door-observe EPCIS Object Event Generator module, as shown in Figure 2-7.

Figure 2-7 Tag List Filter Module Sends Output to EPCIS Object Event Generator



Edit Module door-taglist on /Edge_of_Darkness/Edge_of_Darkness_01

Module Name door-taglist

Module Type Tag List Filter

ECSpec Report Names

Module Outputs

Tag List door-observe:Tag List (Tag List)

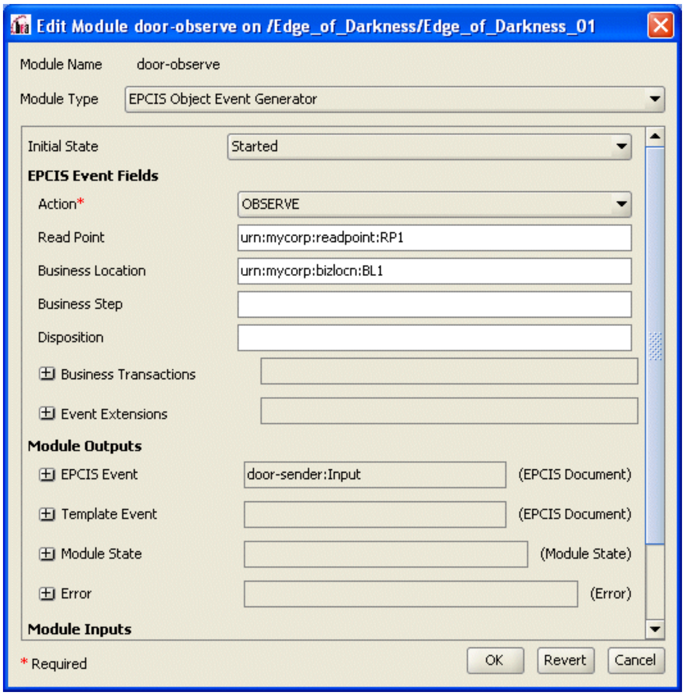
Error (Error)

Module Inputs

Input (ECReports/PCWriteReport)

4. The EPCIS Object Event Generator module (door-observe) creates an OBSERVE object event for **each** tag it receives and sends the object event (in an EPCIS Document) to the External Sender module, as shown in Figure 2-8.

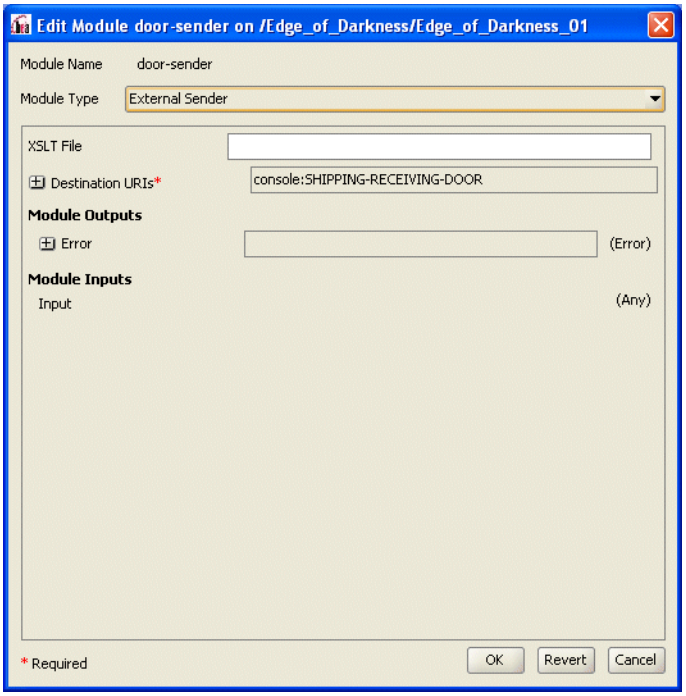
Figure 2-8 EPCIS Object Event Generator Sends Output to External Sender


Edit Module door-observe on /Edge_of_Darkness/Edge_of_Darkness_01

Module Name	door-observe
Module Type	EPCIS Object Event Generator
Initial State	Started
EPCIS Event Fields	
Action*	OBSERVE
Read Point	urn:mycorp:readpoint:RP1
Business Location	urn:mycorp:bizlocn:BL1
Business Step	
Disposition	
Business Transactions	
Event Extensions	
Module Outputs	
EPCIS Event	door-sender:Input (EPCIS Document)
Template Event	(EPCIS Document)
Module State	(Module State)
Error	(Error)
Module Inputs	

5. The External Sender module (door-sender) receives the EPCIS Document, converts it to its XML representation, and sends it to the Edge Server console, as shown in Figure 2-9.

Figure 2-9 External Sender Sends OBSERVE Event to the Edge Server Console



Edit Module door-sender on /Edge_of_Darkness/Edge_of_Darkness_01

Module Name	door-sender	
Module Type	External Sender	
XSLT File		
⊞ Destination URIs*	console:SHIPPING-RECEIVING-DOOR	
Module Outputs		
⊞ Error		(Error)
Module Inputs		
Input		(Any)

* Required

The destination URI for the External Sender module is defined as: *console:SHIPPING-RECEIVING-DOOR.*

You could instead configure this destination URI to use HTTP to send the events, using the following format: *http://<host>:<port>/module/<modulename>:<portname>*.

EXAMPLE: Pallet Building Edgeflow

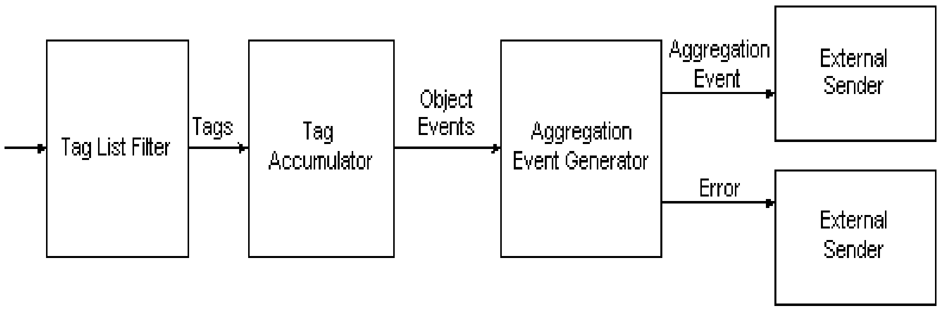
This sample (pallet-building-station) demonstrates a pallet building use case. In the example, the EPCs that are read during the process are accumulated to create and EPCIS Aggregation event for the pallet.

The aggregation event is created when no tags are detected for 10 seconds (the Tag Accumulator module is configured to accumulate for 10 seconds).

The aggregation event is then sent to a specified destination. In this example, the event is sent to the Edge Server console. Any errors that occur during this process are also sent to the Edge Server console.

A diagram of this example is shown in Figure 2-10.

Figure 2-10 Example Pallet Building Station



The following sequence summarizes the function of each component of the Pallet Building station example:

1. An ECSpec (pallet-station-ecspec) is configured to always read from antenna 1 (ConnecTerra1) of the reader simulator. This ECSpec has one ECReport, report_0, which contains an EPC List and a Tag List.
2. A Tag List Filter module (pallet-station-taglist) subscribes to the ECSpec, takes the ECReports produced by the ECSpec, extracts the tag information, and sends it to the Add port of a Tag Accumulator module.
3. The Tag Accumulator module (pallet-station-accumulator) accumulates the tags until it detects that no tags have been read for a 10-second period. This 10-second time period is configured by a value of 10000 milliseconds in the Stable Set field.

   After the 10 seconds have passed, the Final Tag List is sent to an EPCIS Aggregation Event Generator module Tag List input.
4. The EPCIS Aggregation Event Generator module (pallet-station-aggregator) identifies EPCs that contain the pattern `urn:epc:pat:gid-64-i:10.70.*` as being a parent. All other tags are used as child tags.

An aggregation ADD event is generated from the Tag List input and sent to an External Sender module. If an error occurs during this process, an error message is sent to an additional External Sender module.

5. An External Sender module (pallet-station-sender) sends the generated aggregation to the Edge Server console via the Destination URIs configuration field.

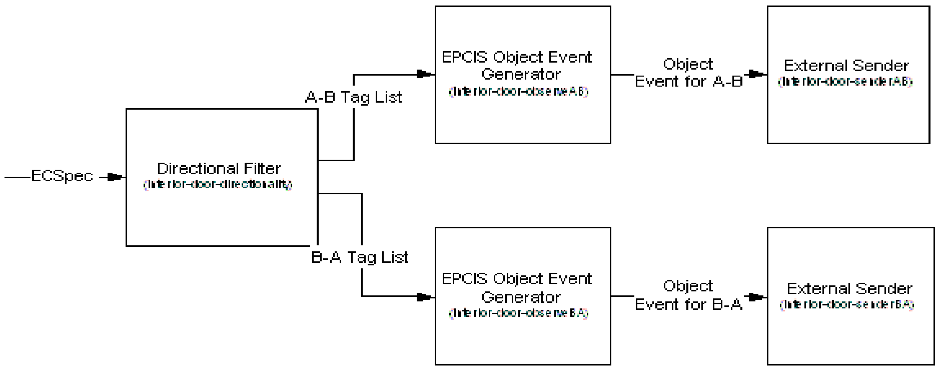
6. An External Sender module (pallet-station-error-sender) sends any errors that occurred to the Edge Server console via the Destination URIs configuration field.

Example: Directionality

This sample (store-interior-door.xml) demonstrates a use case where tags, and the direction they are moving while passing through a door, are detected. For the detected tags, an EPCIS event is generated, and includes the detected direction of a tag while passing through a doorway. The EPCIS event is sent to external destinations.

Figure 2-11 shows a diagram of this example.

Figure 2-11 Directionality Example



The following sequence summarizes the function of each component of the Directionality edgeflow:

1. The examples does not require that the user explicitly create an ECSpec.

   Instead, the Directional Filter (Auto Configuration) module (interior-door-directionality) creates an ECSpec for the read points (configured via the A-side and B-side logical reader names) and subscribes itself to receive notifications of EPC sightings. Logical reader names are required to be configured for each point .

Specifically, it configures two ECSpecs (interior-door-directionality-DirectionalityPointA and interior-door-DirectionalityPointB) that read from antenna 1 (ConnecTerra1) and antenna 2 (ConnecTerra) of the reader simulator, respectively.

Consider the antenna locations to be the two sides of the door.

2. The Directional Filter (Auto Configuration) module then uses the ECReports produced by the ECSpecs to determine tag direction. The following ECReports are generated:
   - Add report for point A
   - Delete report for point A
   - Add report for point B
   - Delete report for point A
3. The Directional Filter (Auto Configuration) module has two tag lists output ports: one for tags moving from point A to point B and one for tags moving from point B to point A.

   The tag lists are sent to two EPCIS Object Event Generator modules: interior-door-observeA2B:Tag List and interior-door-observeB2A:Tag List
4. The EPCIS Object Event Generator module (interior-door-observeA2B) creates `OBSERVE` object events for tags that are read moving from point A to Point B. These events are sent to an External Sender module, interior-door-senderA2B:Input as EPCIS documents.
5. An EPCIS Object Event Generator module (interior-door-observeB2A) creates `OBSERVE` object events for tags that are read moving from point B to point A. These events are sent to an External Sender module, interior-door-senderB2A:Input as EPCIS documents in XML format.
6. External Sender modules (interior-door-senderA2B) and (interior-door-senderB2A) send the Object events to two different ports of the Edge Server console based on their direction, console:TO-SALES-FLOOR or console:TO-STORE-ROOM.

Example: Pallet Reconciliation

This example (pallet-reconcilliation.xml) demonstrates a pallet reconciliation use case that includes sending HTTP messages between edgeflow modules and external systems.

There are two components of this example: the edge server edgeflow component that you import via pallet-reconcilliation.xml, and a target Web container component that generates and returns a manifest.

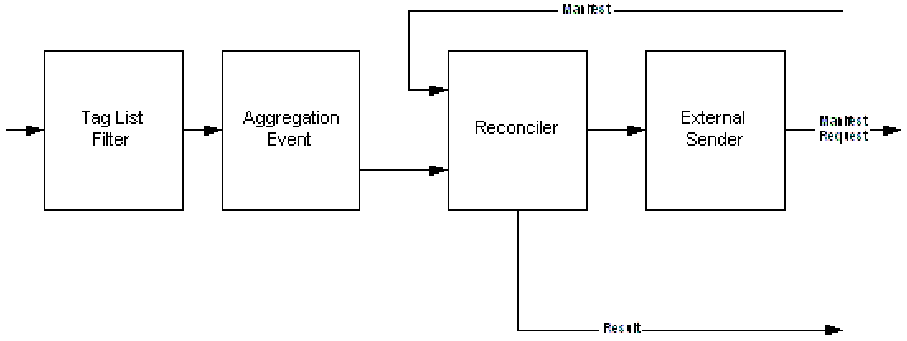
For the Web container component, the example includes Java code, XML files, and .JSP files to create a manifest in response to a request. The example is already built, but includes a build file in case you want to make changes and rebuild.

Note: Your modules would typically request the manifest from an external system. However, to keep the example simple it assumes that the supplied web client on localhost is used to get the manifest.

The directions for running this portion of the example are provided in Configuring the External System to Generate the Manifest.

Figure 2-12 provides a diagram of this example.

Figure 2-12 Pallet Reconciliation



The following sequence summarizes the function of each component of the Pallet Reconciliation example:

1. An ECSpec (reconciliation-ecspec) reads tags added at Antenna 1 of the Reader Simulator.

2. A subscription to the ECSpec is made by a Tag List Filter module (reconciliation-aggregationTagListBuilder). This module takes the ECReports produced by the ECSpec, extracts the tag information and sends it to the Tag List input port of an EPCIS Aggregation Event Generator module.

3. An Aggregation Event Generator module (reconciliation-aggregationEventGenerator) creates an aggregation for the tags. The parent tag is defined as `urn:epc:id:gid:10.10.1` and EPCs that match `urn:epc:pat:gid-64-i:*.*.[2-7]` are defined as children in the aggregation. An aggregation event is generated and sent to an Aggregation Reconciler module. If an error occurs during processing, the error is sent to an External Sender module.
4. An Aggregation Reconciler module (reconciliation-reconciler) receives the aggregation from the aggregation generator and sends to an External Sender to request a manifest for the aggregation. The manifest is received on the Aggregation Reconciler module Manifest input port.

   The acceptable match is 100 percent and extra children are allowed.

   The Aggregation Reconciler module uses five external sender modules to send the results of the reconciliation to different ports on the Administration Server Console:

   reconciliation-result-parentMatched-sender:Input sends the result of reconciliation (boolean indicating success or failure).

   reconciliation-result-childrenMatched-sender:Input sends the result of reconciliation (boolean indicating success or failure).

   reconciliation-result-missing-sender:Input sends a manifest identifying any missing entries.

   reconciliation-result-extra-sender:Input sends a manifest identifying extra entries.

   reconciliation-result-error-sender:Input sends an error report.
5. The External Sender module (reconciliation-aggregationEventExternalSender) receives the aggregation on its input port (reconciliation-aggregationEventExternalSender:Input), and sends the aggregation to the external system (localhost is used in the example for simplicity) as an HTTP message.

   The External Sender also sends console notification so that the Edge Server console displays the aggregation event sent.

Configuring the External System to Generate the Manifest

The Web container portion of this example is already built and you do not generally need to rebuild it. However, if you make changes and need to rebuild, the build file is located in `samples\EdgeFlows\PalletReconciliation\manifestGenerator\build.bat`.

Use the following steps to run this example:

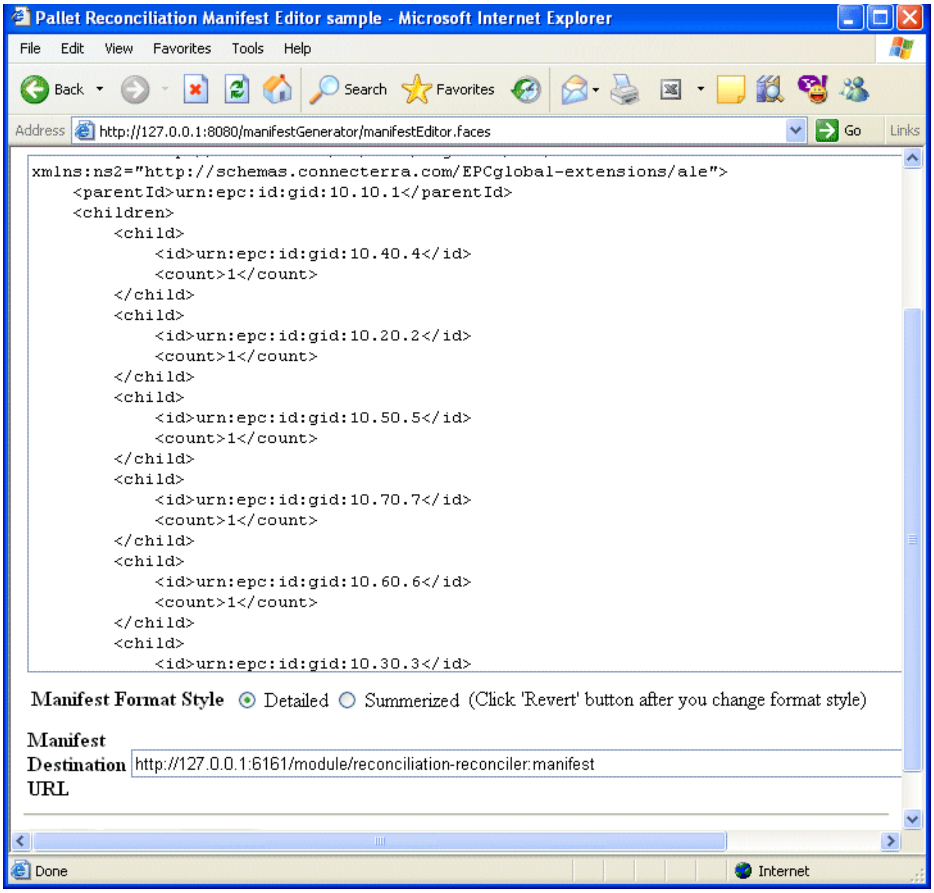
1. Deploy the application to your Web container. For example, you can copy the entire manifestGenerator directory to the WebApps directory of Apache Tomcat and then deploy the manifestGenerator application.
2. The default URL target is: http://localhost:8080/manifestGenerator/eventReceiver.

   If the context path for the application is not http://localhost:8080/manifestGenerator, then you must change the destination URIs field in the reconciliation-aggregationEventExternalSender module.
3. Start a browser window pointing to the sample context, which is http://localhost:8080/manifestGenerator unless you have changed it.
4. Using the Edge Server Administration Console, Select ECSpec > reconciliation-ecspec > Suspend.
5. Start the Reader Simulator and select all tags on Antenna 1.
6. Return to the ECSpec pane, select reconciliation-ecspec > Unsuspend. Wait a few moments and then select Suspend. This creates a notification with tags selected on Antenna 1.
7. On the Edge Server console, you will see the Aggregation Event Sent to Manifest Generator message, such as the following:

```
<!-- AggregationEventSentToManifestGeneratorClient -->

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<ns7:EPCISDocument
xmlns:ns4="http://www.bea.com/ns/rfid/enterprise/epcis-captur

e/xsd/2.0/" xmlns:ns7="urn:epcglobal:epcis:xsd:1"
xmlns:ns3="urn:epcglobal:epcis

-query:xsd:1"
xmlns:ns5="http://www.bea.com/ns/rfid/enterprise/epcis-masterdata/

xsd/2.0"
xmlns:ns2="http://www.unece.org/cefact/namespaces/StandardBusinessDocum

entHeader" xmlns:ns6="urn:epcglobal:epcis-masterdata:xsd:1">

    <EPCISBody>

        <EventList>

            <AggregationEvent>

                <eventTime>2007-07-24T13:59:42.725-04:00</eventTime>

                <parentID>urn:epc:id:gid:10.10.1</parentID>
                <childEPCs>
                    <epc>urn:epc:id:gid:10.40.4</epc>
                    <epc>urn:epc:id:gid:10.20.2</epc>
                    <epc>urn:epc:id:gid:10.50.5</epc>
                    <epc>urn:epc:id:gid:10.70.7</epc>
                    <epc>urn:epc:id:gid:10.60.6</epc>
                    <epc>urn:epc:id:gid:10.30.3</epc>
                </childEPCs>
                <action>OBSERVE</action>
                <bizStep></bizStep>
                <disposition></disposition>
                <readPoint>
                    <id></id>
                </readPoint>
                <bizLocation>
                    <id></id>
                </bizLocation>
                <bizTransactionList/>
            </AggregationEvent>
        </EventList>
    </EPCISBody>
</ns7:EPCISDocument>
```

8. On the browser window, the page displays notification of a new aggregation event arrival and a generated manifest. A sample browser window is shown in Figure 2-13.

Figure 2-13 Browser Window Displaying Generated Manifest


Pallet Reconciliation Manifest Editor sample - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://127.0.0.1:8080/manifestGenerator/manifestEditor.faces
Go Links
Done
Internet

```
xmlns:ns2="http://schemas.connecterra.com/EPCglobal-extensions/ale">
    <parentId>urn:epc:id:gid:10.10.1</parentId>
    <children>
        <child>
            <id>urn:epc:id:gid:10.40.4</id>
            <count>1</count>
        </child>
        <child>
            <id>urn:epc:id:gid:10.20.2</id>
            <count>1</count>
        </child>
        <child>
            <id>urn:epc:id:gid:10.50.5</id>
            <count>1</count>
        </child>
        <child>
            <id>urn:epc:id:gid:10.70.7</id>
            <count>1</count>
        </child>
        <child>
            <id>urn:epc:id:gid:10.60.6</id>
            <count>1</count>
        </child>
        <child>
            <id>urn:epc:id:gid:10.30.3</id>
```

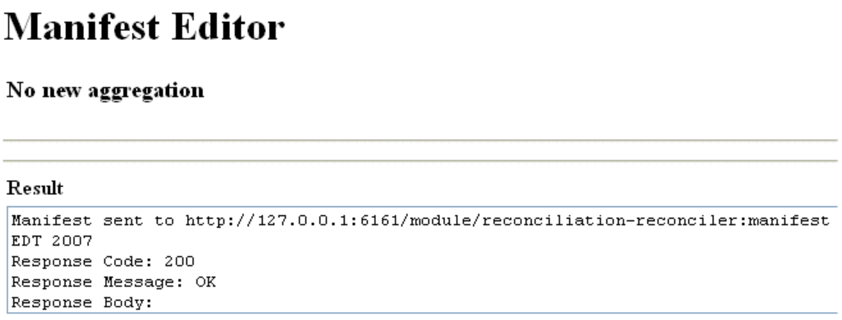
Manifest Format Style [✓ Detailed] Summerized (Click 'Revert' button after you change format style)

Manifest Destination URL: http://127.0.0.1:6161/module/reconciliation-reconciler:manifest

9. You can edit the manifest and click the Send button to send it to the Aggregation Reconciler (reconciliation-reconciler). The manifest for the aggregation is sent as an HTTP message to the manifest input port of the module. Results are produced based on the manifest being used to reconcile the aggregation.

10. You should see results of the send in the status area of the Web browser page and on the Edge Server console. A sample Web browser page is shown in Figure 2-14.

Figure 2-14 Browser Window Confirming Send Status


Manifest Editor

No new aggregation

Result

```
Manifest sent to http://127.0.0.1:6161/module/reconciliation-reconciler:manifest
EDT 2007
Response Code: 200
Response Message: OK
Response Body:
```

11. Sample reconciliation output displayed on the Edge Server console is shown in Listing 2-1.

Listing 2-1 Sample Edge Server Reconciliation Output

```
<!-- ReconciliationResult-ParentMatched -->

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<ns4:PrimitiveType xsi:type="xs:boolean"
xmlns:ns4="http://www.bea.com/ns/rfid/e

dgeflow/xsd/1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns3

="urn:epcglobal:ale:xsd:1" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:ns2

="http://schemas.connecterra.com/EPCglobal-extensions/ale">true</ns4:Pr
imitiveTy

pe>


<!-- ReconciliationResult-ChildrenMatched -->

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<ns4:PrimitiveType xsi:type="xs:boolean"
xmlns:ns4="http://www.bea.com/ns/rfid/e
```

```
dgeflow/xsd/1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns3
="urn:epcglobal:ale:xsd:1" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:ns2
="http://schemas.connecterra.com/EPCglobal-extensions/ale">true</ns4:Pr
imitiveTy
pe>


<!-- ReconciliationResult-Extra -->
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns4:Manifest xmlns:ns4="http://www.bea.com/ns/rfid/edgeflow/xsd/1.0"
xmlns:ns3=
"urn:epcglobal:ale:xsd:1"
xmlns:ns2="http://schemas.connecterra.com/EPCglobal-ex
tensions/ale">
    <children/>
</ns4:Manifest>


<!-- ReconciliationResult-Missing -->
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns4:Manifest xmlns:ns4="http://www.bea.com/ns/rfid/edgeflow/xsd/1.0"
xmlns:ns3=
"urn:epcglobal:ale:xsd:1"
xmlns:ns2="http://schemas.connecterra.com/EPCglobal-ex
tensions/ale">
    <children/>
</ns4:Manifest>
```

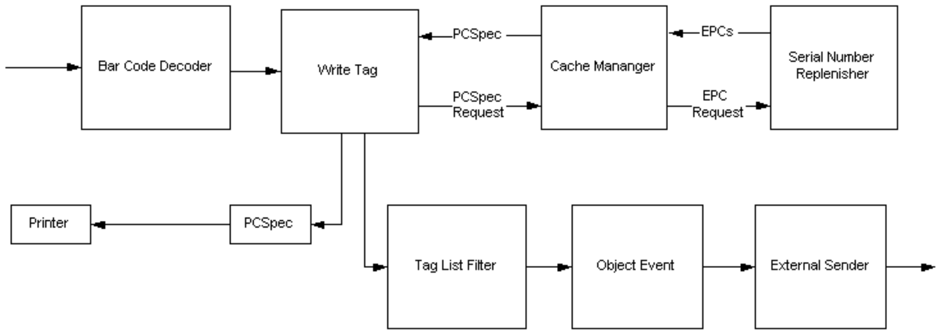
Note: The reconciliation-reconciler module waits indefinitely for manifest messages for every aggregation event. If you cannot send a manifest to this module, you may need to reset the module using the Administration Console: Select reconciliation-reconciler > Configure > OK. This will reset the module and return it to the normal ready state.

Example: Tag Commissioning

This sample demonstrates how to configure printing-related modules to print EPCs, based on bar code input.

As part of this sample, a GUI (invoked via the run.bat file) is provided to allow users to send bar code data to the printing modules. When bar code data is sent to the modules, an EPC is printed to the selected EPC field of Antenna 1 of the Reader Simulator.

Figure 2-15 Tag Commissioning



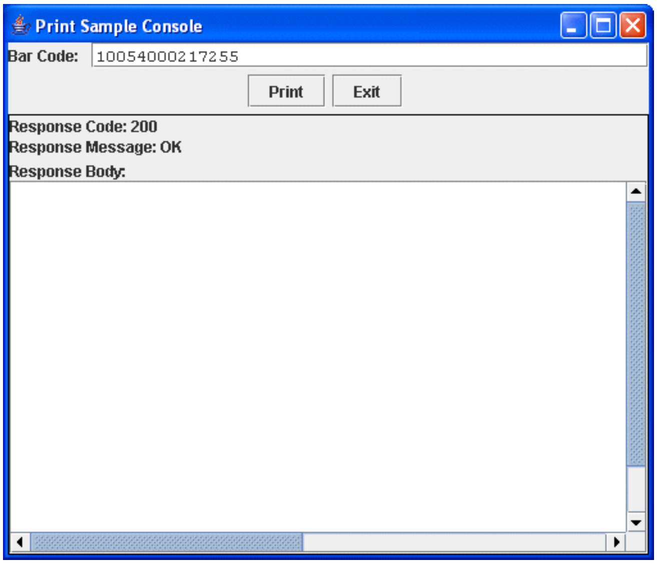
The following sequence summarizes the function of each component of the Tag Commissioning example:

1. An HTTP Receiver (Request Response) module (tag-commissioning-http), accepts barcode input from the user and sends it to a Bar Code Decoder module.

   To run this example, the bar code GUI (invoked via run.bat), which is shown in Figure 2-16, sends the bar code you specify. The status area displays the response.

   You must enter a bar code that conforms to one of the supported formats.

Figure 2-16 Print Sample Console Bar Code GUI



Bar Code: 10054000217255

Response Code: 200
Response Message: OK
Response Body:

2. The Bar Code Decoder module (tag-commissioning-barcode) converts the bar code to an EPC pattern required by the printer modules and sends it to a Write Tag module. For bar codes that already contain a serial number reference (SSCC) the EPC ID is generated. Otherwise, the EPC ID pattern is generated.

3. The Write Tag module (tag-commissioning-write) contacts an EPC Cache Manager (tag-commissioning-cachemgr:PCSpec Request) to create an edge side cache, creates a PCSpec for the item type, and prints an EPC from the created cache.

   a. The EPC Cache Manager module (tag-commissioning-cachemgr) creates and replenishes the cache by contacting a Serial Number Replenisher (Static) module.

   b. The Serial Number Range (Static) module (tag-commissioning-static-replenish) supplies EPCs for the configured range to the EPC Cache Manager module.

4. Returning to the Write Tag module (tag-commissioning-write) module, it sends a PCWriteReport to the input port of a Tag List Filter module. The Tag List Filter module extracts tag information from the PCWriteReports and creates a tag list.

5. The Tag List Filter module (tag-commissioning-taglist) provides the tag list to the Tag List input of an EPCIS Object Event Generator module.
6. The EPCIS Object Event Generator module (tag-commissioning-event-generator) sends Object Events (the response) to the HTTP Receiver module.

Edgeflow Module Reference

This section describes reference information for the WebLogic RFID Edge Server edgeflow modules.

You may notice similarities across modules. Each module requires parameters for operation (which can include information such as timing data and EPCIS data) and parameters that allow it to connect to other modules and send data to outside destinations.

You configure edgeflow modules using the RFID Edge Server Administration Console. See Configuring Workflow Modules in *Using the Administration Console*.

Note: In version 3.0 of WebLogic RFID Edge Server, workflow modules supported in previous releases are supported in a limited manner. See Workflow Module Reference for Versions Prior to 3.0 for more information.

The following edgeflow modules are described:

- "Aggregation Reconciler" on page 3-3
- "Bar Code Decoder" on page 3-6
- "Counter" on page 3-8
- "Directional Filter" on page 3-11
- "Directional Filter (Auto Configuration)" on page 3-14
- "ECSpec Subscriber" on page 3-16
- "EPC Cache Manager" on page 3-17

- "EPCIS Aggregation Event Generator" on page 3-20
- "EPCIS Manifest Query" on page 3-25
- "EPCIS Object Event Generator" on page 3-27
- "External Sender" on page 3-31
- "HTTP Receiver (One Way)" on page 3-33
- "HTTP Receiver (Request Response)" on page 3-34
- "Numeric Display (EDI111 LED)" on page 3-36
- "Preset Message (Dynamic)" on page 3-38
- "Preset Message (Static)" on page 3-39
- "Serial Number Replenisher (BEA Service)" on page 3-40
- "Serial Number Replenisher (Static)" on page 3-41
- "Stack Light" on page 3-43
- "State Filter" on page 3-46
- "Tag Accumulator" on page 3-47
- "Tag Counter" on page 3-50
- "Tag List Filter" on page 3-52
- "Tag Verifier" on page 3-53
- "Trigger Receiver" on page 3-55
- "Wait For Time" on page 3-56
- "Wait For Trigger" on page 3-57
- "Write Tag" on page 3-58
- "XML Generator" on page 3-60
- "XML Parser" on page 3-61
- "XML Transformer" on page 3-62

Aggregation Reconciler

An Aggregation Reconciler module receives an aggregation event and sends to an external sender to request the manifest for the aggregation.

The Aggregation Reconciler module can be used to find out if an aggregation has the expected items or not. The Aggregation Reconciler module compares an aggregation with its manifest (expected elements in an aggregation). The comparison result is based on the properties such as percentage of children-match required.

Consider the following usage:

1. An EPCIS Aggregation Event Generator creates an aggregation for the tags it receives from a Tag Accumulator module or a Tag List Filter module.
2. An Aggregation Reconciler module receives the aggregation from the EPCIS Aggregation Event Generator module and sends to an EPCIS Manifest Query module or a custom module that generates the manifest for the query.
3. The Aggregation Reconciler receives the manifest and performs the reconciliation.

The Aggregation Reconciler module fields are shown in Table 3-1.

Table 3-1 Aggregation Reconciler Module

Field Name	Value/Type	Description
Module Name	String	Unique user-defined identifier for this module.
Module Type	Aggregation Reconciler.	Type of module. Values are chosen from a list of module types.
Acceptable Children Match %	Percentage (Decimal)	Specifies what percentage of children must be matched to produce a “true” matched-children result. Enter a value between 0 and 100 to specify the match percentage required.

Table 3-1 Aggregation Reconciler Module

Field Name	Value/Type	Description
Allow Extra Children	True/False	Determines whether extra children are allowed when deciding if there is a match. If this is true, extra children produce true for the matched children result. If this is false, detection of extra children produces false for the matched children result.
Match Parent	Never If EPC If Not EPC Always	Specifies whether the parent must be matched when the expected parent is, or is not, an EPC URI.
Module Outputs		
Manifest Request	EPCIS Document	Required. This is the output port through which the request for the manifest is sent. For example, in this field you might specify an EPCIS Manifest Query module or a custom module that generates the manifest for the query.
Parent Matched	Boolean	Specify the location to send a Boolean indicating whether the parent matched.
Children Matched	Boolean	Specify the location to send a Boolean indicating whether the required percentage of children IDs was found.

Table 3-1 Aggregation Reconciler Module

Field Name	Value/Type	Description
Missing Entries	Manifest	Specify the location to send an Aggregation manifest containing IDs indicating which entries were missing. For example, reconciliation-result-missing-sender: Input
Extra Entries	Manifest	Specify the location to send an Aggregation Manifest with the IDs that were in the aggregation, but not expected. For example, reconciliation-result-extra-sender:Inp ut
Error	Error	Specify the error message destination. For example, my-error-sender: Input. This port will send an output if an error occurs in the module processing. This port may be used to indicate any kind of error in an edgeflow.
Module Inputs		

Table 3-1 Aggregation Reconciler Module

Field Name	Value/Type	Description
Aggregation	EPCIS Document	The aggregation received from another module. This is the aggregation that is to be reconciled. For example, the EPCIS Aggregation Event Generator might create an aggregation for the tags it receives. It could then send the aggregation to an Aggregation Reconciler module.
Manifest	Manifest	Manifest that is compared against the aggregation. This could come from a custom module or an EPCIS Manifest Query module.

Bar Code Decoder

This module takes in a bar code string or bar code Tag List and converts it to an EPC ID or EPC ID pattern.

For example, you might use an HTTP Receiver (One Way) module to receive bar codes from an external source, or you might receive a bar code tag list from a Tag List Filter module that itself receives ECReports for an ECSpec configured to read from a bar code reader.

This module supports the following bar code types:

- UCC12
- UCC13
- SSCC
- GTIN
- GLN
- GRAI
- GIAI

Note: Symbology character decoding is not supported; SSCC and SGTIN barcode with AI fields are not supported.

Table 3-2 describes how bar codes are processed based on the number of digits in the bar code.

Table 3-2 How Bar Codes Are Processed

Number of Characters in Bar Code	How Processed
12	1. Padded to 14 characters by prepending zeroes 2. Check if it is GTIN/UCC12 3. If not, check if it is GIAI
13	1. Padded to 14 characters by prepending zeroes 2. Check if it is GTIN/UCC13 3. If not, check if it is GIAI
14	1. Check if it is GTIN 2. If not, check if it is GIAI
16	1. Check if is GLN 2. If not, check if it is GIAI
18	1. Check if it is SSCC 2. If not, check if it is GRAI 3. If not, check if it is GIAI

The Bar Code Decoder module fields are shown in Table 3-3.

Table 3-3 Bar Code Decoder

Field Name	Value/Type	Description
Module Name	String	Unique user-defined identifier for this module.
Module Type	Bar Code Decoder	Type of module. Values are chosen from a list of module types.
Module Outputs		
Decoded ID or Pattern	URI String	Specifies the destination for the decoded bar code. For example, you might send the bar code to the Write input of a Write Tag module: tag-commissioning-write:Write For bar codes that already contain a serial number reference (SSCC) the EPC id is generated. Otherwise, the EPC id pattern is generated.
Error	Error	Specify the error message destination. For example, my-error-sender: Input. This port will send an output if an error occurs in the module processing. This port may be used to indicate any kind of error in an edgeflow.
Module Inputs		
Bar Code	Bar Code String/TagList	String type (complying with barcode EPC tag data) or tag list representing barcode data.

Counter

This module is used to display a count, or similar information on a numeric display device.

For example, you can use the output of a Tag Accumulator module with a Counter module to get a count of EPCs seen.

The Counter module fields are shown in Table 3-4.

Table 3-4 Counter Module

Field Name	Value/Type	Description
Module Name	String	Unique user-defined identifier for this module.
Module Type	Counter	Type of module. Values are chosen from a list of module types.
Always Counting	True/False	Specifies if the module is counting on active startup.
Reset on Start	True/False	Specifies if the module resets the count to initial input message on every startup.
Initial Count	Number	Specifies the initial value for the module. This value is used when the module starts and when it is reset.
Count Duration	Milliseconds	Specifies the timeout period for sending output. Must be greater than, or equal to, zero. Zero means infinite time, which means that final count output is sent only on finish message.
Module Outputs		
Final Count	Number	On a finish request or send timeout, specifies where to send the final count. For example, smart-shelf-final-count-sender:Input

Table 3-4 Counter Module

Field Name	Value/Type	Description
Running Count	Number	Specifies where to send the current count. For example, smart-shelf-count-sender:Input This count is sent for every add, subtract, increment, or decrement input.
Module State	Working or Idle	Specifies where to send the module state: • Working - sent when the module starts processing in response to Start input. • Idle - sent when the module stops processing in response to finish input.
Error	Error	Specify the error message destination. For example, my-error-sender: Input. This port will send an output if an error occurs in the module processing. This port may be used to indicate any kind of error in an edgeflow.
Module Inputs		
Add	Number	This port accepts a count (integer) and adds it to the current count.
Subtract	Number	This port accepts a count (integer) and decrements it from the current count.
Increment	Any Message Type	This port takes in any message type and increments the count by one on every message.

Table 3-4 Counter Module

Field Name	Value/Type	Description
Decrement	Any Message Type	This port accepts any message type and decrements the count by one on every message.
Start	Any Message Type	This message starts the count. This port accepts messages of any type.
Finish	Any Message Type	This message triggers the module to send the final count. The count value will be reset to the configured initial count value. If it is configured to be "always count," it will be in count mode. If not, it will wait for a start message to start counting again.

Directional Filter

You can use this module to determine the direction in which an EPC is moving. Specifically, the module is used to determine the direction between two points.

This module is similar to the Directional Filter (Auto Configuration) module. However, in this module the user is responsible for defining the ECSpec and its subscriptions.

The Directional Filter module fields are shown in Table 3-5.

Table 3-5 Directional Filter Module

Name	Type/Value	Description
Module Name	String	Unique user-defined identifier for this module.
Module Type	Directional Filter	Type of module. Values are chosen from a list of module types.
Initial State	Started/Stopped	Specify if the module is started or stopped at startup. The default is stopped.

Table 3-5 Directional Filter Module

Name	Type/Value	Description
Stale Tag Timeout	Time in milliseconds. Must be greater than zero.	Required. Determines how long a tag is kept active after it is not detected at the second read point. Defaults to 180,000.
A-side Addition Report Names	ECReport name	ECReport name that states "I see (add) the tag at position A."
A-side Deletion Report Names	ECReport name	ECReport name that states "I no longer see (delete) the tag at position A."
B-side Addition Report Names	ECReport name	ECReport name that states "I see (add) the tag at position B."
B-side Deletion Report Names	ECReport name	ECReport name that states "I no longer see (delete) the tag at position B."
Module Outputs		
A to B Tag List		Specifies where to send the list of tags that passed from side A to B. For example, interior-door-observeA2B:Tag List
B to A Tag List		Specifies where to send the list of tags that passed from side B to A. For example, interior-door-observeA2B:Tag List.

Table 3-5 Directional Filter Module

Name	Type/Value	Description
Module State	Working Idle	Specifies where to send the module status: • Working - sent when the module starts processing messages in response to Start input. • Idle - sent when the module stops processing messages in response to Stop input.
Error	Error	Specify the error message destination. For example, my-error-sender: Input. This port will send an output if an error occurs in the module processing. This port may be used to indicate any kind of error in an edgeflow.
Module Inputs		
ECReports	ECReports	Receives all add/delete reports indicating tags were added/deleted at point A or B. Typically, either this input or the specific addition/deletion inputs are used, but not both.
A-side Additions	Tag List	To notify module that tags were added at point A
A-side Deletions	Tag List	To notify module that tags were deleted at point A
B-side Additions	Tag List	To notify module that tags were added at point B
B-side Deletions	Tag List	To notify module that tags were deleted at point A

Table 3-5 Directional Filter Module

Name	Type/Value	Description
Start	Any Message Type	Start processing input messages to determine direction.
Stop	Any Message Type	Stop processing input messages to determine direction.

Directional Filter (Auto Configuration)

Use this module to determine the direction in which an EPC is moving. The module can be used to determine the direction between two points.

The module creates an ECSpec for the read points and subscribes itself to receive notifications of EPC sightings. Logical reader names are required to be configured for each side.

The Directional Filter module fields are shown in Table 3-6.

Table 3-6 Directional Filter Module (Auto Configuration)

Name	Type/Value	Description
Module Name	String	Unique user-defined identifier for this module.
Module Type	Directional Filter (Auto Configuration)	Type of module. Values are chosen from a list of module types.
Initial State	Started/Stopped	Specify if the module is started or stopped at startup. The default is stopped.
Stale Tag Timeout	Time in milliseconds. Must be greater than zero.	Required. Determines how long a tag is kept active after it is not detected at the second read point. Defaults to 180,000.
A-side Logical Reader Names	Logical Reader Name	Required. Logical reader names for side A.

Table 3-6 Directional Filter Module (Auto Configuration)

Name	Type/Value	Description
B-side Logical Reader Names	Logical Reader Name	Required. Logical reader names for side B.
Include Filters	EPC patterns EPC ID	EPC patterns. Or, if specified, direction will be calculated only for EPCs that match EPC ID patterns.
Exclude Filters	EPC patterns EPC ID	EPC patterns. Or, if specified, direction will be excluded only for EPCs that match EPC ID patterns.
Module Outputs		
A to B Tag List	Tag List	Specifies where to send the tag list that passed from point A to point B during the current event cycle. For example, interior-door-observeA2B:Tag List
B to A Tag List	Tag List	Specifies where to send the tag list that passed from point B to point A during the current event cycle. For example, interior-door-observeB2A:Tag List
Module State	Working Idle	Specifies where to send the following states: • Working - sent when the module starts processing messages as a result of Start input. • Idle - sent when the module stops processing messages as a result of Stop input.

Table 3-6 Directional Filter Module (Auto Configuration)

Name	Type/Value	Description
Error	Error	Specify the error message destination. For example, my-error-sender: Input. This port will send an output if an error occurs in the module processing. This port may be used to indicate any kind of error in an edgeflow.
Module Inputs		
Start	Any Message Type	Start looking for EPCs to determine direction.
Stop	Any Message Type	Stop looking for EPCs to determine direction.

ECSpec Subscriber

Use this module to subscribe to ECSpecs only when needed: ECSpecs do not stay active.

The ECSpec names parameter allows you to specify from which ECSpecs this module should get ECReports.

A subscribe request adds a subscription for itself for an ECSpec; an unsubscribe request removes the subscription from the ECSpec.

The ECSpec Subscriber module fields are shown in Table 3-7.

Table 3-7 ECSpec Subscriber Module

Name	Type/Value	Description
Module Name	String	Unique user-defined identifier for this module.
Module Type	ECSpec Subscriber	Type of module. Values are chosen from a list of module types.

Table 3-7 ECSpec Subscriber Module

Name	Type/Value	Description
Initial State	Subscribed/Unsubscribed	Specify if the module is subscribed or unsubscribed at startup. The default value is unsubscribed.
ECSpec Names	EPSpec Name	Required. Specify the name of the ECSpec to which to subscribe.
Module Outputs		
ECReports	ECReports	ECReports derived from the subscribed ECSpec
Error	Error	Specify the error message destination. For example, my-error-sender: Input. This port will send an output if an error occurs in the module processing. This port may be used to indicate any kind of error in an edgeflow.
Module Inputs		
Subscribe	Message of any type	Subscribe to ECSpec.
Unsubscribe	Message of any type	Unsubscribe from ECSpec.

EPC Cache Manager

An EPC cache is an ordered list of EPC values used in sequence to program tags. An EPC cache allows a series of tags to be programmed without application intervention or extra tag programming to supply new EPC codes.

You can use this module to create and manage a cache. The configuration values are specific to the caches managed by this module. This module will:

- Create a named cache for a given item.

- Configure the cache by contacting the configured Serial Number Replenisher (BEA Service) or Serial Number Replenisher (Static) module and receiving available EPCs from it.
- Replenish the cache when necessary. The module can receive cache low notification and contact Serial Number Replenisher (BEA Service) or Serial Number Replenisher (Static) modules for replenishment.
- Creates a PCSpec for the cache that it created, and sends it to the configured module.

For example, you might use the EPC Cache Manager together with the Serial Number Replenisher (Static) module (or a Serial Number Replenisher (BEA Service) module) as follows:

1. Send the Replenishment Request output of the EPC Cache Manager module to the Replenishment Request input of a Serial Number Replenisher (Static) module.
2. Send the resulting Replenishment output from a Serial Number Replenisher (Static) module to the Replenishment input of the EPC Cache Manager module.

The EPC Cache Manager module fields are shown in Table 3-8.

Table 3-8 EPC Cache Manager Module

Name	Type/Value	Description
Module Name	String	Unique user-defined identifier for this module.
Module Type	EPC Cache Manager	Type of module. Values are chosen from a list of module types.
Cache Name Prefix	String	Specify a unique, identifying prefix for the cache name. For example, tag-commissioning-sample- Either the Cache Name Prefix or Cache Name Suffix is required.
Cache Name Suffix	String	Specify a unique, identifying suffix for the cache name. Either the Cache Name Prefix or Cache Name Suffix is required.
EPC Length	64-bits/96-bits	Select 64 bit or 96 bit EPCs.

Table 3-8 EPC Cache Manager Module

Name	Type/Value	Description
EPC Filter Value		Required. Specify the EPC filter value between 0 and 7.
PCSpec Request Timeout	Milliseconds	Required. Time to wait for PCSpec. Must be greater than or equal to zero. This is the amount of time it will wait to receive EPCs for the cache it creates to define a PCSpec with the cache.
Auto Replenish	True/False	Replenish the cache when necessary.
Cache Low Threshold	Integer.	When this number of EPC's remain in the cache, replenish it. You set the size of the cache in the Replenishment Size field of the Serial Number Replenisher (BEA Service) and Serial Number Replenisher (Static) modules.
Module Outputs		
Replenishment Request	URI String	Specifies where to send requests to fill the cache. For example, tag-commissioning-static-replenisher :Replenishment Request
PCSpec	PCSpec	Specifies where to send the PCSpec with the created cache.
Error	Error	Specify the error message destination. For example, my-error-sender: Input. This port will send an output if an error occurs in the module processing. This port may be used to indicate any kind of error in an edgeflow.
Module Inputs		

Table 3-8 EPC Cache Manager Module

Name	Type/Value	Description
PCSpec Request	URI String	Request for the PCSpec with the appropriate cache name in it.
Replenishment	URI String List	This port receives EPCs. When it receives EPCs, it checks to see if it is waiting for this response (initiated because of cache low notification or because of cache setup request). If it is waiting for the response, it creates or replenishes the cache.

EPCIS Aggregation Event Generator

You can use this module to generate an EPCIS aggregation event for a given set of EPCs.

An EPCIS event is an observation of EPC-related activity, expressed at the business level. EPCIS events typically have four informational aspects: what (which EPCs or other entities participated in the event), when (the date and time of the event), where (the location where the event took place and where the entities are expected to be afterward), and why (what business transaction was taking place and what is the business condition of the entities afterward).

Consider the following usage:

1. A Tag List Filter module takes the ECReports produced by the ECSpec, extracts the tag information and sends it to the TagList port of an EPCIS Aggregation Event Generator module.
2. The EPCIS Aggregation Event Generator creates an aggregation for the tags it receives from the Tag List Filter. The aggregation event may be sent to an EPCIS system.

Note: As described in the EPCglobal *EPCIS (Electronic Product Code Information Service) Frequently Asked Questions* document, the EPCIS Data Model specifies a standard way to represent visibility information about physical objects, including descriptions of product movements in a supply chain. See that document for information about the Business Step, Disposition, Read Point, Business Location, and Business Transaction fields.

The EPCIS Aggregation Event Generator module fields are shown in Table 3-9.

Table 3-9 EPCIS Aggregation Event Generator Module

Name	Type/Value	Description
Module Name	String	Unique user-defined identifier for this module.
Module Type	EPCIS Aggregation Event Generator	Type of module. Values are chosen from a list of module types.
Initial State	Started/Stopped	Specify the state of the module at startup. The default is Started.
Parent Include Patterns	EPC EPC Pattern Any URI	Required for add or delete action on the aggregation event. The module looks for EPCs matching these patterns in the input it receives and use the EPC that matches as the parent id. EPC patterns, EPC ID patterns, EPC, or any URI may be specified. For example, urn:epc:pat:gid-64-i:*.*.1 If a non-EPC URI is specified, it should be the only value specified and it is used as the parent id field in the generated aggregation event.
Parent Exclude Patterns	EPC Patterns	Valid EPC patterns or EPC identity patterns must be specified. EPCs matching these patterns will be excluded from being the parent ID.
Children Include Patterns	EPC patterns EPC ID Patterns EPC	If specified, only EPCs matching these patterns will be used as child EPCs in the aggregation event.
Children Exclude Patterns	EPC patterns EPC ID Patterns EPC	EPC's that match the patterns will be excluded from the child EPCs list.
EPCIS Event Fields		

Table 3-9 EPCIS Aggregation Event Generator Module

Name	Type/Value	Description
Action	ADD DELETE OBSERVE	Required. Specify ADD, DELETE, or OBSERVE for an event action.
Read Point	URI	Read point URI that is used in the generated Aggregation Event.
Business Location	URI	Business location URI that is used in the generated Aggregation Event. For example, urn:acmecorp:bizLocation:store1.sal esfloor
Business Step	URI	The business process step that was occurring when the event was captured.
Disposition	URI	Disposition URI that is used in the generated Object Event.
Business Transactions	Transaction Type/Transaction ID	Business transaction type and ID.
Event Extensions	Namespace/Name/Value	List of namespace-name-value tuples included as extensions in the object event.
Module Outputs		
EPCIS Event	EPCIS Document	Specifies where to send the EPCIS aggregation event. For example, it might send the aggregation to an Aggregation Reconciler module Aggregation input.

Table 3-9 EPCIS Aggregation Event Generator Module

Name	Type/Value	Description
Template Event	EPCIS Document	The template event received on the Template Event input (and then applied) is sent out from this port. The template event is sent as an acknowledgement to a template event message. This may be used by other modules or external applications to make sure that the template is applied before starting to send tags to the module.
Module State	Working Idle	Specify where to send an output string representing status of the module. There are two possible states: • Working: Sent every time the module moves into the state where it is processing messages. • Idle: Send every time the modules moves into the state where it is not processing messages.
Error	Error	Specify the error message destination. For example, my-error-sender: Input. This port will send an output if an error occurs in the module processing. This port may be used to indicate any kind of error in an edgeflow.
Module Inputs		

Table 3-9 EPCIS Aggregation Event Generator Module

Name	Type/Value	Description
Tag List	Tag List	List of tags used to generate aggregation events. For example, the Tag List might be sent by a Tag List Filter module.
Template Event	EPCIS Document	This port accepts an EPCIS Document with an aggregation event. This port may be used to dynamically change the parameters for generating an aggregation event. The values specified through the template override those specified as configuration on the module. EPC values and action on the template event is ignored. If values are null on the template event, the values configured on the module are used. If the fields Read Point, Business Location, Business Step, and Disposition are non-null on the template event, they override the values specified at module configuration. Business Transactions and Event Extensions specified on the template event are added to those specified as part of the module configuration. The Template Event is typically generated by an external source.

Table 3-9 EPCIS Aggregation Event Generator Module

Name	Type/Value	Description
Start	Any Message Type	Indicates that the module should start processing messages it receives at the Tag List input port. If the module did not come up in started mode (initial state), the module will ignore messages on the Tag List port until it gets start message. This port accepts any message type. A message on this port changes the module state from Idle to Working.
Stop	Any Message Type	Indicates that the module should start processing messages it receives at the Tag List input port. After it receives a stop message, the module ignores messages on the Tag List port until it receives a start message again. A message on this port changes the module state from Working to Idle.

EPCIS Manifest Query

This module gets a manifest, which specifies what the content of an aggregation should be, from an EPCIS repository. It determines the manifest based on the latest ADD and DELETE aggregation event for the parent ID on the input aggregation.

When the exact IDs expected in an aggregation are known, the URI is the EPC ID. If only the item type is known, the URI is an EPC ID Pattern identifying the type. The module produces an error if the (input) aggregation does not have a parent ID.

The EPCIS Manifest Query module fields are shown in Table 3-10. The generated manifest is a map from URI to the expected-count for the URI.

Table 3-10 EPCIS Manifest Query Module

Name	Type/Value	Description
Module Name	String	Unique user-defined identifier for this module.
Module Type	EPCIS Manifest Query	Type of module. Values are chosen from a list of module types.
EPCIS WSDL URL	URL	Required. The WSDL URL to connect to the EPCIS. The default is http://localhost:7001/epcis/EPCIS?WSDL
Username		Username required to access the EPCIS database.
Password		Password required to access the EPCIS database.
Connection Timeout	Milliseconds	Required. Time, in milliseconds, to wait for response from EPCIS. Must be greater than zero.
Module Outputs		
Manifest	Manifest	The generated manifest is sent through this port.
Error	Error	Specify the error message destination. For example, my-error-sender: Input. This port will send an output if an error occurs in the module processing. This port may be used to indicate any kind of error in an edgeflow.

Table 3-10 EPCIS Manifest Query Module

Name	Type/Value	Description
Module Inputs		
Aggregation	EPCIS Document	Aggregation for which manifest is to be calculated. Possibly output from an EPCIS Aggregation Event Generator module.

EPCIS Object Event Generator

This module is used to generate an EPCIS Object Event from a given list of EPCs.

An EPCIS event is an observation of EPC-related activity, expressed at the business level. EPCIS events typically have four informational aspects: what (which EPCs or other entities participated in the event), when (the date and time of the event), where (the location where the event took place and where the entities are expected to be afterward), and why (what business transaction was taking place and what is the business condition of the entities afterward).

You might use a Directional Filter module or output from a Tag List Filter module to send the A to B Tag List to the Object Event Generator module. The generated EPCIS object event may be sent to an EPCIS system.

Note: As described in the EPCglobal *EPCIS (Electronic Product Code Information Service) Frequently Asked Questions* document, the EPCIS Data Model specifies a standard way to represent visibility information about physical objects, including descriptions of product movements in a supply chain. See that document for information about the Business Step, Disposition, Read Point, Business Location, and Business Transaction fields.

The EPCIS Object Event Generator module fields are shown in Table 3-11.

Table 3-11 EPCIS Object Event Generator Module

Name	Type/Value	Description
Module Name	String	Unique user-defined identifier for this module.
Module Type	EPCIS Object Event Generator	Type of module. Values are chosen from a list of module types.
Initial State	Started/Stopped	Specify the state of the module when started.
EPCIS Event Fields		
Action	ADD DELETE OBSERVE	Required. Specify ADD, DELETE, or OBSERVE for an action.
Read Point	URI	Read point URI that is used in the generated Aggregation Event.
Business Location	URI	Business location URI that is used in the generated Aggregation Event. For example, urn:acmecorp:bizLocation:store1.sal esfloor
Business Step	URI	The business process step that was occurring when the event was captured.
Disposition	URI	Disposition URI that is used in the generated Object Event.
Business Transactions	Name/Value	Business transaction type and ID.
Event Extensions	Namespace/Name/Value	List of namespace-name-value tuples that is included as extensions in the object event.
Module Outputs		

Table 3-11 EPCIS Object Event Generator Module

Name	Type/Value	Description
EPCIS Event	EPCIS Document	Specifies where to send the generated event. For example, you might use an External Sender module to send the event to an EPCIS system.
Template Event	EPCIS Document	The template event received on the Template Event input (and then applied) is sent out from this port. The template event is sent as an acknowledgement to a template event message. This may be used by other modules or external applications to make sure that the template is applied before starting to send tags to the module.
Module State	Working Idle	Specify where to send an output string representing status of the module. For example, my-module-accumulator: Finish. There are two possible states: • Working: Sent every time the module moves into the state where it is processing messages as a result of a Start input. • Idle: Send every time the modules moves into the state where it is not processing messages as a result of a Stop input.

Table 3-11 EPCIS Object Event Generator Module

Name	Type/Value	Description
Error	Error	Specify the error message destination. For example, my-error-sender: Input. This port will send an output if an error occurs in the module processing. This port may be used to indicate any kind of error in an edgeflow.
Module Inputs		
Tag List	Tag List	List of tags used to generate object events, such as an A to B Tag List output by the Directional Filter module.
Template Event	EPCIS Document	This port accepts an EPCIS Document with an object event. This is used to dynamically change the parameters for generating an object event. The values specified through the template override most of the ones specified as configuration on the module. EPC values and action on the template event is ignored. If values are null on the template event, the values configured on the module are used. If the fields Read Point, Business Location, Business Step, and Disposition are non-null on the template event, they override those specified in the module configuration. Business Transaction and Event Extensions specified on the template event are added to those specified in the module configuration.

Table 3-11 EPCIS Object Event Generator Module

Name	**Type/Value**	**Description**
Start	Any Message Type	A message to this port is a signal to the module to start generating object events. Depending on how the module is configured, the module will ignore messages on the Tag List port until it gets start message. This port accepts any message type. A message on this port changes the module state from Idle to Working.
Stop	Any Message Type	A message to this port signals the module to stop generating object events. The module ignores messages on the Tag List port until it receives a start message again. A message on this port changes the module state from Working to Idle.

External Sender

This module is used to send a message from edgeflow modules to external destinations.

This module has one input port, Input, which takes messages of any type and processes them as follows:

1. If it receives an XML input, it will apply the XSLT transformation and send the result to the output.
2. If it receives String input, it will send it as-is to the output.
3. If it receives other types, it will try to serialize it. If it serializes successfully, it applies any XSLT transformation and sends the XML to the output port. If an error occurs trying to serialize, error details are sent on the error port.

The External Sender module fields are shown in Table 3-12.

Table 3-12 External Sender Module

Name	Type/Value	Description
Module Name	String	Unique user-defined identifier for this module.
Module Type	External Sender	Type of module. Values are chosen from a list of module types.
XSLT File	Path of File Containing Stylesheet	Stylesheet to transform message. The file that contains the stylesheet must be on the Edge Server system, and the path must be absolute path.
Destination URI's	URI	Required. The external destination to which to send the message. For example, in the case of an error message: console:PALLET-STATION-ERRO R You could instead configure this output to use HTTP or JMS to send the events, using the appropriate format.
Module Outputs		
Error	Error	Specify the error message destination. For example, my-error-sender: Input. This port will send an output if an error occurs in the module processing. This port may be used to indicate any kind of error in an edgeflow.
Module Inputs		
Input	Any message type	The message to send to the external destination.

HTTP Receiver (One Way)

You use the HTTP Receiver (One Way) module to send messages from external systems to edgeflow modules. The full URL specifies the target module and input port. The HTTP client will receive a response whether the message was delivered to the target module or not.

The HTTP Receiver module fields are shown in Table 3-13.

Table 3-13 HTTP Receiver (One Way) Module

Name	Type/Value	Description
Module Name	String	Unique user-defined identifier for this module.
Module Type	HTTP Receiver (One Way)	Type of module. Values are chosen from a list of module types.
TCP Port	Port number between 1 and 65535, inclusive.	Required. The TCP port the module listens on.
URL Suffix	Unique identifier for this HTTP Receiver	Required. The unique URL suffix to be used with this HTTP Receiver. For example, if the URL suffix is specified as tag-commissioning, HTTP messages to URL `http://<link>:<port>/tag-commissioning` will be sent.
Parse Request XML	True False	This parameter indicates whether input XML should be parsed or not. When True, output is an object. When false, output is XML.
Request XSLT File	XSLT	Style sheet to apply to XML. Specify the full path to the file on the Edge Server with the XSLT.
Module Outputs		

Table 3-13 HTTP Receiver (One Way) Module

Name	Type/Value	Description
Output		Specifies where to send the message. For example, if the message is a bar code, you might want to send it to the Bar Code input of a Bar Code Decoder module, such as tag-commissioning-barcode:Bar Code
Error		Specify the error message destination. For example, my-error-sender: Input. This port will send an output if an error occurs in the module processing. This port may be used to indicate any kind of error in an edgeflow.

HTTP Receiver (Request Response)

You can use the HTTP Receiver (Request Response) module to send a message to any module port and return the output from that module (or another module) in the response. The HTTP client will receive a response whether the message was delivered to the target module or not.

The HTTP Receiver (Request Response) module fields are shown in Table 3-14.

Table 3-14 HTTP Receiver (Request Response) Module

Name	Type/Value	Description
Module Name	String	Unique user-defined identifier for this module.
Module Type	HTTP Receiver (Request Response)	Type of module. Values are chosen from a list of module types.
TCP Port	Port number between 1 and 65535, inclusive.	Required. The TCP port the module listens on.

Table 3-14 HTTP Receiver (Request Response) Module

Name	Type/Value	Description
URL Suffix	Unique identifier for this HTTP Receiver	Required. The unique URL suffix to be used with this HTTP Receiver. For example, tag-commissioning.
Parse Request XML	True False	When True, output is an object. When false, output is XML.
Request XSLT File	XSLT	Style sheet to apply to XML.
Response Timeout	Milliseconds	Required. The amount of time to wait to receive the response message that is to be sent back to the HTTP client. The default is 500. Must be greater than zero.
Response XSLT File	XSLT	XSLT file to apply to response XML.
Module Outputs		
Output		Specifies where to send the message. For example, if the message is a bar code, you might want to send it to the Bar Code input of a Bar Code Decoder module, such as tag-commissioning-barcode:Bar Code.
Error		Specify the error message destination. For example, my-error-sender: Input. This port will send an output if an error occurs in the module processing. This port may be used to indicate any kind of error in an edgeflow. .

Table 3-14 HTTP Receiver (Request Response) Module

Name	Type/Value	Description
Module Inputs		
Response	Any message type	The input port can receive any type of message to send as the response to the HTTP client, and it uses the type to determine what to send: • If it receives an object of type `java.lang.String`, it sends the string as the response. • If it receives a message of type `XMLDocument`, it sends the XML as the output. • If it receives any other type, it will make a best effort to serialize, and sends the result of the serialization as the result.

Numeric Display (EDI111 LED)

You can use the Numeric Display module to display a count or similar information on a device.

The Numeric Display module fields are shown in Table 3-15.

Table 3-15 Numeric Display (EDI111 LED) Module

Name	Type/Value	Description
Module Name	String	Unique user-defined identifier for this module.
Module Type	Numeric Display EDI111 LED	Type of module. Values are chosen from a list of module types.
Device Host		Required. The host name for the device.

Table 3-15 Numeric Display (EDI111 LED) Module

Name	Type/Value	Description
Device Port	Port number between 0 and 65535, inclusive.	Required. The TCP port used by the device.
Initial Message	Characters	Character value displayed on the LED when the module is initially started. Can be empty.
Display Duration	Milliseconds. Greater than or equal to zero.	Time that the message will be displayed on the LED. The default is 0, which indicates infinite time.
Flash Period	Milliseconds. Greater than or equal to zero.	Time period for which messages will be on or off in flash mode. The message will flash for “display duration” time with an on and off period defined by “flash period.” The default is 500.
Module Outputs		
Error	Error	Specify the error message destination. For example, my-error-sender: Input. This port will send an output if an error occurs in the module processing. This port may be used to indicate any kind of error in an edgeflow.
Module Inputs		
Steady Message	Number	Display the input message in steady mode for time defined by display duration.
Flash Message	Number	Display the input message in flash mode for time defined by display duration.
Reset	Any message type	Clear the display.

Preset Message (Dynamic)

You can use the Preset Message module to to send a preset message to its destination. The message sending activity may be because of a trigger activation or other event. The message to be sent is preset through the message content input port. Note the following:

- If the send message is received but there is no preset message, the module generates an error message to declare that the message was not set.
- If a clear message is received, the existing preset message is cleared.

The Preset Message (Dynamic) module fields are shown in Table 3-16.

Table 3-16 Preset Message (Dynamic) Module

Name	Type/Value	Description
Module Name	String	Unique user-defined identifier for this module.
Module Type	Preset Message (Dynamic)	Type of module. Values are chosen from a list of module types.
Module Outputs		
Message	Any message type	Specify the destination module and port.
Error	Error	Specify the error message destination. For example, my-error-sender: Input. This port will send an output if an error occurs in the module processing. This port may be used to indicate any kind of error in an edgeflow.
Module Inputs		
Message Content	Any message type.	Sets the content of the message to be sent on a “send” input.

Table 3-16 Preset Message (Dynamic) Module

Name	Type/Value	Description
Clear	Any message type.	Clear signal clears the message content.
Send	Any message type.	Send signal sends the message content.

Preset Message (Static)

You can use the Preset Message module to to send a preset, static message to its destination. The message sending activity may be because of a trigger activation or other event.

If the send message is received but there is no static message, an empty string is sent.

The Preset Message (Static) module fields are shown in Table 3-17.

Table 3-17 Preset Message (Static) Module

Name	Type/Value	Description
Module Name	String	Unique user-defined identifier for this module.
Module Type	Preset Message (Static)	Type of module. Values are chosen from a list of module types.
Message Content	String	Required. User-defined message content.
Module Outputs		
Message	String	Specify the message destination.
Module Inputs		
Send	Any message type	A message on this port sends the configured message.

Serial Number Replenisher (BEA Service)

You can use the Serial Number Replenisher (BEA Service) module to contact the RFID Enterprise Server Serial Number Assignment Service to get unique EPC's.

As a possible use, you might use the EPC Cache Manager together with the Serial Number Replenisher (Static) module or a Serial Number Replenisher (BEA Service) module as follows:

1. Send the Replenishment Request output of the EPC Cache Manager module to the Replenishment Request input of a Serial Number Replenisher (BEA Service) module.
2. Send the resulting Replenishment output from a Serial Number Replenisher module to the Replenishment input of the EPC Cache Manager module.

The Serial Number Replenisher module fields are shown in Table 3-18.

Table 3-18 Serial Number Replenisher (BEA Service) Module

Name	Type/Value	Description
Module Name	String	Unique user-defined identifier for this module.
Module Type	Serial Number Replenisher (BEA Service)	Type of module. Values are chosen from a list of module types.
BEA Service URL	URL	Required. URL of the Serial Number Assignment Service
Username		User name for the Serial Number Assignment Service.
Password		Password for the Serial Number Assignment Service
Replenishment Size	Number. Must be greater than zero.	Required. The number of EPCs to check out of the Serial Number Assignment Service. You use this together with the Cache Low Threshold field of the EPC Cache Manager module to manage the cache.

Table 3-18 Serial Number Replenisher (BEA Service) Module

Name	Type/Value	Description
Range Selection Criteria	Name/Value	Additional range criteria can be specified here.
Module Outputs		
Replenishment	URI String	Specifies where to send the serial numbers to satisfy the replenishment request. For example. tag-commissioning-cachemgr:Reple nishment The EPCs checked out from the serial number assignment service will be sent out on this port. This is a list of Strings representing the EPC range.
Error	Error	Specify the error message destination. For example, my-error-sender: Input. This port will send an output if an error occurs in the module processing. This port may be used to indicate any kind of error in an edgeflow.
Module Inputs		
Replenishment Request	URI String	This port accepts EPC pattern as input. The EPC pattern must have the fields to identify the EPC class. Since this module uses the serial number assignment service, it handles only types supported by that service (SSCC and GTIN).

Serial Number Replenisher (Static)

You can use the Serial Number Replenisher (Static) module to set a fixed range of serial numbers of any EPC type.

As a possible use, you might use the EPC Cache Manager together with the Serial Number Replenisher (Static) module or a Serial Number Replenisher (BEA Service) module as follows:

1. Send the Replenishment Request output of the EPC Cache Manager module to the Replenishment Request input of a Serial Number Replenisher (Static) module.
2. Send the resulting Replenishment output from a Serial Number Replenisher module to the Replenishment input of the EPC Cache Manager module.

The Serial Number Replenisher (Static) module fields are shown in Table 3-19.

Table 3-19 Serial Number Replenisher (Static) Module

Name	Type/Value	Description
Module Name	String	Unique user-defined identifier for this module.
Module Type	Serial Number Replenisher (Static)	Type of module. Values are chosen from a list of module types.
Serial Range Start	Number. Must be greater than or equal to zero.	Required. EPC start range of serial numbers. You use this together with the Cache Low Threshold field of the EPC Cache Manager module to manage the cache.
Serial Range End	Number. Must be greater than or equal to zero.	Required. EPC end range of serial numbers. You use this together with the Cache Low Threshold field of the EPC Cache Manager module to manage the cache.
Module Outputs		
Replenishment	URI String	Specifies where to send the serial numbers to satisfy the replenishment request. For example, tag-commissioning-cachemgr:Replenishment

Table 3-19 Serial Number Replenisher (Static) Module

Name	Type/Value	Description
Error	Error	Specify the error message destination. For example, my-error-sender: Input. This port will send an output if an error occurs in the module processing. This port may be used to indicate any kind of error in an edgeflow.
Module Inputs		
Replenishment Request	URI String	Message requesting cache replenishment. This might typically come from the Replenishment Request output of the EPC Cache Manager module

Stack Light

You can use the Stack Light module to turn the lights on a stack light on or off.

Note: The implementation of the Stack Light module in this release is significantly enhanced from the Stack Light Module of previous versions.

The Stack Light module fields are shown in Table 3-20.

Table 3-20 Stack Light Module

Name	Type/Value	Description
Module Name	String	Unique user-defined identifier for this module.
Module Type	Stack Light	Type of module. Values are chosen from a list of module types.
Logical Reader Name	Reader Name	Required. The logical name for the stack light device.

Table 3-20 Stack Light Module

Name	Type/Value	Description
Turn Other Colors Off	True/False	Required. Specifies whether other colors should be turned off on an input message to turn on one color.
Red Duration	Milliseconds	The time in milliseconds for which the light should be lit. The default is zero, which turns the light on indefinitely. Available values increment by 5000 milliseconds, from zero to 40,000.
Amber Duration	Milliseconds	The time in milliseconds for which the light should be lit. The default is zero, which turns the light on indefinitely. Available values increment by 5000 milliseconds, from zero to 40,000.
Green Duration	Milliseconds	The time in milliseconds for which the light should be lit. The default is zero, which turns the light on indefinitely. Available values increment by 5000 milliseconds, from zero to 40,000.
Blue Duration	Milliseconds	The time in milliseconds for which the light should be lit. The default is zero, which turns the light on indefinitely. Available values increment by 5000 milliseconds, from zero to 40,000.
White Duration	Milliseconds	The time in milliseconds for which the light should be lit. The default is zero, which turns the light on indefinitely. Available values increment by 5000 milliseconds, from zero to 40,000.
Module Outputs		

Table 3-20 Stack Light Module

Name	Type/Value	Description
Error	Error	Specify the error message destination. For example, my-error-sender: Input. This port will send an output if an error occurs in the module processing. This port may be used to indicate any kind of error in an edgeflow.
Module Inputs		
Red On	Any message type	Turn red on.
Red Off	Any message type	Turn red off.
Amber On	Any message type	Turn amber on.
Amber Off	Any message type	Turn amber off.
Green On	Any message type	Turn green on.
Green Off	Any message type	Turn green off.
Blue On	Any message type	Turn blue on.
Blue Off	Any message type	Turn blue off.
White On	Any message type	Turn white on.

Table 3-20 Stack Light Module

Name	Type/Value	Description
White Off	Any message type	Turn white off.
Multi-light Control	URI String	This port is used to specify a complex pattern for turning on multiple color lights. This will take in URI string of the same format as expected by ALEPC API: urn:connecterra:stacklight:update=< update value>. <update value> is a string consisting of 5 digits [0 - 9]. Each digit corresponds to one color. The order of colors in the string is as following: WHITE-BLUE-GREEN-AMBER-RED. The values are: • 0 - turn off the light • 9 - turn on indefinitely, until a different value is written. • X in [1 - 8] - turn on the light for 5 seconds times X. For example, 01119: • Turns white off. • Turns blue, green, and amber on for 5 seconds each. • Turns red on indefinitely.

State Filter

You can use the State Filter module split the Working State output of other modules into distinct Working and Idle outputs.

For example, consider the Module State output of the EPCIS Object Event Generator module. The state for this module can be either Working or Idle, and it is up to the destination to parse the string to determine what the actual current state is.

The State Filter module provides this service for you: it receives the generic Module State as input, parses it, and splits in into distinct Working and Idle outputs.

The State Filter module fields are shown in Table 3-21.

Table 3-21 State Filter Module

Name	Type/Value	Description
Module Name	String	Unique user-defined identifier for this module.
Module Type	State Filter	Type of module. Values are chosen from a list of module types.
Module Outputs		
Working State	Working/Idle	Specifies the destination of the working message.
Idle State	Working/Idle	Specifies the destination of the Idle message.
Error		Specify the error message destination. For example, my-error-sender: Input. This port will send an output if an error occurs in the module processing. This port may be used to indicate any kind of error in an edgeflow.
Module Inputs		
State	Working/Idle	Receives a Working or Idle message from another module.

Tag Accumulator

You can use the Tag Accumulator module to accumulate tags for a period of time, or for a period of time delineated by start and end messages. The module starts collecting and keeping track of tags seen on a start message or at start-up time. The module sends the collected tags on a message to the send port, or after a configured amount of time.

The output of this module may typically be used by modules like EPCIS Object Event Generator or EPCIS Aggregation Event Generator. For example, use the Final Tag List output from the Tag Accumulator as the Tag List input to a EPCIS Object Event Generator to generate an object event for all accumulated tags.

The Tag Accumulator module fields are shown in Table 3-22.

Table 3-22 Tag Accumulator Module

Name	Type/Value	Description
Module Name	String	Unique user-defined identifier for this module.
Module Type	Stack Light	Type of module. Values are chosen from a list of module types.
Always Accumulating	True/False	Specify if accumulation starts when the module is started. The default is false.
Reset on Start	True/False	Specifies if the accumulated tags should be reset to empty on every start input message. The default is true.
Suppress Empty Result	True/False	Specify whether empty results are reported; that is, sent to the Final Tag List output destination. The default is true.
Finish When		
Duration Elapses	Milliseconds	Sends out the final tag list when the specified duration elapses. Must be greater than or equal to zero. The default is zero, which is infinite time.
Stable Set	Milliseconds	If the accumulated list does not change in the time specified, the list is considered final and sent out. Must be greater than or equal to zero. The default is zero, which is infinite time.

Table 3-22 Tag Accumulator Module

Name	Type/Value	Description
Stable Set Count	Milliseconds	If the current count did not change for the Stable Set duration, and the current accumulation count is greater than or equal to this count, then final tag list is sent. A zero value (the default) means disregard the count value and apply the stable set timeout for any count. Must be greater than or equal to zero.
Module Outputs		
Final Tag List	Tag List	The final Tag List is sent when the module receives a send request or when the configured time has expired.
Running Tag List	Tag List	This outputs the Tag List accumulated to this point. Input on the add or delete port will cause this output to be sent out.
Module State	Working/Idle	Specify where to send an output string representing status of the module. For example, my-module-accumulator: Finish. The status may be: • Working: This is sent every time the module moves into accumulating state. • Idle: This is sent every time the module moves into non-accumulating state.

Table 3-22 Tag Accumulator Module

Name	Type/Value	Description
Error	Error	Specify the error message destination. For example, my-error-sender: Input. This port will send an output if an error occurs in the module processing. This port may be used to indicate any kind of error in an edgeflow.
Module Inputs		
Add	Tag List	This port accepts a Tag List and adds them to the list of collected tags. It will ignore messages that are not of this type.
Delete	Tag List	This port accepts a Tag List and deletes those from the list of collected tags. It will ignore messages that are not of this type
Start	Any message type	A message on this port enables the module to accumulate tags. Any message type is accepted
Finish	Any message type	A message on this port triggers the module to output final tags and reset the list. If the module is configured with “Always Accumulating” set to false, the send message will also put the module back to the state where it will not be accumulating Tags. This port accepts messages of any type.

Tag Counter

The Tag Counter module accepts an ECReport, a PCWriteReport, or tag list and outputs a count of the tags in the input.

For example, a Tag Counter module can take the ECReports produced by an ECSpec and output the number of tags in the input.

The Tag Counter module fields are shown in Table 3-23.

Table 3-23 Tag Counter Module

Name	Type/Value	Description
Module Name	String	Unique user-defined identifier for this module.
Module Type	Tag Counter	Type of module. Values are chosen from a list of module types.
ECSpec Report Names	ECSpec Report Names	If specified, tags are counted only from ECReports with the specified name. If nothing is specified here, tags from all reports will be counted.
Module Outputs		
Count	Number	Specify the location to send the tag count. For example, smart-shelf-counter:Add
Error	Error	Specify the error message destination. For example, my-error-sender: Input. This port will send an output if an error occurs in the module processing. This port may be used to indicate any kind of error in an edgeflow.

Table 3-23 Tag Counter Module

Name	Type/Value	Description
Module Inputs		
Input		The module accepts ECReports, PCWriteReport, and tag lists. If the ECSpec Report Names configuration field is supplied, it identifies the particular report name (from the ECReport) from which to get input.

Tag List Filter

The Tag List Filter module extracts tag information from ECReports, or PCWriteReports and creates a tag list.

For example, a Tag List Filter module can take the ECReports produced by the ECSpec, extract the tag information and send it to the Add port of a Tag Accumulator module.

The Tag List Filter module fields are shown in Table 3-24.

Table 3-24 Tag List Filter Module

Name	Type/Value	Description
Module Name	String	Unique user-defined identifier for this module.
Module Type	Tag List Filter	Type of module. Values are chosen from a list of module types.
ECSpec Report Names	ECSpec Report Name	If specified, tags are counted only from ECReports with the specified name. If nothing is specified here, tags from all reports will be counted.
Module Outputs		

Table 3-24 Tag List Filter Module

Name	Type/Value	Description
Tag List	Tag List	Specify the destination for the tag list generated by the module. For example, pallet-station-accumulator:Add
Error	Error	Specify the error message destination. For example, my-error-sender: Input. This port will send an output if an error occurs in the module processing. This port may be used to indicate any kind of error in an edgeflow.
Module Inputs		
Input	ECSpecReport or PCWriteReport	The module accepts ECReports, PCWriteReport, and tag lists. If the ECSpec Report Names configuration field is supplied, it identifies the particular report name (from the ECReport) from which to get input.

Tag Verifier

You can use the Tag Verifier module to verify that tags are readable. It takes in tags to be verified and tags read, and produces a list of verified tags, missing tags, extra tags, and a Boolean indicating whether the verification was successful.

The Tag Verifier module fields are shown in Table 3-25.

Table 3-25 Tag Verifier Module

Name	Type/Value	Description
Module Name	String	Unique user-defined identifier for this module.
Module Type	Tag Verifier	Type of module. Values are chosen from a list of module types.
Message Pattern	One Actual Per Expected/ Multiple Actual Per Expected	Specifies how to match actual tags input with expected tags input. “One” indicates that for every expected input there will be one actual input. Multiple indicates that it will look in subsequent actual messages for tags in the expected input.
Verification Timeout	Milliseconds. Greater than or equal to zero.	Time to wait for verification. If time elapses, an error message is generated.
Module Outputs		
All Expected Verified	Boolean	Boolean indicating whether all tags were verified or not.
Verified Tags	Tag List	Tag list of verified tags.
Missing Tags	Tag List	Tag list of missing tags
Extra Tags	Tag List	Tag list of extra tags.
Error	Error	Specify the error message destination. For example, my-error-sender: Input. This port will send an output if an error occurs in the module processing. This port may be used to indicate any kind of error in an edgeflow.
Module Inputs		

Table 3-25 Tag Verifier Module

Name	Type/Value	Description
Expected	Tag List	Tag list of expected tags.
Actual	Tag List	Tag list of actual tags.

Trigger Receiver

You can use Trigger Receiver modules to configure an edgeflow module to receive a trigger. Triggers are typically used by counted input ports, such as start, stop, clear, and send.

The Trigger Receiver module fields are shown in Table 3-26.

Table 3-26 Trigger Receiver Module

Name	Type/Value	Description
Module Name	String	Unique user-defined identifier for this module.
Module Type	Trigger Receiver	Type of module. Values are chosen from a list of module types.
Initial State	Started/Stopped	Specify if the module accepts triggers (started) or not. The default is started.
Trigger URI	URI	Required. URI for a trigger configured within the Edge Server. See Triggers for additional information.
Module Outputs		
Trigger	Trigger	Specifies where to send the trigger.
Module Inputs		

Table 3-26 Trigger Receiver Module

Name	Type/Value	Description
Start	Any message type.	Accepts any type message and subscribes the module to the trigger.
Stop	Any message type.	Accepts any type message and causes the module to unsubscribe from the trigger.

Wait For Time

You can use the Wait For Time module to hold (queue) input before sending it. It will discard the first message on the hold queue if a discard message is received.

The size of the queue is configurable.

The Wait For Time module fields are shown in Table 3-27.

Table 3-27 Wait For Time Module

Name	Type/Value	Description
Module Name	String	Unique user-defined identifier for this module.
Module Type	Wait For Time	Type of module. Values are chosen from a list of module types.
Wait Duration	Milliseconds	Required. The amount of time to wait before sending message. Must be greater than zero.
Queue Size	Number of messages	Required. Message queue size. Must be greater than zero.
Module Outputs		
Message		The triggered message.

Table 3-27 Wait For Time Module

Name	Type/Value	Description
Error	Error	Specify the error message destination. For example, my-error-sender: Input.
Module Inputs		
Message	Any message type.	Message to be added to the queue.
Discard	Any message type.	Message to discard first message in queue.

Wait For Trigger

You can use the Wait For Trigger module to hold (queue) input until a send or discard trigger is received. If there is no message on the queue and a send trigger is received, an error is generated.

The Wait For Trigger module fields are shown in Table 3-28.

Table 3-28 Wait For Trigger Module

Name	Type/Value	Description
Module Name	String	Unique user-defined identifier for this module.
Module Type	Wait For Trigger	Type of module. Values are chosen from a list of module types.
Queue Size	Number of messages	Required. Size of the message queue. Must be greater than zero.
Timeout	Milliseconds	Time to wait before a discard or send of the message is received. When this time elapses, message is discarded and error is sent. Must be greater than or equal to zero. Default is zero, which means infinite time.
Module Outputs		

Table 3-28 Wait For Trigger Module

Name	Type/Value	Description
Message	Any message type.	The generated message.
Error	Error	Specify the error message destination. For example, my-error-sender: Input. This port will send an output if an error occurs in the module processing. This port may be used to indicate any kind of error in an edgeflow.
Module Inputs		
Message	Any message type.	The message to be sent on "send" input message.
Send	Any message type.	The command to send message. If no message is present when the send message is received, an error is generated.
Discard	Any message type.	Clears the first message in the queue.

Write Tag

This port accepts an EPC or EPC Pattern. If it receives an EPC on this port, it will program a tag with that EPC and output the PCWriteReport.

If the module receives an EPC pattern as input, it will send that to other modules (such as EPC Cache Manager) through the PCspec request output port to construct the PCSpec required to program a tag for the input item type. It will then program a tag using that PCSpec and output the PCWriteReport.

For example, in the case of EPC pattern you might use the Write Tag module together with the EPC Cache Manager module as follows:

1. The Write Tag module sends the PCSpec Request output to the EPC Cache Manager module PCSpec Request input.

2. The EPC Cache Manager module then returns the PCSpec, with the appropriate cache name specified in it, to the PCSpec input of the Write Tag module via the PCSpec output.
3. The write tag module uses the PCSpec to write the tag, and then generates the PCWriteReport.

The Write Tag module fields are shown in Table 3-29.

Table 3-29 Write Tag Module

Name	Type/Value	Description
Module Name	String	Unique user-defined identifier for this module.
Module Type	Write Tag	Type of module. Values are chosen from a list of module types.
Logical Reader Name	Reader Name	Required. Name of printer performing the write operation.
PCSpec Request Timeout	Milliseconds	Required. Time to wait for the PCSpec. Must be greater than or equal to zero. Default is zero.
Module Outputs		
PCSpec Request	URI String	Specifies where to send the PCSpec Request. If the module receives an EPC pattern as input, it will send that to other modules (such as EPC Cache Manager) through the PCspec request output port to construct the PCSpec required to program a tag for the input item type.
PCWriteReport	PCWriteReport	PCWriteReport produced by the print request.
Error	Error	Specify the error message destination. For example, my-error-sender: Input.
Module Inputs		

Table 3-29 Write Tag Module

Name	Type/Value	Description
Write	URI String	This port accepts an EPC or EPC Pattern. If it receives an EPC on this port, it will program a tag with that EPC and output the PCWriteReport. If it receives a pattern, it will send that to other modules (such as the EPC Cache Manager Module) through the PCspec request output port to construct the PCSpec required to program a tag for the input item type.
Setup	URI String	This port accepts EPCs or an EPC Pattern and creates and defines the PCSpec for the EPC Class of the input. After being set up, the module is ready for further writes and those write operations for the same EPC class will be fast. This is useful for high speed conveyor-type scenarios, where you want to set everything up before starting the operation.
PCSpec	PCSpec	This port accepts PCSpec from modules such as the EPC Cache Manager Module.

XML Generator

You can use the XML Generator module to accept any input type (except Lists) and generate XML output. The Input port takes messages of Any (excluding Lists), String List, or Tag List.

The XML Generator module fields are shown in Table 3-30.

Table 3-30 XML Generator Module

Name	Type/Value	Description
Module Name	String	Unique user-defined identifier for this module.
Module Type	XML Generator	Type of module. Values are chosen from a list of module types.
Input Type	Any (Excluding Lists) String List Tag List	Specify the input type from the drop-down list.
Module Outputs		
XML	XML	Specify the destination for the generated XML.
Error	Error	Specify the error message destination. For example, my-error-sender: Input. This port will send an output if an error occurs in the module processing. This port may be used to indicate any kind of error in an edgeflow.
Module Inputs		
Input	Any message type	The input from which to generate XML.

XML Parser

You can use the XML Parser module to convert XML to a Java object. This module supports all of the object types known to the edgeflow framework or registered with the naming service.

The XML Parser module fields are shown in Table 3-31.

Table 3-31 XML Parser Module

Name	Type/Value	Description
Module Name	String	Unique user-defined identifier for this module.
Module Type	XML Parser	Type of module. Values are chosen from a list of module types.
Module Outputs		
Output	Based on Input	Specifies where to send the output.
Error	Error	Specify the error message destination. For example, my-error-sender: Input.
Module Inputs		
Input	XML	XML input.

XML Transformer

This module takes XML as input and applies an XSLT to it to create transformed XML output.

The XML Transformer module fields are shown in Table 3-32.

Table 3-32 XML Transformer Module

Name	Type/Value	Description
Module Name	String	Unique user-defined identifier for this module.
Module Type	XML Transformer	Type of module. Values are chosen from a list of module types.

Table 3-32 XML Transformer Module

Name	Type/Value	Description
XSLT File		Required. The style sheet that is applied to the input XML file. The fully specified path to the file on the Edge Server with the XSLT must be specified here.
Module Outputs		
XML	XML	The transformed XML file.
Error	Error	Specify the error message destination. For example, my-error-sender: Input.
Module Inputs		
Input	XML	The XML file to be transformed by the style sheet.

Writing Custom Modules

This section describes how to use the edgeflow APIs to write and deploy custom edgeflow modules.

The following topics are described in this section:

Sample custom workflow modules are available in the samples directory (RFID_EDGE_HOME\samples\EdgeFlows\CustomModules) in the Edge Server installation hierarchy. In order to use the custom modules, they must be deployed to the Edge Server. See Building and Deploying the Sample.

Edgeflow Module Overview

The Edge Server installation creates a standard set of edgeflow modules, as described in Edgeflow Module Reference.

You may want to write custom modules to satisfy specific use cases not handled by the standard set of modules. Java docs for the interfaces used within the edgeflow engine are available for your use. This section, when used along with the Java docs, describes how to write and deploy custom edgeflow modules.

Different edgeflow modules carry out different tasks, which are defined by the module's implementation. At run time, instances of workflow modules may be created, configured, and wired together to carry out an overall use case. Table 4-1 shows design time vs. run time tasks:

Table 4-1 Design Time Versus Run-time Tasks

Design Time Tasks	Run-time Tasks
Choose name for module	Create an instance of a module
Define module configuration parameters	Choose name for module instance
Define module input and output ports	Set module configuration parameter values
Write module code	Wire input and output ports to other module instances
Deploy module into Edge Server	

Interaction Between Edgeflow Engine and Module When Creating a New Module

The sequence of calls that occur when the edgeflow engine receives a request to create a new edgeflow module (initiated by definePlugin on WorkflowManager by the Edge Server when you create a new instance in the console) is as follows:

1. Look up the name of the class to be instantiated from the PluginMeta instance.
2. Create the `AbstractWorkflowModulePlugin` module instance by invoking the zero-argument constructor.
3. Call the `initialize` method on the module instance.
4. Send the configuration parameters using the module’s configure method.
5. Call `startup` for any module startup functions that must happen before the module starts receiving messages.

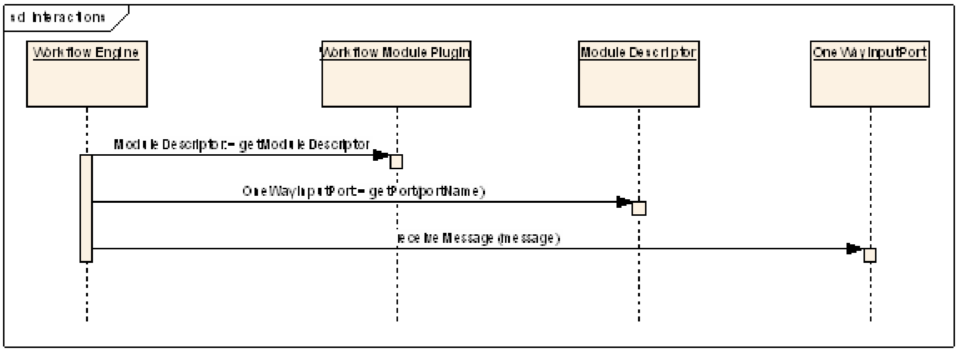
Sequence of Calls When Sending Message to Edgeflow Module Port

As described in Configuring and Using Edgeflows, an edgeflow module instance can receive data from, and send data to, other module instances. It sends data to other module instances through output ports and receives data from other module instances through input ports. A module may have multiple input ports and multiple output ports.

The sequence of calls that occur when the edgeflow engine sends a message to a module's port is shown in Figure 4-1.

As described in Extend the AbstractWorkflowModulePlugin Class, your custom module should extend the `AbstractWorkflowModulePlugin` class, including providing an implementation for the abstract method `createModuleDescriptor`. This method should return a `ModuleDescriptor` that contains `com.bea.rfid.edge.api.workflow.OneWayInputPort` for all the input ports of the module and `com.bea.rfid.edge.api.workflow.Port` for all the output ports.

Figure 4-1 Sequence of Edgeflow Calls



Writing Custom Modules

When you create a custom module, you must:

- Implement the Required Interface and Static Methods
- Implement Custom Serializers and Deserializers, if needed

- Build and Deploy Your Custom Module

These tasks are described in the sections that follow.

Implement the Required Interface and Static Methods

At the very high level, a custom module class must satisfy the following requirements:

- Extend the class `com.bea.rfid.edge.api.workflow.AbstractWorkflowModulePlugin`.
- Implement the static method to get PluginMeta.

  The configuration framework uses a static method getPluginMeta() to get the PluginMeta object with the configuration parameter names and other configuration parameters of the module.

Extend the AbstractWorkflowModulePlugin Class

The abstract base class, `com.bea.rfid.edge.toolkit.workflow.AbstractWorkflowModulePlugin` implements `WorkflowModulePlugin`.

Your module should extend the `AbstractWorkflowModulePlugin` class.

The abstract base class provides the following:

- It uses the Edge Server event handling mechanism, sets up the event queue using the queue size specified in the Edge Server configuration and using resources available within the Edge Server.
- Startup function and receiving input messages to the module are done using the event queuing mechanism, which ensures that the module receives only one message at a time.
- If an exception is thrown by an input port's `processMessage` method, error output is sent with the exception information in it.
- If an exception occurs during `doStartup`, a warning level message is logged.
- Provides the utility method, `sendToModules`, to send output to other modules.
- Provides the log methods, which adds module and/or port names to the log message. The log methods use a logger created by this class, with the fully qualified class name of the module appended with the module name as the name of the logger. This allows logging to be filtered for the specific instance of the module or for all modules of that class.

Perform the following steps to implement an edgeflow module using this abstract base class:

1. The class must have a zero-argument constructor, either explicitly implemented or the default one for the class when no others are implemented. The edgeflow framework relies on this constructor being there to instantiate a module.
2. Provide an implementation for the configure() interface method.
3. Provide an implementation for the abstract method `createModuleDescriptor`. This method should return a `ModuleDescriptor` that contains `com.bea.rfid.edge.api.workflow.OneWayInputPort` for all the input ports of the module and `com.bea.rfid.edge.api.workflow.Port` for all the output ports.
4. If the module has startup functions to perform, override the `doStartup` method.
5. Provide an input port implementation for each input port of the module. This can be done by extending the inner class `OneWayPortImpl` in `AbstractWorkflowModulePlugin`. The only method that needs to be implemented in this implementation is `processMessage`.
6. If the module has an error output port, it should call `setupErrorOutputs(PluginConfig)` during the configuration step to make sure that the error output connection is set up correctly in the abstract class.

Example of Extending the AbstractWorkflowModulePlugin Class

This section provides an example of extending the AbstractWorkflowModulePlugin class.

The example code is from the ProvisioningSerialRangeModule.java file in the samples directory (`EDGE_SERVER_HOME\samples\EdgeFlows\CustomModules\`).

Note: Only code fragments are shown here for example purposes; see the actual samples for complete code.

1. Implement a zero-argument constructor for the class.

   The example uses the default zero-argument constructor provided by Java.
2. Provide an implementation for the configure() interface method, as shown in Listing 4-1.

Listing 4-1 Sample Implementation of Configure() Method

```
// interface method implementation

    public void configure(PluginConfig config)
```

```
        throws PluginException
    {
        long rangeStart = Long.parseLong(getStringValue(config, 
RANGE_START_PARAM));
        long rangeEnd = Long.parseLong(getStringValue(config, 
RANGE_END_PARAM));
        if (rangeEnd < rangeStart)
            throw new PluginException("Range end cannot be less than range 
start");
        m_rangeBO.setBegin(rangeStart);
        m_rangeBO.setEnd(rangeEnd);

        // output connections
        m_outModules = getOutputModulesList(config, OUT_PORT_KEY);
        setupErrorOutputs(config);
    }
```

3. Provide an implementation for the abstract method `createModuleDescriptor`. This method should return a `ModuleDescriptor` that contains `com.bea.rfid.edge.api.workflow.OneWayInputPort` for all the input ports of the module and `com.bea.rfid.edge.api.workflow.Port` for all the output ports, as shown in Listing 4-2.

Listing 4-2 Example of Implementing the createModuleDescriptor() Method

```
// abstract method implementation
    protected ModuleDescriptor createModuleDescriptor()
    {
        List <Port> inPorts = new ArrayList <Port> (1);
        inPorts.add(new EPCStringInputPort());
        return new ModuleDescriptorImpl(inPorts, getOutputPorts());
    }
:
private static List<Port> getOutputPorts()
    {
        List<Port> outPorts = new ArrayList(2);
        outPorts.add(new PortImpl(OUT_PORT_KEY,
                                  "Replenishment",
                                  ListOneWayPortType.URI_STRING_LIST));
        outPorts.add(ERROR_OUT_PORT);
        return outPorts;
    }
```

4. If the module has startup functions to perform, override the `doStartup` method.

   The example does not use the `doStartup` method.

5. Provide an input port implementation for each input port of the module. This can be done by extending the inner class `OneWayPortImpl`. The only method that needs to be implemented in this implementation is `processMessage`, as shown in Listing 4-3.

Listing 4-3 Example Input Port Implementation

```
/**
     * Implementation for the object input port.
     */
    private class EPCStringInputPort
        extends OneWayInputPortImpl
    {
        public EPCStringInputPort()
        {
            super(IN_PORT);
        }


        public void processMessage(Object msg)
            throws Exception
        {
            String inStr = (String)msg;
            log(Level.FINE, "processing message: " + inStr);
            // get EPC range for this EPC string
            List<String> ranges = m_rangeBO.getRanges(inStr);
            // get EPC range for this EPC string
            log(Level.FINE, "retrieved EPC ranges: " + ranges.toString());

            // send it out
            if ((m_outModules!=null && m_outModules.size()>0))
            {
                sendToModules(ranges, m_outModules);
            }
        }
    }
```

6. If the module has an error output port, it should call `setupErrorOutputs(PluginConfig)` during the configuration step to make sure that the error output connection is set up correctly in the abstract class:

   ```
   setupErrorOutputs(config);
   ```

Implement Static Method to Get PluginMeta

Your custom module needs to implement a static method to get the PluginMeta.

Every module has configuration parameters that determine the behavior of the module instance. For example, the External Sender module has parameters that let the user specify an XSLT file, destination URI(s), input data connections, and error output connections for a given instance of that module.

The configuration framework uses the static getPluginMeta() method to get the PluginMeta object with the configuration parameter names and other configuration parameters of the module.

The PluginMeta object contains a list of PluginParameterMeta objects with the details of its configuration parameters. The order of the PluginParameterMeta objects in the list is the order in which the administration console will display the parameters.

You use the com.connecterra.ale.dynamicconfig.bean.DynamicConfigBeanFactory(); constructor to create an instance of the com.connecterra.ale.dynamicconfig.api.DynamicConfigFactory interface (DynamicConfigBeanFactory implements DynamicConfigFactory), and then use the resulting DynamicConfigFactory.createPluginMeta() method to create an instance of PluginMeta. This sequence is shown in Listing 4-4.

When adding parameters to the PluginMeta object you created, you need to define all the output ports under the sub-configuration Module Outputs and all the input ports under sub-configuration Module Inputs.

To do this, the `AbstractWorkflowModulePlugin` class has utility methods, `createOutputPortsSubConfig()` and `createInputPortsSubConfig()`, to create the sub-configuration entries from a given list of ports. These utility methods create the ports under the appropriate sub-configuration entries and add constraints to enable correct wiring with other module instances. All of the required information is extracted from the ports that are passed in.

Listing 4-4 Example of Implementing PluginMeta

```
private static com.connecterra.ale.dynamicconfig.api.PluginMeta
s_pluginMeta;


    public static  com.connecterra.ale.dynamicconfig.api.PluginMeta
getPluginMeta()
```

```
        throws PluginException
    {
        if (s_pluginMeta != null)
            return s_pluginMeta;

       com.connecterra.ale.dynamicconfig.api.DynamicConfigFactory factory =
                     new
com.connecterra.ale.dynamicconfig.bean.DynamicConfigBeanFactory();
        s_pluginMeta = factory.createPluginMeta(
                     ProvisioningSerialRangeModule.class.getName(),
                     MODULE_NAME,
                     I18N_MODULE_NAME,

com.connecterra.ale.dynamicconfig.api.PluginMeta.Role.WORKFLOW_MODULE);
        s_pluginMeta.setDescription("Provisioning Serial Range Module");

        // range start
        s_pluginMeta.addParameterMeta(factory.createRequiredNumericMeta(
            RANGE_START_PARAM, // key name
            I18N_RANGE_START_PARAM, // display name
            "EPC range start", // desc
            0,
            Long.MAX_VALUE));

        // range end
        s_pluginMeta.addParameterMeta(factory.createRequiredNumericMeta(
            RANGE_END_PARAM, // key name
            I18N_RANGE_END_PARAM, // display name
            "EPC range end", // desc
            0,
            Long.MAX_VALUE));

        // Output ports
        s_pluginMeta.addParameterMeta(createOutputPortsSubConfig(factory,
                                                                   getOutputPorts()));

        // Input ports
        List<Port> inPorts = new ArrayList(1);
        inPorts.add(IN_PORT);
        s_pluginMeta.addParameterMeta(createInputPortsSubConfig(factory,
inPorts));

        return s_pluginMeta;
    }

    private static List<Port> getOutputPorts()
    {
        List<Port> outPorts = new ArrayList(2);
        outPorts.add(new PortImpl(OUT_PORT_KEY,
                                  "Replenishment",
                                  ListOneWayPortType.URI_STRING_LIST));
        outPorts.add(ERROR_OUT_PORT);
        return outPorts;
    }
```

Implement Custom Serializers and Deserializers

Edge flow modules may accept XML or string messages from outside the edgeflow system. These messages must be converted to the corresponding Java type before they are sent to the modules.

Edge flow messages can also be sent out externally (via HTTP or JMS, for example). The messages will be serialized to their XML or String over-the-wire format before they are sent outside.

Custom modules may have their own data types that need to be passed over the wire and sent to and from the custom modules using the standard framework mechanisms.

To do this, you need to write serializers that implement the `com.bea.rfid.workflow.api.MessageSerializer` interface, and deserializers that implement the `com.bea.rfid.workflow.api.MessageDeserializer` interface.

You then need to register these classes with the naming service (described in the Edgeflow API Overview section).

The `com.bea.rfid.edge.plugin.workflow.epcis.encoding.EPCISDocumentSerializer` class, which is shown in Listing 4-5, implements both the `com.bea.rfid.workflow.api.MessageSerializer` and `com.bea.rfid.workflow.api.MessageDeserializer` interfaces.

Once the serializers and deserializers are created and registered, the framework will look up the appropriate serializer based on the Java type, and the appropriate deserializer based on the XML namespace and root element name, and perform the conversion.

Listing 4-5 shows an example of registering serializers and deserializers.

Listing 4-5 Example of Registering Serializers and Deserializers

```
import
com.bea.rfid.edge.plugin.workflow.epcis.encoding.EPCISDocumentSerializer;

:

private static void registerSerializers(WorkflowNamingService ns)

    {

        if (s_serializersRegistered)
```

```
            return;

        EPCISDocumentSerializer ser = new EPCISDocumentSerializer();
        ns.registerDeserializer(ser);
        ns.registerSerializer(ser);
        s_serializersRegistered = true;
    }
registerSerializers(getWorkflowManager().getWorkflowNamingService());
```

Build and Deploy Your Custom Module

Use the following steps to build and deploy the your custom modules:

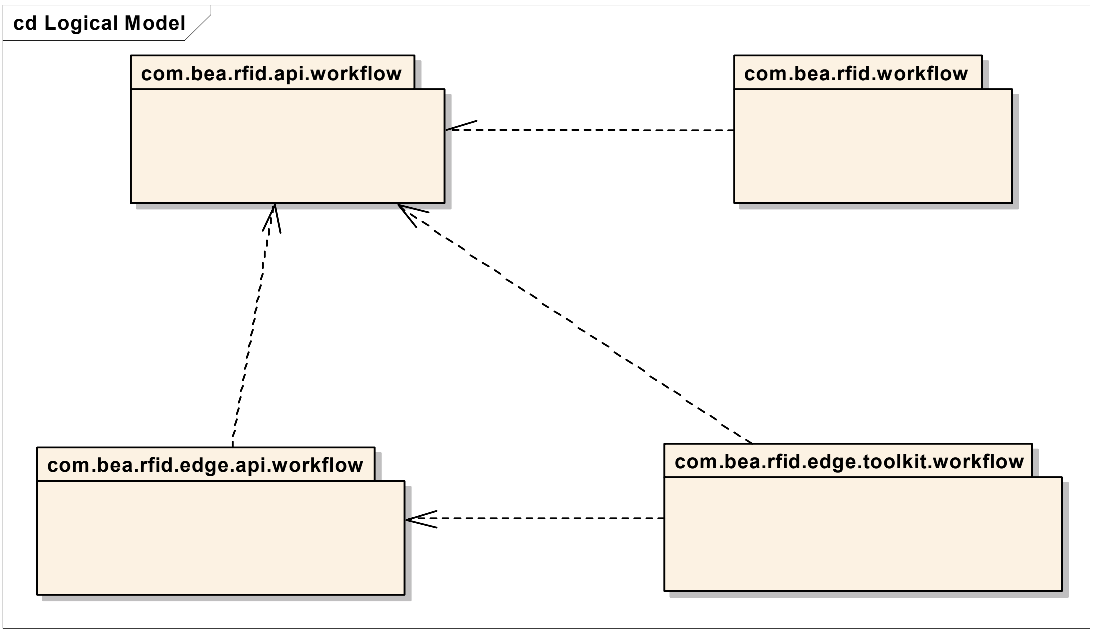
1. Modify the example build script (`EDGE_SERVER_HOME\samples\EdgeFlows\CustomModules\build.bat)` to compile your custom module Java files and package them into a jar file. In particular, make sure that rfta-common.jar and edgeserver.jar are in your classpath.
2. Copy your custom module jar file to the `RFID_EDGE_HOME\plugin\edgeflows` directory.
3. Restart the Edge Server.

The Edge Server Administration Console should now list the custom module.

Edgeflow API Overview

Figure 4-2 shows the high level packages in edgeflow API and the dependencies between the packages. More details of the packages and the classes in them are in the sections that follow.

Figure 4-2 Edgeflow API High-Level Packages


cd Logical Model
com.bea.rfid.api.workflow
com.bea.rfid.workflow
com.bea.rfid.edge.api.workflow
com.bea.rfid.edge.toolkit.workflow

Package com.bea.rfid.api.workflow

This package defines public interfaces that may be used on the client side or the server side of the edgeflow system.

Types

The types sub-package contains the data types used by various BEA-provided modules. This package defines interfaces and classes for common message data types sent or received by edge flow modules and messaging framework. The following interfaces and classes are defined:

Interfaces:

- ErrorOutput — Interface representing error output produced by edge flow modules.
- Manifest — Interface representing the manifest (parent and children identifiers expected) for an aggregation. This may just contain the error details, if an error occurred while trying to get the manifest.

- TriggerOutput — Represents trigger output generated by trigger receiver modules. This is an empty interface for now. This may be extended in the future to include additional information.

Classes :

- WorkflowModuleState — Class representing the different states of an edge flow module. A module may be in one of the following states:

  WORKING: A module that is actively processing inputs that it receives is in this state.

  IDLE: A module that is not processing its inputs is in this state.

Most modules have an error output port, which is used to send the details of errors during message processing. The ErrorOutput interface defines a type for the error output data. Similarly, an interface is defined for trigger output, and a class for module state.

Encoding

The encoding sub-package contains interfaces for encoding and decoding edge flow message types.

Custom module developers must write serializers and deserializers for custom message types if you want the serialization and deserialization to be handled transparently by the system. These serializers and deserializers must implement the interfaces and must be registered with the edgeflow naming service (described in the Edgeflow API Overview section).

For example, an External Sender will try to find a serializer for any message type that it receives by looking at the serializers registered with the system.

The following interfaces are defined:

- MessageDeserializer — Interface that must be implemented by edge flow message deserializers for messages should be automatically deserialized by the framework.

- MessageSerializer — Interface that must be implemented by edge flow message serializers for messages should be automatically serialized by the framework.

Edgeflow modules may accept XML or other messages from outside the edgeflow system. These messages must be converted to the corresponding Java type before they are sent to the modules. Classes that convert messages from their over-the-wire format to the Java types must implement the `MessageDeserializer` interfaces.

Edge flow messages can be sent externally (via HTTP or JMS, for example). The messages will be serialized to their XML or other over-the-wire format, before they are sent outside. Classes that serialize the Java types must implement the `MessageSerializer` interfaces.

Config

The Config sub-package defines constants for the dynamic set types handled by the edgeflow framework. You should use the interface and class in this package when creating configuration parameters whose values come from a dynamically computed set.

For example, the `DynamicSetConstants.DynamicSetID.LOGICAL_READER` constant should be used as the set ID for a parameter whose value must be one of the logical readers defined in the system.

The following interface and class are defined:

Interface:

- DynamicSetConstants — Interface that defines enumerations for identifiers and parameter names of dynamic sets in WORKFLOW_MODULE role.

Class:

- DynamicSetConstants.DynamicSetID — Defines identifiers for the Dynamic Set types supported by the edge flow system.

Package com.bea.rfid.edge.api.workflow

The interface for an edgeflow module and its properties as required by the edgeflow engine is defined here. These interfaces are used by the edgeflow engine to create a module instance or to get to an input port of a module and send a message to it.

Note: You do not need to implement these interfaces. Implementations are provided for your convenience in the com.bea.rfid.edge.toolkit.workflow package.

The following interfaces are defined in this package:

- ModuleDescriptor — Contains information about about a module's input and output ports.
- OneWayInputPort — Represents input port of an edge flow modules that receives one-way messages. This is the only input port type that is currently supported.
- Port — Represents an input or output port of a module.
- PortType — Holds information about what type of messages a port may send or accept. This interface has methods that will help determine if it is valid to connect two port types or if a port type will process a certain message.

- WorkflowManager — Edgeflow manager interface used by the edge server to define, reconfigure or delete edge flow modules. This interface is implemented by the edgeflow engine.
- WorkflowModulePlugin — This interface defines methods that are required by the edgeflow engine to instantiate, initialize, configure and startup an edgeflow module. The sequence in which initialization/startup methods are called is as follows:

  a. constructor

  b. initialize

  c. configure

  d. startup
- WorkflowNamingService — Interface to get to workflow related objects like workflow modules and serializers and deserializers for the different types. This interface is implemented by the edgeflow engine.

Package com.bea.rfid.edge.toolkit.workflow

This package defines a set of abstract classes and utility classes to assist in writing custom modules. The modules provided with the Edge Server installation are implemented using these classes.

The following classes are provided:

- AbstractWorkflowModulePlugin — Abstract base class for edge flow modules.
- AnyPortType — Represents the type of a Port that sends or receives messages of any type.
- ListOneWayPortType — Represents a port type that sends or receives one-way list messages containing a certain class of objects.
- ModuleDescriptorImpl — A simple implementation of ModuleDescriptor.
- MultiTypeOneWayPortType — Represents the type of a one-way Port that accepts/sends multiple types of input.
- PortImpl Simple — Implementation of Port interface.
- SerializationHelper — Singleton class with utility methods that help in serialization and deserialization of objects within the edge flow system.

- SimpleOneWayPortType — Represents a port type that sends or receives one-way scalar type messages.

Package com.bea.rfid.workflow

This package defines a set of Java classes that do not rely on any edgeflow framework classes. Many of the modules provided with the Edge Server installation use these POJO classes. These classes isolate the core tasks of the module from the edgeflow framework. This means that if a custom module is to be written that behaves slightly differently from a standard module, it can be done easily by making use of these Java classes. Similarly, these classes make it easy to create similar modules in some other workflow framework outside of the Edge Server.

Three sub-packages are provided:

- com.bea.rfid.workflow.epcis
- com.bea.rfid.workflow.print
- com.bea.rfid.workflow.util

com.bea.rfid.workflow.epcis Sub-Package

The object and aggregation event generators include Java classes that generate EPCIS Object or Aggregation events for tag read inputs.

The following classes are provided:

- AggregationEventGenerator — Java class to generate EPCIS Aggregation Events for input tags.
- AggregationReconciler — Compares a given manifest and aggregation and produces a result based on the property settings.
- EventGenerator — Abstract base class for generating EPCIS events for list of com.connecterra.ale.api.ECReportGroupListMember containing tags read.
- ObjectEventGenerator — Plain java class to generate EPCIS Object Events for input tags.
- ReconciliationResult — Utility class to hold Aggregation event reconciliation result.

com.bea.rfid.workflow.print Sub-Package

This is the Java class for programming tags. The following interface and class are provided:

Interfaces

- PrintObserver — Interface that must be implemented by classes that should process callbacks from PrintTag.

Classes

- PrintTag — Java class for programing tags for specified EPC class or with the specified EPC.

com.bea.rfid.workflow.util Sub-Package

The following interfaces and classes are defined:

Interfaces

- AccumulationObserver — Interface for observers of Accumulator. AccumulationObserver is the interface that Accumulator uses to notify regarding the final accumulated tags.
- DirectionObserver — Interface for observers for A to B direction.

Classes

- Accumulator — This Java class encapsulates the Tag Accumulator module functionality. AccumulationObserver is the interface that Accumulator uses to notify regarding the final accumulated tags.
- BarcodeDecoder — Converts bar code string or EPC to EPC ID pattern.
- BarcodeDecoder.BarcodeEncoding Converts EPC ID pattern to UCC12 or UCC13???????
- Directionality — Class to determine direction of tags moving between two points. This Java class determines the direction of tags based on ECReports with addition and deletion reports, or based on tag reads inputs (tag list) at different points.
- TagListBuilder — The tag list builder is used to get list of tags from an ECReports or PCWriteReport.

Custom Module Sample

The RFID Edge Server 3.0 software includes a custom module sample in the following directory:
`EDGE_SERVER_HOME\samples\EdgeFlows\CustomModules`

The sample includes Java files that demonstrate how to write custom modules. These files include:

- Module that generates an EPCIS Object Event from a list of tags.

- Module that generates an EPCIS Aggregation Event from a list of tags.
- Module that provides EPC serial numbers in the configured range for an EPC type
- Module that generates manifest for an EPCIS Aggregation Event using the EPCIS query interface.

All of the source code for the example is provided.

Building and Deploying the Sample

Use the following steps to build and deploy the sample modules:

1. Run the build script (`EDGE_SERVER_HOME\samples\EdgeFlows\CustomModules\build.bat)` to compile the Java files and package them into the `custom-modules.jar` file.
2. Copy the `custom-modules.jar` file to the `RFID_EDGE_HOME\plugin\edgeflows` directory.
3. Restart the Edge Server.

The Edge Server Administration Console should now list the custom modules, with names that begin with “Sample”.

Workflow Module Reference for Versions Prior to 3.0

This section describes the WebLogic RFID Edge Server workflow modules that were released prior to Version 3.0.

Note: Starting with version 3.0 of the Edge Server, limited support is provided for the workflow modules. You cannot use the Administration Console to create workflow modules, and messaging between the workflow modules and the new edge flow modules (described in Edgeflow Module Reference) is not supported.

However, you can use the Console to import and export existing workflow modules.

The following sections describe the WebLogic RFID Edge Server workflow modules that were released prior to the Version 3.0 release of the product:

Overview of the Legacy RFID Edge Server Workflow Modules

The following sections describe workflow module configuration parameters. You may notice similarities across modules. Each module requires parameters for operation (which can include information like timing data and EPCIS data) and parameters that allow it to connect to other modules and send data to outside destinations.

You configure workflow modules using the RFID Edge Server Administration Console. See Configuring Workflow Modules in *Using the Administration Console*.

Output Destinations

Output destinations are destinations for workflow messages. Workflow modules can send messages to multiple destinations; you do this by separating entries in the *Output Destinations* field with commas. The possible destinations and configuration information are shown in Table 5-1.

Table 5-1 Output Destinations Configuration Properties

Destination	Field Contents
Another workflow module	Destination module name (from the *Module Name* field).
A notification destination	A URI with the appropriate format for this destination. Drivers for each kind of destination must be configured within the RFID Edge Server. See *Installing WebLogic RFID Edge Server* for more information on the types of notification mechanisms available in the RFID Edge Server and their URI formats.

Bidirectional Portal Module

The Bidirectional Portal module monitors the movement of EPC tags between two read points and detects directional item movement. Based on tag direction, it maintains tag counts, generates EPCIS ObjectEvents, and manipulates stack lights and LED displays. If a parent tag is detected, it generates an EPCIS AggregationEvent.

Note: Avoid sharing numeric display modules between business modules; unexpected behavior will result.

Several fields in this module configuration are repeated for both read points (locations). These are indicated below by the notation *(Define one for first location and one for second location)*.

Table 5-2 Bidirectional Portal Module Configuration Properties

Field Label	Description
Module Name	Unique user-defined identifier for this module.
Module Type	Type of module. Values are chosen from a list of module types.
Object Event Output Destinations	Destinations for EPCIS ObjectEvent information. See "Output Destinations" on page 5-2. Note that this information can be sent to other destinations as well as to EPCIS.
EPCIS Read Point URI	Location that identifies how or where the EPCIS event was detected.
Stale Tag Timeout	How long (in milliseconds) a tag is kept *active* (included in the current state management tracking) after it is no longer detected at the second read point. Defaults to 180000.
Tag Addition ECReport Name	*(Define one for Side A and one for Side B)* Required. Name of the ECReport containing tags that have been newly read during this event cycle. Configure this ECReport to have *Additions* as its Report Set. Selecting *Tag List* in the Report Contents is required.
Tag Deletion ECReport Name	*(Define one for Side A and one for Side B)* Required. Name of the ECReport containing tags that have appeared during prior event cycles, but do not appear in the current event cycle. Configure this ECReport to have *Deletions* as its Report Set. Selecting *Tag List* in the Report Contents is required.
(EPCIS) Object Event Action	*(Define one for each direction: Side A to Side B, and Side B to Side A)* Required. One of the EPCIS actions: ADD, DELETE, or OBSERVE. Defaults to OBSERVE.
(EPCIS) Aggregation Event Action	*(Define one for each direction: Side A to Side B, and Side B to Side A)* Required. One of the EPCIS actions: ADD, DELETE, or OBSERVE. Defaults to OBSERVE.
(EPCIS) Business Step URI	*(Define one for first location and one for second location)* Business context of an EPCIS event; the business process step that was occurring when the event was captured.

Table 5-2 Bidirectional Portal Module Configuration Properties (Continued)

Field Label	Description
(EPCIS) Disposition URI	*(Define one for each direction: Side A to Side B, and Side B to Side A)* Business state of the object participating in the EPCIS event (for example, "available for sale").
(EPCIS) Business Location URI	*(Define one for each direction: Side A to Side B, and Side B to Side A)* Uniquely identified place where the object(s) observed will be located after the EPCIS event.
(Numeric Display) Display Duration	*(Define one for each direction: Side A to Side B, and Side B to Side A)* Duration (in milliseconds) that data should be displayed or flashed on the LED, after which time the LED will go blank. Defaults to 0 (display infinitely).
(Numeric Display) Reset Interval	*(Define one for each direction: Side A to Side B, and Side B to Side A)* How long (in milliseconds) to wait after last LED update before resetting the LED count to 0. Defaults to 15000.
(Numeric Display) Output Destinations	*(Define one for each direction: Side A to Side B, and Side B to Side A)* Destinations for numeric display (LED) information. See "Output Destinations" on page 5-2.
(Stack Light) Display Duration	*(Define one for each direction: Side A to Side B, and Side B to Side A)* Duration (in milliseconds) that each light combination should be displayed on the stack light. Defaults to 0 (display infinitely).
(Stack Light) Output Destinations	*(Define one for each direction: Side A to Side B, and Side B to Side A)* Destinations for stack light information. See "Output Destinations" on page 5-2.
Control Tag EPC Patterns	One or more EPC patterns belonging to a control tag. Entering a pattern in this field indicates to the workflow that aggregation (for example, pallet) data is expected. For more information on EPC patterns, see the *EPC Tag Data Standards* document, available at *http://www.epcglobalinc.org/standards/*.
Parent Tag EPC Patterns	One or more EPC patterns belonging to a parent tag. Entering a pattern in this field indicates to the workflow that there is an EPC hierarchy.

Table 5-2 Bidirectional Portal Module Configuration Properties (Continued)

Field Label	Description
Delay After Detecting Direction	Delay (in milliseconds) between the control tag being read at the second read point, and the AggregationEvent being sent. Defaults to 0.
Output Destinations	Destinations for EPCIS AggregationEvent information. See "Output Destinations" on page 5-2. Note that this information can be sent to other destinations as well as to EPCIS.

Observe Portal Module

The Observe Portal module monitors EPC tag traffic across a read point. It maintains tag counts, generates EPCIS ObjectEvents, and manipulates stack lights and LED displays.

Table 5-3 Observe Portal Module Configuration Properties

Field Label	Description
Module Name	The unique user-defined identifier for this module.
Module Type	The type of module. Values are chosen from a list of module types.
Tag Addition ECReport Name	Required. Name of the ECReport containing tags that have been newly read during this event cycle. This ECReport should be configured to have *Additions* as its Report Set.
(EPCIS) Object Event Action	Required. One of the EPCIS actions: ADD, DELETE, or OBSERVE. Defaults to OBSERVE.
(EPCIS) Business Step URI	Business context of an EPCIS event; the business process step that was happening when the event was captured.
(EPCIS) Disposition URI	Business state of the object participating in the EPCIS event (for example, "available for sale").
(EPCIS) Read Point URI	Location that identifies how or where the EPCIS event was detected.
(EPCIS) Business Location URI	Uniquely identified place where the object(s) observed will be located after the EPCIS event.
(EPCIS) Output Destinations	Destinations for EPCIS ObjectEvent information. See "Output Destinations" on page 5-2. Note that this information can be sent to other destinations as well as to EPCIS.

Table 5-3 Observe Portal Module Configuration Properties (Continued)

Field Label	Description
(Numeric Display) Display Duration	Duration (in milliseconds) that data should be displayed or flashed on the LED. Defaults to 0 (display infinitely).
(Numeric Display) Tag Deletion ECReport Name	Name of the ECReport containing tags that have appeared during prior event cycles, but do not appear in the current event cycle. This ECReport should be configured to have *Deletions* as its Report Set.
Reset Interval	How long (in milliseconds) to wait after last LED update before resetting the LED count to 0.
(Numeric Display) Output Destinations	Destinations for numeric display (LED) information. See "Output Destinations" on page 5-2. **Note:** If Output Destinations is configured, you must provide either the Tag Deletion ECReport Name or the Reset Interval.
(Stack Light) Display Duration	Duration (in milliseconds) that each light combination should be displayed on the stack light. Defaults to 0 (display infinitely).
(Stack Light) Output Destinations	Destinations for stack light information. See "Output Destinations" on page 5-2.

Message Notifier Module

The Message Notifier module defines the parameters needed to send a message to a notification destination, using a driver previously defined in the RFID Edge Server. Only messages that support XML serialization can be sent. The module sends information from other workflow modules to a destination specified in the *Destination URI* field. It returns a result, which it sends to the destinations specified in the *Failure Output Destinations and Success Output Destinations* fields.

Table 5-4 Message Notifier Module Configuration Properties

Field Label	Description
Module Name	The unique user-defined identifier for this module.
Module Type	The type of module. Values are chosen from a list of module types.
Destination URI	Required. URI of the notification destination.

Table 5-4 Message Notifier Module Configuration Properties (Continued)

Field Label	Description
XSLT File Name	Path and file name of an XSLT file used to transform the message. This file must be available to the RFID Edge Server.
Failure Output Destinations	Destinations for module status messages. See "Output Destinations" on page 5-2. This field may not contain a reference to a notification destination.
Success Output Destinations	Destinations for module status messages. See "Output Destinations" on page 5-2. This field may not contain a reference to a notification destination.

Numeric Display Module

The Numeric Display module defines the parameters needed to connect to an LED display. It connects to the LED device using the host and port specified, and attempts to display a value (specified in the calling module). The module returns a result, which it sends to the destinations specified in the *Failure Output Destinations and Success Output Destinations* fields.

Table 5-5 Numeric Display Module Configuration Properties

Field Label	Description
Module Name	Unique user-defined identifier for this module.
Device Host	Required. Host name of the device.
Device Port	Required. TCP port used by the device.
Device Connection Timeout	Required. The amount of time (in milliseconds) the device maintains an idle connection with the Edge Server before closing the connection.
Initial Display	Value displayed when module comes on.
Failure Output Destinations	Destinations for module status messages. See "Output Destinations" on page 5-2. This field may not contain a reference to a notification destination.
Success Output Destinations	Destinations for module status messages. See "Output Destinations" on page 5-2. This field may not contain a reference to a notification destination.

Stack Light Module

The Stack Light module defines the parameters needed to connect to a stack light. It connects to the stack light using the logical name provided, and attempts to set the lights as specified in the *Initial Color* field. The module returns a result, which it sends to the destinations specified in the *Failure Output Destinations and Success Output Destinations* fields.

Table 5-6 Stack Light Module Configuration Properties

Field Label	Description
Module Name	Unique user-defined identifier for this module.
Module Type	Type of module. Values are chosen from a list of module types.
Logical Reader Name	Required. Logical name for the stack light device.
Initial Color	Color displayed when module comes on. Allowed values are `GREEN`, `RED`, and `NONE` (no communication with the Stack Light).
Failure Output Destinations	Destinations for module status messages. See "Output Destinations" on page 5-2. This field may not contain a reference to a notification destination.
Success Output Destinations	Destinations for module status messages. See "Output Destinations" on page 5-2. This field may not contain a reference to a notification destination.