

# IloTA™ industrial IoT Platform Manual



elliTek, Inc.

7139 Regal Lane | Knoxville, TN 37918 | 865.409.1555

# IIoTA industrial IoT Platform Manual

## Table of Contents:

### [IIoTA industrial IoT Platform Introduction](#)

- [The Workbench](#)
  - [Starting and shutting down the Workbench](#)
  - [Scanning for local or remote nodes](#)
  - [Parts of the Workbench window](#)
  - [Managing the Workbench Nodes list](#)
  - [Backing up and Restoring a node's configuration](#)
  - [Node alert states](#)
  - [Exiting safe mode](#)
  - [Keyboard Navigation](#)
  - [Cross References](#)
  - [Viewing product documentation](#)
- [Projects and triggers](#)
  - [Projects](#)
  - [Defining a trigger](#)
    - [Trigger event type](#)
    - [Trigger local variables, static variables, macros and event variables](#)
      - [Trigger macros](#)
      - [Trigger event variables](#)
    - [Trigger settings](#)
    - [Trigger details](#)
    - [Trigger actions](#)
    - [Trigger validation](#)
    - [Trigger search](#)
  - [Using an individual project's tab](#)
  - [Exporting a project or trigger](#)
  - [Importing a project or trigger](#)

- [Using the Canvas Editor](#)
  - [Adding actions using the Canvas Editor](#)
  - [Action statement execution paths](#)
  - [The Canvas Editor toolbar](#)
  - [The Canvas Editor Context Menus](#)
- [Trigger event type reference](#)
  - [Data](#)
    - [Example data event type trigger conditions](#)
  - [Listener](#)
  - [On-Demand](#)
  - [Schedule](#)
  - [SubTrigger](#)
  - [User Operation](#)
  - [Variable Group](#)
- [Store and Forward event](#)
- [Internal events](#)
  - [Device State Change](#)
  - [Logging Event](#)
  - [Mapping Log File Rolled Event](#)
  - [System](#)
  - [Time Change](#)
  - [Trigger Status Change](#)
- [Networking events](#)
  - [FTP Server Activity](#)
  - [HTTP Page Listener](#)
  - [MQTT Publish Receive](#)
  - [Receive TCP Message](#)
  - [Receive UDP Message](#)
  - [TCP Listener Status](#)
- [PLC Logic Events](#)
- [Trigger actions reference](#)

- [Expression](#)
- [Generate Random Number](#)
- [Set](#)
  - [Examples of the Set action](#)
  - [Set action and data conversion considerations](#)
- [Wait](#)
- [Array](#)
  - [Array Average](#)
  - [Array Fill](#)
  - [Array Get](#)
  - [Array Set](#)
  - [Find Array Maximum](#)
  - [Find Array Minimum](#)
- [Binary](#)
  - [Binary Fill](#)
  - [Binary Get](#)
  - [Binary Set](#)
  - [Decode Binary Buffer](#)
  - [Encode Binary Buffer](#)
- [Bit Manipulation](#)
  - [Get Bit](#)
  - [Set Bit](#)
  - [Bit numbering when using strings](#)
- [Date and Time](#)
  - [Date Diff](#)
  - [Get Date & Time](#)
  - [Get Device Date & Time](#)
  - [Set Date & Time](#)
  - [Set Device Date & Time](#)
  - [Example Get Device Date & Time](#)
  - [Example Set Device Date & Time](#)

- [Device](#)
  - [BCD Conversion](#)
  - [Check Variable Status](#)
  - [Demand Read](#)
  - [Demand Write](#)
  - [Device Commit](#)
  - [Device Control](#)
  - [Enhanced Demand Read](#)
  - [Enhanced Demand Write](#)
  - [Execute Device Command](#)
  - [Get Device State](#)
  - [Variable Group Add Variable](#)
  - [Variable Group Remove Variable](#)
- [Enterprise Communication](#)
  - [Control Transport State](#)
  - [Get Message From Controlled Listener](#)
  - [Logical Unit of Work Begin](#)
  - [Logical Unit of Work Commit](#)
  - [Logical Unit of Work Rollback](#)
  - [Transaction](#)
- [Hash Map](#)
  - [Add to Hash Map](#)
  - [Delete from Hash Map](#)
  - [Find in Hash Map](#)
  - [List Keys from Group in Hash Map](#)
- [Internal](#)
  - [Log Message](#)
  - [Modify Attention Bit](#)
  - [Operating System](#)
  - [Security Policy Control](#)
  - [Set Alert](#)

- [System Variable Get](#)
- [Local Database actions](#)
  - [Local DB Count](#)
  - [Local DB Delete](#)
  - [Local DB Insert](#)
  - [Local DB Select](#)
    - [Example using Local DB Select action](#)
  - [Local DB Update](#)
- [Lua Scripting](#)
  - [Execute Lua Function From File](#)
  - [Execute Lua Script](#)
- [Networking](#)
  - [E-mail](#)
  - [MQTT Publish](#)
  - [Ping](#)
  - [Send UDP Message](#)
  - [TCP Connection Check](#)
- [Routing actions](#)
  - [Do Once](#)
  - [End Execution \(Failure\)](#)
  - [End Execution \(Success\)](#)
  - [Error Handling](#)
  - [For](#)
  - [If](#)
  - [Regular Expression](#)
  - [Jump](#)
  - [Connector](#)
- [Staging File System](#)
  - [Compute File Checksum](#)
  - [Configuration Import](#)
  - [File Operation](#)

- [File Read](#)
- [File Write](#)
- [FTP](#)
- [Local DB Export](#)
- [Local DB Import](#)
- [Directory Operation](#)
- [String](#)
  - [Character Position](#)
  - [Convert to Hex String](#)
  - [Get String Length](#)
  - [Get SubString](#)
  - [Parse String to Timestamp](#)
  - [Split String at Delimiter](#)
  - [Split String at Offset](#)
  - [String Builder](#)
  - [String Compare](#)
  - [Time Formatter](#)
- [Trigger](#)
  - [Execute Remote SubTrigger](#)
  - [Execute SubTrigger](#)
  - [Fire Trigger](#)
  - [Get Trigger Statistics](#)
  - [Project Control](#)
  - [Trigger Control](#)
- [Using compound strings](#)
- [Enterprise connectivity](#)
- [Transports](#)
  - [Learning the basics](#)
  - [Using tabs on the Transport window](#)
    - [Timeout tab](#)

- [Store & Forward tab](#)
- [Mapping Log tab](#)
- [Custom Payloads tab](#)
- [Using the Transports tab](#)
  - [Editing a transport](#)
  - [Deleting a transport](#)
  - [Exporting a single transport](#)
  - [Exporting multiple transports](#)
  - [Importing one or more transports](#)
  - [Suspending a transport](#)
  - [Purging data from store and forward](#)
  - [Using the payload error log](#)
- [Transport types](#)
  - [Database transports](#)
    - [Default local database transports](#)
    - [Creating the database transport](#)
  - [TCP transport](#)
    - [Defining a TCP transport](#)
- [Transport maps](#)
  - [Overview of the transport map process](#)
  - [Reviewing the Transport Map window](#)
  - [Specifying transport map characteristics](#)
  - [Creating map variables \(Input or Output tab\)](#)
    - [Creating each map variable separately](#)
    - [Creating map variables automatically](#)
    - [Keeping map variables when changing to another table](#)
  - [Transport map macros](#)
  - [About payloads](#)
  - [Using the Transport Maps tab](#)
    - [Understanding transport map execution times](#)
    - [Using the pop-up menu of the Transport Maps tab](#)

- [Exporting a transport map](#)
- [Exporting multiple transport maps](#)
- [Importing a transport map](#)
- [Testing the transport map](#)
- [Transport map types](#)
  - [Transport map for a database transport](#)
    - [Database destined payloads](#)
    - [Creating the transport map for a database transport](#)
    - [Inserting data \(Insert\)](#)
    - [Inserting multiple rows of data \(Batch Insert\)](#)
    - [Changing data \(Update\)](#)
    - [Using a stored procedure](#)
    - [Using the distinct operator with Select](#)
    - [Updating a set of columns \(Select with Update\)](#)
    - [Determining the number of rows on a table \(Count Rows\)](#)
    - [Deleting a set of rows after select \(Select with Delete\)](#)
    - [Deleting rows in a table \(Delete\)](#)
    - [Using SQL aggregate functions](#)
    - [Referencing a local database transport](#)
    - [Your first transport map with a database transport](#)
      - [Adding a database table](#)
      - [Defining map variables](#)
      - [Associating map variables with table columns](#)
- [Listeners](#)
- [Creating a TCP listener](#)
  - [Allowing and Denying access to the subnet for the TCP listener](#)
  - [ASCII payload considerations for the TCP listener](#)
  - [TCP listener might not receive first message](#)
  - [Example client application to interact with a TCP listener](#)
- [XML listener commands](#)
- [Creating a listener trigger](#)

- [Using the Listeners tab](#)
- [Editing a listener](#)
- [Exporting and importing listeners](#)
  - [Exporting a single listener](#)
  - [Exporting multiple listeners](#)
  - [Importing one or more listeners](#)
- [Using the Listener Maps tab](#)
- [Configuring a default listener map](#)
- [Tabs on the Listener window](#)
  - [Listener Mapping Log tab](#)
  - [Listener Payload tab](#)
    - [Listener ASCII format](#)
    - [Listener Custom format](#)
    - [Listener Map format](#)
    - [Listener XSD format](#)
- [Listener maps](#)
  - [Defining JMS Header attributes](#)
  - [Creating a listener map with an XSD payload](#)
    - [XSD limitations](#)
  - [Creating a listener map with a map message payload](#)
  - [Creating a listener map with a custom format](#)
- [Understanding data types](#)
- [Device connectivity](#)
- [Defining, viewing, and controlling devices](#)
  - [Starting a device](#)
  - [Enabling per device variable security](#)
- [Accessing device variables](#)
  - [Reading the value of a device variable](#)
  - [Writing the value of a device variable](#)
  - [Watching the value of a device variable](#)

- [Defining, viewing, and controlling data mappings](#)
- [Defining, viewing, and controlling variable groups](#)
- [Device types](#)
  - [Aliases device](#)
    - [Aliases device example](#)
  - [Global Variables device](#)
    - [Configuring Global Variables device variables](#)
    - [Configuring Global Variables device structures](#)
      - [Assigning a structure as a data type](#)
      - [Understanding nested structures](#)
- [TCP Listener](#)
  - [TCP Listener device definition](#)
    - [Using the Workbench to define a TCP Listener device](#)
    - [Using the Variables window to access TCP Listener device variables](#)
  - [TCP Listener troubleshooting](#)
- [Logs & Reports](#)
  - [Exceptions Log](#)
  - [Audit Log](#)
  - [Reports](#)
  - [Mapping Logs](#)
- [Local Database](#)
- [Local Database Tables](#)
  - [Inserting, updating, deleting, and viewing rows in a table](#)
  - [Exporting and Importing Local Database tables](#)
  - [Altering Existing Tables](#)
- [Local Database Management](#)
- [Local Database Execute SQL](#)

# IIoTA industrial IoT Platform Introduction

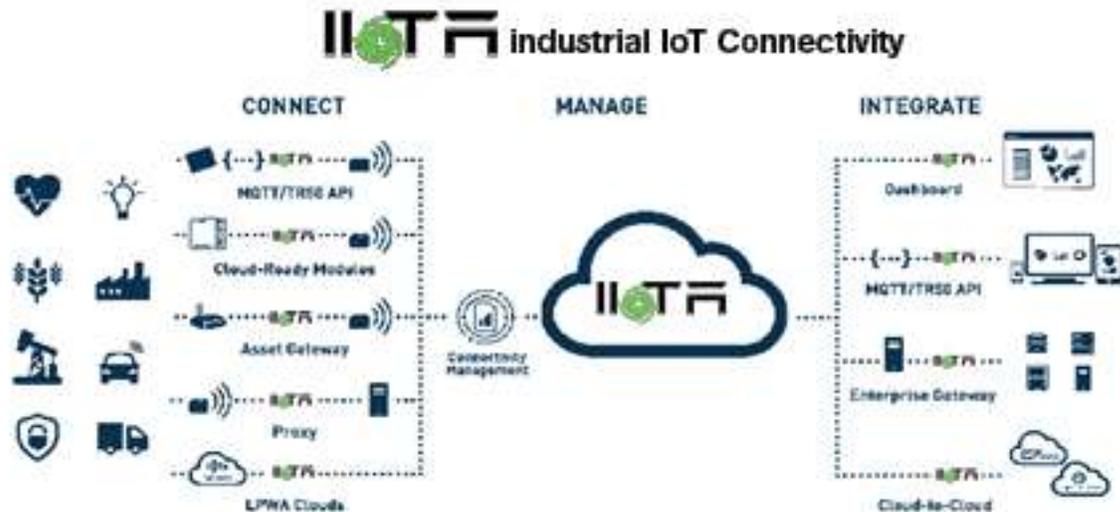
## MES Overview

This introduction section will introduce you to the IIoTA industrial IoT Platform. The IIoTA industrial IoT Platform is a Manufacturing Execution Appliance which helps you connect your *things* (the remote assets with which you want to interact) to your applications (the application services that you want to interface to your assets). The IIoTA industrial IoT Platform does this in a way that is easy to use and allows you to deploy production-ready solutions in a matter of weeks instead of months.

IIoTA is part of a Manufacturing Execution System (MES), which enables factories to quickly create Machine to Machine (M2M) connections and change them as manufacturing lines and work cells are modified.

*elliTek's IIoTA™ platform replaces the Data Commander line. The new features now available with the release of the IIoTA™ platform include:*

- *Higher performance*
- *A 3x increase in memory*
- *The ETH1 RJ45LAN port has been upgraded from 10/100 baseT to 1 GB*
- *Optional addon's for customized user-configurable dashboards, integrated product traceability & skip check, and a fully functioning SQL Database Server*



## Contents

- Management and development tools

- The Workbench
- Common design patterns
  - Enterprise Gateway communicating directly with sensors or devices within a company LAN
    - Example: Devices or sensors connected directly to an Enterprise Gateway

### **IloTA Enterprise Gateway**

The IloTA Enterprise Gateway is unique in the industry. Where most platforms only provide you with APIs so that you can write code to interface your enterprise applications, the IloTA industrial IoT Platform goes a step further and provides enterprise-grade software that can be installed in your back-office and instantly map data from the IloTA industrial IoT Platform to your enterprise applications.

- To review the list of supported enterprise transports, see **Enterprise transports**.

## **Management and development tools**

### **The Workbench**

The IloTA Workbench is the main tool that you will use to configure the Enterprise Gateway. The Workbench provides full development capabilities and the ability to administer and debug an individual device in the field. It is also the same tool that is used to configure how to interface data to your enterprise applications through its easy to use click-to-configure features.

Following is a list of the most common tasks that you can perform using the Workbench:

- Build, manage, and monitor triggers and projects
- Configure and manage connections to devices and sensors
- Configure network settings
- View logs and reports
- Configure connections to enterprise application programs.

## **IloTA industrial IoT Platform: The Workbench**

The Workbench is used to configure the Enterprise Gateway. It provides functionalities to develop, administer, and debug devices in the field. With the *click-to-configure*, you will be able to configure how to interface data in the platform to your enterprise applications.

You can do the following using the Workbench:

- Build, manage, and monitor projects and triggers (your application logic)
- Configure and manage connections to devices and sensors
- Configure network settings

- Configure connections to enterprise application programs
- View logs and reports.

## Assumptions

Before using information in this section, the following should have occurred:

- There is a node installed on the network.
- The Workbench was installed on a computer that has TCP network connectivity to the node.

Download the IIoTA Workbench software for free at <https://www.iiota.net/resources>

## What's Inside

This section contains the following pages:

- [Starting and shutting down the Workbench](#)
- [Scanning for local or remote nodes](#)
- [Parts of the Workbench window](#)
- [Managing the Workbench Nodes list](#)
- [Backing up and Restoring a node's configuration](#)
- [Node alert states](#)
- [Exiting safe mode](#)
- [Keyboard Navigation](#)
- [Cross References](#)
- [Viewing product documentation](#)

# IIoTA industrial IoT Platform: Starting and shutting down the Workbench

## Starting the Workbench

To start the Workbench and log on, do the following:

1. Use the Workbench icon installed on the Windows desktop.  
Or alternatively, use the Windows **Start** menu, select **All Programs > IIoTA > Workbench > Workbench**.
2. The Workbench main window will appear.

Node's user name and password

The Workbench itself does not have a concept of a user logging in before being authorized to access features in the Workbench.

Rather the Workbench uses the default authentication settings when connecting to each and every node and will prompt for a User Name and Password if security settings have been modified from the defaults on that node. The user names, passwords and access control configuration is a feature of each node.

This allows flexibility, if needed, in defining separate access control configurations for each node.

The Nodes list may be empty if this is the first time the Workbench has been started, or the Node list will contain the nodes that were connected to the last time the Workbench was used.

This example shows a few nodes in the Nodes list:

Name	Address	Status	Description
Acme Products Enterprise Gateway	192.168.2.96	Running	
Acme Products Line 1	192.168.2.55	Alert: Warning	
Acme Products Line 2	192.168.2.107	Running	
Acme Products Line 3	192.168.2.185	Running	
Acme Products Line 4	192.168.2.208	Running	

## Shutting down the Workbench

When the Workbench is shutdown, the current nodes in the Nodes list are saved. Subsequent restarts of the Workbench use the Nodes list saved from the last time the Workbench was used. To shutdown (exit) the Workbench, do the following:

1. From the menu bar at the top of the Workbench window, select **User**.
2. Select **Exit**.  
A confirmation dialogue window is displayed, select **Yes** to exit the Workbench.

To change the current user name and password, do the following:

1. From the menu bar at the top of the Workbench window, select **User**.
2. Select **Set Gateway Node Credentials**.  
The Workbench credentials window will appear, allowing you to enter a user name and password for use on each future node connection.

To clear the current user name and password and revert to default security settings, do the following:

1. From the menu bar at the top of the Workbench window, select **User**.
2. Select **Clear Gateway Node Credentials**.

## IIoTA industrial IoT Platform: Scanning for local or remote nodes

When the Workbench is started, the Nodes list contains the nodes from the last time the Workbench was used. The Nodes list will be empty the first time the Workbench is started after it is installed.

To connect to nodes and access the node's features, the Workbench must first scan for the node.

- For local nodes, this is usually done using the node's IP address for nodes that are on the same LAN as the computer running the Workbench.

Scanning a node to access its resources

Each node's configuration settings and application definitions are contained in internal database files on the node. The Workbench uses a **Scan** function to locate the node in the network. The Workbench then uses a **connect** function to log on to the node using the user name and password supplied by the user of the Workbench.

When configuration settings or application definitions are changed, they are updated and stored in the internal database files on the node.

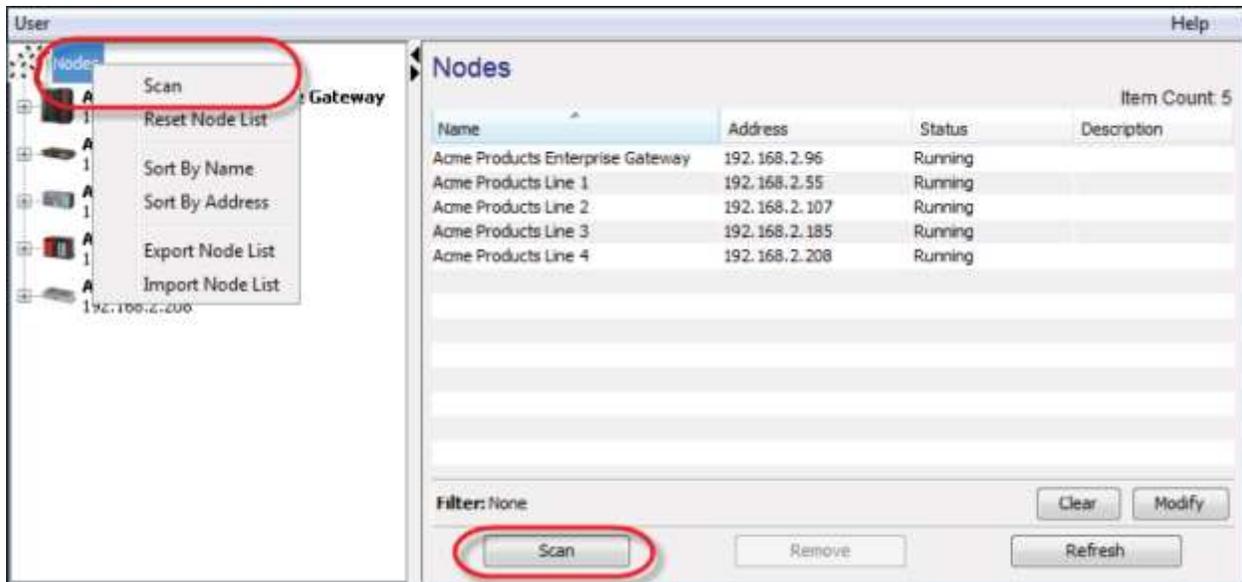
The Workbench does not have an online and offline mode concept or a concept of storing a node's settings and definitions in files on the computer that is running the Workbench.

The runtime products do have Node Back Up and item definition Export features that are described in their sections.

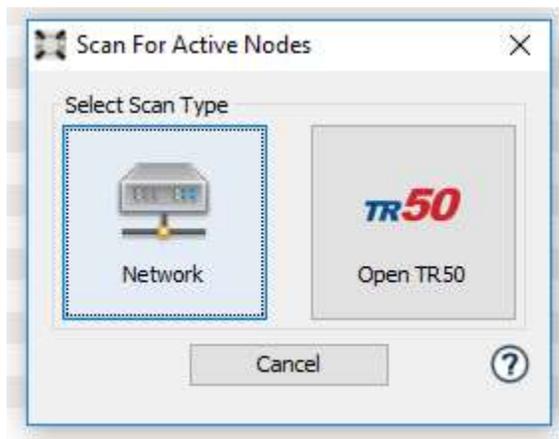
### Scanning for a node by IP address

To scan for a node using its IP address, do the following:

1. From the Workbench left pane, select **Nodes** at the top of the left-hand pane, then right click to display its pop-up menu and select **Scan**.  
Or alternatively, select the **Scan** button at the bottom of the right-hand pane.



- A Scan type window is displayed, allowing different ways for the Workbench to connect to (or scan) the node.  
Select the **Network** icon.



- A dialog window is displayed, allowing different options for scanning over a Local Area Network (LAN).  
Select **Address / Host Name**, enter the IP address of your node, and then select **Scan**.

The screenshot shows a 'Network' configuration window with three radio button options: 'Subnet', 'Network ID/Mask', and 'Address / Host Name'. The 'Address / Host Name' option is selected and highlighted with a red oval. The 'Subnet' dropdown menu is set to '192.168.0.0 / 255.255.252.0'. The 'Network ID/Mask' field contains four dots separated by periods and a slash. Below the options are three buttons: 'Scan', 'Reselect Type', and 'Cancel'.

The node's icon will be added to the Nodes list.

If the node's icon is not added to the Nodes list, check the following possible causes:

- Make sure the IP address was entered correctly.
- Make sure the node is powered on and connected to the network.  
You should be able to use the Workbench computer's ping command to successfully ping the node using its IP address.
- Make sure the runtime processes are started on the node.
- Make sure the node has its operating system firewall configured to allow the Workbench to access the runtime processes.

## Scanning for a node by host name

To scan for a node using its host name, follow the same steps as above for an IP address, but use the host name instead.

The host name must be able to be resolved by a DNS server for the computer running the Workbench.

## Specifying an alternative port number

The Workbench, by default, uses port number 4012 over TCP when connecting to a node. While not common, it is possible to change the TCP port number that a node will use by specifying a different port number in the node's property file.

If there is a node that is configured to use port number different from the default port 4012, the Workbench scan option for **Address / Host Name** can be used to specify the alternative port number.

For example, if there is a node that cannot use the default port 4012, you would change that node's `../dwcore/dwcore.properties` file to have a different port for this statement:

```
listener.2=Private/0.0.0.0:4012/SECURE
```

In the Workbench **Address / Host Name** parameter, you would add the alternative port number when scanning for that node. For example: `192.168.2.208:4019`.

## Scanning a subnet for nodes

You can scan for multiple nodes at once by using the **Network ID/Mask** option. The Network ID portion of the input field and the Subnet Mask portion is used to identify the subnet to scan. For example:

Network

**Subnet:** 192.168.0.0 / 255.255.252.0

**Network ID/Mask:** . . . / . . .

**Address / Host Name:** \_\_\_\_\_

Scan Reselect Type Cancel

The Workbench will scan for all nodes within the subnet and add each node found to the Nodes list.

After you have entered and scanned a subnet using a **Network ID/Mask** value, the value is saved and is available in the selection list for the **Subnet** option.

## IIoTA industrial IoT Platform: Parts of the Workbench window

### Workbench Nodes list view

When the Workbench **Nodes** icon is selected, the Workbench window displays a Nodes list in the left-hand pane and a Nodes table in the right-hand pane, for example:

Title bar  
Menu bar

Node list

Filter

Left hand pane Right hand pane

Name	Address	Status	Description
Acme Products Enterprise Gateway	192.168.2.96	Running	
Acme Products Line 1	192.168.2.55	Running	
Acme Products Line 2	192.168.2.107	Running	
Acme Products Line 3	192.168.2.185	Running	
Acme Products Line 4	192.168.2.208	Running	

The parts of the Workbench window for this view are described in the following sections.

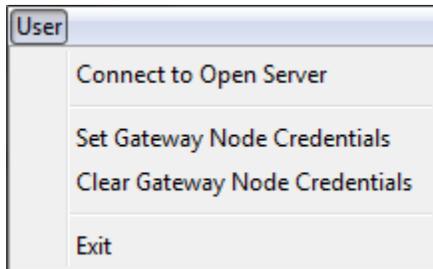
### Title bar

The Title bar displays the version of the Workbench and the user name used when the user logged on to the Workbench.

### Menu bar

The menu bar has the following options.

- User



The User option has the following options:

- **Set Gateway Node Credentials** - used to set the default user id and passwords to use when connecting to nodes. If the credentials are not set, then the default user id of "admin" and the password of "admin" are used.
- **Clear Gateway Node Credentials** - used to clear the default user id and password to use when connecting to nodes.
- **Exit** - used to shut down the Workbench. The nodes that are currently in the Nodes list are saved and used the next time the Workbench is started.

### Left hand pane

The left-hand pane displays the Nodes list of all the nodes that have been connected to by the Workbench. If a node's icon is greyed out, you can reconnect to it by selecting the icon.

A node can be expanded by selecting its [+] icon, which will display the features available on that node.

Each node in the Nodes list shows a node icon that is specific to the product installed on the node, the node name, or the IP address used to connect to the node, and if applicable the highest-level active alert state or the attention bit for the node. The alert state or attention bit is displayed as a small icon overlaid on top of the node icon.

### Right hand pane

The right-hand pane displays a Nodes table with additional information for each node.

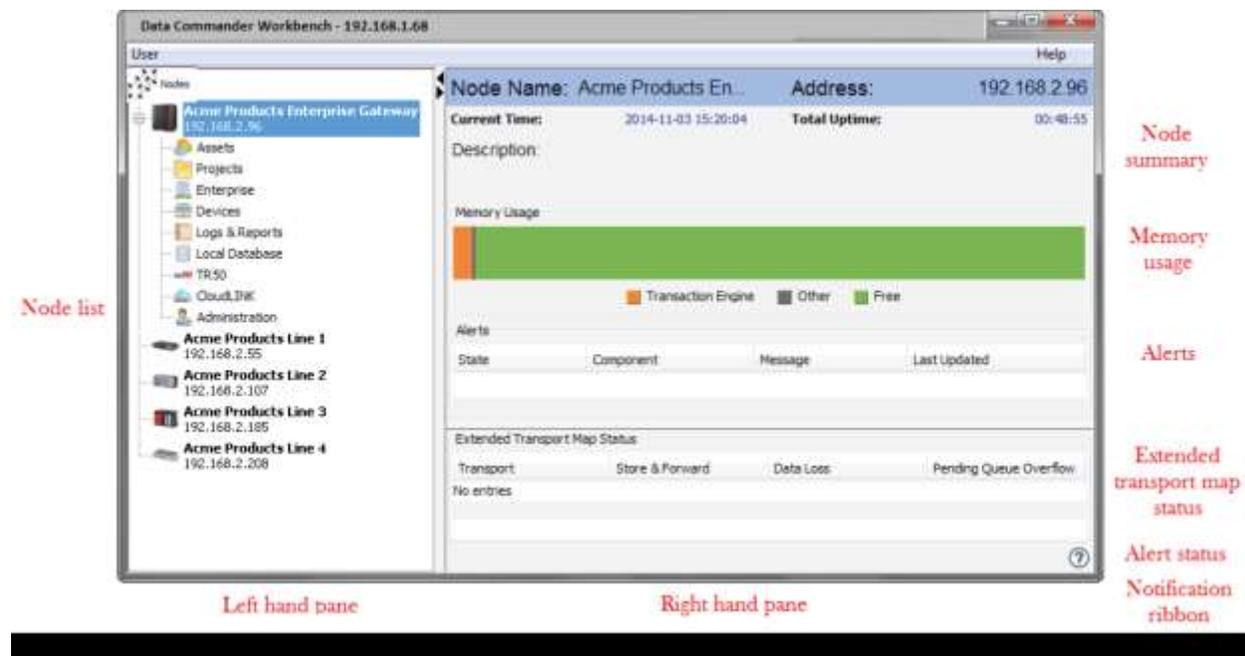
The columns are sortable, by selecting the column heading. The columns can also be reordered by dragging and dropping to a new position in the table.

- The **Filter** option allows you to input filter parameters to reduce the number of nodes displayed in the Nodes table.

Select the **Modify** button to enter the filter parameters. The **Clear** button will remove all filter parameters and display all the nodes in the Nodes list.

## Selected node right hand pane view

When a node's icon is selected, the Workbench window displays details about the node in the right-hand pane, for example:



The parts of the Workbench window for this view are described in the following sections. The Title bar and Menu bar are the same as in the Nodes list view.

### Node summary information

The node summary section includes the node name and description, the node's current time (considering the node's time zone) or the node's IP address, and the node's total uptime (the amount of time since the node was last started).

### Memory Usage section

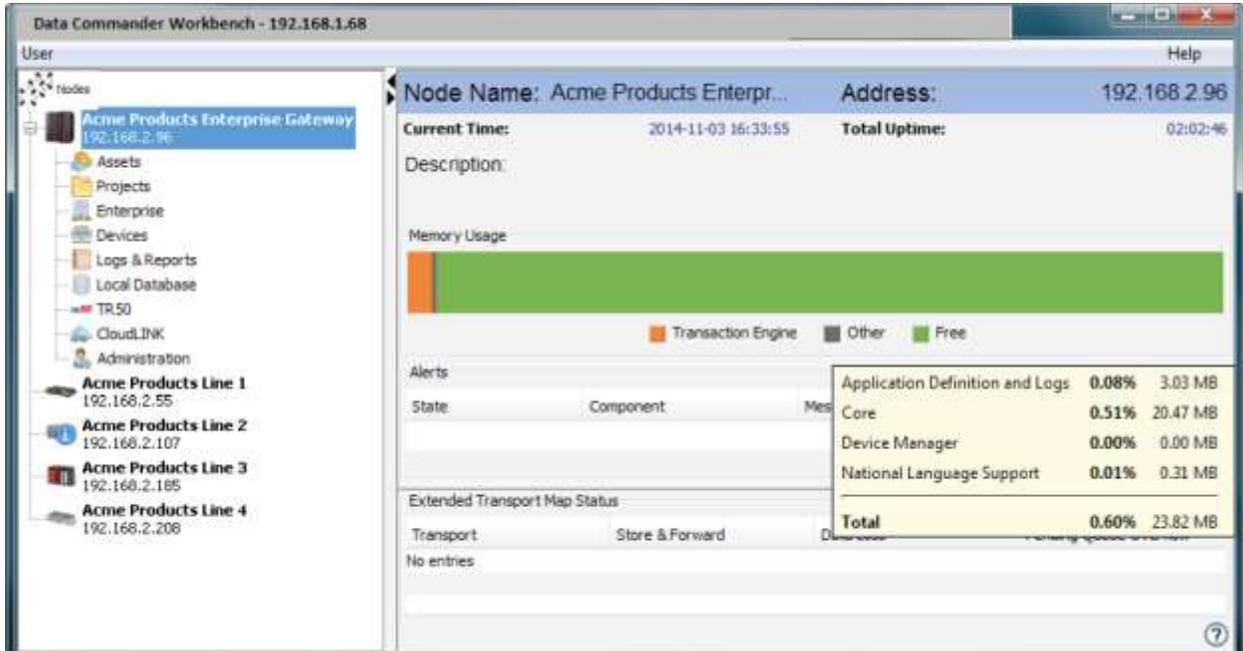
The Memory Usage section displays the memory (RAM) usage of the runtime components and the amount of currently available free memory.

This information is calculated from the memory available when the runtime is started, with those values being the baseline. The percentages are calculated relative to that baseline.

This information can be used to understand the relative increase in system resources as devices, triggers, transports and other runtime applications are defined and started.

Hovering the mouse over the colored bar chart or the legend will display a popup with additional details.

For example:



### Alerts section

The Alerts section displays a list of currently active alerts. For more information on the Alerts feature, see Node alert states.

### Extended Transport Map Status section

The Extended Transport Map Status section displays a list of the transport that have been or are in a Store and Forward state. Nodes that do not have the Enterprise connectivity feature will not have this section. For more information about the Transaction Server and the Store and Forward feature, see Enterprise connectivity.

### Alert Status level

The highest-level active alert is displayed in the Alert Status field. This field is displayed in the right-hand pane of all windows for the selected node.

### Notification ribbon

A colored notification ribbon will display the highest-level active notification for the selected node, if there is an active notification. The notifications include:

Color	Description
Red	Emergency licenses in use

Blue	System execution suspended
Aqua	Safe mode
Green	Evaluation license in use
Yellow	Not for resale licenses in use

## Windows XP Professional and Windows 7, 8, 10 and the Workbench look and feel

The visual style of the Workbench will depend on the windows and buttons theme you selected for your computer. The images captured for this information site use a combination of Windows XP, Windows 7, and Windows Classic styles.

The images captured may show differences in the icons, colors or tabs displayed compared to the Workbench you are using. This might be based on the version of the Workbench you are using, or it might be based on the features available in the nodes the Workbench is accessing. Some products do not have all the features described in this information site.

## IIoTA industrial IoT Platform: Managing the Workbench Nodes list

The Workbench Node List allows you to select nodes and access the nodes settings and application definitions.

As you work with several nodes, both on the local network and at remote locations, you can manage the Node List maintained by the Workbench and perform some tasks on the nodes in the list.

### The Node List icon's tasks

You can perform Node List tasks by selecting the Nodes icon, then right click to display its pop-up menu:



## Scanning for local and remote nodes

The **Scan** option allows you to scan and connect to nodes. This can be for nodes in your local network environment. For more information, see [Scanning for local or remote nodes](#).

## Resetting the Nodes list

The **Reset Node List** option will remove all nodes from the Node List. You can use this option to remove all nodes from the list and then **Scan** for the specific nodes you are interested in working with. Removing nodes from the Workbench's Node List does not change the status of node's run time processes.

## Sorting the Nodes list

The **Sort by Name** and **Sort by Address** options sort the Node List by the node names or by the IP address used to connect to the node.

## Exporting and Importing the Node List

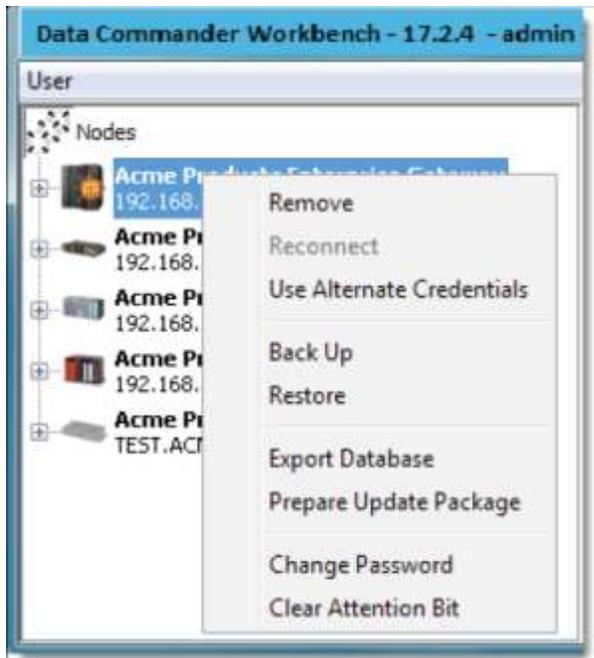
The **Export Node List** option will save information about the nodes currently in the Node List to a text file.

The **Import Node List** option will read the information from a previously saved Node List export file and add those nodes to the current Node List.

These options can be used if you have different node lists that you need to work with, where scanning or removing individual nodes becomes unwieldy.

## A selected node icon's tasks

You can perform node specific tasks by selecting a node's icon, then right click to display its pop-up menu. The options available depend on the product installed on that node. For example:



## Removing a node from the Node List

The **Remove** option will remove the node from the Node List. It can be added back to the Nodes list by using the **Scan** option.

Removing nodes from the Workbench's Node List does not change the status of node's run time processes.

## Reconnecting to a node

If a node's icon is greyed out, then the Workbench has ended its connection with that node.

The **Reconnect** option will establish a new connection from the Workbench to the node.

## Using alternate log on credentials

The Workbench itself does not have a concept of a user logging on before being authorized to access features in the Workbench.

Rather the Workbench uses the user name and password credentials when it connects to every node. The user names, passwords and access control configuration is a feature of each node.

This allows flexibility, if needed, in defining separate access control configuration for each node.

If a node has a user name and password configured that is different from the user name and password credentials set in the Workbench, then the **Use Alternate Credentials** option can be used to specify the user name and password when connecting to that node.

When the Workbench attempts to connect to a node with credentials that are not recognized, a red circle and diagonal line "No" symbol is overlaid on the node's icon. For example:



## Backing up and restoring node's settings and definitions

The **Back Up** and **Restore** options are used to save (back up) and retrieve (restore) a node's system configuration settings and application definitions.

For more information, see [Backing up and Restoring a node's configuration](#).

## Exporting a node's database

### Preparing an update package

A software update package can be created from the resources on a node.

### Changing the current user's password

The password for the current user can be changed on a node by using the **Change Password** option. As described in previous sections, the user name and password are characteristics of each node and are not characteristics of the Workbench.

### Clearing the Attention bit

The attention bit icon overlaid on the node's icon can be removed by using the **Clear Attention Bit** option. The attention bit is separate from the **Alert** feature.

# IIoTA industrial IoT Platform: Backing up and Restoring a node's configuration

## Backing up a node's settings and definitions

The **Back Up** option will save all the node's system configuration settings and application definitions into a file. There may be a question to include the node's network settings, depending on the product installed on the node.

The **Back Up** option is in the Workbench's pop-up menu for a node. Using the Workbench, right click the node's icon in the Node List to display the node's pop-up menu.

This node backup file can then be saved to a file on the computer running the Workbench.

## Node Back Ups

Doing a node **Back Up** periodically during your application development and debug cycle is an important habit to have.

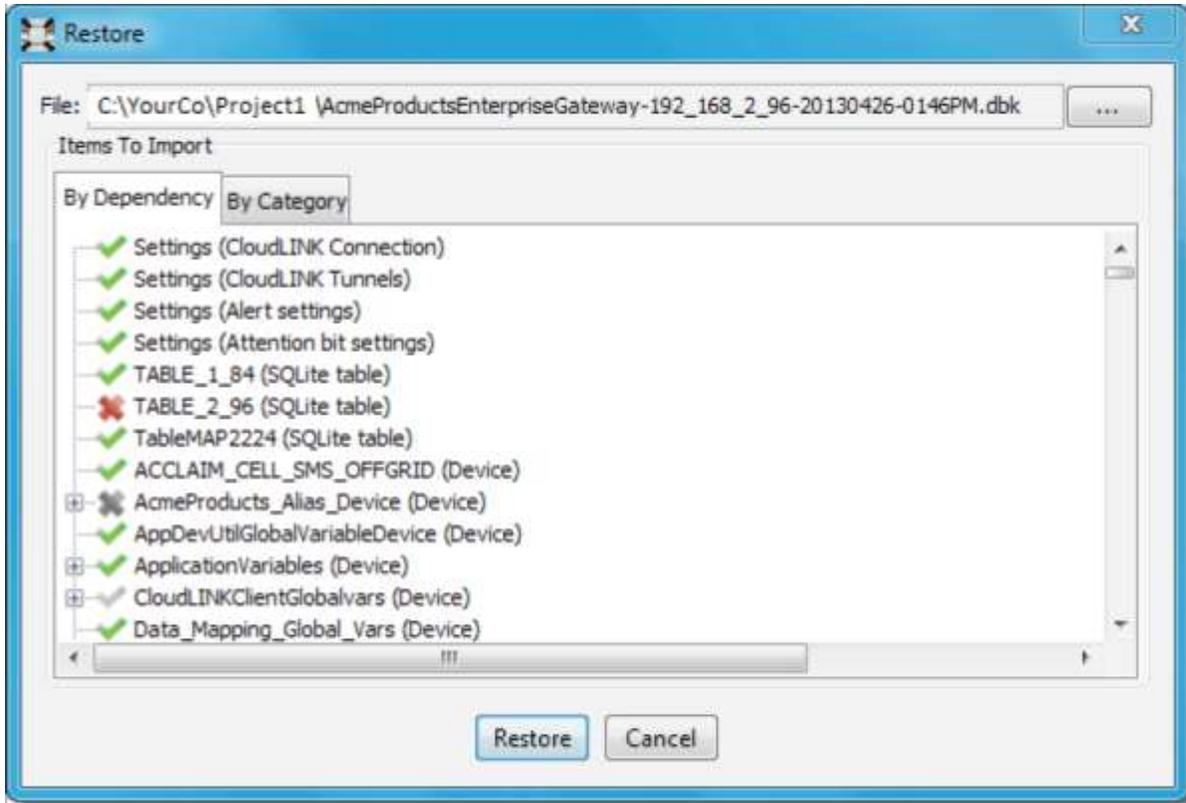
The backup files can be named something that is meaningful to the stage of your development cycle, or the default file name of the node name appended with the IP address and a date and time serializer will help with the backup file management.

The node **Back Up** process uses all the node's configuration settings and application definitions. It is like the item **Export** function.

## Restoring a nodes backed up settings and definitions

The **Restore** option will read a previously saved node back up file and display the settings and definitions in a list that allows you to select **All** or specific items to restore to the node.

The **Restore** option is in the Workbench's pop-up menu for a node. Using the Workbench, right click the node's icon in the Node List to display the node's pop-up menu.



The two tabs available on the **Node Restore** window present the saved information in difference ways:

1. The **By Dependency** tab displays the items in a dependency list that can be expanded and collapsed through the dependencies of an item. The individual items and its dependencies can be selected, represented by a green check mark, or not selected, represented by a red X. A grey check mark or X indicates that an item's dependencies are partially met or the item is not selected, but one of its dependencies is selected.

Examples of dependencies can best be seen by doing a node back up and then a node restore and selecting different items in the tab.

For example:

- A trigger is dependent on the project that it resides in.
  - A trigger is dependent on a device if the trigger references a variable in that device.
  - A trigger is dependent on a transport map that it references.
  - A transport map is dependent on the transport that it references.
  - And so forth.
2. The **By Category** tab displays the items by the type (or category) of the items. If the **By Category** tab is used to select items, each item's dependencies are not automatically selected as they are with the **By Dependency** tab selections.

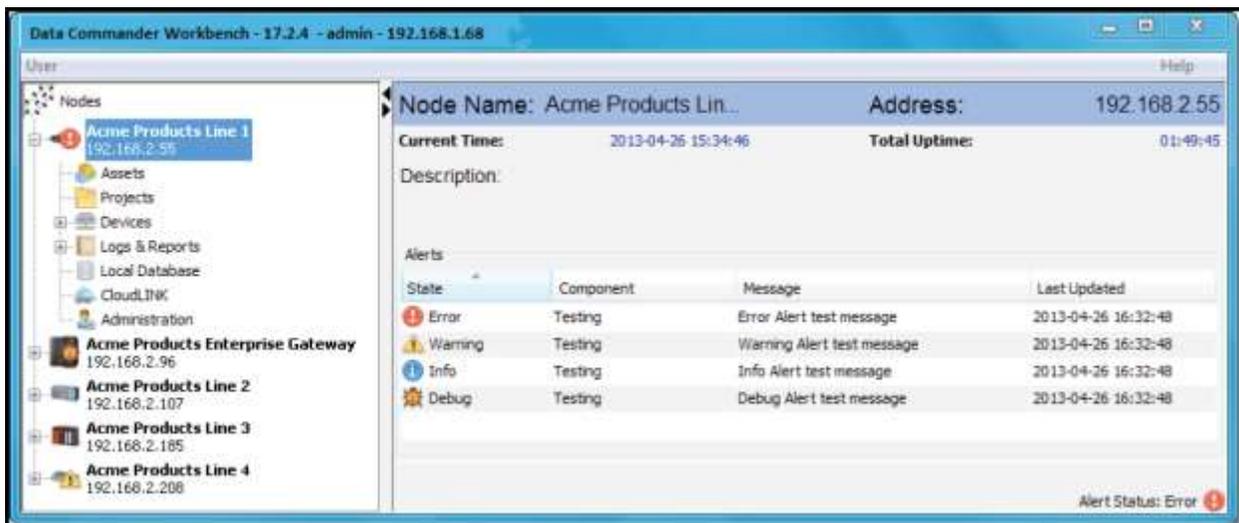
You can right click on any item to display a pop-up menu that has the options to **Select All** or **Deselect All**.

After the restore completes, a window will display any errors encountered during the restore process.

## IIoTA industrial IoT Platform: Node alert states

Each node icon in the Nodes list will be overlaid with an icon for the highest-level active alert or with an icon for the attention bit.

When a node is selected from the Nodes list, the right-hand pane of the Workbench window displays an **Alerts** section. For example:



In this example, the selected node has an error alert icon overlaid on its node icon and the **Alerts** section displays a table of the active alerts for the selected node (showing an example of each alert state).

The columns in the Alerts table are as follows:

Column	Description
<b>State</b>	The alert state with its associated icon. The states in order of priority level are:  <b>Error</b> - An error usually requiring action to correct.

	<p><b>Warning</b> - A potential problem that might be important or might not need immediate action.</p> <p><b>Info</b> - An informational item about the system or the application.</p> <p><b>Debug</b> - A debug item generated by the application.</p>
<b>Component</b>	The component that is the source of the alert. This could be a system component, or an application defined component.
<b>Message</b>	The message text associated with the alert.
<b>Last Updated</b>	The date and time the alert was last updated.

Alerts can be generated by system components and generated by an application using a trigger action.

The attention bit is a separate feature from the Alerts feature. The attention bit icon is the same as the warning alert icon.

Read more: <https://html.com/tags/comment-tag/#ixzz4yQLolMHh>.

For more information about generating an alert from an application, see Set Alert.

## IIoTA industrial IoT Platform: Exiting safe mode

A node will enter into a safe mode when it detects that it is being restarted within five minutes of its last start up time.

Safe mode may be useful when developing and debugging your application and you are experiencing problems with your triggers and device access that does not allow you to access the node using the Workbench. You can force the node to enter safe mode by restarting the node (or by powering off and then on) within the five-minute period since its last start up time.

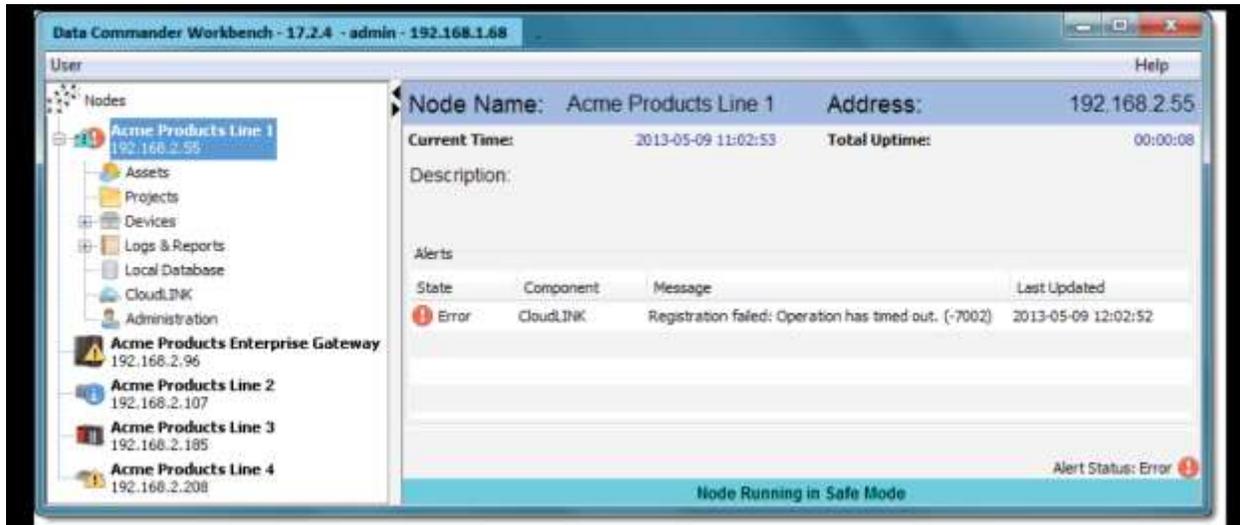
Safe mode offers a one-minute recovery period during which you can troubleshoot triggers, data mappings, devices, and other items that require intervention. While the node is in safe mode, triggers will not execute. After the one-minute time period expires, the node will automatically return to normal running mode.

If you require additional time beyond the one-minute recovery period, you can disable system execution by using the System administration Diagnostics System Execution function to toggle to Suspended mode.

You will be alerted that a node has entered into safe mode by the following:

- The node's icon the Nodes list will be overlaid with a safe mode icon .

- The bottom of the Workbench right hand pane will show a aqua colored notification ribbon indicating the node has entered safe mode.



## Manually exiting safe mode

You can manually exit safe mode using the node's pop-up menu.

From the Workbench left hand pane, right-click the node to display its pop-up menu, and then select **Exit Safe Mode**.



The safe mode icon is removed from the node and the System Execution mode is changed to Running.

You can also exit safe mode by using the System administration Diagnostics System Execution function to toggle to Running mode.

# IIoTA industrial IoT Platform: Keyboard Navigation

On a panel with buttons, pressing the **Alt** key will show an underlined character on each button. Pressing that character will cause that button's function to be executed. For example, if the button shows "**Start**", pressing **Alt-s** would perform the start action.

The following keys allow for additional navigation in the workbench:

- **Tab** - move forward through fields/buttons. Shift-TAB to go backward
- **Enter** - Enter value or perform edit/open action depending upon context
- **Delete** - Perform delete action for selected items
- **Control-Tab** - cycle through tabbed panels and buttons. Control-Shift-Tab reverse cycle
- **Home/End** - go to first/last row in table
- **F5** - Refresh list window
- **Shift-F10** - View context menu
- **F11** - Maximize list subpanel when viewing a panel with a list and preview. F11 again to restore split view

## IIoTA industrial IoT Platform: Cross References

In the following panels you can view a list of what items reference the selected object as well as what items the selected object references.

- Projects
- Triggers (Project tab for an individual project)
- Transports
- Transport Maps
- Listeners
- Listener Maps
- Devices
- Variables
- Data Mapping
- Variable Groups

Selecting one or more items in the list window, right clicking and selected "References" will display a list of tables displaying Name, Type, and Reference.

The values in the reference column:

- **Outgoing:** The selected items reference this item. (e.g., selecting references on a trigger, items such as variables and transport maps which the trigger calls)
- **Incoming:** The selected items have references to this item (e.g., selecting references on a transport, items such as transport maps which use the transport).

Items in **bold** can be directly viewed. Double clicking on one or selecting one and clicking the "View" button will bring up the list view of that item type with the item selected.

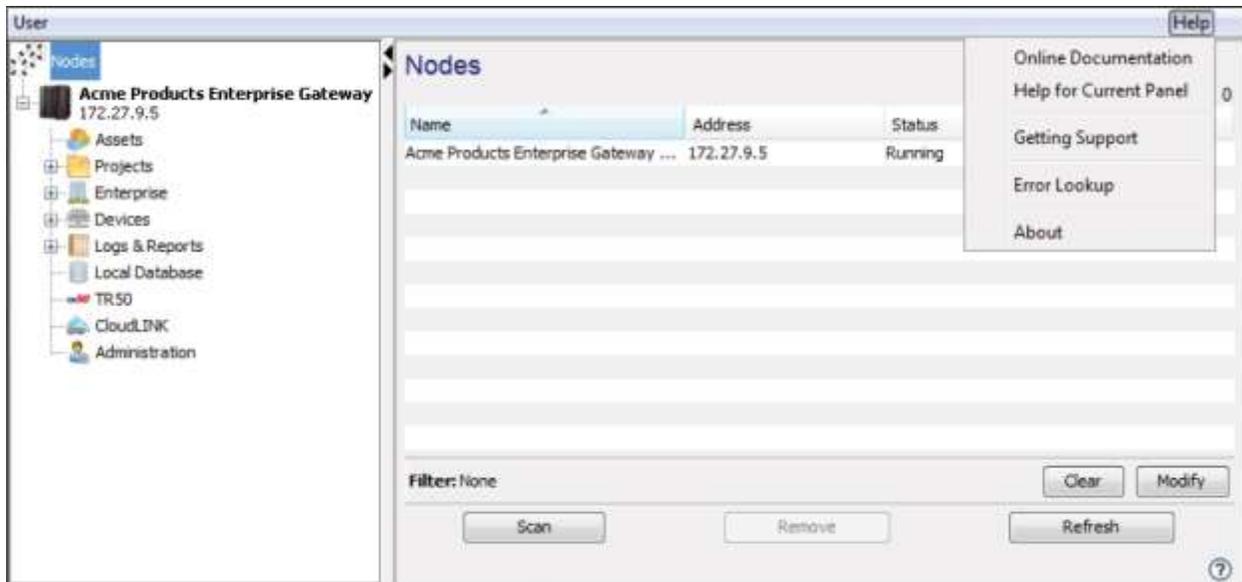
## IIoTA industrial IoT Platform: Viewing product documentation

The Workbench provides links to view the product documentation.

When the Workbench links to the product documentation, it will display the requested page in a new web browser tab.

There are several ways to link to the product documentation:

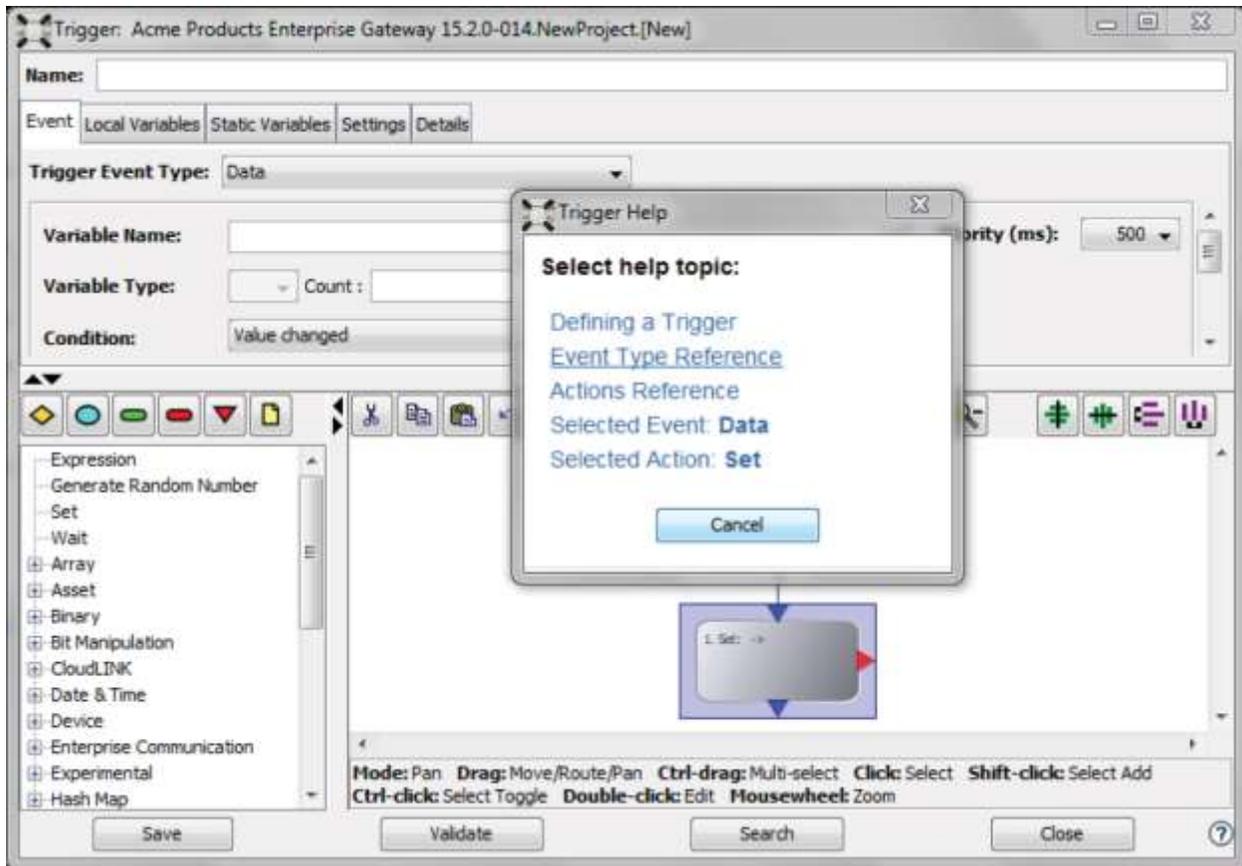
- The **Help -> Online Documentation** option will display the home page for the product documentation.
- To display the documentation page for the Workbench's current panel:
  - Select the **Help -> Help for Current Panel** option
  - Select the "help" icon (the question mark in a circle) in the lower right corner
  - Press the **F1** key.



## Trigger editor help

The Workbench's trigger editor will display additional options when the help icon or F1 is selected.

A dialog will be displayed with links to the trigger definition, trigger event reference and trigger action reference documentation pages. In addition, links for the current trigger event type and current action are also displayed. For example, a **Data** trigger event type with a **Set** action are the current event type and action when the help icon is selected:



## Navigating the product documentation

Once the production documentation page is displayed in a web browser, you can navigate all the topics by using the left-hand navigation pane.

To find the content you need, expand and then browse the contents of the left-hand navigation pane, or use the Search box at the top left of any page.

# IIoTA industrial IoT Platform: Projects and triggers

## Overview

Your application logic is defined in a *trigger*.

The available trigger event types and trigger actions on a node depends on the product installed and the packages or extensions installed on the node.

## Highlights

The first several sections of this **Projects and Triggers** guide describe the general concepts and tasks that apply to projects and triggers.

The reference sections: Trigger event type reference and Trigger actions reference provide the reference information along with concepts and examples of the available trigger event types and trigger actions.

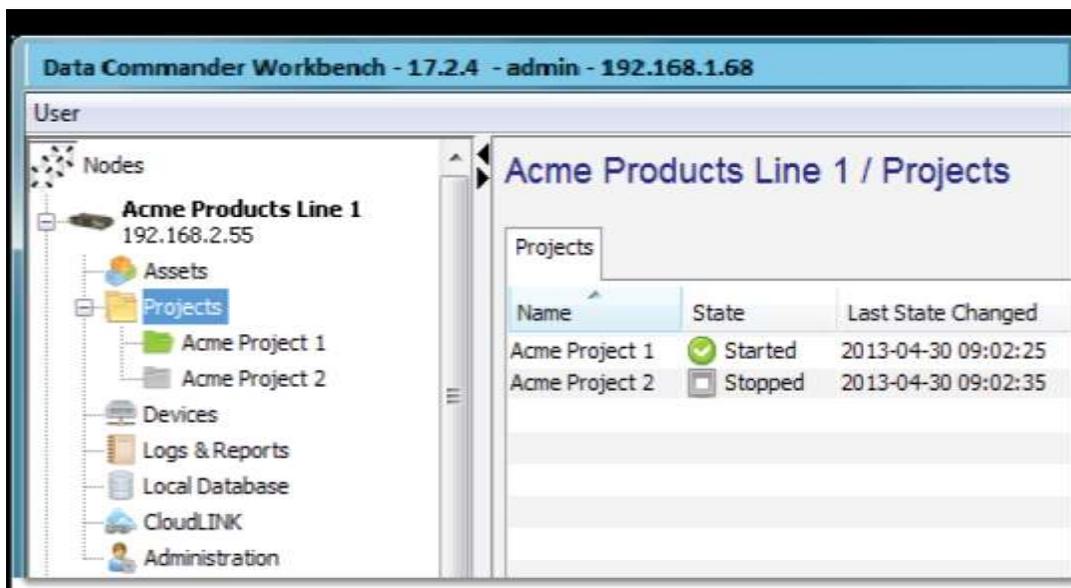
## IIoTA industrial IoT Platform: Projects

### About projects

Each trigger resides in, or belongs to, a project. A project is a container item that is used to organize and control triggers. You can create as many projects as needed for a node and define the triggers needed within each project. The organization of triggers within a project can be based on the physical characteristics of the solution (for example: Line 1, Line 2, etc.) or based on the logical functions that the triggers provide (for example: device access, database access, utility functions, etc.).

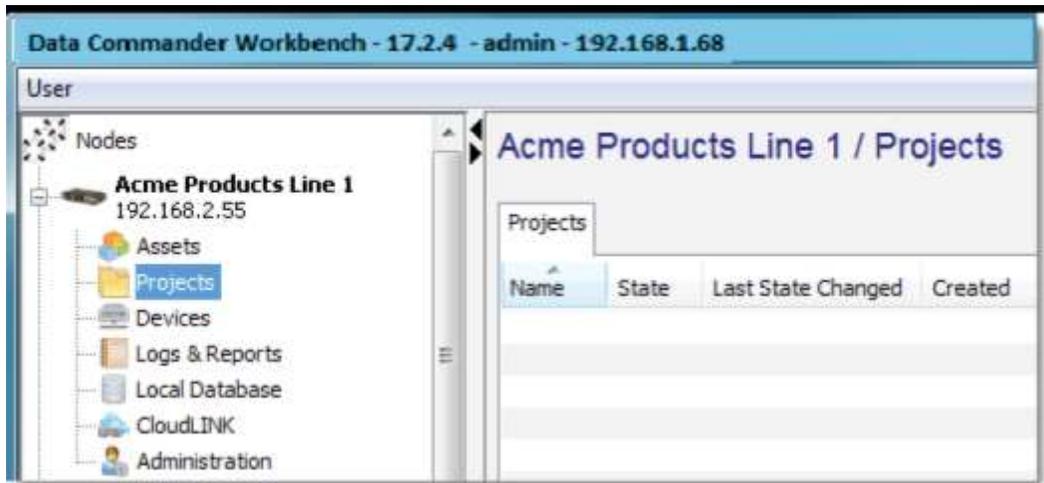
The organization of projects and triggers does not limit the availability or access of the triggers, it is just a mechanism to organize the application logic in a manner that makes sense to the application developer.

When you select the **Projects** icon for a node, the Projects window is displayed in the right-hand pane with the list of projects that have been defined for that node:



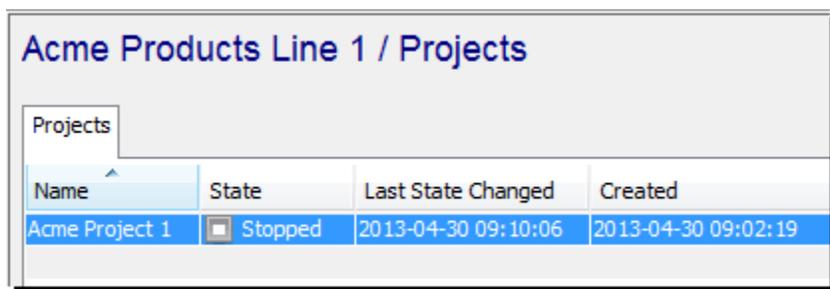
## Defining a project

1. From the Workbench left pane, select and then expand the node that you want to add a project to.  
An expanded tree view of the selected node appears.
2. Select the **Projects** icon.  
The **Projects** window appears as the right-hand pane.

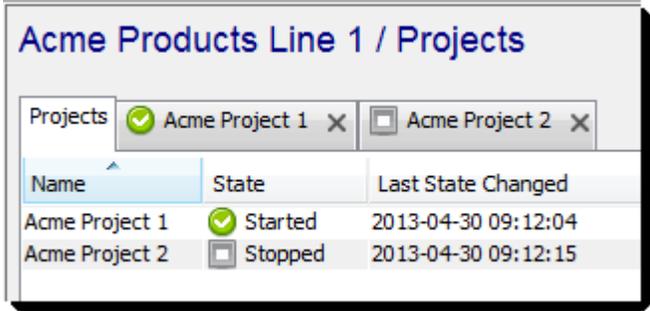


The list of projects will be empty if no projects have been defined.

3. From the bottom of the Projects tab, select **New**.  
The Create Project window appears.
4. Enter a name for the project, an optional description, and then select **OK**. A project name can be up to 64 characters in length and can include letters, numbers, and the underscore character. Spaces are allowed.  
The project name is added to the **Projects** tab on the Workbench right-hand pane.



The Projects tab has a table format with these columns:

Column	Description									
<b>Name</b>	The name of the project.									
<b>State</b>	<p>Projects have a state that is separate from the state of its triggers. The state of a project is:</p> <ul style="list-style-type: none"> <li>• <b>Started</b> - The project is started. A project must be started in order for its triggers to be loaded and available to be executed</li> <li>• <b>Stopped</b> - The project is stopped. All of the triggers in the project will have a status of unloaded and are not available to be executed.</li> </ul> <p>This example shows two projects. For the <b>State</b> column, in addition to the text <b>Started</b> and <b>Stopped</b> the green check mark icon on <b>Acme Project 1</b> indicates the project is started.</p> <p>Likewise, the gray square icon on <b>Acme Project 2</b> indicates the project is stopped.</p>  <table border="1" data-bbox="402 852 1052 1163"> <thead> <tr> <th>Name</th> <th>State</th> <th>Last State Changed</th> </tr> </thead> <tbody> <tr> <td>Acme Project 1</td> <td>Started</td> <td>2013-04-30 09:12:04</td> </tr> <tr> <td>Acme Project 2</td> <td>Stopped</td> <td>2013-04-30 09:12:15</td> </tr> </tbody> </table>	Name	State	Last State Changed	Acme Project 1	Started	2013-04-30 09:12:04	Acme Project 2	Stopped	2013-04-30 09:12:15
Name	State	Last State Changed								
Acme Project 1	Started	2013-04-30 09:12:04								
Acme Project 2	Stopped	2013-04-30 09:12:15								
<b>Last State Changed</b>	Displays the date and time the project was last started or stopped.									
<b>Created</b>	Displays the date and time the project was originally defined.									

## Project Overview

Below the project list table is a panel showing an overview of the project.

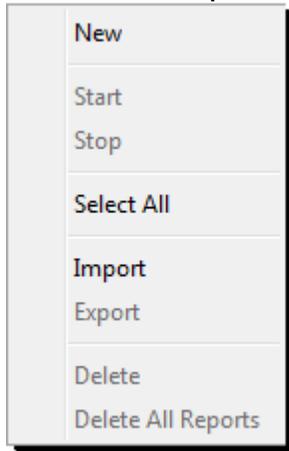
This includes an editable description as well as a list of the triggers in the project and a brief summary of their status.

AcmeProject1				
Description				
This is the description for myproject				Edit Description
Trigger	State	Last Triggered	Successes	Failures
Trigger1	Started	2017-07-12 09:47:30		0
Trigger2	Started	2017-07-12 09:46:31		1

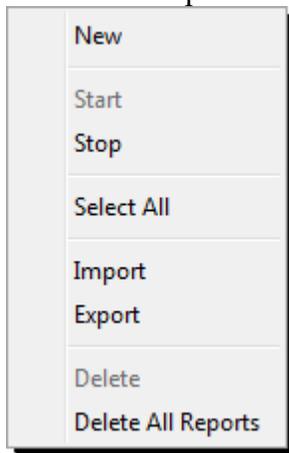
## Managing projects

When the **Projects** window is displayed as the right-hand pane, a **Projects** tab is always displayed as the first tab in the window. The projects tab displays all the defined projects, along with their state, the date and time last modified and the date and time the project was defined.

If you right-click in the empty part of the **Projects** tab list (not on a project row), a pop-up menu with available options is displayed:



If you right-click on a project in the list, a pop-up menu with available options is displayed, some of which are specific to the selected project:



The options from either of these pop-up menus are:

Option	Description
<b>New</b>	Defines a new project. Alternatively, the <b>New</b> button at the bottom of the Projects window can be used.
<b>Start</b>	Starts the selected project. Alternatively, the <b>Start</b> button at the bottom of the projects window can be used. This option is only available if the selected project is in a stopped state.
<b>Stop</b>	Stops the selected project. Alternatively, the <b>Stop</b> button at the bottom of the projects window can be used. This option is only available if the selected project is in a started state.
<b>Select All</b>	Selects all projects
<b>Import</b>	Displays the Import window, allowing the selection of a previously exported export file.
<b>Export</b>	Displays the export window, allowing the selection of the items in the project (triggers) and any of the triggers' dependencies to be exported.
<b>Delete</b>	Deletes the selected project and all of the project's triggers. Alternatively, the <b>Delete</b> button at the bottom of the projects window can be used. This option is only available if the selected project is in a stopped state.
<b>Delete All Reports</b>	Deletes all trigger reports generated by triggers in the selected project.

## IIoTA industrial IoT Platform: Defining a trigger

The main concepts of a trigger are:

- The trigger's event type
- The trigger's local variables, static variables, macros and event variables
- The trigger's settings
- The trigger's actions, including the success and failure routes between actions.

When you define a trigger, you name the trigger, identify the event type (Data, Schedule, On-Demand and so forth), define the event parameters, and then configure one or more actions.

As the definition of the trigger progresses, you can use the **Validate** button to check for correctness and completeness of the definition.

When the trigger is saved, it is written to an internal database file on the node.

You can edit, validate and save the trigger definition multiple times as the trigger's application logic is defined. You can use the trigger report feature to generate a report of the trigger's execution to help understand how the trigger's execution progresses through its actions and the actions success and failure routes.

## Defining an example trigger

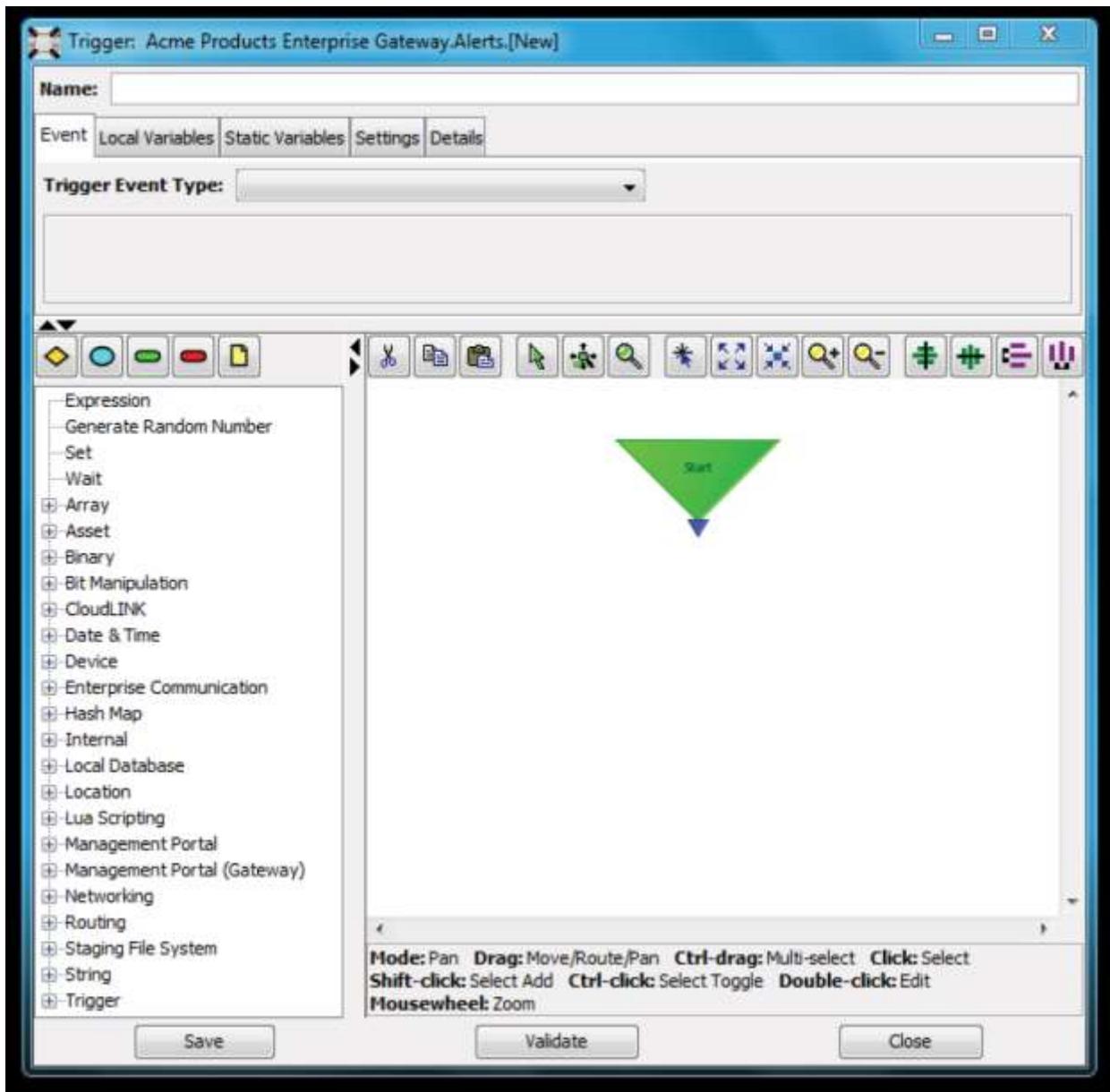
This example will quickly step you through the defining and execution of a sample trigger. The concept and reference information for each of the trigger components is in their specific sections, so all the details and variations will not be covered in this quick example.

### Adding actions to the Canvas

1. Select and expand the node where the trigger will be defined and executed.
2. Select the project where the trigger will reside.  
This example assumes that you will use the project defined in the Projects example.
3. Select the New button at the bottom of the right-hand pane of the project's tab to start the definition of a new trigger.

This example will use the trigger Canvas Editor.

The New trigger window appears.



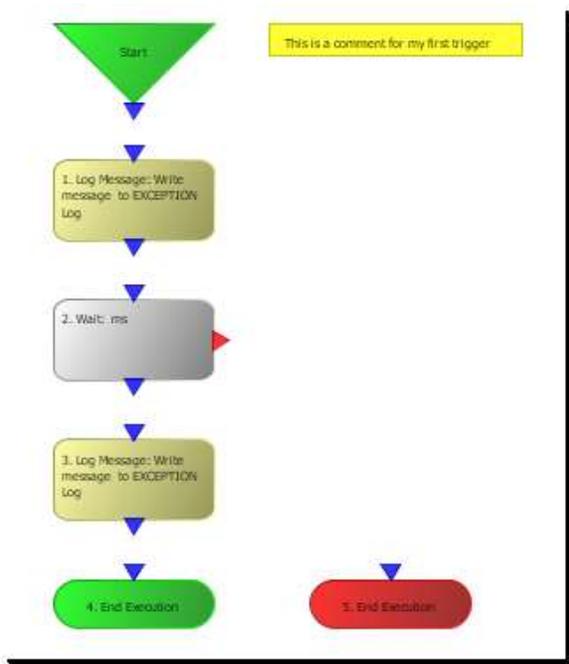
4. Enter *MyFirstTrigger* for the **Name**.
5. Select On-demand for the **Trigger Event Type**.
6. From the left hand pane list of actions, expand the **Internal** category and then select (click once) the **Log Message** action.  
 Select (click once) a location below the **Start** block to place the action.  
 Alternatively, you can drag (click and hold) the action and then position its location and drop (release the mouse button).



7. From the left-hand pane, select the **Wait** action and position it on the Canvas below the **Log Message** action.
8. From the left-hand pane, select the **Log Message** action and position it below the **Wait** action.
9. From the tool bar above the left-hand pane, select the **End Execution (Success)** action and position it on the Canvas below the second **Log Message** action.
10. From the tool bar above the left-hand pane, select the **End Execution (Failure)** action and position it on the Canvas to the right of the **End Execution (Success)** action.
11. From the tool bar above the left-hand pane, select the **Comment** block and position it on the Canvas to the right of the **Start** block.

Enter *This is a comment for my first trigger* into the comment block

The Canvas should look similar to this:



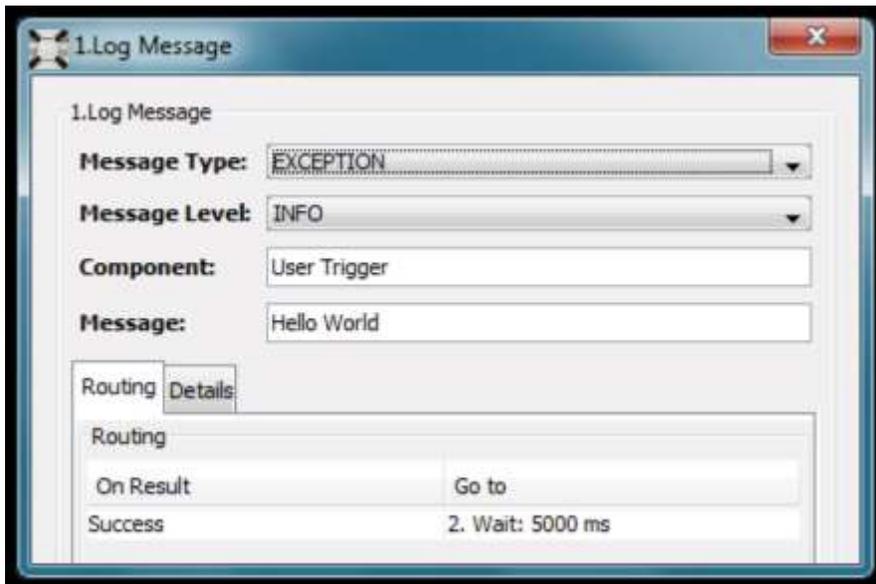
## Entering the parameter details for each action

The two Log Message actions and the Wait action have parameters that need to be entered.

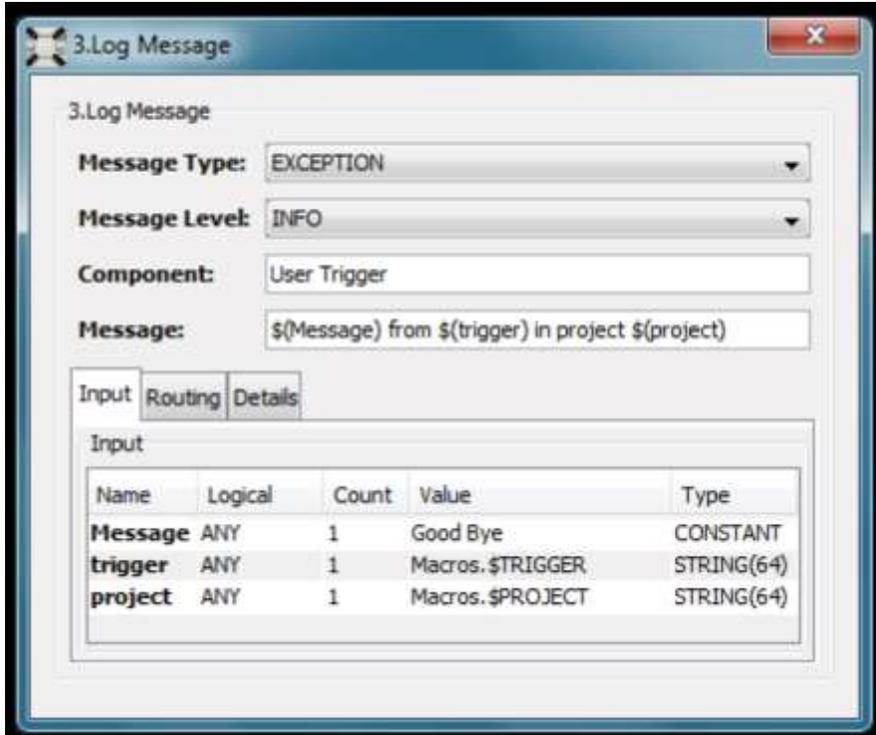
To enter the parameter information for an action, double click the action in the Canvas area.

For the actions, do the following:

1. Double click the first **Log Message** action.  
The action's parameters details are displayed in a window.
2. In the **Message** parameter, delete the \$(Message) text and enter *Hello World*.



3. In the **Details** tab, enter a comment for this action.
4. Close the window by selecting the red close icon (X) in the upper right of the window.
5. Double click the **Wait** action.
6. In the Time to Wait(ms) parameter, enter *5000* for 5000 milliseconds (5 seconds).
7. Close the window by selecting the red close icon (X).
8. Double click the second Log message action.
9. In the **Message** parameter, enter *\$(Message) from trigger \$(trigger) in project \$(project)*.  
Each of the substitution parameters enclosed in a \$( ) becomes an input parameter on the Input tab.
10. Enter variables for each of the input parameters as follows:



11. Close the window by selecting the red close icon (X).

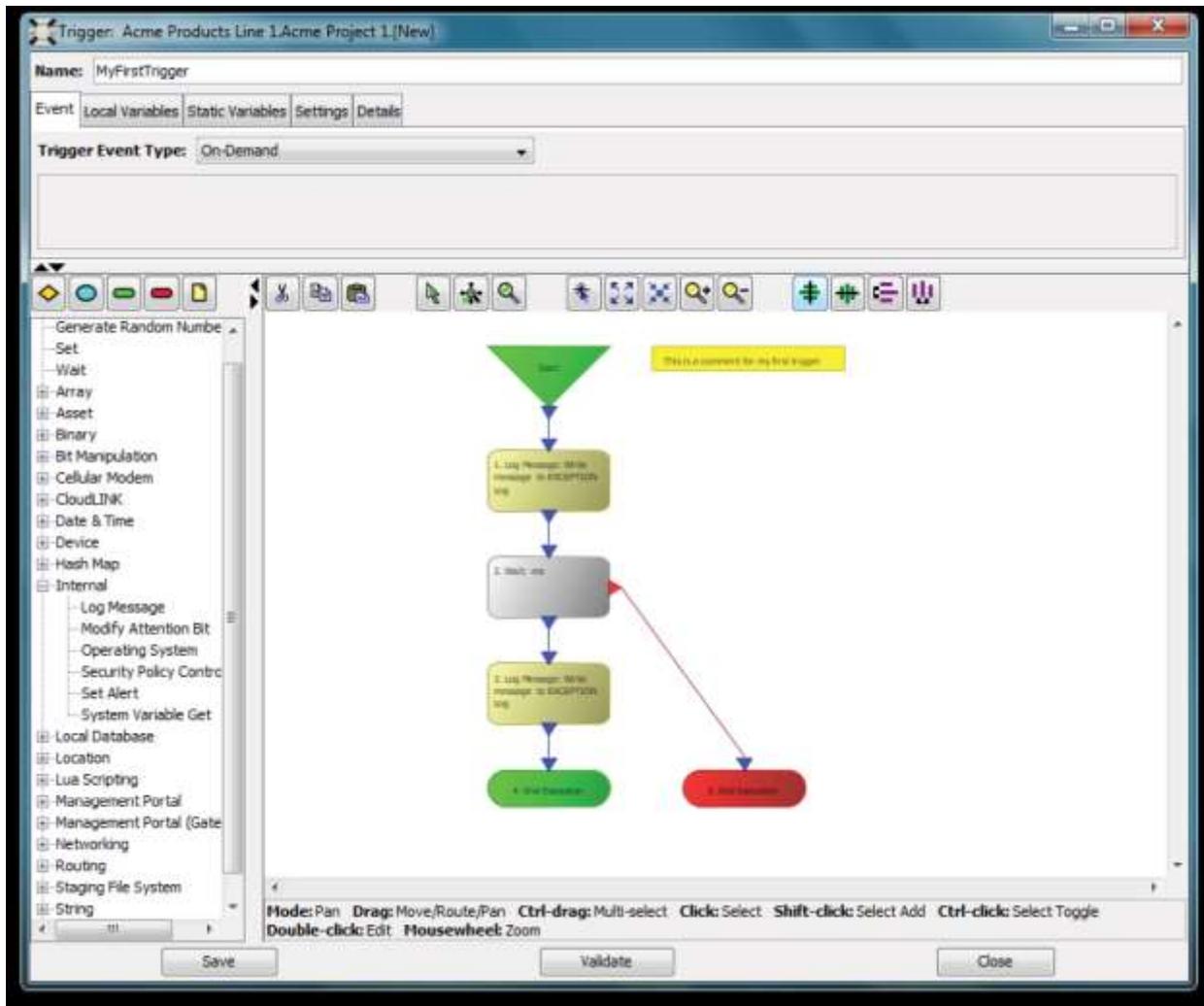
## Specifying the routing for each action

The actions and blocks need routings for each of the input and output ports.

For the routings, do the following:

1. Click and hold the output port at the bottom of the **Start** block, then drag the mouse cursor to the input port at the top of the first **Log Message** action.  
When the input port turns from blue to yellow, release the mouse button.  
A route connection line should be drawn between the **Start** block and the first **Log Message** action.
2. Draw a connection line from the bottom of the first **Log Message** action to the top of the **Wait** action.
3. Draw a connection line from the bottom of the **Wait** action to the top of the second **Log Message** action.
4. Draw a connection line from the bottom of the second **Log Message** action to the top of the **End Execution (Success)** action.
5. Finally draw a connection line from the red side exit of the **Wait** action to the top of the **End Execution (Failure)** action.

The completed Canvas should look similar to this:



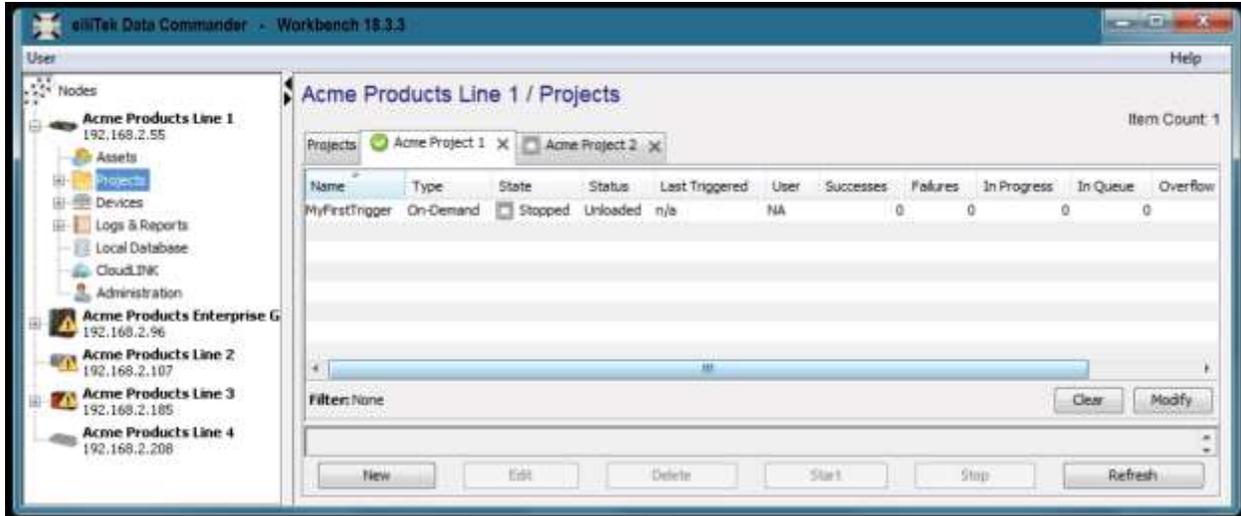
## Validating and saving the trigger

1. Select the **Validate** button at the bottom of the right-hand pane. The validation function will check each action for correctness and completeness.

If errors are found they will be displayed in a window for review and correction. Your trigger should validate successfully, if not review any errors and make the corrections.

2. Select the **Save** button at the bottom of the right-hand pane to save the trigger definition and close the trigger Canvas Editor.

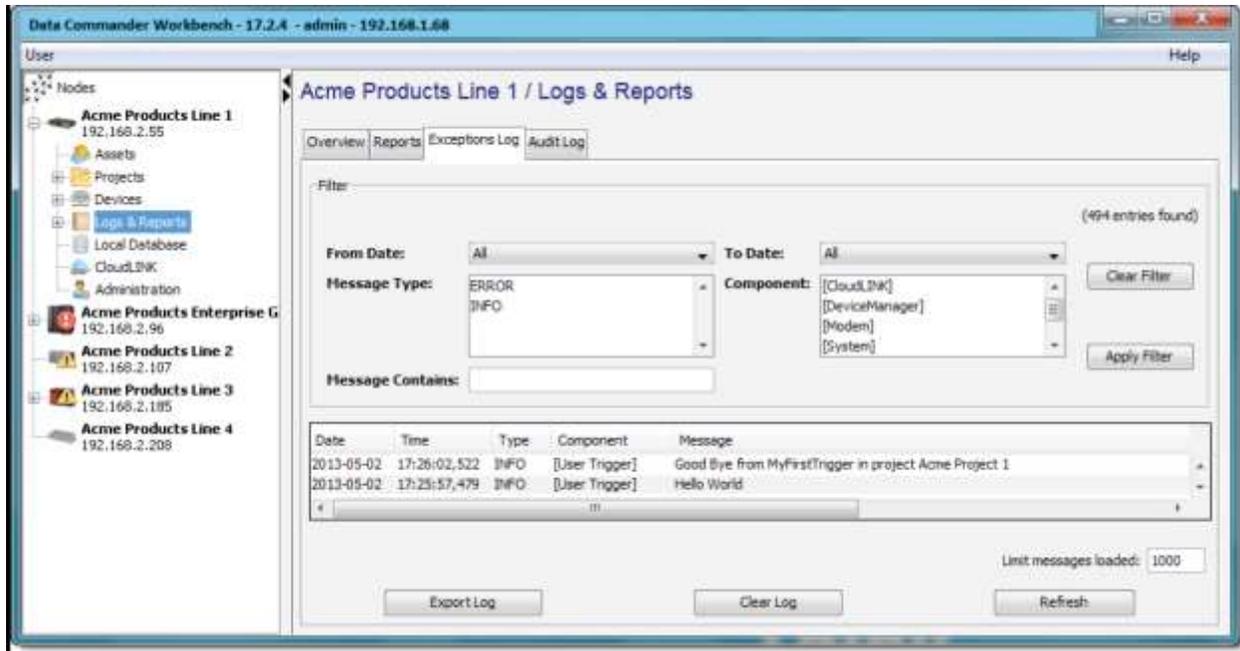
The trigger will be listed in the project tab's list of triggers in a **Stopped** state:



## Starting and executing the trigger

1. Select the trigger and then select the **Start** button.  
Alternatively, you can select the trigger, then right-click to display a pop-up menu and then select the **Start** option.
2. The trigger's state should be **Started** and the trigger's status should be **Loaded**.
3. If the trigger's status is **Unloaded**, then the project needs to be started. To do this, right-click on the project's tab to display a pop-up menu and then select the **Start** option.  
The project should be **Started** and the trigger should be **Started** and **Loaded**.
4. Right-click on the trigger, and then select the **Fire Trigger** option.
5. The trigger (an On-Demand event type) will be executed.  
You may see the **In Progress** count change to 1, and then you should see the **Successes** count change to 1.
6. Select the **Refresh** button a few times until you see the completion of the trigger's execution.
7. You will notice the updates to the **Last Triggered** and **Avg Time (ms)** values.
8. Since this trigger added messages to the Exceptions log, we will view it to see the messages.
9. In the left-hand pane, select the **Logs & Reports** icon for this node.
10. In the right-hand pane select the **Exceptions Log** tab.

The Exceptions Log messages are display, including the two from this trigger:



11. The Audit Log can also be viewed to see the types of auditing messages that are logged by the system when events occur.

### Example trigger summary

- The trigger Canvas Editor was used; the trigger List Editor could have been used instead.
- The trigger's logic was very simple, more complex triggers could include access to device variables, access to enterprise applications, interaction with the M2M Service and much more.

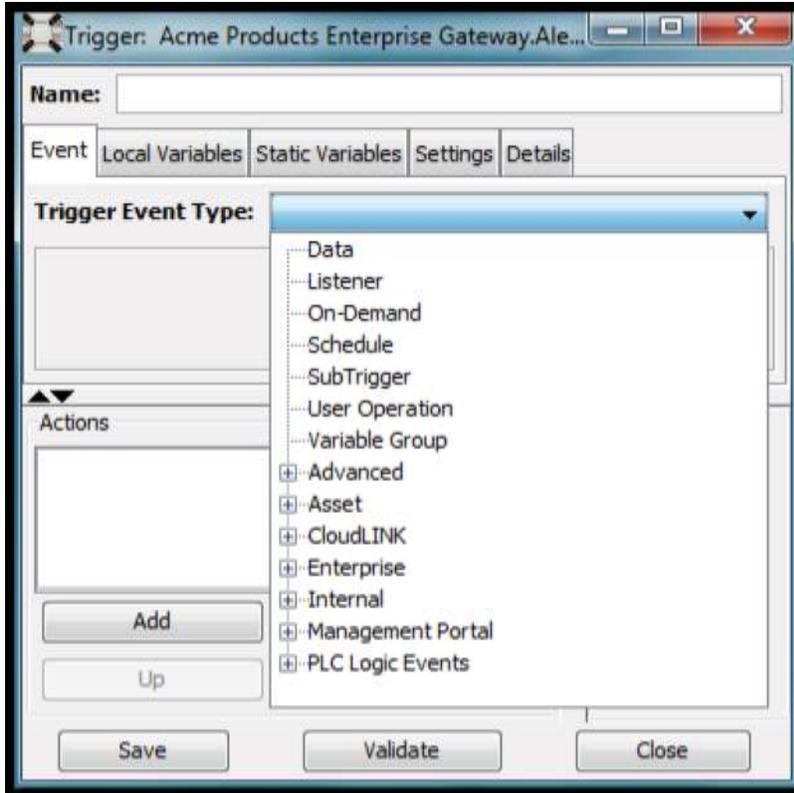
That completes this simple example trigger. The details of each trigger component are described in the following sections.

## IIoTA industrial IoT Platform: Trigger event type

The trigger event type *identifies when* the trigger is executed.

For every trigger's definition, the **Event** tab provides a **Trigger Event Type** drop down-list with options that determine the trigger event type.

Once a specific trigger event type is selected, the event tab becomes active with parameters that accommodate that specific event type.



The event types available in the **Trigger Event Type** list depends on the type of product installed on the Enterprise Gateway, as well as the packages and extensions installed on the node (for example the device drivers).

The trigger event types are documented in the [Trigger event type reference](#).

## IIoTA industrial IoT Platform: Trigger local variables, static variables, macros and event variables

### Overview

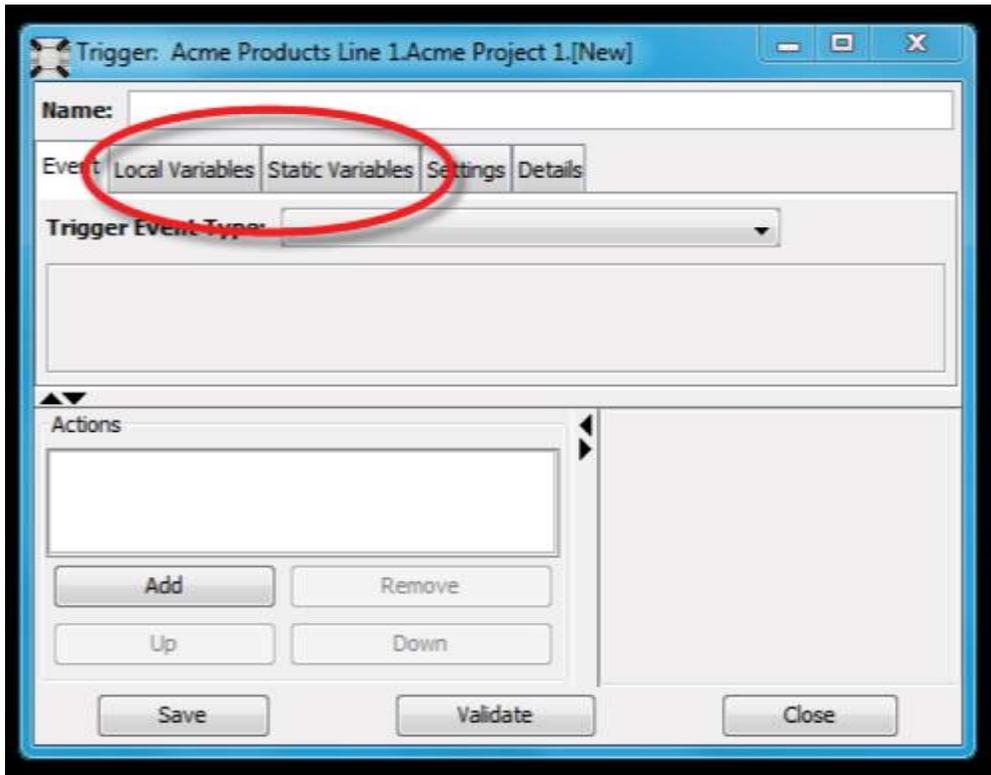
A trigger definition can include local variables and static variables for use as part of the trigger's application logic (in the trigger's actions). These local variables and static variables belong to the trigger and its actions and cannot be referenced outside of the trigger.

A group of trigger macros are available for use as part of the trigger's application logic. These trigger macros can be used as the source variable for the trigger's actions.

The trigger's event type also identifies the event variables, both source variables and destination variables, that are available to the trigger's actions.

## Local variables and static variables

A trigger's local variables and static variables are defined using their tabs in the trigger window.



These trigger variables can be used as part of the trigger's application logic when a variable is needed to hold a run time value that does not need to be available outside of the trigger.

Examples of local and static variable uses include:

- A temporary calculation
- A loop counter

An important difference between these trigger (local and static) variables and device variables is that the trigger variables are fully contained within the trigger's execution and are not accessible outside of the trigger. Device variables reside in a device and can be accessed by anything that can access the device, including different triggers, the device's application code, and the device's programming tool.

A trigger's local variables and static variables have the same definition and run time behavior, with the following differences:

- Local variables have a life cycle, or scope, of a single execution instance of the trigger. This means that the variable is created when an instance of the trigger is executed, and the variable is destroyed when that instance of the trigger ends its execution. A running

counter would lose its value when the trigger instance ends and be reinitialized to its default value when the next instance of the trigger is executed.

- Static variables have a life cycle, or scope, of the duration that the trigger is in a Started state and a Loaded status. This means that the trigger is started, and its project is also started. A static variable's value is retained during this life cycle, so a running counter would retain its value when the trigger instance ends and be available to the next execution instance of the trigger.

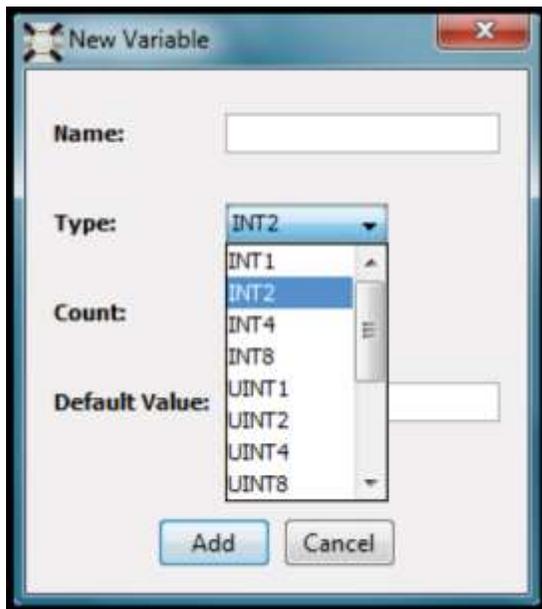
### Adding local or static variables to a trigger

1. Using either the List Editor or the Canvas Editor, from the trigger window select the **Local Variables** or **Static Variables** tab.

This example will use the **Local Variables** tab.

2. Select **Add**.

The New Variable window appears.

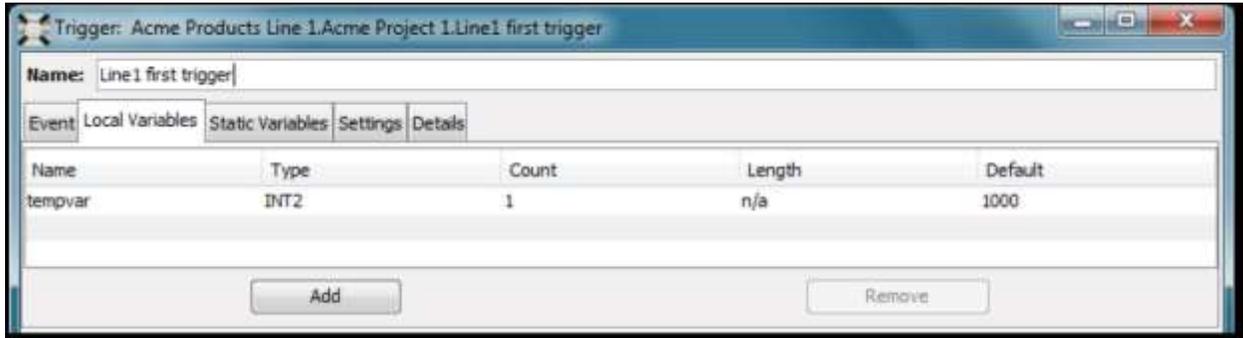


3. **Name** the variable, and then select the **Type** (for this example INT2).
4. In the **Count** parameter, enter a value of 1 (the default) to indicate a scalar variable, or enter a value greater than 1 to indicate an array variable.

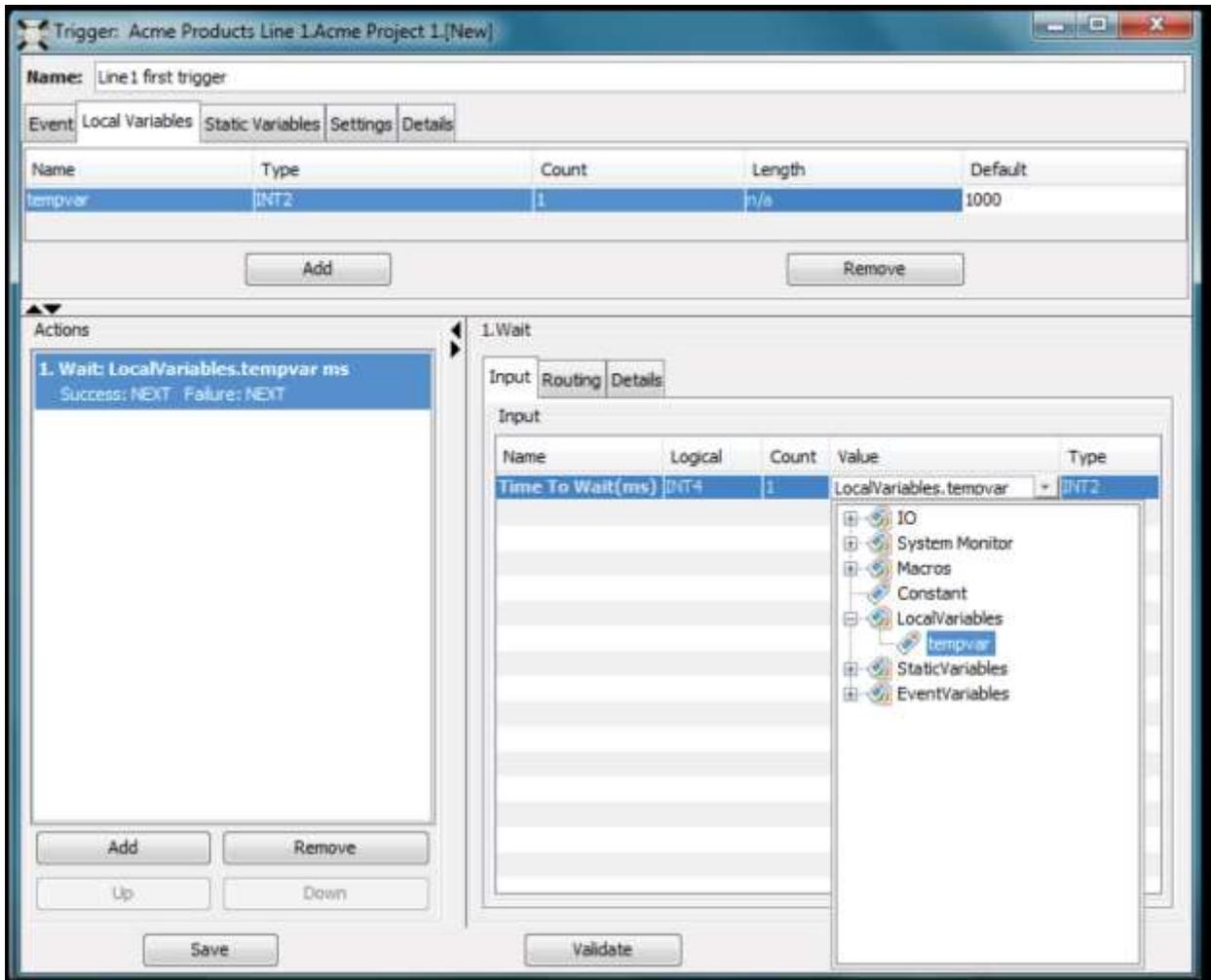
5. In the **Default Value** parameter, type the value to use to initialize the variable.

6. Select **Add**.

The variable is added to the **Local Variables** tab.



The defined local and static variables will then be available as source or destination variables for the trigger's actions, for example:



## Additional variable considerations

Additional considerations when using local variables, static variables or device variables include:

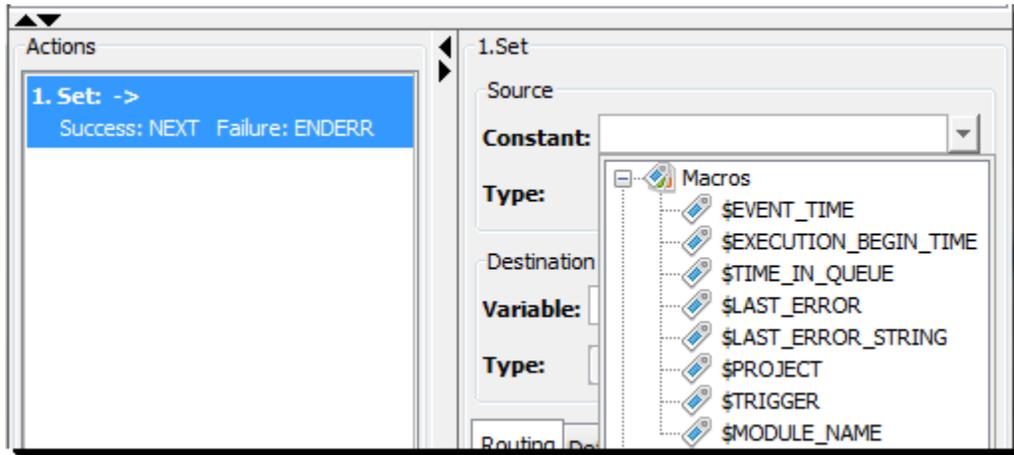
- A trigger's local or static variables are useful for temporary variables that do not need to be accessed outside of the trigger. They do not require the access overhead of a device driver code path, network overhead, and device response time to read or write.
- Local variables have a scope of a trigger's one execution instance, so the variable's value cannot be retained and passed between trigger execution instances. This does allow the local variable, since it is private to one execution instance of the trigger, to not be affected by multiple concurrent execution instances of the trigger.
- Static variables have a scope of the trigger being in the Loaded status until it changes to the Unloaded status. This means that the variable's value is retained and is shared by all execution instances of the trigger. If multiple instances of the trigger can be executed concurrently, the variable may have indeterminate values based on the timing of the different execution instances referencing the variable.
- **Caution:** When editing a started trigger, the trigger is reloaded upon save. This causes all static variables to be reset to their initial value.
- Device variables referenced in any of a trigger's actions are read once when the trigger execution instance begins. The device variables are written to an internal buffer during the trigger's execution of its multiple actions and are only written (flushed) to the device when the trigger execution instance ends.
- This initial reading of all device variables and final flushing of the buffered writes may not result in the desired device variable access that the trigger's application logic requires. For more information on device variable access, see Demand Read, Demand Write, and Device Commit.

## IIoTA industrial IoT Platform: Trigger macros

All triggers have access to a set of trigger macros that provide information related to the execution of the trigger.

These trigger macros are available to trigger actions as a source or input variable.

The following example shows the reference to the trigger macros as the source to a **Set** action:



Trigger macros are never available to trigger actions as a destination or output variable.

The trigger macros are:

Trigger macro	Data type	Description
<b>\$EVENT_TIME</b>	TIMESTAMP	The date and time that the trigger event occurred.
<b>\$EXECUTION_EVENT_TIME</b>	TIMESTAMP	The date and time that the trigger execution occurred. In most cases, this will be the same as \$EVENT_TIME. When the execution of the trigger is delayed because of trigger instance queuing, the two times could vary.
<b>\$TIME_IN_QUEUE</b>	INT4	The time that the trigger instance waited in the trigger instance queue before starting execution.
<b>\$LAST_ERROR</b>	INT4	The last error value encountered by the trigger.
<b>\$LAST_ERROR_STRING</b>	STRING	The last error value encountered by the trigger in a STRING data type.
<b>\$PROJECT</b>	STRING	The name of the project that this trigger belongs to.
<b>\$TRIGGER</b>	STRING	The name of the trigger.

<b>\$MODULE_NAME</b>	STRING	The node name where this trigger is executing.
<b>\$NULL</b>	STRING	Use this macro to specify a NULL value to an input variable of an action. This can be used to set a column or stored procedure parameter value to NULL when executing a database transaction action.

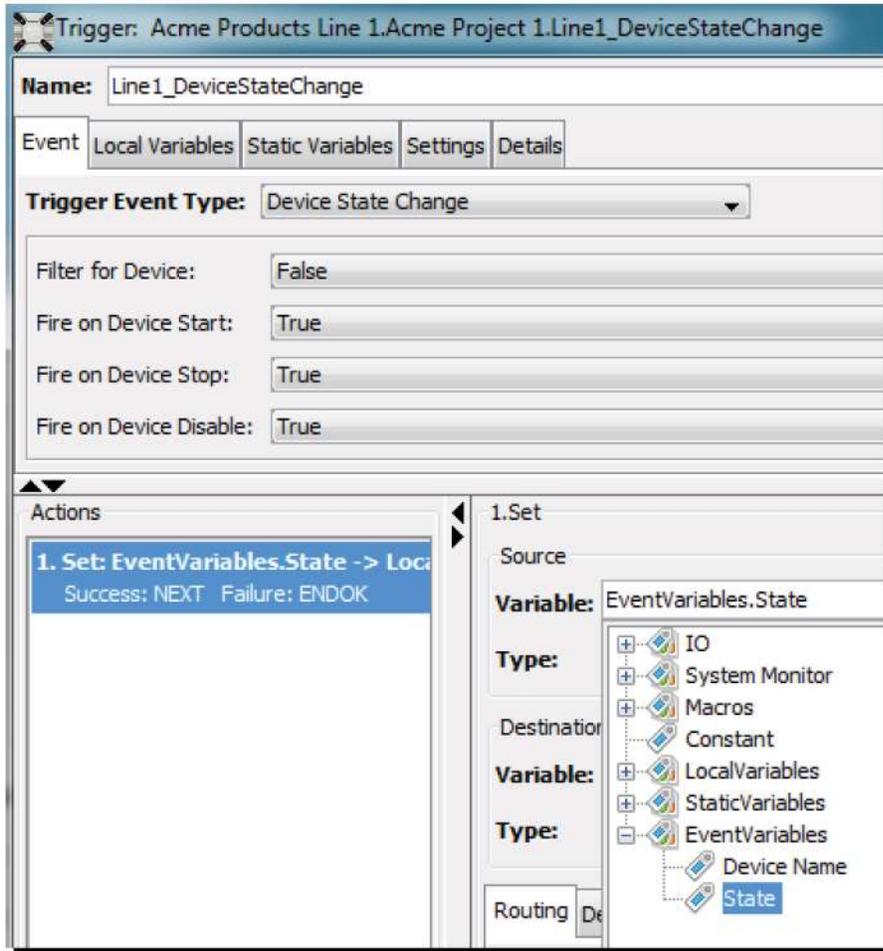
## IIoTA industrial IoT Platform: Trigger event variables

The trigger event variables available to a trigger when it executes are variables that relate to the current execution of a trigger, such as the data value of a monitored device variable for a data event trigger or the name of a device for a device state change event trigger.

The trigger event variables are either input event variables or output event variables (but not both):

- Input variables can be used for the source variable in an action but cannot be changed.
- Output variables can be used for the destination variable in an action. They can be changed and be used to pass information back to the component that caused the trigger to execute.

An example of the device state change event type shows the available input event variables available as the source of a Set action:



For this example of the device state change event type, there are no output event variables. The specific trigger event variables information is covered in the section for each trigger event type in the Trigger event type reference.

## IIoTA industrial IoT Platform: Trigger settings

The **Settings** tab allows you to set parameters for the trigger that provide specific execution processing such as the number of triggers that can be processed at the same time.

**Name:** Line1\_DeviceStateChange

Event | Local Variables | Static Variables | **Settings** | Details

**Max in Progress:**  **Queue Size:**

**Max Exec Time (ms):**  **Editor mode:**

**Reporting:**  **Editor Layout (Global):**

Apply reporting settings to subtriggers

## Parameter descriptions

Parameter	Description
<b>Max in Progress</b>	<p>Sets the number of trigger instances that are allowed to execute concurrently. The default value is one.</p> <p>If more than this number are scheduled to execute concurrently, the trigger instances will be queued for later execution based on the <b>Queue Size</b> parameter.</p> <p>For example, a schedule event type, with a periodic frequency set to 100 milliseconds, but the trigger executions are taking longer than 100 milliseconds. This will lead to multiple trigger instances that are scheduled to execute concurrently.</p>
<b>Max Exec Time (ms)</b>	<p>The maximum execution time for the trigger, in milliseconds. If the trigger execution exceeds this time, a warning message is logged in the Exceptions Log (even though the <b>Reporting</b> parameter might be set to off). This warning message in the Exceptions Log is for informational purposes, it does not end or change the trigger's execution.</p>
<b>Reporting</b>	<p>The trigger reporting option controls when a trigger report is generated and written to the Reports Log.</p> <p>Care should be taken when specifying the reporting option for triggers, based on the need for the additional information, the frequency of the execution of the triggers, and the availability of system resources on the node (CPU, memory, disk). For all options other than Off, the trigger is generated as the trigger steps through the trigger's actions. Once the trigger execution completes as successful or an error, the trigger report may be discarded or written to the Reports Log.</p>
<b>Apply reporting settings to subtriggers</b>	<p>An option to have the trigger reporting setting selected for this trigger applied to all "called" subtriggers. If selected, the trigger reporting selection (On, Off, etc.) is passed to a subtrigger that is "called" using the <b>Execute SubTrigger</b> action. This passing of the reporting selection</p>

	<p>applies to all levels of called subtriggers. For example: trigger 1 calls subtrigger 2, which calls subtrigger 3 and subtrigger 4. Trigger 1's reporting selection is passed to, and overrides, the reporting setting in subtrigger 2, subtrigger 3, and subtrigger 4.</p>
<b>Queue Size</b>	<p>The number of trigger instances that can be queued for later execution when the number of concurrent trigger instance execution reaches the <b>Max in Progress</b> parameter. Trigger instances queued are executed once another executor becomes available (its trigger instance ends execution). The default value is blank, which is the same as a zero, which indicates the queue is disabled.</p> <p>If the <b>Queue Size</b> value is reached, then additional trigger instances are marked as <b>Overflow</b> and are not executed.</p>
<b>Editor mode</b>	<p>The trigger editor to use when editing this trigger:</p> <p><b>List</b> - use the List Editor. Trigger actions are specified in a list form, with the action's routes set by selecting from a list.</p> <p><b>Canvas</b> - use the Canvas Editor. Trigger actions are positioned in a drag and drop flow chart form, with action routes indicated by connecting action output ports to other action's input ports.</p> <p><b>User preference</b> - Remembers the last editor mode used for the trigger.</p>
<b>Editor Layout (Global)</b>	<p>A Global setting, for all triggers, for the trigger editor's display layout style.</p>

## Use of trigger settings during development, debug and production

As an application's triggers are developed and debugged, the settings can be used to aid the development process and to understand a trigger's execution path.

Once the application is put into production, the settings can be monitored to understand how the triggers are executing in different production environment scenarios.

For example:

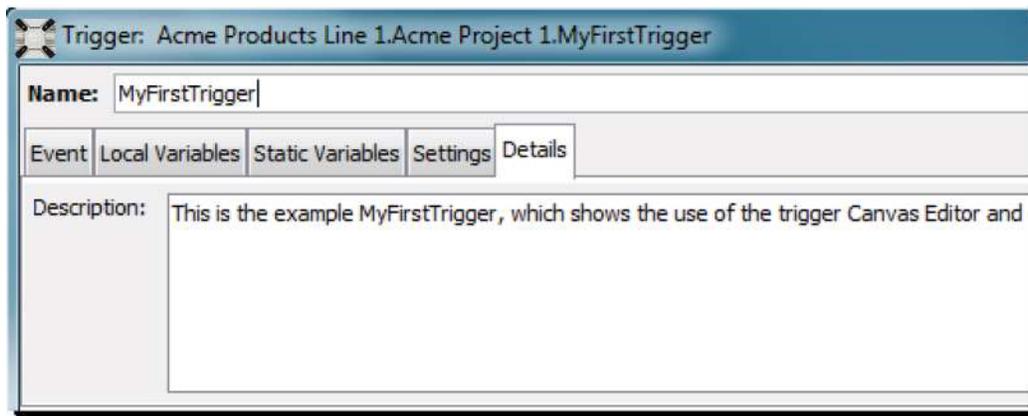
- The **Reporting** parameter can be set to On when developing a trigger and then set to Off before going into production.
- The **Max in Progress** parameter can be used to ensure that only one instance of a trigger is executing concurrently (value set to 1) or it can be used to allow multiple concurrent instances of the trigger to execute concurrently. Allowing multiple instances

of a trigger to execute concurrently may require serialization or concurrency concepts in the trigger's application logic.

- The **Queue Size** parameter can be used to allow temporary spikes in trigger instances being scheduled for execution to be queued for later execution, instead of discarding the instances as an **Overflow**.
- An increase in a trigger's **Overflow** counter (in the trigger list in the project's tab on the Projects window) may mean that the **Max in Progress** and **Queue Size** parameters need to be adjusted based on the trigger's event frequency and the trigger's execution time.

## IIoTA industrial IoT Platform: Trigger details

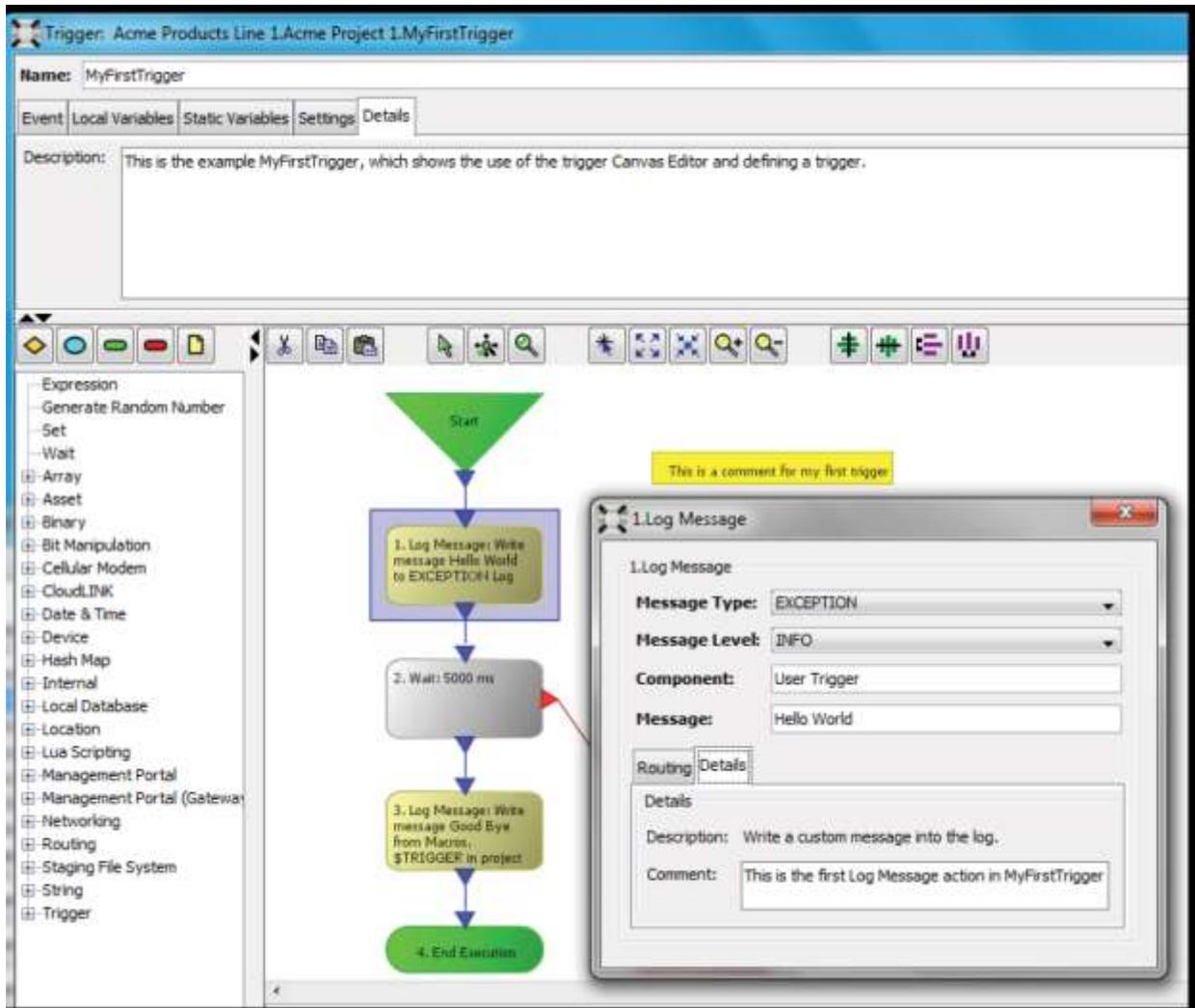
The **Details** tab allows you to enter a text description for the trigger.



This trigger description is available for viewing when the trigger is edited.

In addition to the information in the **Details** tab, the trigger Canvas Editor supports a **Comment** block that can be placed anywhere on the canvas. The **Comment** blocks added to a trigger can have any information desired to document the trigger's application logic and flow.

Each action in a trigger also has a **Details** tab that can be used for information specific to that action within in the trigger's application logic. For example:



## IIoTA industrial IoT Platform: Trigger actions

A trigger's actions define the trigger's application logic or *what the trigger does*.

Trigger actions are individual function blocks that provide a single step in the trigger's application logic. The execution flow, or route, through the trigger's actions can be controlled based on an action's result and run time values of variables.

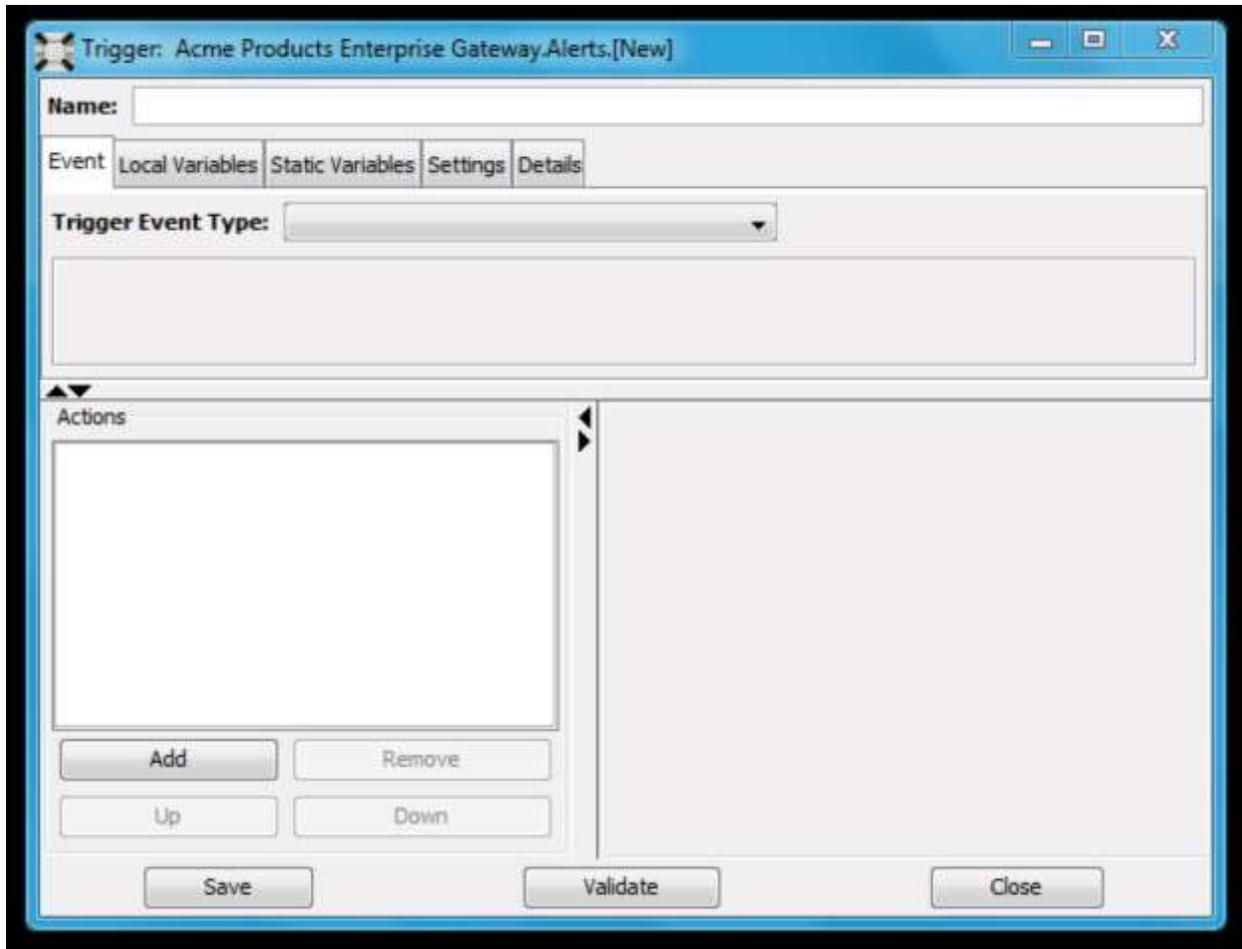
The trigger actions available in the trigger editor depends on the type of product installed on the Enterprise Gateway, as well as the packages and extensions installed on the node (for example the device drivers).

The trigger actions are documented in the Trigger actions reference.

The Workbench supports two trigger editors: the List Editor and the Canvas Editor. Triggers can be initially defined using either editor and then edited and saved with the other editor. For consistency of the underlying trigger definition's action location (Canvas Editor) and action number (List Editor), it is usually best to consistently use one editor or the other for a trigger.

## Specifying actions using the List Editor

When you use the trigger List Editor to define a new trigger, the Trigger window is displayed:



Actions are added to the trigger by selecting the **Add** button in the **Actions** section of the window, and then selecting the specific action to add.

After an action is added to the trigger, the left-hand pane displays the parameters for that action.

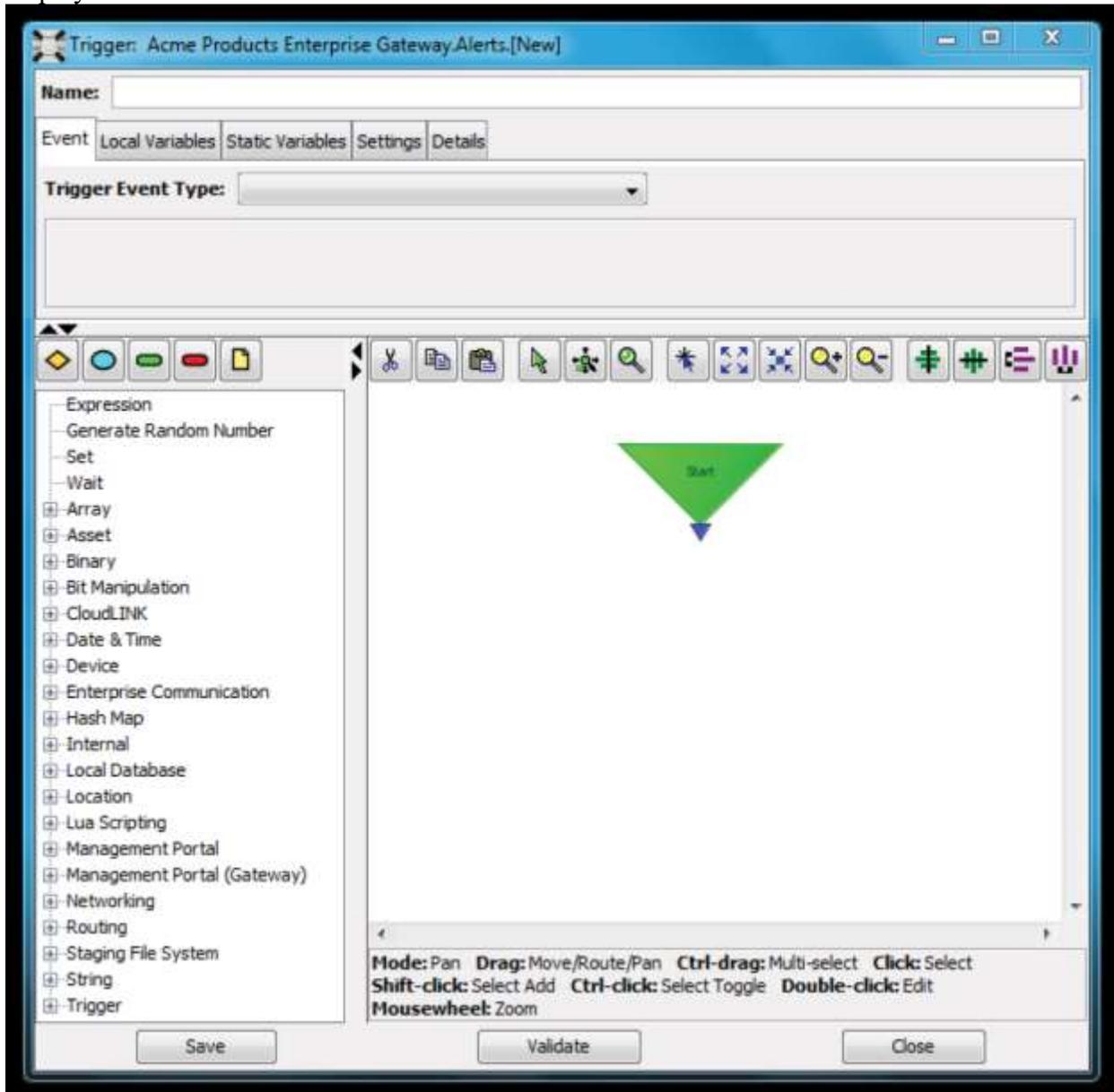
After multiple actions are added to the list, the actions can be reordered by using the **Up** and **Down** buttons.

Actions can be removed using the Remove button.

Right clicking an action will display a pop-up menu with the options to **Duplicate** the action or **Renumber** the actions in the list to their current sequential order.

## Specifying actions using the Canvas Editor

When you use the trigger Canvas Editor to define a new trigger, the Trigger window is displayed:



Actions are added to the trigger by selecting (click once) the specific action from the left-hand pane and then selecting (click once) the location in the right-hand pane canvas to place the action block.

A drag and drop style is also supported by selecting (click once and hold) an action and dragging the cursor to the location in the right hand pane canvas and dropping (release the mouse button) to place the action block.

After an action is added to the canvas in the left-hand pane, double clicking the action displays the parameters for that action.

Actions that have been added to the canvas in the left-hand pane can be re-positioned by dragging and dropping them.

Actions can be removed by selecting them (click once) and then using the delete key. Other navigation functions are listed below the canvas and provided by the icons in the toolbar above the canvas.

For more information on using the Canvas Editor, see [Using the Canvas Editor](#).

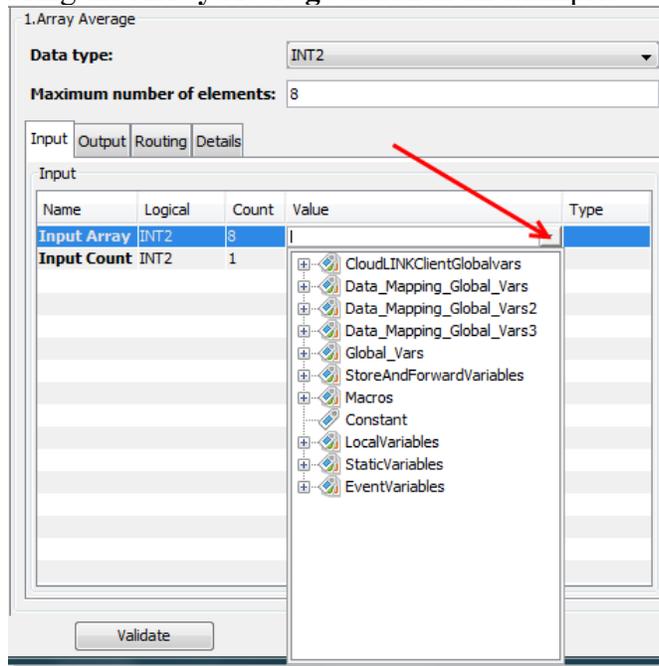
## Entering action properties

Each action has the specific properties needed for the action to provide its function. These properties can be provided in the action's:

- Parameters
- Input tab
- Output tab

An action may or may not have parameters, an input tab or an output tab depending on the action's provided function.

Using the **Array Average** action as an example:



The main pane has two parameters:

- **Data Type** - which is a drop-down selection list parameter
- **Maximum number of elements** - which is a numeric entry parameter.

This action has both an **Input** tab and an **Output** tab. Each tab has rows for each of its parameters.

When a parameter can be specified by a device variable, trigger macro, constant, trigger local variable, trigger static variable or trigger event variable, the available options are displayed by clicking once in the **Value** cell for the parameter and then selecting the down arrow icon.

Only applicable options are displayed. For example:

- Macros and Constant do not apply for an output parameter
- Device variables are only available if the device is in a started state.

## Parameters

Parameter fields can either be an input field where a value is entered, a pop out editor for more complex parameters, or an option list.

Lists may refer to a predefined set of options that will not change, or other item definitions within the IIoTA such as transport maps, other triggers, etc., that may be dynamic.

These lists are stored when first accessed in an editor session. For a dynamic list, refreshing them may be desired if new items are defined or imported while the trigger editor is active.

Using the **Fire Trigger** action as an example:

1.Fire Trigger

**Action Input Type:** Static

**Project Name:** == Current Project ==

**On-Demand Trigger:** test1

**Wait for Completion:** False

The **Project Name** list can be refreshed. The button to the right of the list (indicated by the green arrow) will cause the contents of that list to be retrieved from the gateway.

The **On-Demand Trigger** list is dependent on the project field and will be refreshed when the project field is. This applies to any option list with contents that is dependent on the value of another field.

For parameters referring to a list of triggers, an edit button will be available (indicated by the red arrow in the image below). Selecting this button will open a trigger editor window with that trigger, or switch to an existing window with that trigger if it is already being edited.

## Input/Output Context Menu

Right clicking on a row in the input or output tables will bring up a context menu with the following options:

Option	Result
Map Selected > To Local Variables	Creates local variables using the row name (or references existing variables)
Map Selected > To Static Variables	Creates static variables using the row name (or references existing variables)
Populate Array	Fills in table with array elements. This enhancement was added in release 16.1.0. See below for details.
Clear	Clears values from the selected rows.
Clear All	Clears values from all rows.

### Populate Array

Populate array allows a large table to be filled in quickly. Select a row with an array element and select "Populate Array".

If the rows below are not all empty you'll be given the option to overwrite existing values or stop at the first row with a value, otherwise the command will be immediately performed.

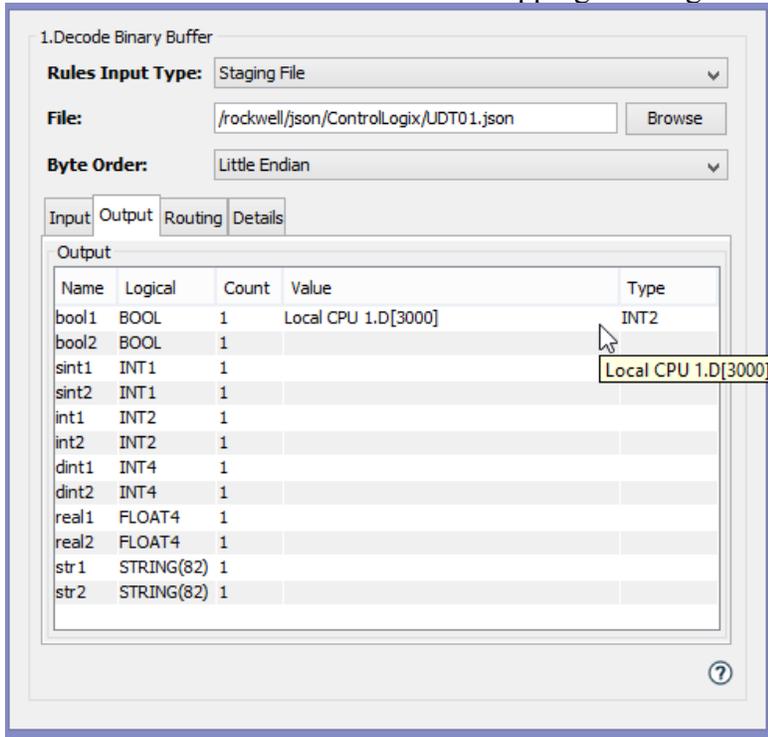
Populating using BINARY values or Siemens STRING data using a 2-byte prefix is not supported.

This function will determine the type of each row and how it can be mapped into an array element. For example, an array of type INT2 that can be read/written as an INT4, will have its type changed to INT4 and the index of the next row will increment by 2 instead of 1 when the logical type is INT4. A more detailed example follows:

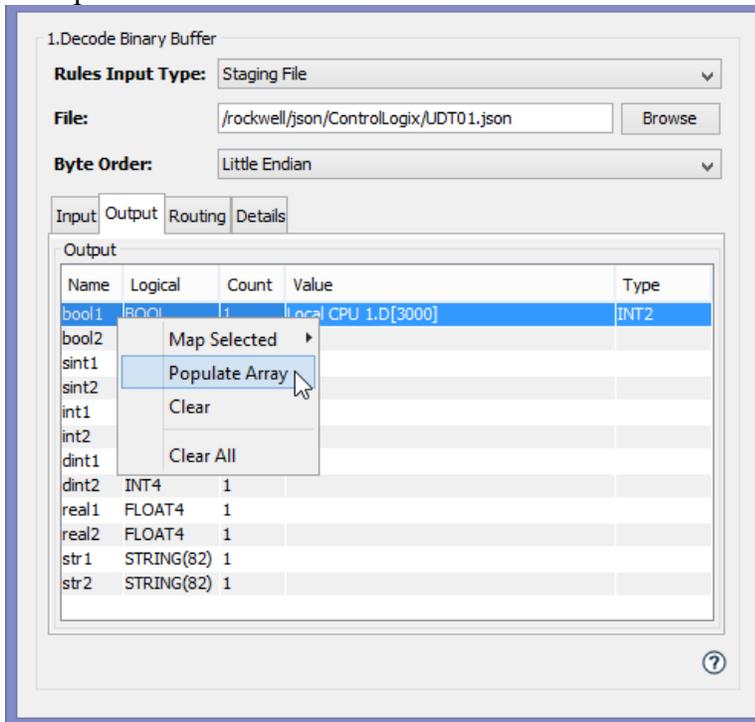
### Populate Array Example

This example uses the Decode Binary Buffer action. The feature works with all actions, following the same steps.

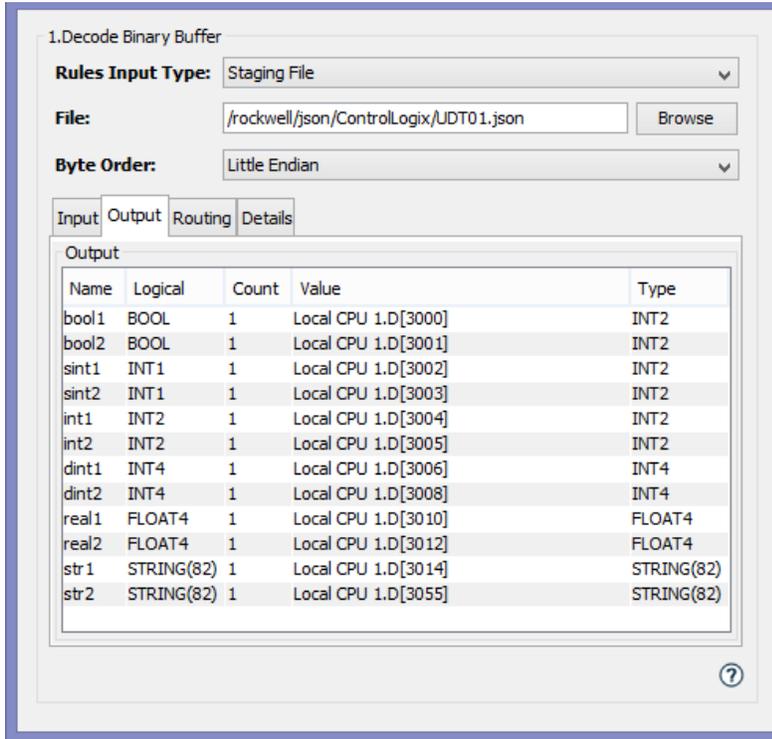
1. Select the First variable where the mapping will begin.



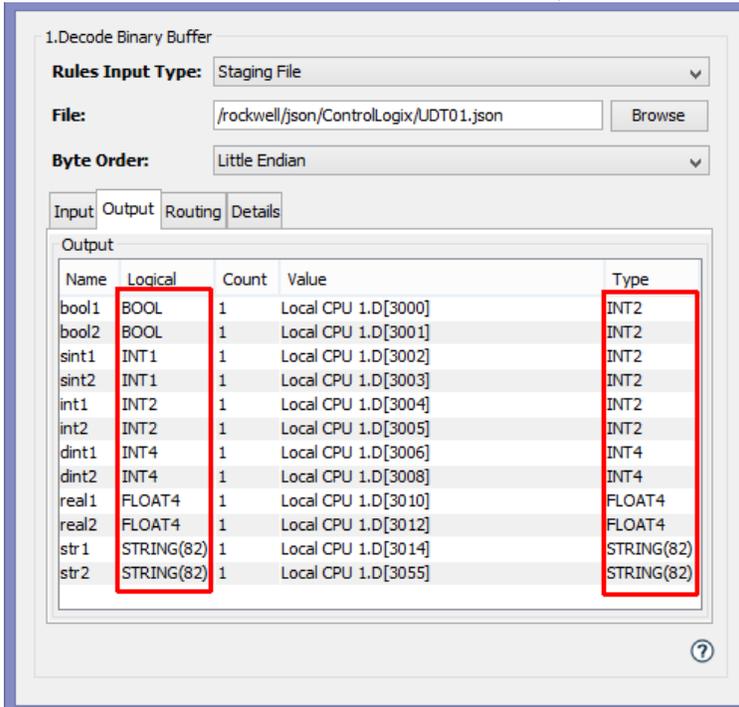
2. Right mouse button click on the variable name of the first item that was mapped, in this example the variable named *bool1*.



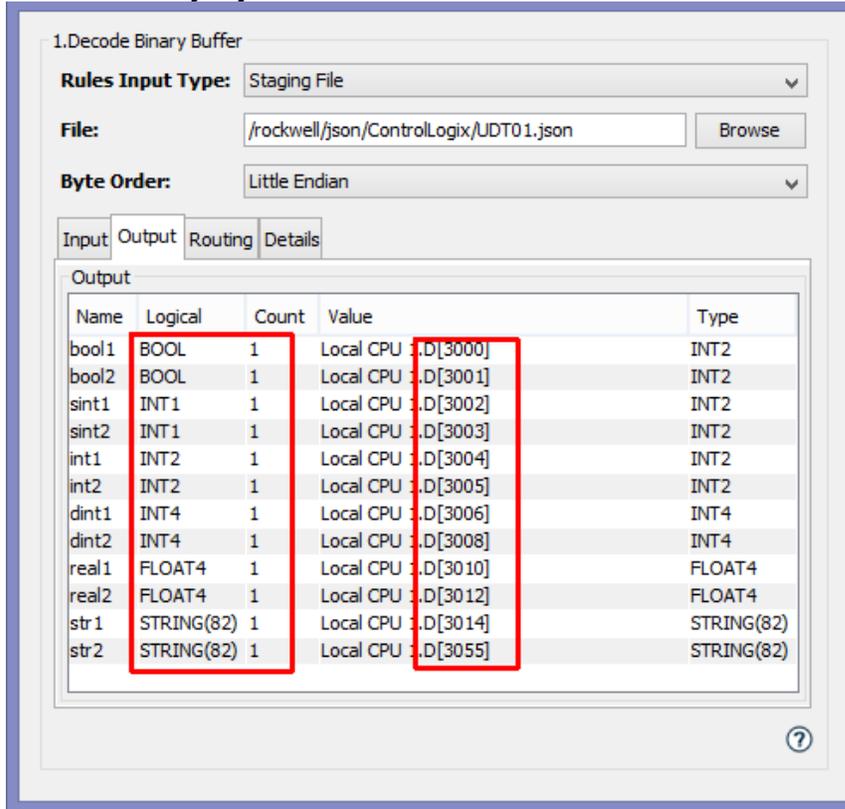
- This will display a pop-up menu with *Populate Array* option. This is only available when an arrayed value is in the selected row. Select this option to map the remaining variables. The result will look similar to this:



- The data types for the source variables, which are displayed in the Logical category are transferred over to destination variables, shown in the **Type** field.



- The offsets in the variable destination are changed based on the data type, the count, and the memory layout of the destination variable.



## Specifying a routing for an action

Each action has a routing, which determines what action to go to next. The specific routes available for an action depend on the different outcomes that are possible for that action.

Most actions have a **Success** route. Most actions have a **Failure** route. Actions have other routes based on the action's possible outcomes, such as the **Transaction** action having a **Store and Forward** route or the **If** action having **True** and **False** routes.

When using the List Editor, an action's routes are selected using the **Routing** tab in the left-hand pane and selecting which action to go to next.

When using the Canvas Editor, an action's routes are specified by drawing connecting lines between an action's output ports to another action's input port.

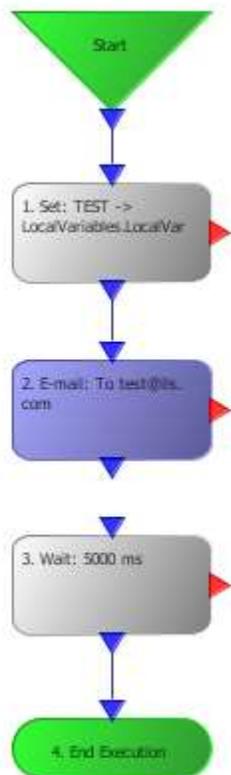
Since a trigger's actions' routes specifies the trigger's execution path through the actions, it is important to understand the routes and how the application logic will be directed when the trigger is executed. Dynamic run time conditions must be taken into account, for example:

- A device not being started and its variable not being accessible.
- An external database server not being accessible.

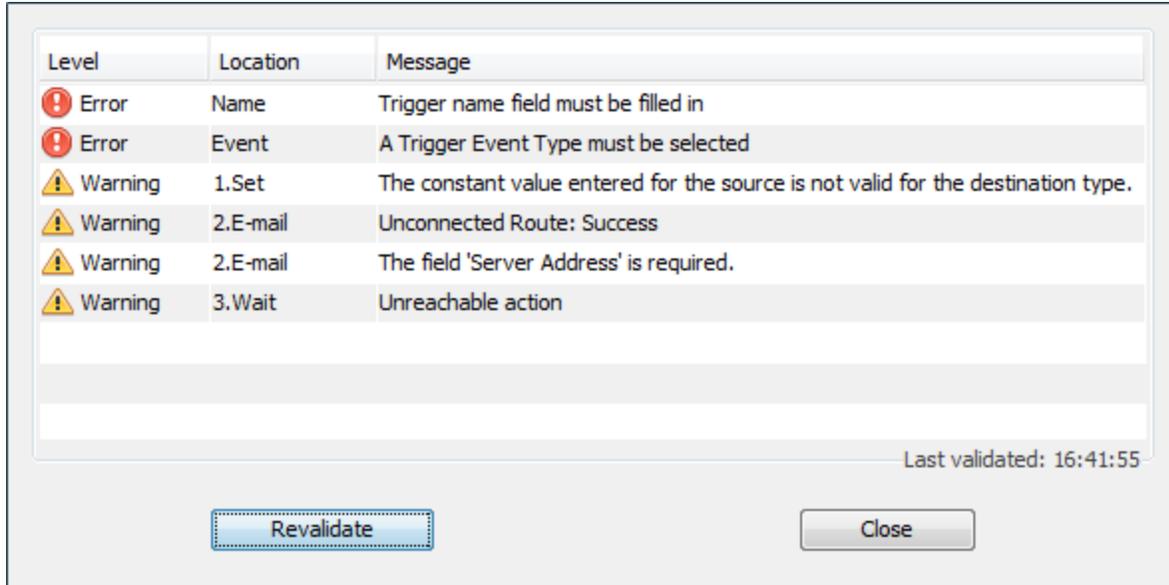
Related topics  
Trigger actions reference  
Using the Canvas Editor

## IIoTA industrial IoT Platform: Trigger validation

When a trigger is validated or saved, a validation dialog will be displayed.  
As an example, for the Canvas Editor, the trigger shown below has several errors and warnings with its definition:



When the **Validate** or **Save** functions are selected, the validation dialog is displayed.  
For this example trigger, the dialog would be similar to the following:



Once the validation dialog is displayed, it will remain until closed by the user. It will not interfere with editing of the trigger and can be moved to the side for convenience.

A timestamp at the bottom right of the dialog shows the last time validation was performed.

There are two levels of messages in the validation dialog:

- **Error** - An error that prevents this trigger from being saved.
- **Warning** - A problem that still allows the trigger to be saved but may cause an issue when the trigger is executed.

Double-clicking on any entry in the list will select the corresponding action in the Canvas Editor. If the List Editor is being used to define the trigger, it will change to the selected action.

The **Revalidate** button will perform the validate again and is useful after changing the actions that were listed as having an error or warning to see if the problem has been completely corrected.

## IIoTA industrial IoT Platform: Trigger search

A trigger can be searched for string values matching a search parameter. Clicking on the **Search** button will open a dialog, for example:



1. Action Name
2. Properties
  1. Name
  2. Value
3. Input
  1. Name
  2. Value
4. Output
  1. Name
  2. Value
5. Routing
  1. Routing value

Parameters in the Event and Details sections are not searched.

The **Result Table** will display matches for the search parameters

- Matching string will be highlighted in **Match** column, with entire value for that field displayed
- Selecting a row (single click) will show that location in the editor
- Opening a row (double click) will perform the same action as select, but will also open an editor dialog in the canvas editor if it is not already open
- If a row is a local variable or static variable location, that tab will be selected
- If a row is a input/output/route for an action, that tab will be selected.
- The time of the last search and the number of result rows will be displayed below the table. One row can have multiple matches.

Once the search dialog is displayed, it will remain until closed by the user. It will not interfere with editing of the trigger and can be moved to the side for convenience.

## IIoTA industrial IoT Platform: Using an individual project's tab

### A project's tab

In addition to the Projects tab, each individual project will have a tab when the project is opened. A project being opened in the Workbench is just so its tab is displayed and information about the project's triggers is available. The project tab does not have to be opened to allow triggers to execute.

When a project's tab is displayed in the foreground in the **Projects** window, you can manage the triggers within the project.

Follow these steps to display a project's tab:

1. From the Workbench left hand pane, select and then expand the node you want to work with.

An expanded tree view of the selected node appears.

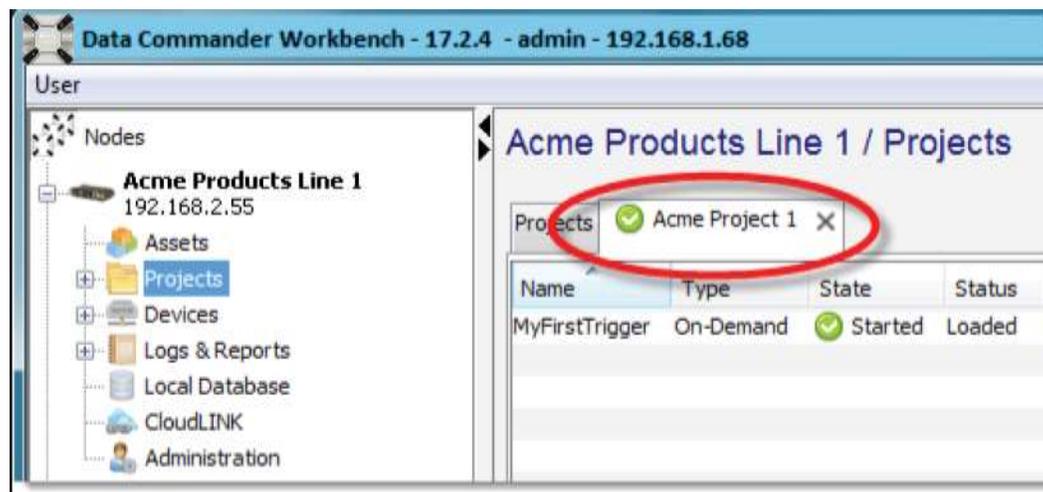
2. Select the **Projects** icon.

The **Projects** window appears as the right-hand pane with the **Projects** tab listing all of the defined projects.

3. Double click the project.

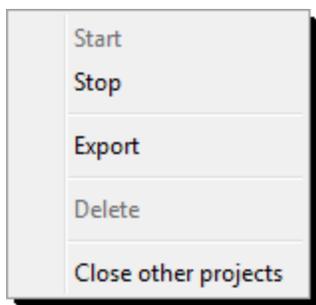
Alternatively, you can select the project and then select the **Open** button at the bottom of the **Projects** tab.

The name of the project appears on its tab along with an indication of the project's state, either Started or Stopped.



### A project's tab pop-up menu

If you right-click on a individual project tab, a pop-up menu with options is displayed:



The options from this project pop-up menu are:

Option	Description
<b>Start</b>	Starts the selected project. Alternatively, the project can be selected on the <b>Projects</b> tab and the <b>Start</b> button at the bottom of the <b>Projects</b> tab can be used. This option is only available if the selected project is in a stopped state.
<b>Stop</b>	Stops the selected project. Alternatively, the project can be selected on the <b>Projects</b> tab and the <b>Stop</b> button at the bottom of the <b>Projects</b> tab can be used. This option is only available if the selected project is in a started state.
<b>Export</b>	Displays the export window, allowing the selection of the items in the project (its triggers) and any of the triggers' dependencies to be exported.
<b>Delete</b>	Deletes the selected project and all of the project's triggers. Alternatively, the project can be selected on the <b>Projects</b> tab and the <b>Delete</b> button at the bottom of the <b>Projects</b> tab can be used. This option is only available if the selected project is in a stopped state.
<b>Close other projects</b>	All other project tabs are removed from the <b>Projects</b> window. The state of the projects is not changed, just the removal of the projects' tabs.

## A project tab's triggers

When a project's tab is displayed in the foreground, it provides a table displaying its triggers with these columns:

Column	Description
<b>Name</b>	The name of the trigger.
<b>Type</b>	The trigger event type, such as <b>Data</b> , <b>Scheduled</b> , <b>On-Demand</b> and so forth.
<b>State</b>	The trigger's state:

	<p> <b>Started</b> - The trigger is ready to execute. In order for a trigger to execute, its status must also be <b>Loaded</b>.</p> <p> <b>Stopped</b> - The trigger is not ready to execute; its status is <b>Unloaded</b>.</p> <p> <b>Disabled</b> - The trigger was disabled by a fatal error when processing an event or action during execution. The system will periodically attempt to start triggers that are <b>Disabled</b>.</p> <p><b>Starting</b> - A temporary transition state while going from stopped to started.</p> <p><b>Stopping</b> - A temporary transition state while going from started to stopped.</p>
<b>Status</b>	<p>The trigger's status:</p> <p><b>Loaded</b> - The trigger is loaded into memory within the trigger component and ready to execute when the trigger event condition occurs. The project must be <b>Started</b>, and the trigger must be <b>Started</b> for the trigger's status to be <b>Loaded</b>.</p> <p><b>Unloaded</b> - The trigger is currently not loaded into memory because the project is stopped, or the trigger is stopped or disabled.</p> <p><b>Loading</b> - The trigger is in the process of loading its actions and registering the trigger event within the trigger component.</p> <p><b>Unloading</b> - The trigger is in the process of releasing its resources. A trigger will not complete unloading until all execution instances listed as <b>In Progress</b> have completed.</p>
<b>Last Triggered</b>	The date and time that the trigger was last executed.
<b>Last Failure</b>	The date and time the last execution failure of the trigger occurred.
<b>User</b>	The user that last changed the trigger's state and status. All security requests that must be evaluated by the trigger are based on the access privileges of this user.
<b>Successes</b>	The number of times the trigger has ended its execution with an end success indication.
<b>Failures</b>	The number of times the trigger has ended its execution with an end failure indication.
<b>In Progress</b>	The number of instances of the trigger that are currently executing. The <b>Max in Progress</b> parameter in the trigger definition defines the maximum number of concurrent trigger executions allowed for this trigger. If more than this number are scheduled to execute concurrently, the trigger instances will be queued for later execution based on the <b>Queue Size</b> parameter.
<b>In Queue</b>	The number of trigger executions that are queued waiting to be executed.

<b>Overflow</b>	The number of triggers execution instances that could not be executed because the number of trigger execution instances that are in the <b>In Queue</b> reaches the <b>Queue Size</b> parameter in the trigger definition. An increase in a trigger's <b>Overflow</b> counter may mean that the <b>Max in Progress</b> and <b>Queue Size</b> parameters need to be adjusted based on the trigger's event frequency and the trigger's execution time.
<b>Avg Time</b>	The average execution time of the trigger for the last 10 executions. This value is in milliseconds.

### Summary of trigger characteristics and status

When a trigger is from the table of trigger, a summary of the trigger's characteristics and status is displayed on the bottom half of the project's tab. You can see at a glance the characteristics of the trigger and its execution statistics:

<b>Name:</b>	MyFirstTrigger	<b>Last Modified:</b>	2013-05-03 09:49:35
<b>State:</b>	Started	<b>Last State Change:</b>	2013-05-03 12:54:19
<b>Status:</b>	Loaded	<b>Last Triggered:</b>	2013-05-03 12:54:24
<b>Successes:</b>	1	<b>Average Execution Time:</b>	5095 ms
<b>Failures:</b>	0	<b>Last Execution Time:</b>	5095 ms
<b>In Progress:</b>	0	<b>Max Execution Time:</b>	5095 ms
<b>Max in Progress:</b>	1	<b>Min Execution Time:</b>	0 ms
<b>In Queue:</b>	0	<b>Queue Watermark:</b>	0
<b>Queue Size:</b>	0	<b>Queue Watermark Timestamp:</b>	n/a
<b>Overflow Count:</b>	0		
<b>Event Type:</b> On-Demand			
<b>Actions:</b>			
1. Log Message: Write message Hello World to EXCEPTION Log			<i>Success: 2</i>
2. Wait: 5000 ms			<i>Success: 3 Failure: 5</i>
3. Log Message: Write message Good Bye from Macros.\$TRIGGER in project Macros.\$PROJECT to EXCEPTION Log			<i>Success: 4</i>
4. End Execution (Success)			
5. End Execution (Failure)			

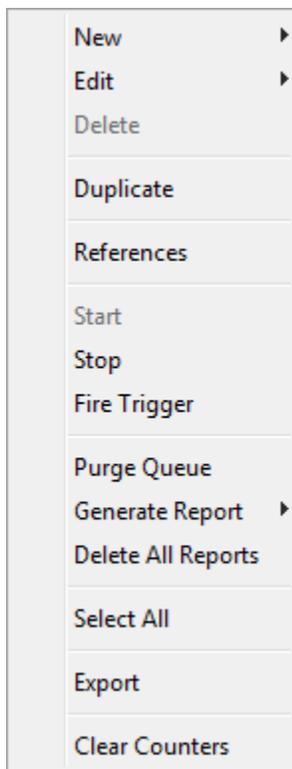
The additional fields beyond what is displayed in the table of triggers include:

Name	Description
<b>Max in Progress</b>	The Max in progress parameter from the trigger definition.
<b>Queue Size</b>	The Queue Size parameter from the trigger definition.
<b>Event Details</b>	One or more characteristics of the trigger event type. For example: a data event type trigger will display the device variable, a scheduled event type trigger will display the frequency.
<b>Last Modified</b>	The date and time the trigger was last modified.
<b>Last State Change</b>	The date and time the trigger last changed its state.

<b>Last Execution Time</b>	The last instance execution time in milliseconds.
<b>Max Execution Time</b>	The maximum instance execution time in milliseconds.
<b>Min Execution Time</b>	The minimum instance execution time in milliseconds.
<b>Queue Watermark</b>	The maximum value that the In Queue parameter contained.
<b>Queue Watermark Timestamp</b>	The date and time that the Queue Watermark was updated to its maximum value.
<b>Actions</b>	A summary of the trigger's actions, the actions' parameters and the actions' routes.

### A trigger's pop-up menu

If you right-click on an individual trigger in the trigger table, a pop-up menu with options is displayed:



Some menu options will not be available depending on the trigger's state and the event type. The following describes the menu options available from the pop-up menu:

Option	Description
<b>New</b>	Displays a New Trigger window to define a new trigger. The option includes selecting either the List Editor or the Canvas Editor.
<b>Edit</b>	<p>Displays a Trigger window to view and edit the trigger's definition. The option includes selecting either the List Editor or the Canvas Editor. When editing a trigger, keep the following in mind:</p> <p>When editing and then saving a started trigger, the trigger is reloaded upon save, causing all static variables to be reset to their default value.</p> <p>When editing and then saving a started trigger, any trigger execution instances that are currently in progress will be completed before the new trigger definition changes take effect.</p> <p>If the trigger is being edited by another person, you will receive a message indicating the trigger is locked but still available for view only access.</p>
<b>Delete</b>	Deletes the trigger. The trigger must be in a Stopped state to allow deleting.
<b>Duplicate</b>	Duplicates (copies) the trigger definition. The option allows entering a new name for the copy of the trigger and selecting the project for the new trigger.
<b>References</b>	Displays a list of items that have a reference to the trigger and a list of items that are referenced by the trigger.
<b>Start</b>	Starts the trigger. The trigger must be in a Stopped state to allowing starting.
<b>Stop</b>	Stops the trigger. The trigger must be in a Started or Disabled state to allowing stopping.
<b>Fire Trigger</b>	<p>Executes an On-Demand event type trigger.</p> <p>Also can be used to execute a Schedule event type trigger. In the case of executing a Schedule event type trigger, manually selecting the <b>Fire Trigger</b> option does not impact the trigger's schedule for being executed (Periodic, Hourly, etc.).</p>
<b>Purge Queue</b>	Removes all pending trigger execution instances. The number of pending trigger executions is indicated in the trigger's <b>In Queue</b> value.
<b>Generate Report</b>	<p>Generates a trigger report for the next execution of the trigger. The options are:</p> <p><b>Selected Triggers</b> - a trigger report is generated for the next execution of the trigger.</p>

	<p><b>Selected Triggers Including SubTriggers</b> - a trigger report is generated for the next execution of the trigger and all subtriggers "called" with the <b>Execute SubTrigger</b> action. This one-time generation of the trigger report applies to all levels of called subtriggers. For example: trigger 1 calls subtrigger 2, which calls subtrigger 3 and subtrigger 4.</p> <p>You can view the report using the Reports tab.</p>
<b>Delete All Reports</b>	Deletes all trigger reports generated by the trigger.
<b>Select All</b>	Selects all the triggers in the project. If you want to exclude one or more triggers from your selection, after using select all, press and hold down the Ctrl key, and then click on the trigger's row in the table.
<b>Export</b>	Displays the Export window with the selected trigger and allows the trigger and its dependencies to be selected for export to a file on the Workbench's computer. For more information on the Export function, see Exporting a project or trigger.
<b>Clear Counters</b>	Clears the execution counters from the trigger table.

## Trigger table and option considerations

- A trigger's state must be **Started** and the trigger's status must be **Loaded** in order for the trigger to be executed. If the trigger's status is **Unloaded**, then the project needs to be **Started**.
- The triggers **Successes** and **Failures** counts are based on the trigger ending its execution with an end success or an end failure indication. The counts are not determined based on a trigger action having a failure.
- When editing and then saving a started trigger, the trigger is reloaded upon save, causing all static variables to be reset to their default value.
- When editing and then saving a started trigger, any trigger execution instances that are currently in progress will be completed before the new trigger definition changes take effect.
- If the trigger is being edited by another person, you will receive a message indicating the trigger is locked but still available for view only access.

- Some of the options can be used when multiple triggers are selected. You can use Ctrl-A, Select All, Shift-click, Ctrl-click, or press and hold the mouse button while swiping multiple rows to select multiple triggers. Then right-click to display the pop-up menu and selecting an option will attempt to apply the option to the multiple selected items. If the option cannot be applied, a message window will indicate the selections that will be applied.

For example, select three stopped triggers and then select **Start**.

## IIoTA industrial IoT Platform: Exporting a project or trigger

The **Export** option is available from a project's and a trigger's pop-up menu. It is also available for other items, such as devices, transports, transport maps, and so forth.

An export is a copy of the item's definition, written to a file on the Workbench's computer. The file can be used when using the Import function, to read in and apply the item's definition to the same or a different node.

The export and import functions can be used for various reasons, including:

- Making a copy of an item before making planned changes
- Making a copy of an item in case changes are inadvertently made to the item
- Making a copy of an item from one node to import into another node to "duplicate" the item.

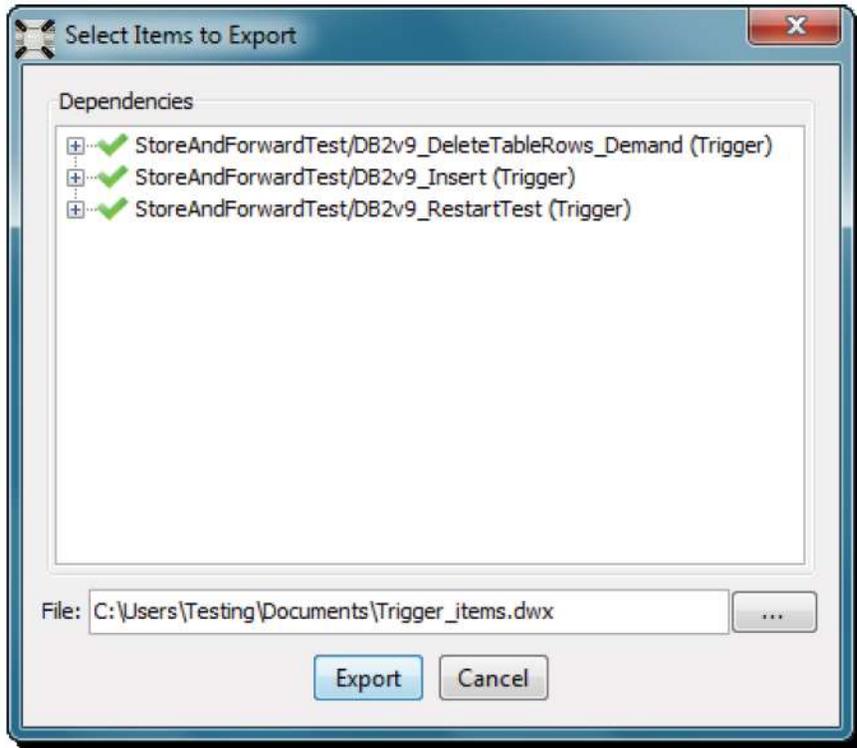
The export and import functions have the concept of an item's dependencies, or other items that are referenced or relied upon. Some examples are:

- A trigger references its project
- A project references all of the triggers in it
- A trigger that accesses a device variable references the device
- A trigger that uses a transport map in a Transaction action references the transport map
- A transport map references the transport it uses
- Etc.

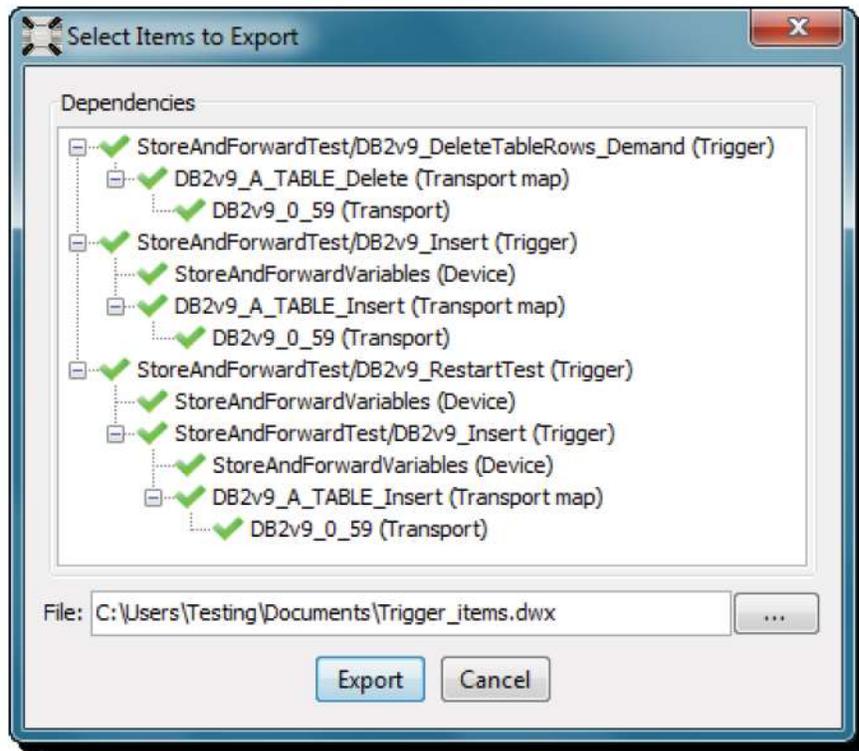
Items can be in a Started or Stopped state when exported.

When one or more items are selected and then the **Export** option is selected, an **Export** window is displayed with the items and their dependencies in an expandable tree listing.

This example has three triggers that were selected and then the Export option selected:



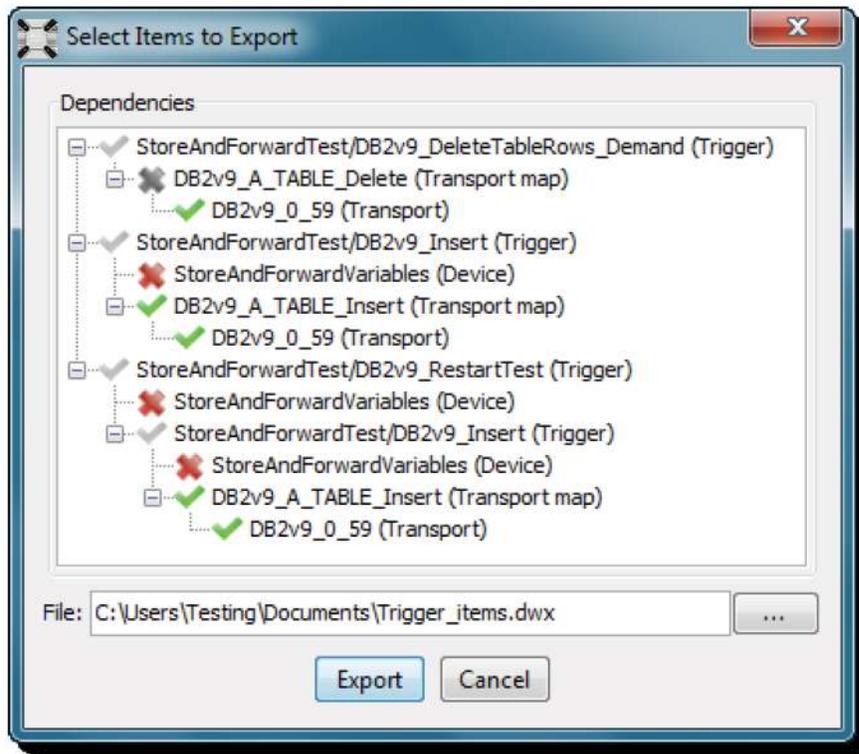
The three triggers are listed with an expand [+ ] icon next to them to indicate that they have dependencies. If all of the items are expanded, the display becomes:



By default, all items and their dependencies are selected for the export as shown by the green check mark icon next to each item.

Items change from selected or deselected by clicking on its icon.

- A green check mark icon means an item and all of its dependencies are selected for export.
- A grey check mark icon means an item is selected for export, but one or more of its dependencies are not selected.
- A red X mark icon means an item and all of its dependencies are not selected for export.
- A grey check mark icon means an item is not selected for export, but one or more of its dependencies are selected.



After the items for export selections are made, select the **Export** button.

A file dialogue window is displayed, allowing you to name the file and the location to write the export file.

- The export file will have a .DWX extension, which should not be changed.
- The export file may or may not be encrypted, depending on the Workbench and node's properties.
- The export files should not be edited and changed, since the format needs to remain intact if it is to be used as part of an **Import** function.
- The node **Back Up** function can be thought of as a complete export of all the items in a node, including system configuration items and application definition items, such as projects and triggers.

Related Topic

Backing up and Restoring a node's configuration

# IIoTA industrial IoT Platform: Importing a project or trigger

The **Import** option is available from the Projects tab window pop-up menu. It is also available for other items' pop-up menus, such as devices, transports, transport maps, and so on.

An import is the reverse of an export. The import reads an export file and creates or updates the definition of the items in the export file.

The export and import functions can be used for various reasons, including:

- Restoring a copy of an item after making changes that did not work correctly
- Restoring a copy of an item after an inadvertently changes are made to it
- To duplicate an item from one node to another node.

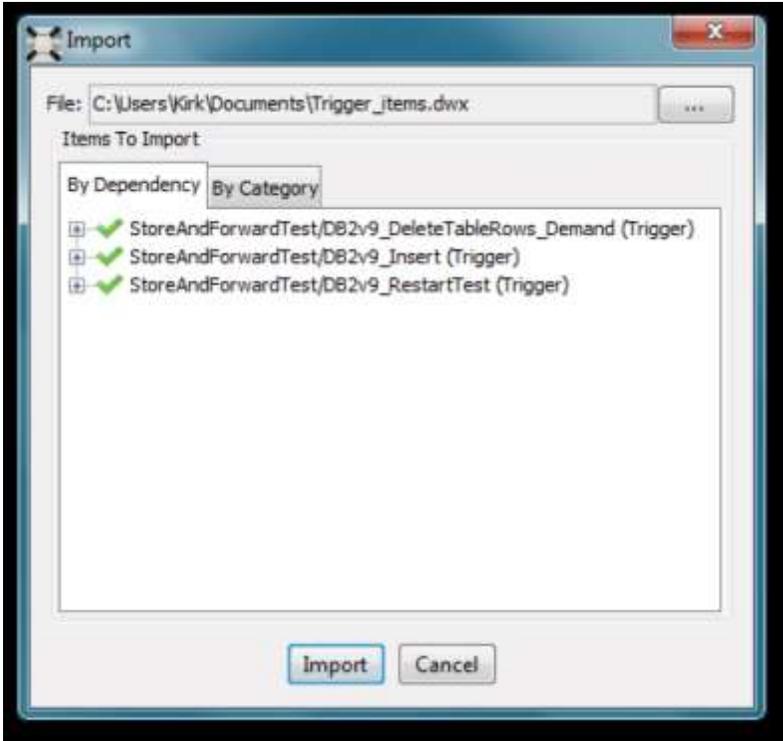
The export and import functions have the concept of an item's dependencies, or other items that are referenced or relied upon. Some examples are:

- A trigger references its project
- A project references all the triggers in it
- A trigger that accesses a device variable references the device
- A trigger that uses a transport map in a Transaction action references the transport map
- A transport map references the transport it uses
- Etc.

Existing Items can be in a Started or Stopped state when imported

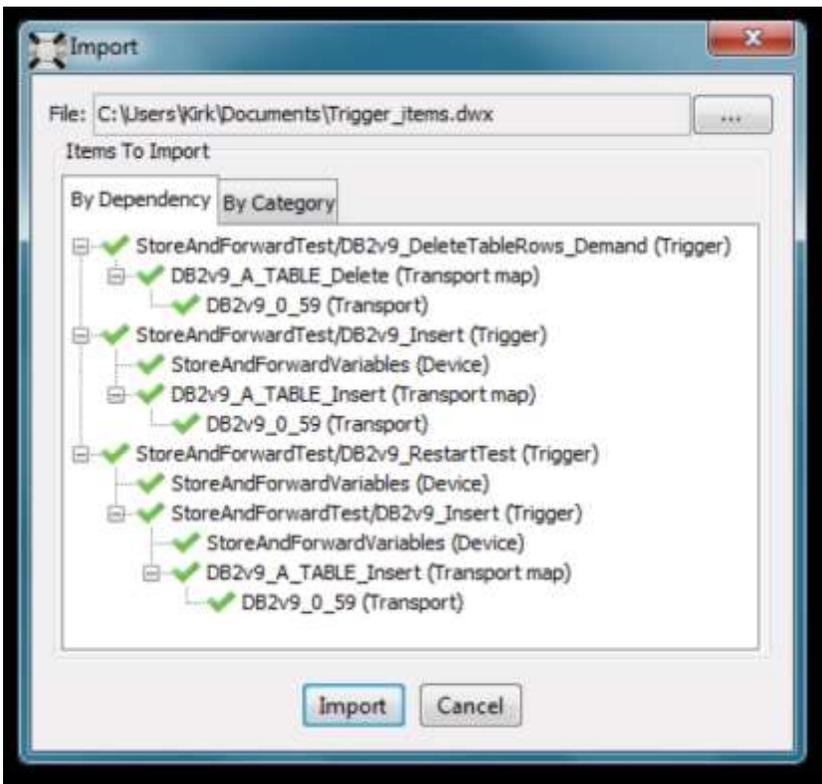
When the **Import** option is selected, a file dialogue is displayed allowing you to select a previously exported .DWX file.

The selected export file is then read and displayed in an **Import** window:



The **Import** window has two tabs, a **By Dependency** tab and a **By Category** tab.

The **By Dependency** tab functions similar to the **Export** function window, where items and their dependencies can be expanded and selected.

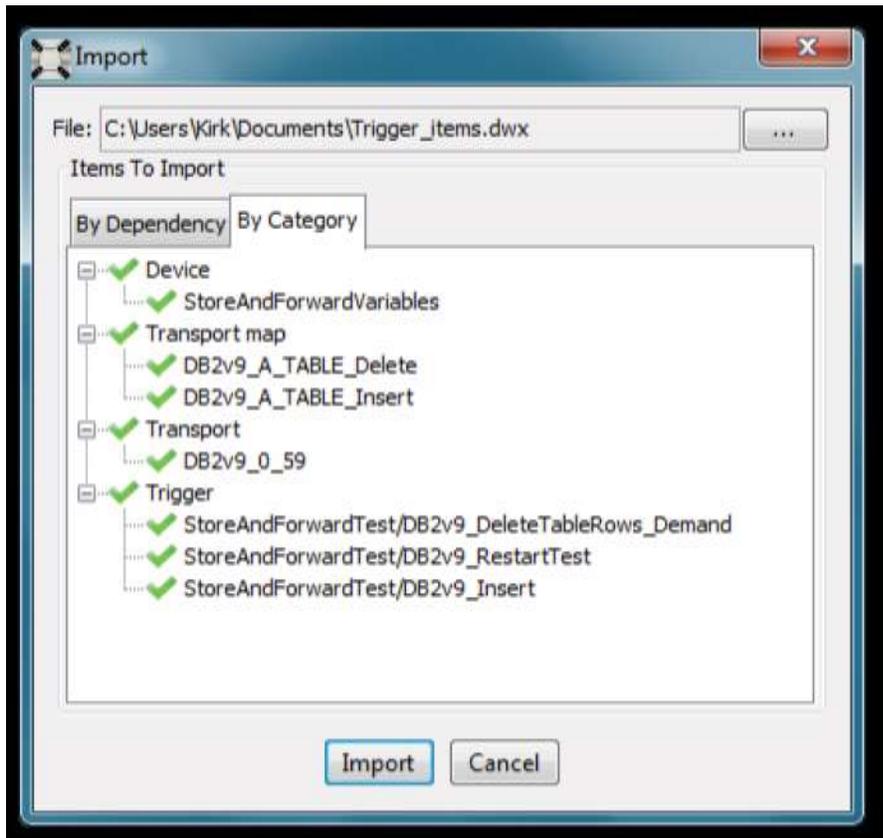


By default, all items and their dependencies are selected for the import as shown by the green check mark icon next to each item.

Items changes be selected or deselected by clicking on its icon.

- A green check mark icon means an item and all of its dependencies are selected for import.
- A grey check mark icon means an item is selected for import, but one or more of its dependencies are not selected.
- A red X mark icon means an item and all of its dependencies are not selected for import.
- A grey check mark icon means an item is not selected for import, but one or more of its dependencies are selected.

The **By Category** tab displays items by type or category for selection or deselection:



When the **By Category** tab is used, an item itself is selected for import. Its dependencies are not referenced or selected automatically.

Related topics

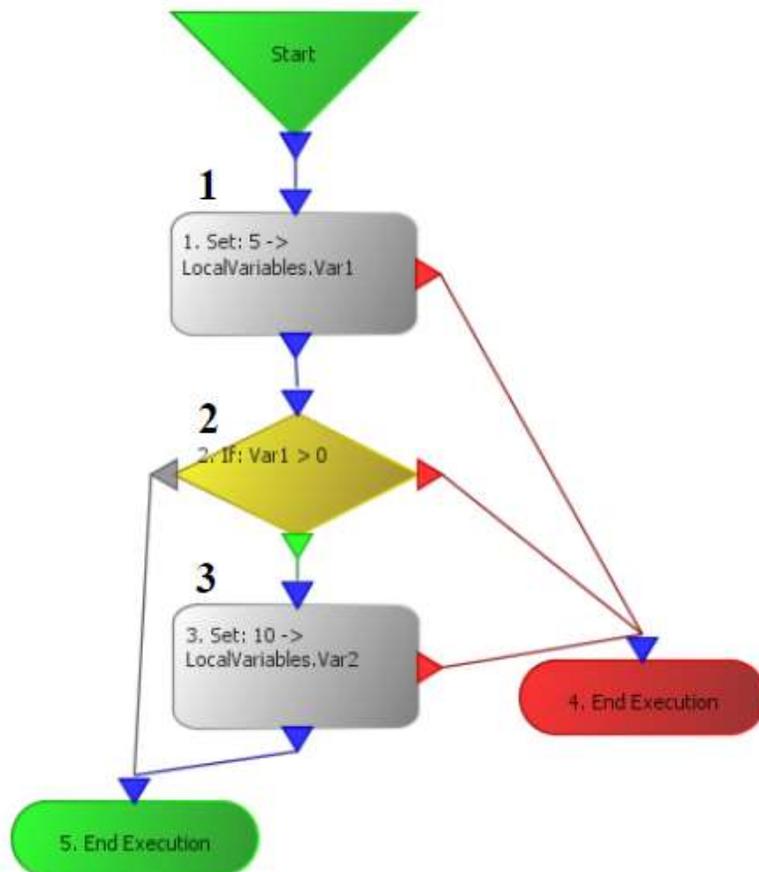
Backing up and Restoring a node's configuration

# IIoTA industrial IoT Platform: Using the Canvas Editor

The trigger Canvas Editor provides a graphical flow chart style tool to define, view and edit triggers.

Both the Canvas Editor and the List Editor provide the same ability to define triggers, they just use a different trigger definition approach: flow chart style (Canvas Editor) or list style (List Editor).

The following example shows a trigger's action diagram.



The simple example trigger is defined with a **Set** action, an **If** action, and a second **Set** action.

The diagram also shows the **Start** block and **End Execution (Success)** and **End Execution (Failure)** actions.

1. The **Start** action block indicates the beginning of the trigger's execution and is routed to the first action in the trigger.  
When the Set action is successful, the output is routed to the next action (the **If** action).

The **Set** action has a red output point used for failure handling.

2. When the **If** expression evaluates to true, the output is routed to the next action (the second **Set** action).

When the **If** expression evaluates to false, the output is routed to **End Execution (Success)**.

The **If** action has a red output point used for failure handling. The **End Execution (Failure)** path is taken when the **If** expression fails (an example would be trying to access a device variable that could not be accessed)

3. When the second **Set** action is successful, the output is routed to **End Execution (Success)**.

The second **Set** action also has a red output point used for failure handling.

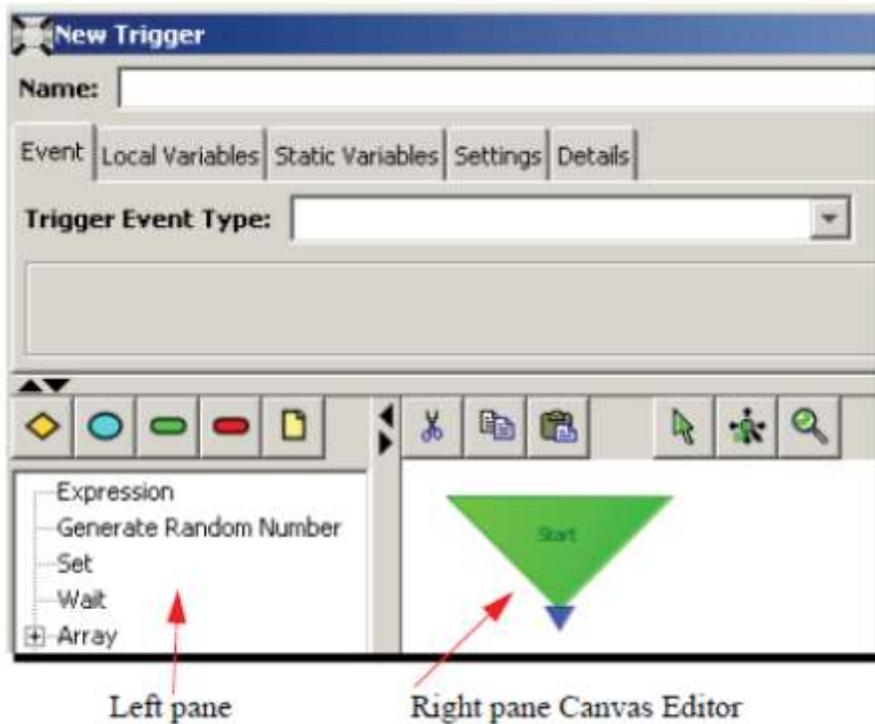
4. The **End Execution (Failure)** action is used by the canvas Editor to indicate the trigger has ended its execution with a **Failure** completion. This is equivalent to the **ENDERR** route when using the List Editor. The trigger statistics on its project tab will have an increase in the **Failures** counter.

5. The **End Execution (Success)** action is used by the canvas Editor to indicate the trigger has ended its execution with a **Success** completion. This is equivalent to the **ENDOK** route when using the List Editor. The trigger statistics on its project tab will have an increase in the **Successes** counter.

The reference sections: Trigger event type reference and Trigger actions reference provide the reference information along with concepts and examples of the available trigger event types and trigger actions.

## IIoTA industrial IoT Platform: Adding actions using the Canvas Editor

The bottom of the right-hand pane of New Trigger window provides the canvas or drawing area of the Canvas Editor.

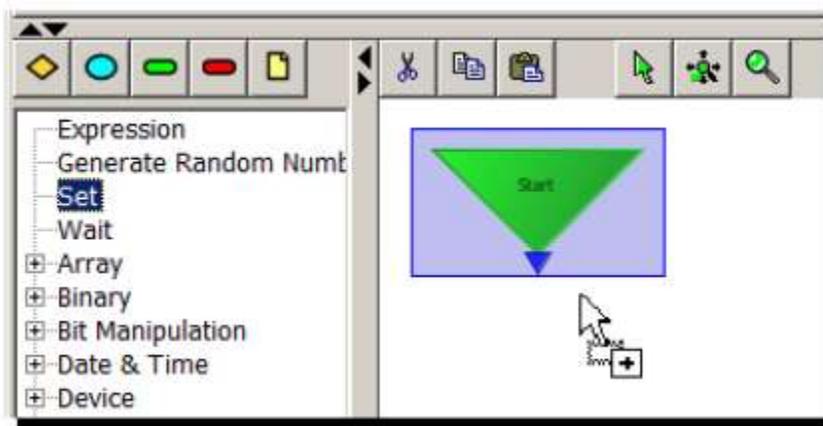


The left-hand pane provides an expandable list of actions that you can drag to the canvas area. The same set of actions are available when using the List Editor.

### Adding parameters to the action

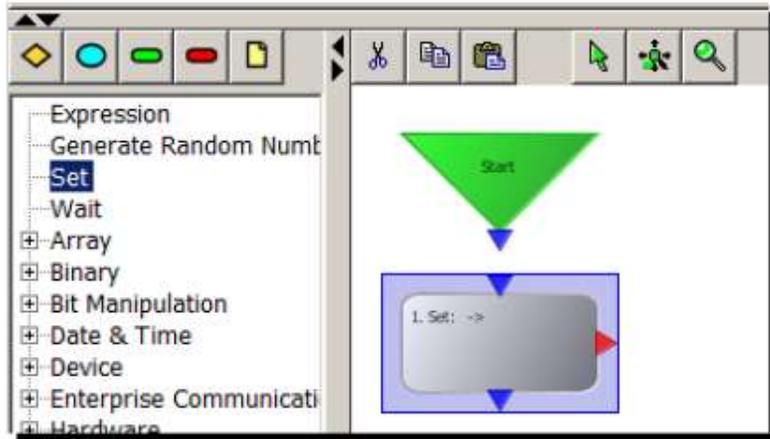
Using the Canvas Editor, you can add actions to a trigger and then specify parameter values as follows:

1. From the Canvas Editor left hand pane, expand a category (if necessary), and then select the action you want to place on the right-hand pane canvas.

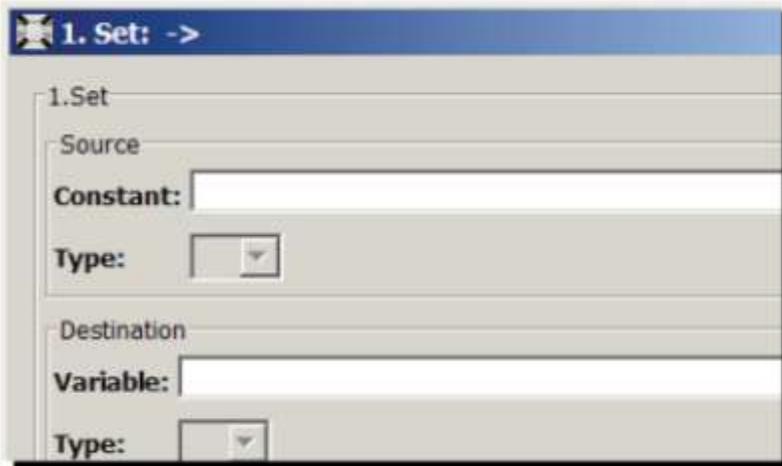


- Position the mouse cursor where you want the action block to appear, and then select with the left mouse button.

For this example, the **Set** action statement appears.

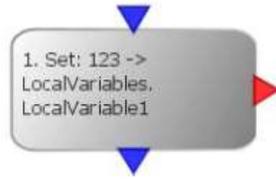


- Double-click the action.  
A window for the action appears.



Note that this is the same window that appears whenever you add an action using the List Editor.

- Specify each parameter value as appropriate, and then select close (X).  
The values appear on the action statement.



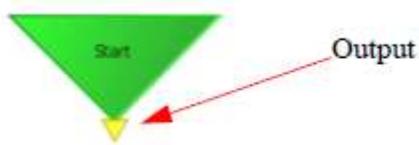
5. To add more actions to the trigger, return to step 1.

### Connecting actions

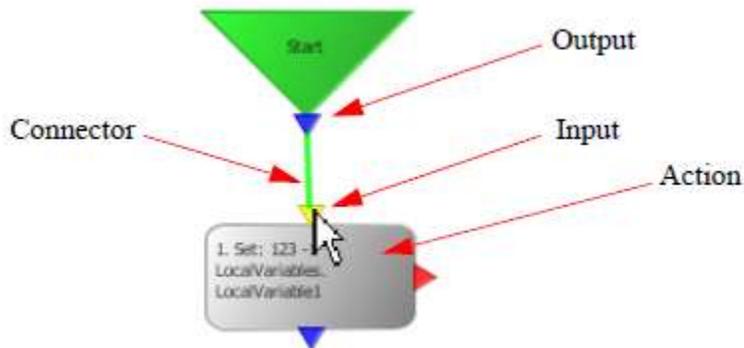
In the Canvas Editor, you draw the routes by connecting the output of one action to the input of another action. This operation makes it easy to visualize the logic flow in the trigger.

For this example, the operation begins with the **Start** statement. Follow these steps:

1. Move the mouse pointer over the Output point.



2. When the output point turns yellow, drag the mouse pointer onto an input for the action.



3. When the input point turns yellow, release the mouse button. The connection is made.

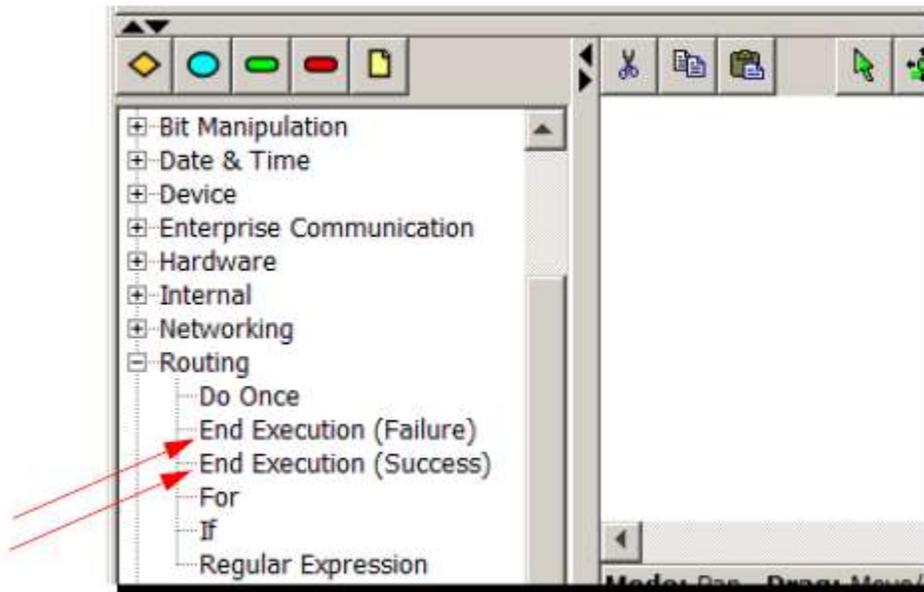
### Ending an action

Every trigger must terminate with at least one **End Execution** action.

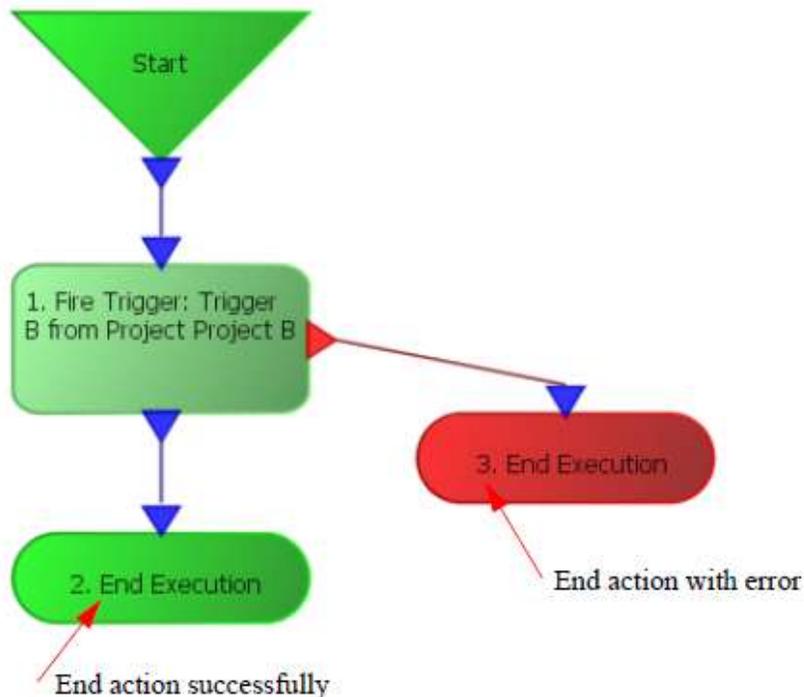
The Canvas Editor provides two end action execution path buttons from the toolbar.



Likewise, you can add an **End Execution (Success)** and **End Execution (Failure)** action from the Canvas Editor left hand pane under the **Routing** category as follows:



A trigger can have more than one **End Execution**, but it must have at least one. The following shows an action diagram configured with **End Execution (Success)** and **End Execution (Failure)** actions.



## End Execution (Success)

**End Execution (Success)** signals that the trigger successfully ended. This route will immediately complete the trigger as a success. There must be at least one end action successfully route for the trigger to end successfully.

To add an **End Execution (Success)** action, follow these steps:

1. From the Canvas Editor left hand pane under the **Routing** category, locate and then expand **Routing**, and then select **End Execution (Success)**.

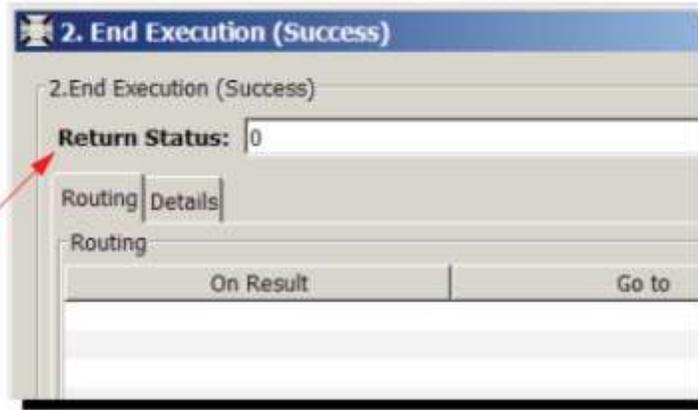
You can also select  from the toolbar.

2. Drag to where you want the action to appear. The mouse pointer changes to a crossbar.
3. Select again.  
The End Execution (Success) action appears on the right hand pane.



4. Double-click the action.

The End Execution (Success) window appears.



5. You can accept the default value of zero for the **Return Status** parameter or type a custom return code value to identify the successful completion of the action. A custom code enables you to identify where the success occurred in the trigger if there are numerous end success exits. When the trigger has reporting turned on, you can view the report (from the **Reports** tab) and then check the return status of the **End Execution (Success)** action. You can also manually run a report for the trigger.

6. Close the window.

Once the trigger is saved and started, you can track its success using the project tab associated with the trigger.

## End Execution (Failure)

**End Execution (Failure)** signals that the trigger ended with errors.

To add an **End Execution (Failure)** action, follow these steps:

1. From the Canvas Editor left hand pane under the **Routing** category, locate and then expand **Routing**, and then select **End Execution (Failure)**.

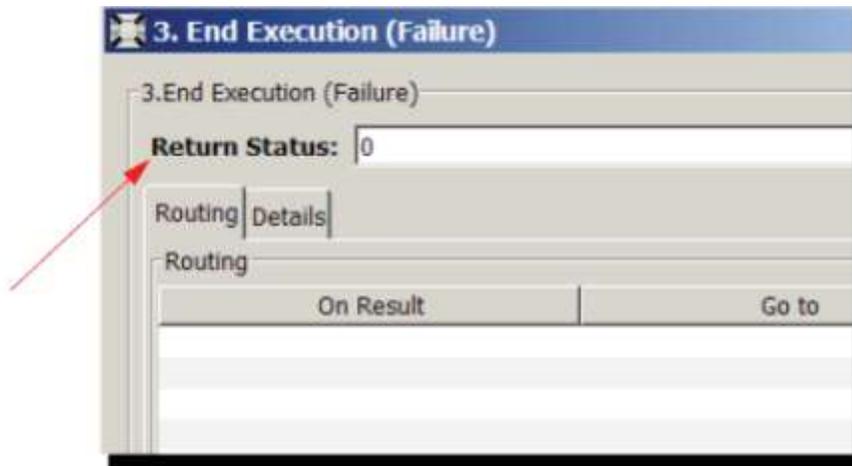
You can also select  from the toolbar.

2. Drag to where you want the action to appear. The mouse pointer changes to a crossbar.
3. Select again.  
The **End Execution (Failure)** action appears on the right-hand pane.



4. Double-click the action.

The End Execution (Failure) window appears.



5. You can accept the default value of zero for the **Return Status** parameter or type a specific error code to identify the error. A custom error code enables you to identify where the error occurred in the trigger if there are numerous end error exits.

When the trigger has reporting turned on, you can view the report (from the **Reports** tab) and then check the return status of the **End Execution (Failure)** action. You can also manually run a report should a failure become apparent on the Failures column for the trigger.

6. Close the window.

## IIoTA industrial IoT Platform: Action statement execution paths

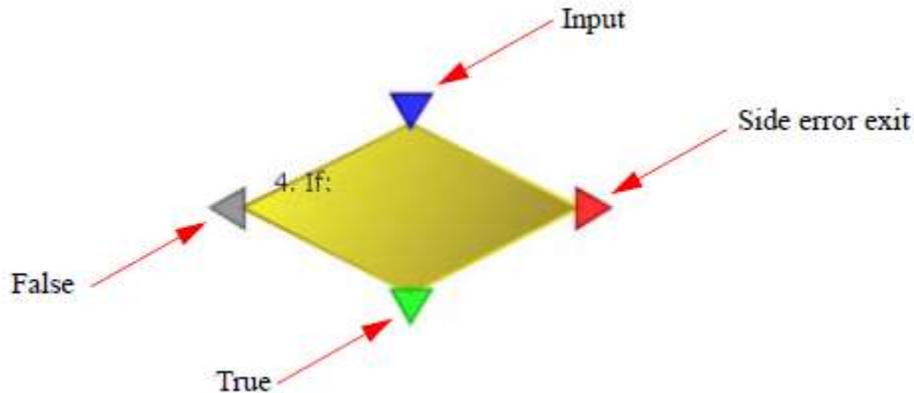
The Canvas Editor uses different shapes and colors to represent different actions.

An action shape will have one input point and one or more output points. The input point

indicates where the flow of control enters the action and the output points indicate where the flow of control leaves the action. This section describes the different action shapes and the corresponding input or output points that are used to specify the actions' routing.

### Example 1 - If

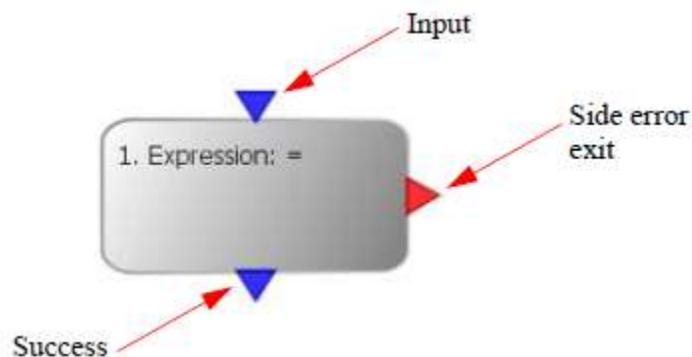
The following shows the **If** action, which provides one input and three output points:



- **Input** - Indicates the input into the action.
- **Side error exit** - Indicates an exit route for error handling. You can specify other actions, or use the **End Execution (Failure)** action, as the exit path.
- **True** - Indicates the route to take when the expression defined in the action evaluates to True.
- **False** - Indicates the route to take when the expression defined in the action evaluates to False.

### Example 2 - Expression

The following shows the **Expression** action, which provides one input and two output points:

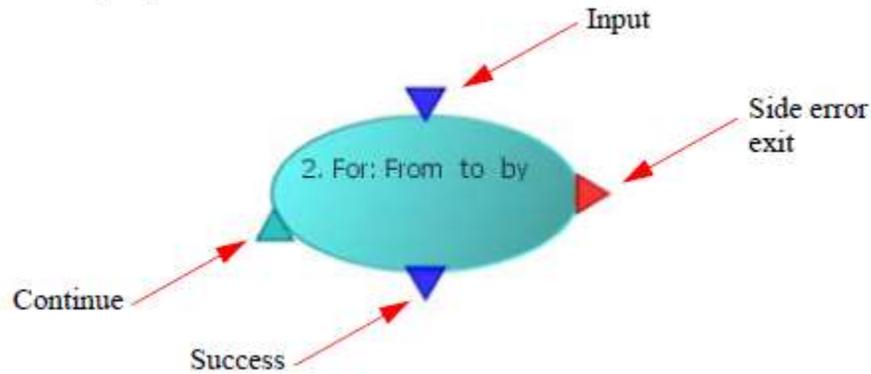


- **Input** - Indicates the input into the action.
- **Side error exit** - Indicates an exit route for error handling. You can specify other actions, or use the **End Execution (Failure)** action, as the exit path.

- **Success** - Indicates the route to take when the expression evaluates successfully, and the output value is stored in the output parameter.

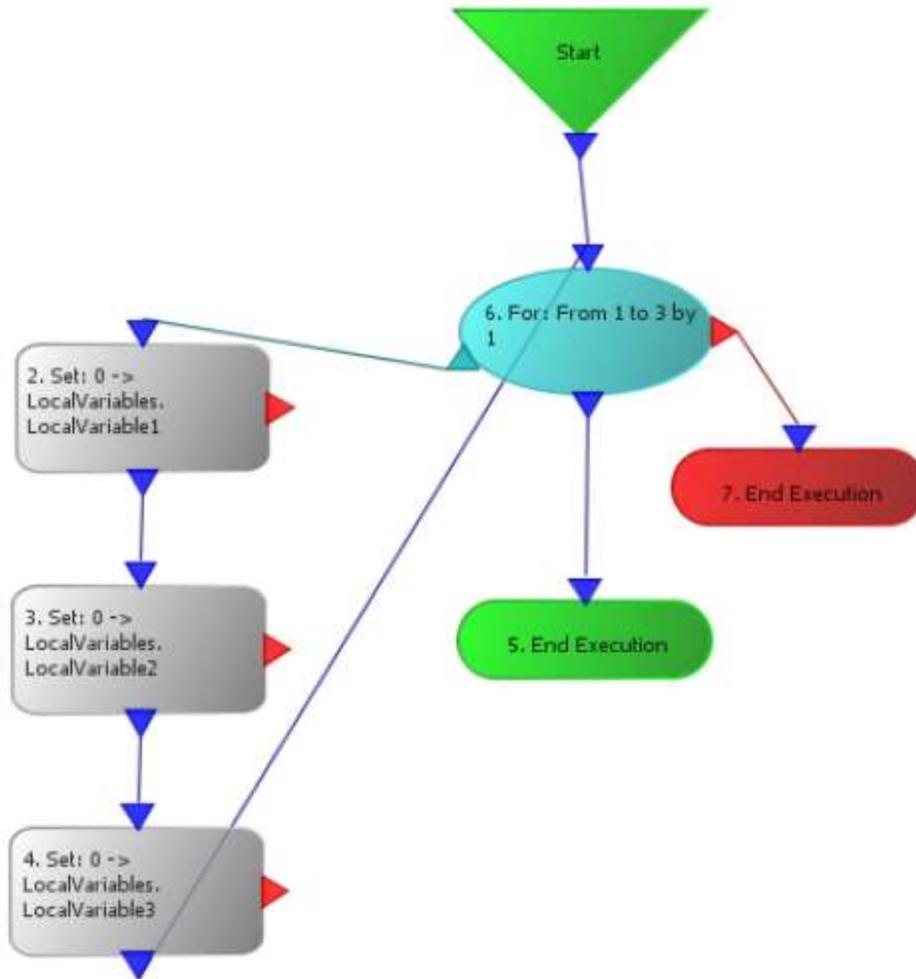
### Example 3 - For

The following shows the **For** action, which provides one input and three output points:



- **Input** - Indicates the input into the action.
- **Side error exit** - Indicates an exit route for error handling. You can specify other actions, or use the **End Execution (Failure)** action, as the exit path.
- **Continue** - Indicates the first route to take within the loop. When an action routes back to the **For** action, the loop will continue and increment the counter. Upon completion of the loop the **Success** route will be taken.
- **Success** - Indicates the route to take upon a successful completion of the **For** action counter.

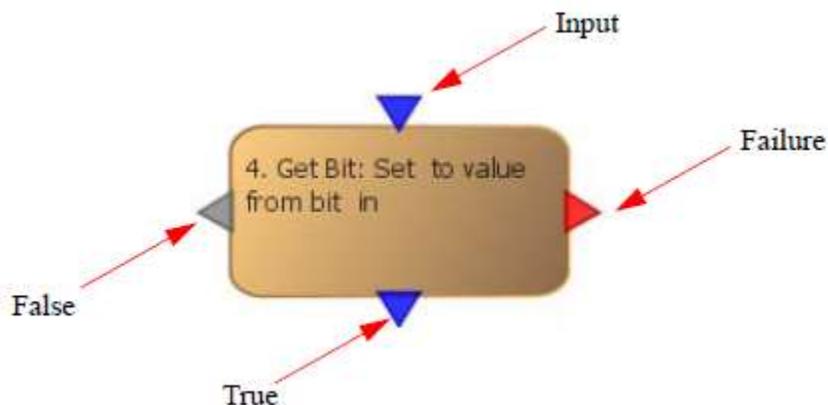
The following example shows a **For** action configured with three **Set** actions.



For this example, the starting number of the For action is set to 1 using a constant. The loop will end execution when the value reaches 3 (also set using a constant).

### Example 4 - Get Bit

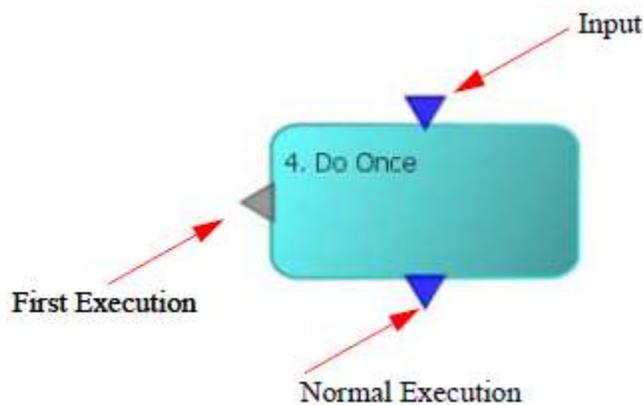
The following shows the **Get Bit** action, which provides one input and three output points.



- **Input** - Indicates the input into the action.
- **Side error exit** - Indicates an exit route for error handling. You can specify other actions, or use the **End Execution (Failure)** action, as the exit path.
- **True** - Indicates the route to take when the expression defined in the action evaluated to True (for the **Get Bit** action, if the bit is one).
- **False** - Indicates the route to take when the expression defined in the action evaluated to False (for the **Get Bit** action, if the bit is zero).

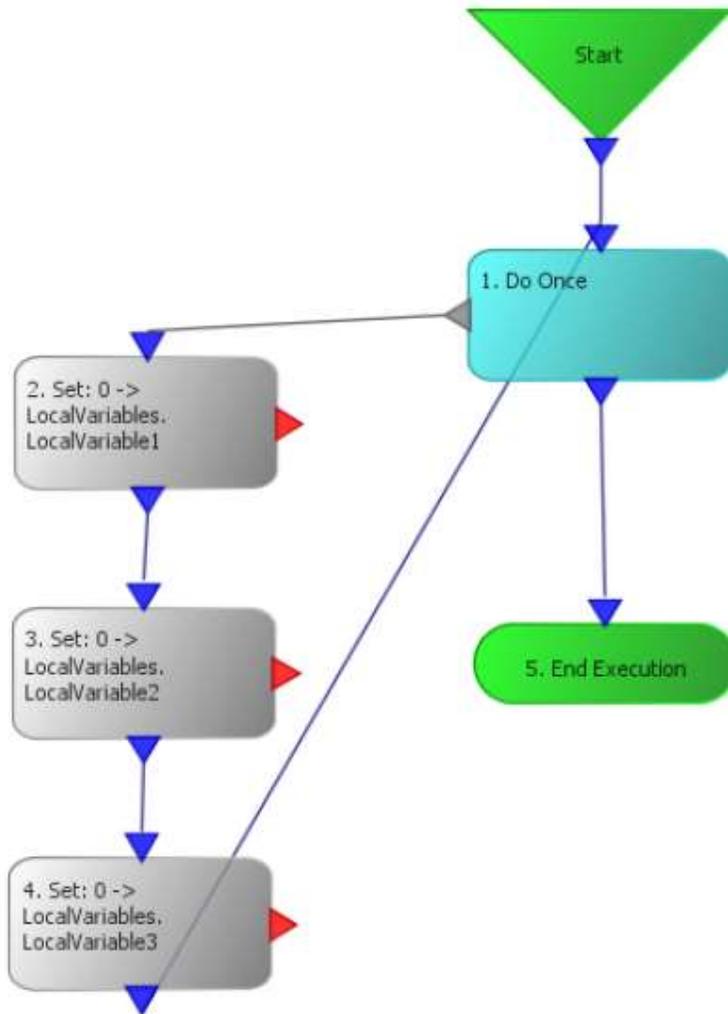
### Example 5 - Do Once

The following shows the **Do Once** action, which provides one input and two output points.



- **Input** - Indicates the input into the action.
- **First Execution** - Indicates the route to take the first time the trigger executes. Subsequent executions of the trigger will bypass this action and move immediately to **Normal Execution**.
- **Normal Execution** - Indicates the route to take for all executions after the first execution.

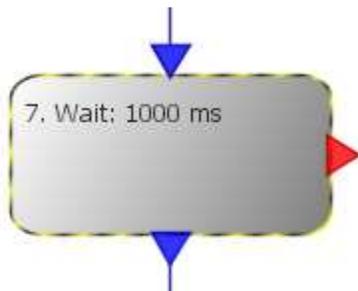
The following shows a **Do Once** action configured with three **Set** actions used for the first execution of the trigger.



For this example, subsequent executions of the trigger will bypass the Set actions and immediately end execution.

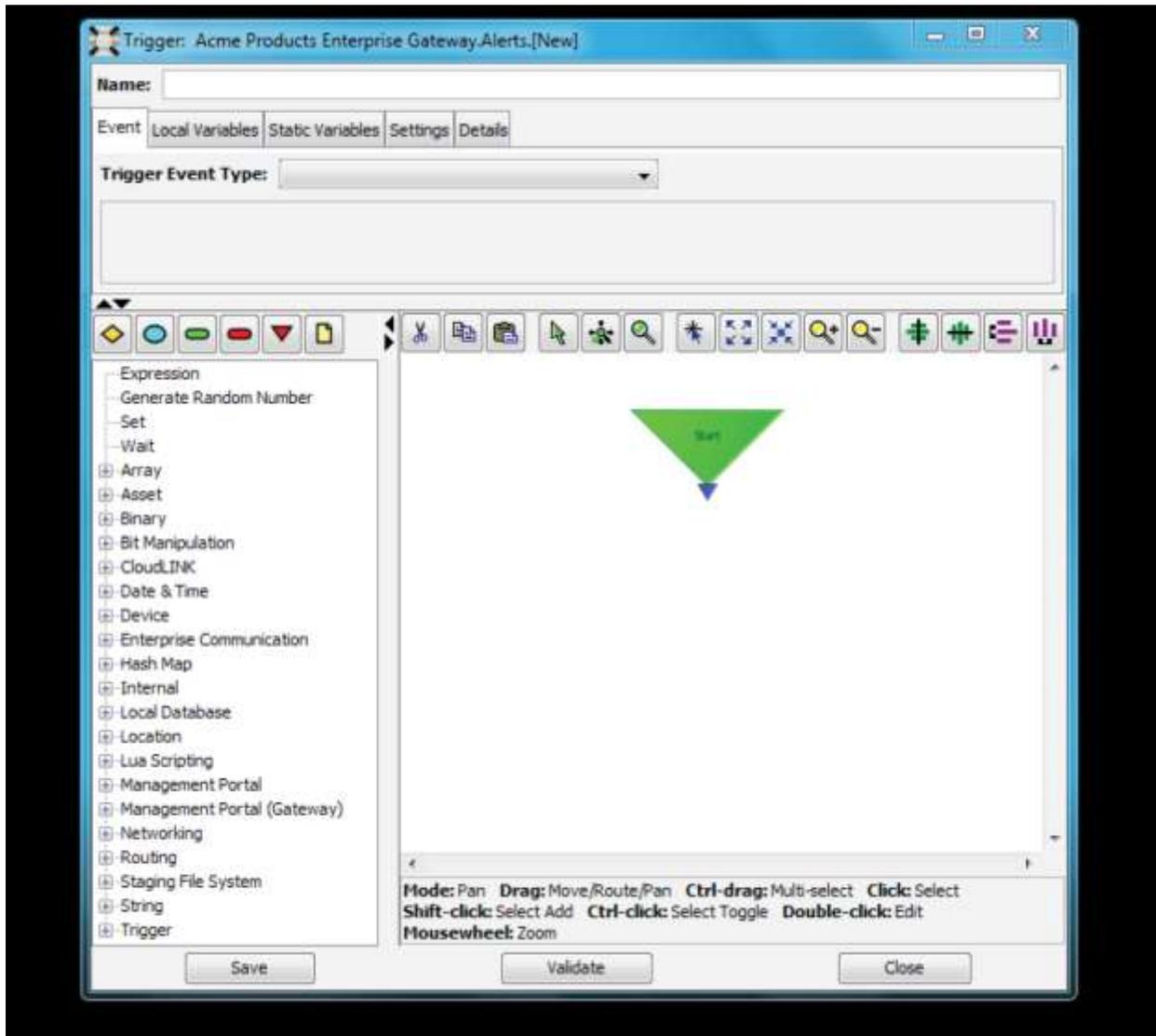
### Disabled Actions

Actions which are disabled (via the context menu command) will appear with a dashed border as shown. These actions will not be performed during execution, instead the action will be skipped, and the execution path will continue down the action's **success** route.



# IIoTA industrial IoT Platform: The Canvas Editor toolbar

The Canvas Editor toolbar is located above the left-hand pane (with the list of actions) and the right-hand pane (with the canvas or drawing area):



## Routing actions and comment form

The following describes the buttons at the far left of toolbar (above the list of actions). They are also available from the list of actions in the left-hand pane under the **Routing** category (except for the comment box).



**If** - Performs a logical comparison. The **If** action is similar to an expression action; however, there is no result value.

	<b>For</b> - Enables a loop within a trigger execution until a certain condition is met.
	<b>End Execution (Success)</b> - Ends the trigger as a Success.
	<b>End Execution (Failure)</b> - Ends the trigger as a Failure.
	<b>Error Handling</b> - Handles all unconnected failure routes for the trigger. Any unconnected failure route will jump to this action. Only one instance of this is allowed per trigger.
	<b>Comment</b> - Used to add a comment to the trigger diagram. You can paste text into the Comment from the clipboard.

### Cut, copy, paste, select, pan, and zoom

The following describes buttons at the left of the toolbar (above the canvas or drawing area).

	<b>Cut</b> - Deletes one or more actions and then copies them to the clipboard.
	<b>Copy</b> - Copies one or more actions to the clipboard.
	<b>Paste</b> - Pastes one or more actions from clipboard.
	<b>Select</b> - Right-click to enclose one or more actions and create a multiple selection. You can copy, paste, and delete the multiple selection.
	<b>Pan</b> - Alters the view of the action. Use the Pan feature when you need more space on the canvas area. With pan, the drawing window automatically scrolls whenever you drag within or beyond its borders.
	<b>Zoom</b> - Increases or decreases the display size of the action statements within the canvas area. Zoom to the size that best suits your work.

### Zoom in and out

The following describes the buttons located at the center of the toolbar.

	<b>Reset Zoom</b> - Returns the action statements to their original size. Whatever was centered on the drawing area remains centered.
	<b>Full Zoom Out</b> - Shrinks the drawing to fit entirely on the drawing area.

	Center Zoom on Selected - When you select one or more actions, the selected actions are enlarged and centered on the drawing area. To return the action to a normal view, use Full Zoom Out.
	Increase Zoom - Increases the size of all the action statements. In order to make the action statements larger and larger, select the Increase Zoom button multiple times until the action statements are the size you want.
	Decrease Zoom - Decreases the size of all the action statements. In order to make the action statements smaller and smaller, select the Decrease Zoom button multiple times until the action statements are the size you want.
<b>Mouse wheel</b>	If you have a scroll wheel on your mouse, you can use the wheel to zoom in and out.

### Alignment and distribute

The following describes the buttons located at the far right of the toolbar (above the canvas or drawing area):

	Center Vertically - Centers selected actions to other actions in a vertical position.
	Center Horizontally - Centers selected actions to other actions in a horizontal position.
	Distribute Vertically - Spaces selected actions equally on a vertical axis.
	Distribute Horizontally - Spaces selected actions equally on a horizontal axis.

### Keyboard shortcuts

The following describes a few keyboard shortcuts that can interact with the Canvas Editor.

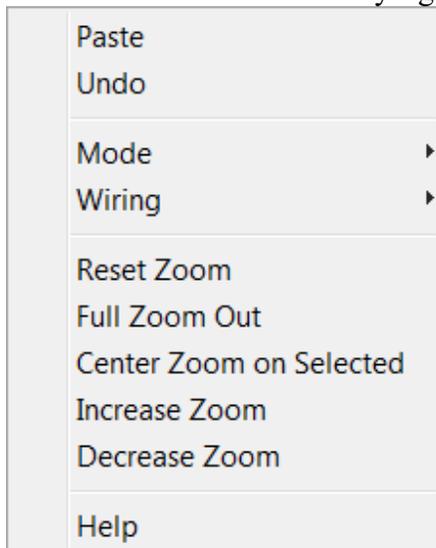
Ctrl+X keys	Delete and copy to the clipboard.
Ctrl+C keys	Copy to the clipboard.
Ctrl+V keys	Paste from the clipboard.
Ctrl+Z keys	Undo.
Ctrl+A keys	Select all.
Del key	Deletes the selected action(s).

# IIoTA industrial IoT Platform: The Canvas Editor

## Context Menus

### Canvas Context Menu

This menu can be accessed by right-clicking in the canvas area.



Paste	Pastes one or more actions from clipboard.
Undo	Undo the last action edit
Mode	Sets the editor into one of three modes - Select, Pan, or Zoom. The status area at the bottom of the canvas will change to detail what mouse and keyboard combinations will perform which functionality.
Wiring	Sets wiring display preference. Direct: straight line from port to port. Right Angle: Vertical/Horizontal lines only with 90 degree turns.
Reset Zoom	Returns the action statements to their original size. Whatever was centered on the drawing area remains centered.
Full Zoom Out	Shrinks the drawing to fit entirely on the drawing area.
Center Zoom on Selected	When you select one or more actions, the selected actions are enlarged and centered on the drawing area. To return the action to a normal view, use Full Zoom Out.

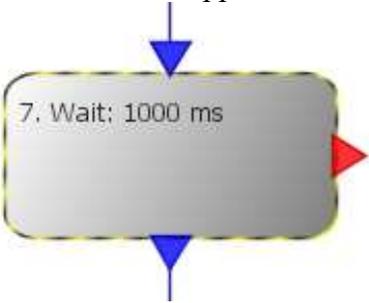
Increase Zoom	Increases the size of all the action statements. In order to make the action statements larger and larger, select the Increase Zoom button multiple times until the action statements are the size you want.
Decrease Zoom	Decreases the size of all the action statements. In order to make the action statements smaller and smaller, select the Decrease Zoom button multiple times until the action statements are the size you want.
Help	Display help

## Action Context Menu

This menu can be accessed by right-clicking on any action.

Cut
Copy
Paste
Undo
Center Horizontally
Center Vertically
Distribute Vertically
Distribute Horizontally
Renumber All Actions
Disable Action
Find Instances
Help

Cut	Deletes one or more actions and then copies them to the clipboard.
Copy	Copies one or more actions to the clipboard.
Paste	Pastes one or more actions from clipboard.
Undo	Undo the last action edit
Center Vertically	Centers selected actions to other actions in a vertical position.
Center Horizontally	Centers selected actions to other actions in a horizontal position.
Distribute Vertically	Spaces selected actions equally on a vertical axis.

Distribute Horizontally	Spaces selected actions equally on a horizontal axis.
Renumber All Actions	Renumbers all the actions
Disable Action	<p>Disables selected actions. These actions will be skipped over during trigger execution. A disabled action appears with a dashed border as shown:</p>  <p><b>Start</b> and <b>Error Handling</b> actions cannot be disabled.</p>
Find Instances	Find instance of selected actions used in other triggers

## IIoTA industrial IoT Platform: Trigger event type reference

### Overview

The main concepts of a trigger are:

- The trigger's event type
- The trigger's local variables, static variables, macros and event variables
- The trigger's settings
- The trigger's actions, including the success and failure routes between actions.

Every trigger identifies the trigger event type, which identifies when the trigger will be executed.

The information in this Trigger event type reference section covers the different trigger event types available across the different products. While the trigger event type identifies when the trigger is executed, the actions used in a trigger define the application logic. The trigger actions are covered in the Trigger actions reference. The general project and trigger concepts are covered in the first few sections of Projects and triggers.

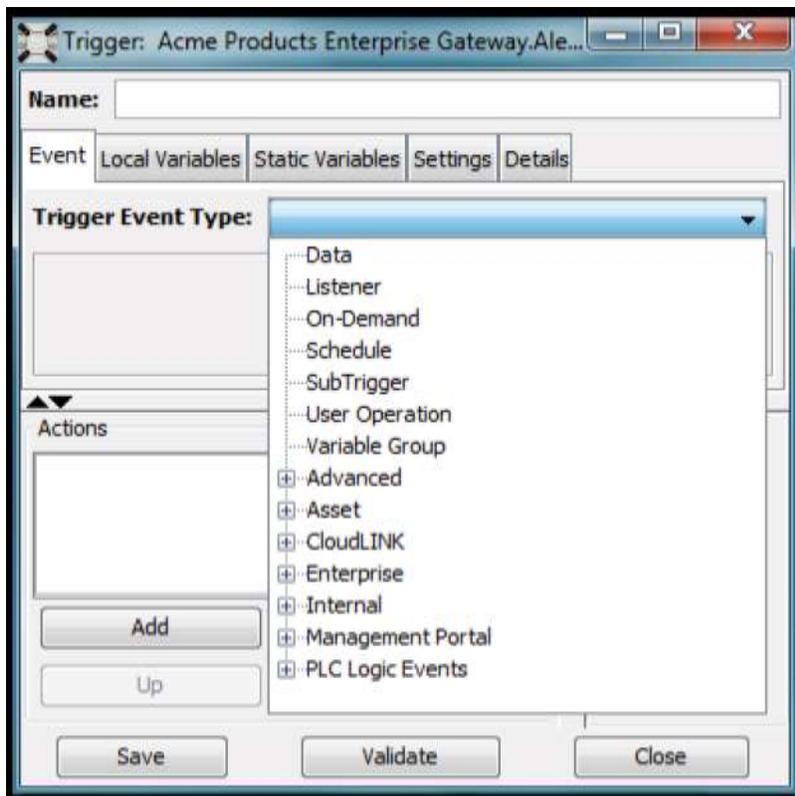
## Assumptions

Before using information in this Trigger event type reference section, the following should have occurred:

- The Workbench was installed on a computer that has TCP connectivity to the node.
- You have a user ID and password to log on and use features of the Workbench.
- You understand how to create a trigger.

## How this reference is organized

The trigger event types are described in the following sections in the order that they are displayed in the **Trigger Event Type** list in the Workbench.



## IIoTA industrial IoT Platform: Data

A **Data** event trigger executes when a device variable's data value meets a defined condition.

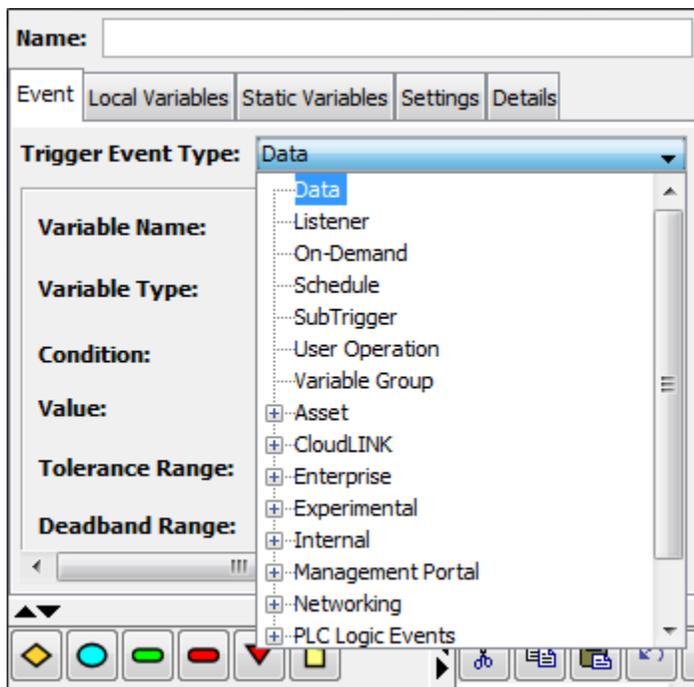
The conditions available include being equal to, greater than or less than a value. The conditions also include concepts of dead band and tolerance.

The trigger component handles the periodic reading of the device variable's value and applying the defined condition. If the condition is met, the data event trigger is scheduled to execute. This alleviates the trigger's application logic from reading the variable's value and comparing it to the condition before deciding to execute the application logic in the trigger's actions.

## Defining a data event trigger

To define a data event trigger, follow these steps:

1. From the Workbench left pane, expand the node where you want to define the data event trigger.
2. Select the **Project** icon to display the **Projects** window, right click a specific project tab to display its pop-up menu, and then select **New**.  
You also can select the **New** button at the bottom of the **Projects** window when a specific project tab has already been selected.
3. The new **Trigger** window appears.  
Name the trigger. The trigger name can be up to 64 characters and include letters, numbers, and the underscore character. Spaces are allowed.
4. From the **Event** tab, select the **Trigger Event Type** down-arrow, and then select **Data**.



The **Event** tab becomes active with parameters that accommodate the data event.

The screenshot shows a configuration window for an event. At the top, there is a 'Name' field. Below it are tabs for 'Event', 'Local Variables', 'Static Variables', 'Settings', and 'Details'. The 'Event' tab is selected. Under 'Trigger Event Type', a dropdown menu is set to 'Data'. The 'Variable Name' field is empty. To its right, 'Priority (ms)' is set to 500. Below these are 'Variable Type' (with a dropdown), 'Count' (with a text input), 'Condition' (set to 'Value changed'), 'Value' (with a dropdown), 'Tolerance Range' (with a dropdown), and 'Deadband Range' (with a dropdown). An 'On Edge' checkbox is present and unchecked.

5. Select the device variable that will be monitored to determine when the trigger should be executed.

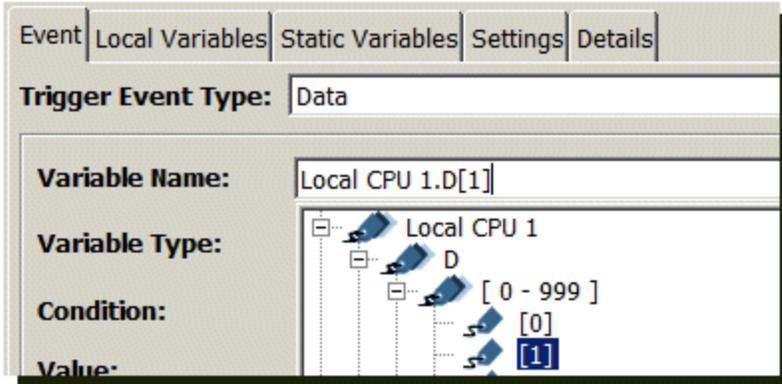
On the **Event** tab, use the down-arrow next to **Variable Name**.

A list of the currently started devices on this node appears.

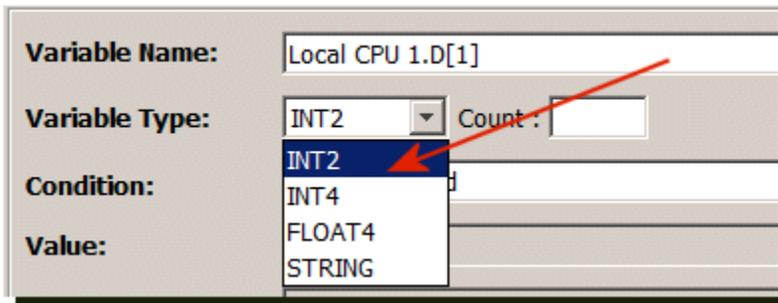
The screenshot shows a 'New Trigger' dialog box. The 'Name' field contains 'MyDataTrigger'. The 'Event' tab is selected. 'Trigger Event Type' is set to 'Data'. The 'Variable Name' field has a dropdown arrow. Below it, a list of device variables is shown: 'ControlLogix\_1\_67', 'Local CPU 1', and 'StoreAndForwardVariables'. Each item has a small icon and a plus sign to its left. The 'Condition' field is empty.

6. Expand the list, and then select the variable you want to use.  
For this example, Local CPU 1.D[1].

The name of the device variable is displayed in the **Variable Name** parameter.



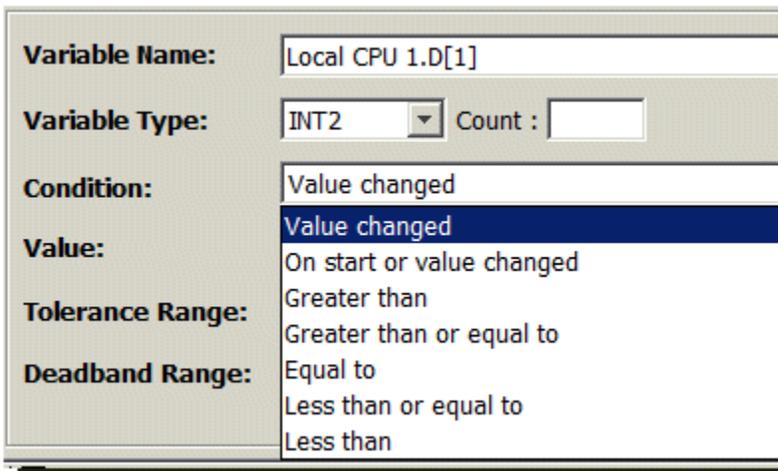
- Use the **Variable Type** down-arrow to display a list of device data types. For this example, accept the default INT2.



- Now that you have added the device variable and specified the variable data type, your next step is to select the condition to be evaluated.

Use the **Condition** down-arrow to display a list of conditions.

For this example, **Value changed** is selected. Therefore, whenever the Local CPU 1.D[1] device variable 's value changes, the trigger will execute.



The condition can be specified as follows:

Condition	Description
Value changed	The value of a device variable changed from its previous value.
On start or value changed	The value of the device variable changed from its previous value. Also, the trigger is executed once as soon as it is loaded.
Greater than	The device variable is greater than a defined value.
Greater than or equal to	The device variable is greater than or equal to a defined value.
Equal to	The device variable is equal to a defined value.
Less than or equal to	The device variable is less than or equal to a defined value.
Less than	The device variable is less than a defined value.

- When **Value changed** or **On Start or Value Changed** are specified as the condition, the **Value**, **Tolerance Range**, and **Deadband Range** parameters are not used.
- When **Equal to** is specified as the condition, the **Tolerance Range**, and **Deadband Range** parameters are not used.

9. Use the **Priority** parameter to select a value in milliseconds. The **Priority** parameter is used to specify the frequency to read the value of the device variable. Values are in milliseconds.

For this example, the device variable is read every 500 milliseconds and evaluated against the condition.

For this example, the condition is Value Changed from the last value read.

The screenshot shows a configuration window for a data event trigger. The fields are as follows:

- Variable Name:** Local CPU 1.D[1]
- Variable Type:** INT2
- Count:** (empty field)
- Condition:** Value changed
- Value:** (empty dropdown)
- Tolerance Range:** (empty dropdown)
- On Edge:**
- Deadband Range:** 0
- Priority (ms):** 500

10. For other conditions, select values for the remaining condition parameters:

Parameter	Description
On Edge	Limits the number of times a trigger executes. Used in conjunction with Greater than and Less than conditions and Tolerance Range.
Tolerance Range	Limits the number of times the trigger executes. The interpretation of the value is based on the selected event condition (Greater than, Less than, Greater than or equal to, and Less than or equal to). The value that you specify will indicate how much you want to allow the value of the device variable to fluctuate before the system acts on it. The value can be either a constant value or device variable.
Deadband Range	Specify a deadband range value when you want to filter insignificant changes. If the value read does not differ from the last stored value by at least the deadband range value, the trigger is not executed. The value can be either a constant value or device variable. Works in conjunction with Value changed, On Start or Value Changed, Greater than, Less than, Greater than or equal to, and Less than or equal to conditions.
Value	A numeric value used in conjunction with <b>Greater than, Less than, Equal to, Greater than or equal to, Less than or equal to</b> conditions. The value can be either a constant value or device variable.

## Data event type trigger event variables

The input event variables available to a data event trigger are:

Event variable	Data type	Description
DATA	The data type of the device variable	The data value from the device variable.

PREVIOUS_DATA	The data type of the device variable	The data value from the device variable on the previous read.
PREVIOUS_TIME	TIMESTAMP	The date and time from the previous read.

## Complete the trigger definition

Complete the definition of the trigger, including the follow trigger components:

- The trigger's actions
- The trigger's local variables, static variables, macros and event variables
- The trigger's settings.

Use the **Validate** button to check the parameters and then the **Save** button to save the trigger's definition.

The trigger will be listed in the Project's tab list of triggers in the **Stopped** state.

## Data event trigger priority parameter considerations

The **Priority** parameter is used to specify the frequency, in milliseconds, to read the value of the device variable. Since this reading of the device variable takes system resources, the requirements of the application and the capability of the system need to be understood. Considerations include:

- Every data event trigger defines a device variable that will be read by the system at the defined priority (frequency).
- Every data mapping defines a device variable that will be read by the system at the defined priority (frequency).
- Every device variable has an expected change rate. This may be constant, or it may be variable. For example:
  - Once every 5 minutes at the end of a manufacturing process is relatively constant.
  - Every time a sensor's temperature value changes .1 degrees is most likely a variable rate.
- The application logic's latency requirement for the data event's condition being met. For example:
  - The end of the 5-minute manufacturing process needs to be recognized within 10 seconds.
  - The change in the temperature value needs to be recognized within 1 second.

- The more frequent the device variable is read, (a lower priority value) the lower the latency for the data event trigger's condition being met and the data trigger logic being executed.
- The more frequent the device variable is read, (a lower priority value) the higher the load on system resources.

The overall application's requirements and the capability of the system need to be monitored while the application is being developed and while it is in production.

## IIoTA industrial IoT Platform: Example data event type trigger conditions

The following examples show different conditions for a data event trigger.

### Example: Greater than and Tolerance Range

For a data event trigger using the **Greater than** condition, the trigger will execute every time the device variable is greater than the defined value.

You can use the **On Edge** option with a corresponding **Tolerance Range** to limit the overall condition being met to include the concept of resetting the event condition.

The screenshot shows a configuration window for an event trigger. The 'Trigger Event Type' is set to 'Data'. The 'Variable Name' is 'Local CPU 1.D[5]' and the 'Variable Type' is 'INT2'. The 'Condition' is 'Greater than', the 'Value' is '10', and the 'Tolerance Range' is '3'. The 'On Edge' checkbox is checked. Red circles highlight the 'Condition', 'Value', 'Tolerance Range', and 'On Edge' fields.

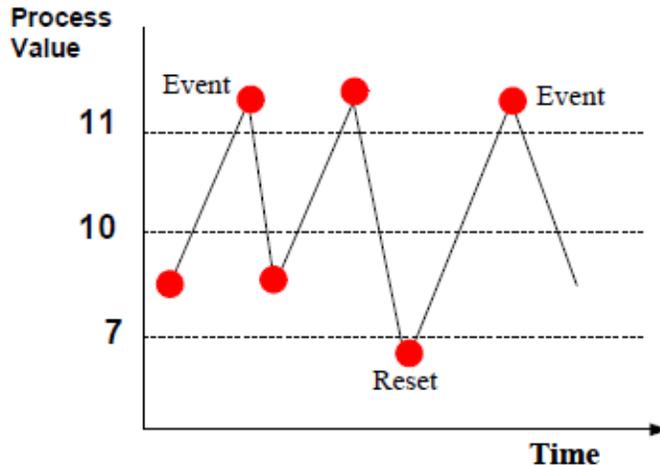
**Example:** The condition is **Greater than**, the value is 10, and a tolerance range of 3.

### Explanation:

- The trigger will execute when the condition is true,  $\text{Local CPU1.D}[5] > 10$ .

- Since a tolerance range is used, the trigger will not execute again until it is reset.
- The trigger will reset when the Local CPU 1.D[5] reaches the condition value minus the tolerance range value, which in this case is 7 (10 minus 3).

The following illustrates **Greater than**, the value is 10, and a tolerance range of 3:



Where:

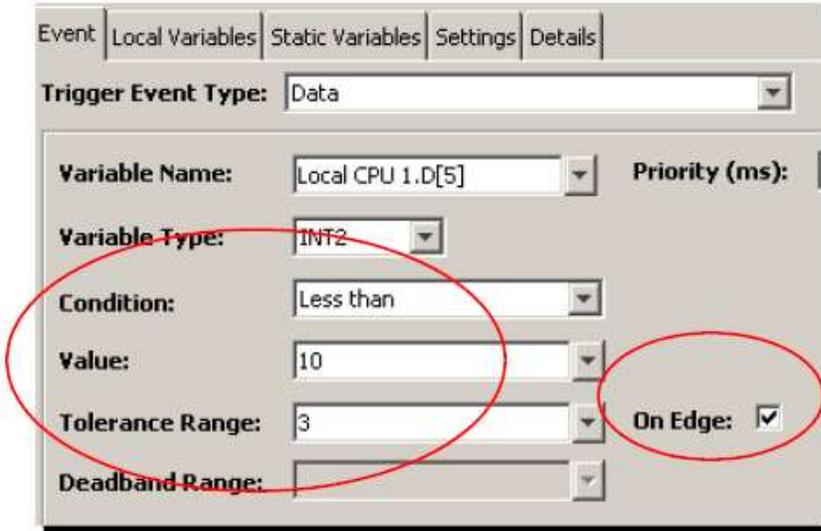
-  are sampled values of Local CPU 1.D[5].
- Event indicates the trigger will execute.
- Reset indicates the trigger event condition has been reset and the trigger will execute again when Local CPU 1.D[5] > 10.

If you do not specify a Tolerance Range or a Deadband Range with the **Greater than** condition, the trigger will execute every time the variable is greater than the defined value.

### Example: Less than and Tolerance Range

For a data event trigger using the **Less than** condition, the trigger will execute every time the device variable is less than the defined value.

You can use the **On Edge** option with a corresponding **Tolerance Range** to limit the overall condition being met to include the concept of resetting the event condition.

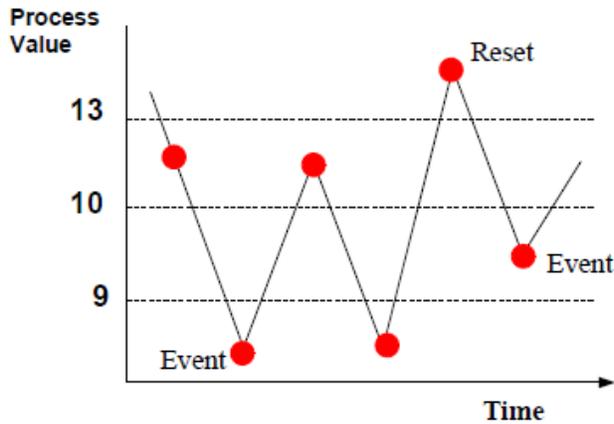


**Example:** The condition is **Less than**, the value is 10, and a tolerance range of 3.

**Explanation:**

- The trigger will execute when the condition is true,  $\text{Local CPU1.D}[5] < 10$ .
- Since a tolerance range is used, the trigger will not execute again until it is reset.
- The trigger will reset when the  $\text{Local CPU 1.D}[5]$  reaches the condition value plus the tolerance range value, which in this case is 13 (10 plus 3).

The following illustrates **Less than**, the value is 10, and a tolerance range of 3.



Where:

- are sampled values of  $\text{Local CPU 1.D}[5]$ .
- Event indicates the trigger will execute.

- Reset indicates the trigger event condition has been reset and the trigger will execute again when Local CPU 1.D[5] < 10.

If you do not specify a Tolerance Range or a Deadband Range with the **Less than** condition, the trigger will execute every time the variable is less than the defined value.

### Example: Equal to

For a data event trigger using the **Equal to** condition, the trigger will execute every time the device variable is equal to the defined value.

The screenshot shows a configuration window for a trigger event. The 'Trigger Event Type' is set to 'Data'. The 'Variable Name' is 'Local CPU 1.D[5]' and the 'Variable Type' is 'INT2'. The 'Condition' is set to 'Equal to' and the 'Value' is '10'. The 'Tolerance Range' and 'Deadband Range' are empty. The 'On Edge' checkbox is unchecked. A red circle highlights the 'Condition' and 'Value' fields.

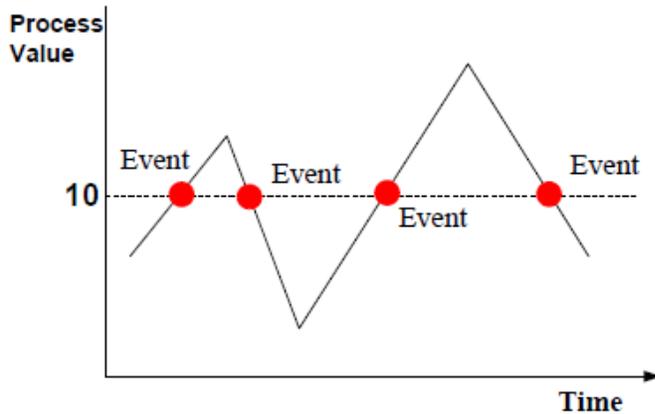
**Example:** The condition is **Equal to** and the value is 10.

### Explanation:

- In the case of an **Equal to** condition, the event condition will be reset on either side of the condition.

Note that **On Edge** is always on when the condition is set to **Equal to**.

The following illustrates **Equal to** and the value is 10.



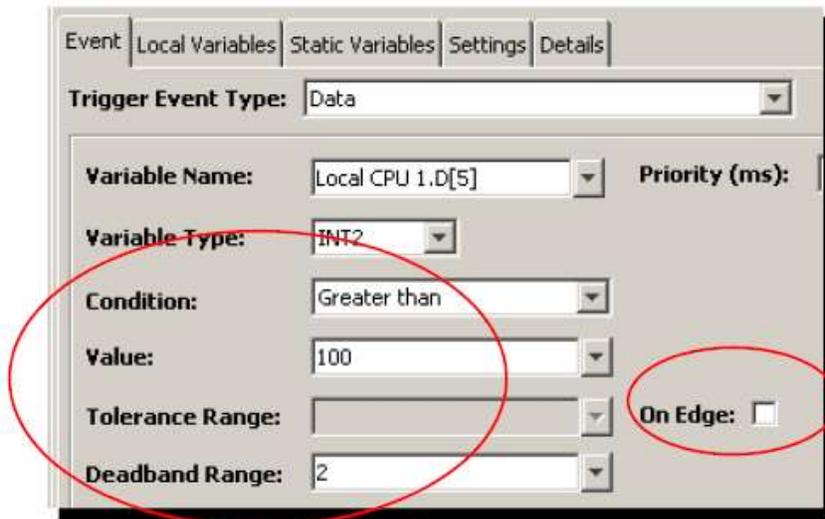
Where:

- are sampled values of Local CPU 1.D[5].
- Event indicates the trigger will execute.
- The condition resets when  $D[5] = 10$ .

### Example: Greater than and Deadband Range

For a data event trigger using the **Greater than** condition, the trigger will execute every time the device variable is greater than the defined value.

You can use the **On Edge** option with a corresponding **Deadband Range** to limit the overall condition being met to include the concept of resetting the event condition.

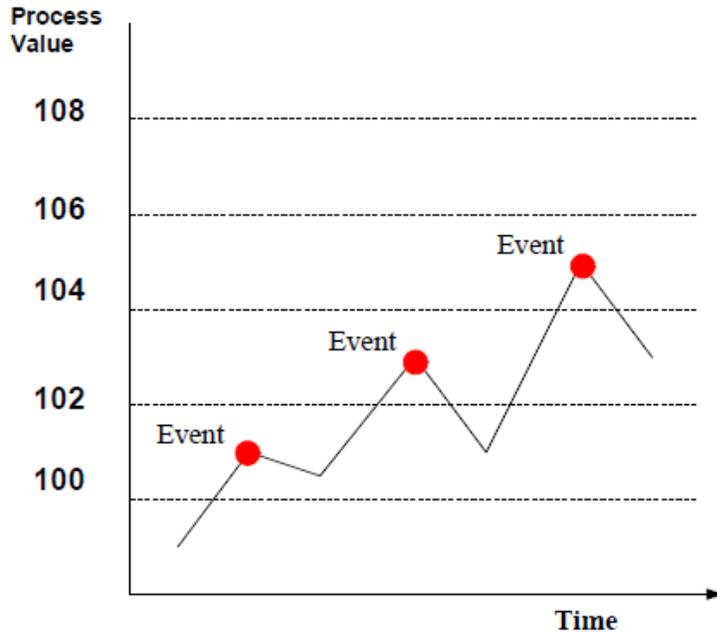


**Example:** The condition is **Greater than**, the value is 100, and a deadband range of 2.

**Explanation:**

- If Local CPU 1.D[5] = 101, then the trigger will execute since the condition is true. Then if Local CPU 1.D[5] assumes another value where current Local CPU 1.D[5] - previous Local PU 1.D[5] > 2, the trigger will execute.

The following illustrates **Greater than**, the value is 100, and a deadband range of 2.



If you specify a **Deadband Range** value instead of a **Tolerance Range** value, the trigger will execute if the difference between any 2 successive values exceeds the deadband range value. The deadband range value is an absolute value and not a percentage.

## IIoTA industrial IoT Platform: Listener

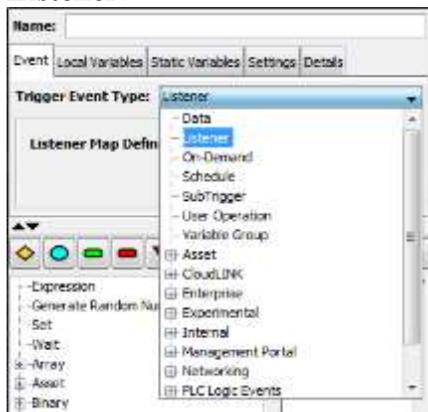
A **Listener** event trigger executes when a message arrives from an external application. A Listener event trigger is part of a listener component and can optionally send data back to the external application in a reply message. See Listeners for more information on the listener feature.

### Defining a listener event trigger

To define a listener event trigger, follow these steps:

1. From the Workbench left pane, expand the node where you want to define the listener event trigger.

2. Select the **Project** icon to display the **Projects** window, right click a specific project tab to display its pop-up menu, and then select **New**.  
You also can select the **New** button at the bottom of the **Projects** window when a specific project tab has already been selected.
3. The new **Trigger** window appears.  
Name the trigger. The trigger name can be up to 64 characters and include letters, numbers, and the underscore character. Spaces are allowed.
4. From the **Event** tab, select the **Trigger Event Type** down-arrow, and then select **Listener**.



The **Event** tab becomes active with parameters that accommodate the listener event.

Parameter	Description
<b>Listener Map Definition</b>	Select a listener map from the list of listener maps currently defined on the node. The listener map contains the definition of the input message's format and variables, and any optional output message's format and variables.

## Listener event type trigger event variables

The event variables available to a listener event trigger are based on the definition of the listener map:

Event variable	Data type	Description
Input variables	The data type of the listener map input variable	Each listener map input variable will be available as an input event variable in the listener event trigger. These input event variables can be used as the source variable in the trigger's actions.

Output variables	The data type of the listener map out variable	If the corresponding listener definition includes the Send reply message option, then the listener map can define output variables. Each listener map output variable will be available as an output event variable in the listener event trigger. These output event variables can be used as the destination variable in the trigger's actions.
------------------	--	---

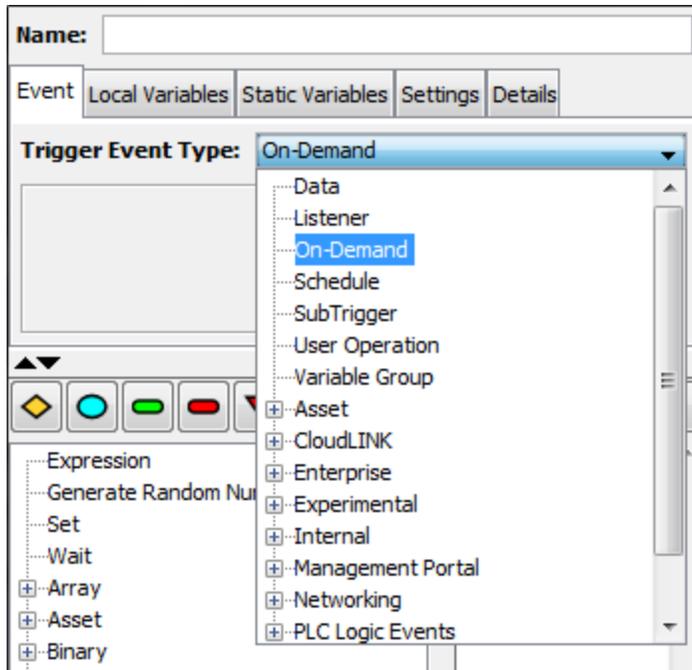
## IIoTA industrial IoT Platform: On-Demand

An **On-Demand** event trigger executes when the Fire Trigger option is used from the Workbench Projects window or from another trigger using the Fire Trigger action.

### Defining an on-demand event trigger

To define an on-demand event trigger, follow these steps:

1. From the Workbench left pane, expand the node where you want to define the on-demand event trigger.
2. Select the **Project** icon to display the **Projects** window, right click a specific project tab to display its pop-up menu, and then select **New**.  
You also can select the **New** button at the bottom of the **Projects** window when a specific project tab has already been selected.
3. The new **Trigger** window appears.  
Name the trigger. The trigger name can be up to 64 characters and include letters, numbers, and the underscore character. Spaces are allowed.
4. From the **Event** tab, select the **Trigger Event Type** down-arrow, and then select **On-Demand**.



The **Event** tab becomes active for the on-demand event, but there are no additional parameters to define.

## On-Demand event type trigger event variables

There are no event variables for an on-demand event trigger.

## Complete the trigger definition

Complete the definition of the trigger, including the following trigger components:

- The trigger's actions
- The trigger's local variables, static variables, macros and event variables
- The trigger's settings.

Use the **Validate** button to check the parameters and then the **Save** button to save the trigger's definition.

The trigger will be listed in the Project's tab list of triggers in the **Stopped** state.

## IIoTA industrial IoT Platform: Schedule

A **Schedule** event trigger executes when a schedule frequency (type of schedule) and occurrence criteria are met.

The schedule frequency options include: Periodic, Hourly, Day of Month and Weekdays.

The occurrence criteria are specific to the schedule frequency.

The trigger component handles the calculation of the schedule frequency and occurrence criteria. If the occurrence criteria are met, the schedule event trigger is scheduled to execute.

This alleviates the trigger's application logic from determining the frequency and occurrence criteria since the last instance of the trigger's execution before deciding to execute the application logic in the trigger's actions.

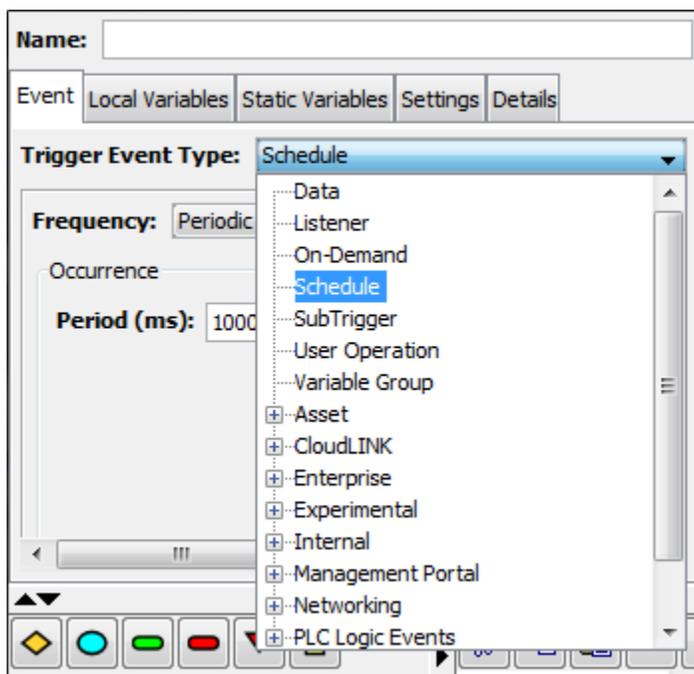
A Schedule event trigger can also be manually executed using the Workbench and the trigger's pop-up menu (right-click the trigger) and the **Fire Trigger** option.

Using the Workbench to manually execute a Schedule event trigger does not impact the trigger's schedule for being executed (Periodic, Hourly, etc.).

## Defining a schedule event trigger

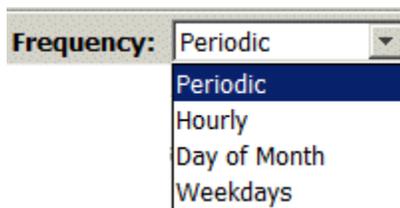
To define a schedule event trigger, follow these steps:

1. From the Workbench left pane, expand the node where you want to define the schedule event trigger.
2. Select the **Project** icon to display the **Projects** window, right click a specific project tab to display its pop-up menu, and then click **New**.  
You also can select the **New** button at the bottom of the **Projects** window when a specific project tab has already been selected.
3. The new **Trigger** window appears.  
Name the trigger. The trigger name can be up to 64 characters and include letters, numbers, and the underscore character. Spaces are allowed.
4. From the **Event** tab, select the **Trigger Event Type** down-arrow, and then select **Schedule**.



The Event tab becomes active with parameters that accommodate the schedule event.

5. Use the **Frequency** down-arrow to display a list of options, then select the frequency option you want to use.



Each frequency option has specific occurrence parameters.

The following describes each frequency option and its occurrence parameters:

Option	Description
--------	-------------

**Periodic**

The **Periodic** option is used to specify a millisecond interval.

The screenshot shows a configuration window with tabs for 'Event', 'Local Variables', 'Static Variables', 'Settings', and 'Details'. Under 'Settings', 'Trigger Event Type' is 'Schedule'. Below it, 'Frequency' is set to 'Periodic' (circled in red). Under the 'Occurrence' section, 'Period (ms)' is set to '1000'.

In the **Period (ms)** parameter, enter a whole number of milliseconds between 0 and 3600000. 3600000 is the number of milliseconds in an hour. For example, if you enter a value of 5000, the trigger will execute every 5 seconds.

**Hourly**

The **Hourly** option is used to specify the number of minutes after the hour and the days of the week.

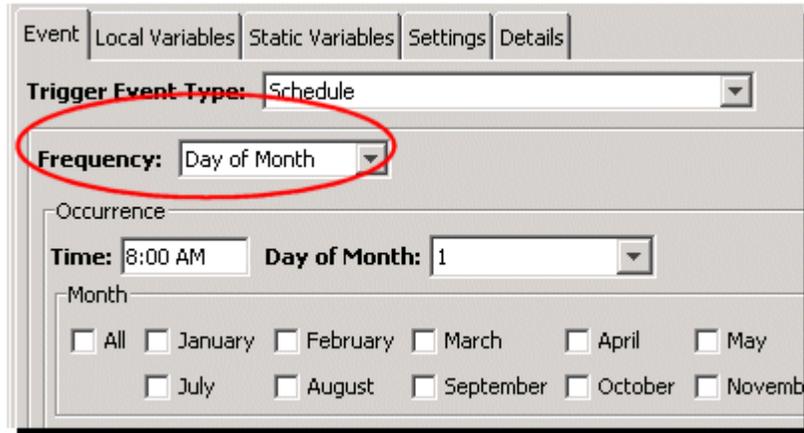
The screenshot shows a configuration window with tabs for 'Event', 'Local Variables', 'Static Variables', 'Settings', and 'Details'. Under 'Settings', 'Trigger Event Type' is 'Schedule'. Below it, 'Frequency' is set to 'Hourly' (circled in red). Under the 'Occurrence' section, 'Minutes after the hour' is set to '1'. Under the 'Day' section, the 'All' checkbox is selected.

In the **Minutes after the hour** parameter, enter a whole number between 0 and 59.

At least one day of the week check box must be selected. Multiple check boxes or the **All** check box can be selected.

**Day of Month**

The **Day of Month** option is used to specify a time and day of the month, for one or more months.



In the **Time** parameter, enter the time of day that you want the trigger to execute. Use a HH:MM (hour:minute) and AM or PM for the format. Time set in military format will be converted to the corresponding AM PM time format.

Select the **Day of Month** down-arrow, and then select a day option of 1 through 31.

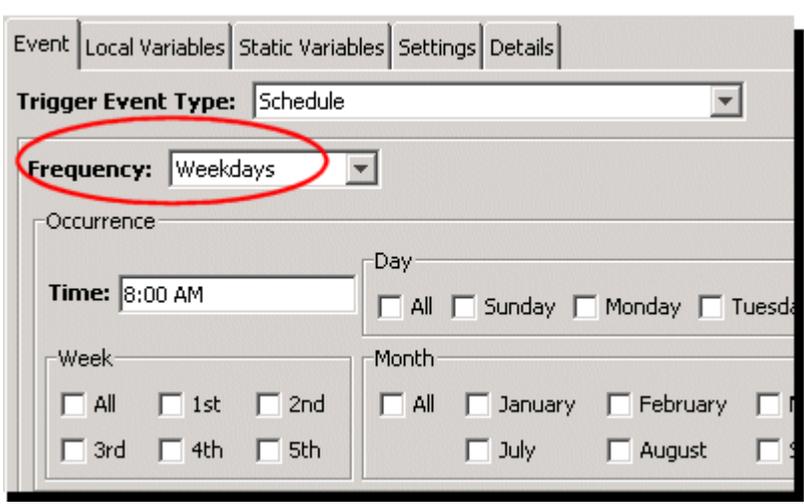
If you select 31, those months that do not contain 31 days become unavailable. Likewise, selecting 29 or 30 will make February unavailable.

There is also a **Last Day of Month** option.

At least one month check box must be selected. Multiple check boxes or the **All** check box can be selected.

**Weekdays**

The **Weekdays** option is used to specify a time, the day, the week, and the month.



In the **Time** parameter, enter the time of day that you want the trigger to execute. Use a HH:MM (hour:minute) and AM or PM for the format. Time set in military format will be converted to the corresponding AM PM time format.

At least one day of the week check box must be selected. Multiple check boxes or the **All** check box can be selected.

At least one week check box must be selected. Multiple check boxes or the **All** check box can be selected.

At least one month check box must be selected. Multiple check boxes or the **All** check box can be selected.

## Schedule event type trigger event variables

The input event variable available to a schedule event trigger is:

Event variable	Data type	Description
Scheduled Time	TIMESTAMP	The date and time that the trigger was scheduled to execute. This is usually the same as the <code>\$EVENT_TIME</code> macro.

## Complete the trigger definition

Complete the definition of the trigger, including the follow trigger components:

- The trigger's actions
- The trigger's local variables, static variables, macros and event variables
- The trigger's settings.

Use the **Validate** button to check the parameters and then the **Save** button to save the trigger's definition.

The trigger will be listed in the Project's tab list of triggers in the **Stopped** state.

## Schedule event trigger frequency and occurrence criteria considerations

The **Frequency** option and **Occurrence** criteria parameters control how frequently a schedule event trigger executes.

The requirements of the application, the characteristics of any device variables' updates and the capability of the system need to be understood when designing the application solution.

Considerations include:

- If the event requirement for when a trigger executes naturally aligns with one of the available trigger event types, then use that trigger event type instead of using a schedule event trigger.

For example, if a trigger needs to execute when a device variable's value meets a certain condition, then use a data event trigger.

If a schedule event trigger was used instead, then every time it was executed, it would need to determine if the device variable's value current value meets the condition before continuing with the application logic. This is an inefficient use of system resources.

- If you have a requirement to execute a trigger for a shift-based configuration (for example, 3 times a day, at 8 hour intervals, beginning at 8:30 AM) that has a frequency that cannot be directly satisfied with the frequency and occurrence options, you will have to use the options available to satisfy the requirement.

For this example, you could create three separate schedule event triggers that use the **Weekdays** option. One trigger could be called **shift1** with a trigger time of 8:30 AM. A second trigger called **shift2** with a trigger time of 4:30 PM, and a third trigger called **shift3** with a trigger time of 12:30 AM. You further can use the schedule event triggers along with a subtrigger event trigger to organize common trigger logic. The three schedule event triggers could all call a single subtrigger event trigger where the common application logic was defined.

## IIoTA industrial IoT Platform: SubTrigger

A **SubTrigger** event trigger executes when referenced in an Execute SubTrigger action from another trigger. The calling trigger must be executing on the same node.

A subtrigger event trigger can have input variables passed to it and can output variables back to the calling trigger.

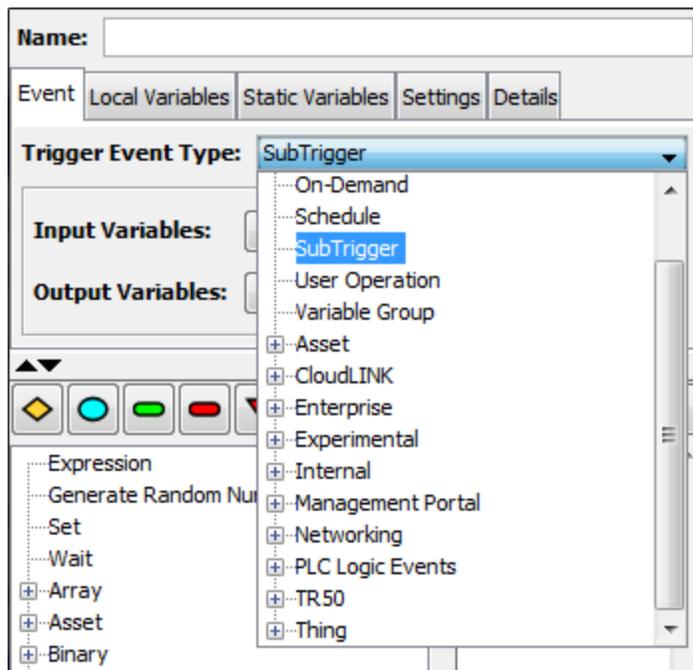
The calling trigger defines in the Execute SubTrigger action whether it will wait for the subtrigger to complete its execution before continuing to its next action.

If multiple triggers can execute the subtrigger at the same time, make sure the **Max In Progress** setting in the subtrigger is defined to allow the multiple concurrent instances of the subtrigger. For more information, see Trigger settings.

## Defining a subtrigger event trigger

To define a subtrigger event trigger, follow these steps:

1. From the Workbench left pane, expand the node where you want to define the subtrigger event trigger.
2. Select the **Project** icon to display the **Projects** window, right-click a specific project tab to display its pop-up menu, and then select **New**.  
You also can select the **New** button at the bottom of the **Projects** window when a specific project tab has already been selected.
3. The new **Trigger** window appears.  
Name the trigger. The trigger name can be up to 64 characters and include letters, numbers, and the underscore character. Spaces are allowed.
4. From the **Event** tab, select the **Trigger Event Type** down-arrow, and then select **SubTrigger**.



The **Event** tab becomes active with parameters that accommodate the subtrigger event.

Parameter	Description
<b>Input Variables</b>	Select the <b>Configure...</b> button to display the Variables window and add the variables that will be the input variables for the subtrigger event trigger. The variables can be single scalar variables or arrays. These input variables will be mapped on the input tab of the <b>Execute SubTrigger</b> action when this subtrigger is referenced as the subtrigger to execute. They also will be available to the subtrigger event trigger as input event variables.
<b>Output Variables</b>	Select the <b>Configure...</b> button to display the Variables window and add the variables that will be the output variables for the subtrigger event trigger. The variables can be single scalar variables or arrays. These output variables will be mapped on the output tab of the <b>Execute SubTrigger</b> action when this subtrigger is referenced as the subtrigger to execute. They also will be available to the subtrigger event trigger as output event variables.

## Subtrigger event type trigger event variables

The input event variables available to a subtrigger event trigger are:

Event variable	Data type	Description
Input variables	The data type of the configured input variable	Each subtrigger input variable will be available as an input event variable in the subtrigger event trigger. These input event variables can be used as the source variable in the trigger's actions.
Parent Project Name	STRING	The name of the project where the trigger resides that "called" the subtrigger.
Parent Trigger Name	STRING	The name of the trigger that "called" the subtrigger

The output event variables available to a subtrigger event trigger are:

Event variable	Data type	Description
Output variables	The data type of the configured output variable	Each subtrigger output variable will be available as an output event variable in the subtrigger event trigger. These output event variables can be used as the destination variable in the trigger's actions.

## IloTA industrial IoT Platform: User Operation

The **User Operation** event trigger is not supported on the IloTA industrial IoT Platform.

## IloTA industrial IoT Platform: Variable Group

A **Variable Group** event trigger executes when the data value of any device variable in a variable group meets a defined condition.

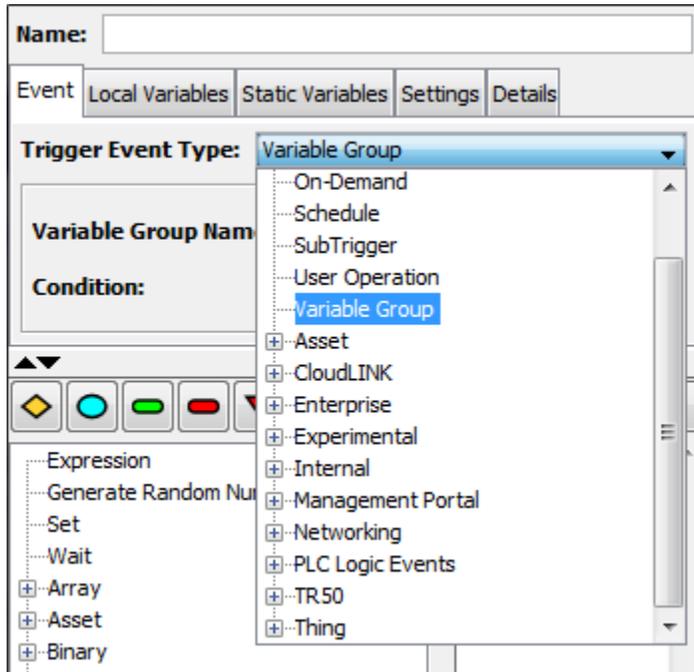
You must have a variable group defined in order to use this event. For more information, see [Defining, viewing, and controlling variable groups](#).

### Defining a variable group event trigger

To define a variable group event trigger, follow these steps:

1. From the Workbench left pane, expand the node where you want to define the variable group event trigger.
2. Select the **Project** icon to display the **Projects** window, right-click a specific project tab to display its pop-up menu, and then select **New**.  
You also can select the **New** button at the bottom of the **Projects** window when a specific project tab has already been selected.
3. The new **Trigger** window appears.  
Name the trigger. The trigger name can be up to 64 characters and include letters, numbers, and the underscore character. Spaces are allowed.

- From the **Event** tab, select the **Trigger Event Type** down-arrow, and then select **Variable Group**.



The **Event** tab becomes active with parameters that accommodate the variable group event.

Parameter	Description
<b>Variable Group Name</b>	A list of variable groups. A variable group is created using the Variable Groups tab.
<b>Condition</b>	Currently, there is only the <b>Value changed</b> condition. When any of the variables' value within the variable group changes from its previous value, the trigger will execute. For the trigger to execute, the variable group selected must be in a <b>Started</b> state.

## Variable group event type trigger event variables

The input event variables available to a variable group event trigger are:

Event variable	Data type	Description
DATA	The data type of the device variable	The data value from the device variable.

Device Name	STRING	The name of the device where the device variable resides.
Variable Group Name	STRING	The name of the variable group.
Variable Key	STRING	The key associated with the device variable, when the device variable was added to the variable group. This variable key can be used as a reference ID, or correlation ID, to help identify which variable's value in the group has changed.
Variable Name	STRING	The name of the device variable.

#### Related Topics

Defining, viewing, and controlling variable groups

Variable Group Add Variable

Data

## IIoTA industrial IoT Platform: Store and Forward event

A **Store and Forward** event trigger executes when a transport's store and forward state changes.

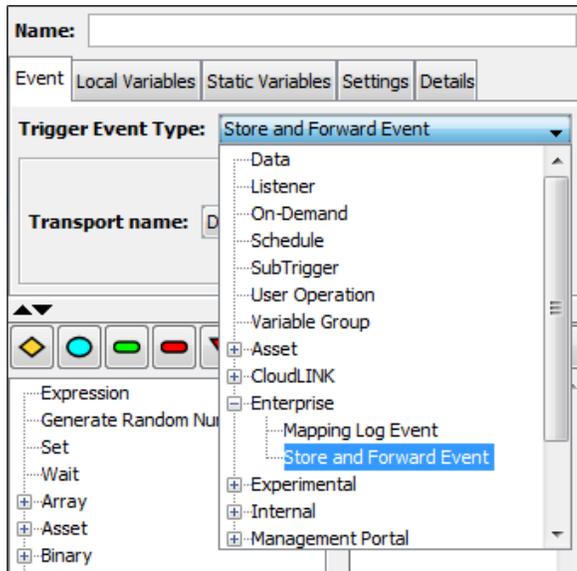
The transport must be configured for store and forward support. For more information, see [Store & Forward](#) tab.

### Defining a store and forward event trigger

To define a store and forward event trigger, follow these steps:

1. From the Workbench left pane, expand the node where you want to define the store and forward event trigger.
2. Select the **Project** icon to display the **Projects** window, right-click a specific project tab to display its pop-up menu, and then select **New**.  
You also can select the **New** button at the bottom of the **Projects** window when a specific project tab has already been selected.

3. The new **Trigger** window appears.  
Name the trigger. The trigger name can be up to 64 characters and include letters, numbers, and the underscore character. Spaces are allowed.
4. From the **Event** tab, select the **Trigger Event Type** down-arrow, expand the **Enterprise** category, and then select **Store and Forward Event**.



The **Event** tab becomes active with parameters that accommodate the store and forward event.

Parameter	Description
<b>Transport name</b>	Select one of the transports from the list of currently defined transports for the node. Only the transport types that support the store and forward feature are displayed in the list.

## Store and forward event type trigger event variables

The input event variables available to a store and forward event trigger are:

Event variable	Data type	Description
SAF State	INT2	The transport's store and forward state. The values and their meaning are:

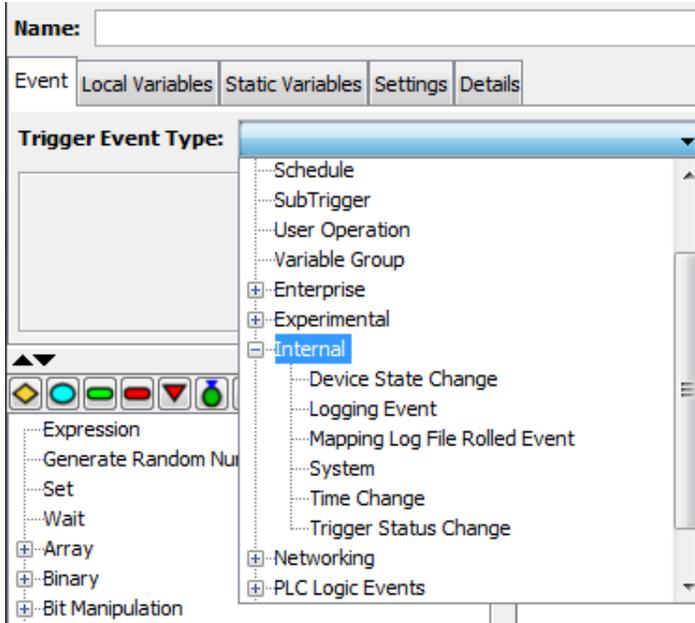
		<p>100 - The transport has entered store and forward</p> <p>101 - The transport has exited store and forward.</p> <p>102 - The store and forward file for the transport has exceeded 50 percent capacity</p> <p>103 - The store and forward file for this transport is at or below 50 percent capacity</p> <p>104 - The store and forward file for this transport is full. Data is lost.</p>
Transport Name	STRING	The name of the transport whose store and forward state has changed.

Related topics

Using tabs on the Transport window

## IIOA industrial IoT Platform: Internal events

The **Internal** category contains the event types related to the base runtime system.



# IIoTA industrial IoT Platform: Device State Change

A **Device State Change** event trigger executes when the state of one or more devices changes.

Since multiple devices can change state at the same time, make sure the **Max In Progress** setting in the device state change trigger is defined to allow concurrent instances of the trigger. For more information, see Trigger settings.

## Defining a device state change event trigger

To define a device state change event trigger, follow these steps:

1. From the Workbench left pane, expand the node where you want to define the device state change event trigger.
2. Select the **Project** icon to display the **Projects** window, right-click a specific project tab to display its pop-up menu, and then select **New**.  
You also can select the **New** button at the bottom of the **Projects** window when a specific project tab has already been selected.
3. The new **Trigger** window appears.  
Name the trigger. The trigger name can be up to 64 characters and include letters, numbers, and the underscore character. Spaces are allowed.
4. From the **Event** tab, select the **Trigger Event Type** down-arrow, expand the **Internal** category, and then select **Device State Change**.

The screenshot shows a configuration window for a trigger. At the top, there is a text input field labeled "Name:". Below this is a tabbed interface with four tabs: "Event", "Local Variables", "Static Variables", and "Settings", with "Event" selected. Under the "Event" tab, there is a dropdown menu labeled "Trigger Event Type:" with "Device State Change" selected. Below the dropdown are four rows of settings, each with a label and a dropdown menu:

- Filter for Device: False
- Fire on Device Start: True
- Fire on Device Stop: True
- Fire on Device Disable: True

The **Event** tab becomes active with parameters that accommodate the device state change event.

Parameter	Description
<b>Filter for Device</b>	<p><b>True</b> - Displays a list of devices that reside on the current node. Select a device from the list to have the trigger execute when that device changes state.</p> <p><b>False</b> - The trigger will execute when any device on the current node changes state.</p>
<b>Fire on Device Start</b>	<p><b>True</b> - The trigger will execute when the device state changes to Started.</p> <p><b>False</b> - The trigger will not execute when the device state changes to Started.</p>
<b>Fire on Device Stop</b>	<p><b>True</b> - The trigger will execute when the device state changes to Stopped.</p> <p><b>False</b> - The trigger will not execute when the device state changes to Stopped.</p>
<b>Fire on Device Disable</b>	<p><b>True</b> - The trigger will execute when the device state changes to Disabled.</p> <p><b>False</b> - The trigger will not execute when the device state changes to Disabled.</p>

## Device state change event type trigger event variables

The input event variables available to a device state change event trigger are:

Event variable	Data type	Description
Device Name	STRING	The name of the device whose state has changed.
State	STRING	This new state of the device. The value can be:  Started Stopped Disabled

Related Topics  
Trigger Status Change

## IIoTA industrial IoT Platform: Logging Event

A **Logging** event trigger executes when a message log entry is written to the exceptions log or audit log.

### Defining a logging event trigger

To define a logging event trigger, follow these steps:

1. From the Workbench left pane, expand the node where you want to define the logging event trigger.
2. Select the **Project** icon to display the **Projects** window, right-click a specific project tab to display its pop-up menu, and then select **New**.  
You also can select the **New** button at the bottom of the **Projects** window when a specific project tab has already been selected.
3. The new **Trigger** window appears.  
Name the trigger. The trigger name can be up to 64 characters and include letters, numbers, and the underscore character. Spaces are allowed.
4. From the **Event** tab, select the **Trigger Event Type** down-arrow, expand the **Internal** category, and then select **Logging Event**.

The screenshot shows a configuration window for a trigger. At the top, there is a text input field labeled "Name:". Below this is a tabbed interface with four tabs: "Event", "Local Variables", "Static Variables", and "Settings", with "Event" being the active tab. Under the "Event" tab, there is a dropdown menu labeled "Trigger Event Type:" with "Logging Event" selected. Below this are three more dropdown menus: "Message Type:" with "EXCEPTION" selected, "Minimum Loglevel:" with "FATAL" selected, and "Maximum Loglevel:" with "INFO" selected. At the bottom, there is a text input field labeled "Component:".

The **Event** tab becomes active with parameters that accommodate the logging event.

Parameter	Description
-----------	-------------

<b>Message Type</b>	Selects the type of log message entry to cause the trigger event. The options are: <b>EXCEPTION</b> - to specify the Exceptions log. <b>AUDIT</b> - to specify the Audit log
<b>Minimum Loglevel</b>	When the message Type is EXCEPTION only, selects the minimum log level to cause the trigger event. The log levels are in order from lowest number (and most severe) to highest number (and least severe): FATAL, ERROR, WARN, INFO.
<b>Maximum Loglevel</b>	When the message Type is EXCEPTION only, selects the maximum log level to cause the trigger event. The log levels are in order from lowest number (and most severe) to highest number (and least severe): FATAL, ERROR, WARN, INFO.
<b>Component</b>	Optional. A case sensitive field to filter on a single component value to cause this logging event trigger to execute. For example, specifying "Transaction Engine" (without the quotes) will cause this trigger to execute if the log message has Transaction Engine for the component. The component value can be a user defined value (from a Log Message action) or a system defined value from one of the system components. Leaving the component parameter blank will cause the logging event trigger to execute for all log message component values.

## Logging event type trigger event variables

The input event variables available to a logging event trigger when the message type is EXCEPTION are:

Event variable	Data type	Description
Code	INT4	The error code used in the log message.
Component	STRING	The component used in the log message.
Level	STRING	The log level used in the log message.
Message	STRING	The log message text used in the log message.
Timestamp	TIMESTAMP	The date and time that the log message was written to the exceptions log.

The input event variables available to a logging event trigger when the message type is AUDIT are:

Event variable	Data type	Description
Code	INT4	The error code used in the log message.
Component	STRING	The component used in the log message.
Message	STRING	The log message text used in the log message.
Timestamp	TIMESTAMP	The date and time that the log message was written to the audit log.
User	STRING	The user associate with the log message. This can be the user who used the Workbench to select a function, or it could be one of the system components.

Related Topics  
 Logs & Reports  
 Log Message

## IIoTA industrial IoT Platform: Mapping Log File Rolled Event

A **Mapping Log File Rolled** event trigger executes when a Transaction Server mapping log file becomes full.

### Defining a mapping log file rolled event trigger

To define a mapping log file rolled event trigger, follow these steps:

1. From the Workbench left pane, expand the node where you want to define the mapping log file rolled event trigger.
2. Select the **Project** icon to display the **Projects** window, right-click a specific project tab to display its pop-up menu, and then select **New**.  
 You also can select the **New** button at the bottom of the **Projects** window when a specific project tab has already been selected.

- The new **Trigger** window appears.  
Name the trigger. The trigger name can be up to 64 characters and include letters, numbers, and the underscore character. Spaces are allowed.
- From the **Event** tab, select the **Trigger Event Type** down-arrow, expand the **Internal** category, and then select **Mapping Log File Rolled**.

The **Event** tab becomes active with parameters that accommodate the mapping log file rolled event.

Parameter	Description
<b>Log Type</b>	Selects the type of mapping log. The options are:  <b>Transport</b> - to specify a transport mapping log and filter the Transport name parameter. <b>Listener</b> - to specify a listener mapping log and filter the Listener name parameter.
<b>Transport name</b> or <b>Listener name</b>	Based on the <b>Log Type</b> selection, a list of Transports or Listeners on the current node that have their mapping log option selected. If no transports or listeners have their mapping log option selected, the list will be blank. Select the transport or listener that will cause the mapping log file rolled event.

## Mapping log file rolled event type trigger event variables

The input event variables available to a mapping log file rolled event trigger are:

Event variable	Data type	Description
----------------	-----------	-------------

File Path	STRING	<p>The file name of the mapping log file that rolled (became full). If the <b>Mapping Log</b> tab parameter <b>Number of Log Files</b> is defined to be greater than 1, then the next file in the rotation will be used.</p> <p>If the <b>Mapping Log</b> tab parameter <b>Copy rolled log to staging</b> option is selected, then a copy of the rolled mapping log file will be created in the staging browser.</p>
-----------	--------	--

Related Topics

Logs & Reports

Mapping Log tab

Listener Mapping Log tab

## IIoTA industrial IoT Platform: System

A **System** event trigger executes when the node starts or when the attention bit is set or cleared.

### Defining a system event trigger

To define a system event trigger, follow these steps:

1. From the Workbench left pane, expand the node where you want to define the system event trigger.
2. Select the **Project** icon to display the **Projects** window, right-click a specific project tab to display its pop-up menu, and then select **New**.  
You also can select the **New** button at the bottom of the **Projects** window when a specific project tab has already been selected.
3. The new **Trigger** window appears.  
Name the trigger. The trigger name can be up to 64 characters and include letters, numbers, and the underscore character. Spaces are allowed.
4. From the **Event** tab, select the **Trigger Event Type** down-arrow, expand **Internal**, and then select **System**.

Name:

Event **Local Variables** Static Variables Settings Details

Trigger Event Type:

Event Condition:

The Event tab becomes active with parameters that accommodate the system event.

Parameter	Description
Event Condition	<p>Select the system event condition for this trigger:</p> <p><b>Node Start</b> - executes when the node starts. There can be multiple system event triggers defined with the Node Start event condition.</p> <p><b>Attention Bit Set</b> - executes when the attention bit is set for any reason.</p> <p><b>Attention Bit Cleared</b> - executes when the attention bit is cleared.</p>

## System event type trigger event variables

There are no event variables for a system event trigger.

Related Topics

Modify Attention Bit

Attention Bit

The alert feature provides functionality beyond the attention bit feature.

## IIoTA industrial IoT Platform: Time Change

A **Time Change** event trigger executes when the system date and time are changed.

### Defining a Time Change event trigger

To define a Time Change event trigger, follow these steps:

1. From the Workbench left pane, expand the node where you want to define the Time Change event trigger.

2. Select the **Project** icon to display the **Projects** window, right-click a specific project tab to display its pop-up menu, and then select **New**.  
You also can select the **New** button at the bottom of the **Projects** window when a specific project tab has already been selected.
3. The new **Trigger** window appears.  
Name the trigger. The trigger name can be up to 64 characters and include letters, numbers, and the underscore character. Spaces are allowed.
4. From the **Event** tab, select the **Trigger Event Type** down-arrow, expand the **Internal** category, and then select **Time Change**.

The screenshot shows a configuration window for a trigger. At the top, there is a text input field labeled 'Name:'. Below this, there are four tabs: 'Event', 'Local Variables', 'Static Variables', 'Settings', and 'Details'. The 'Event' tab is selected and active. Under the 'Event' tab, there is a dropdown menu labeled 'Trigger Event Type:' which is currently set to 'Time Change'. Below the dropdown is a large empty rectangular area.

The **Event** tab becomes active to accommodate the Time Change event.

There are no event parameters for the Time Change event.

## Time Change event type trigger event variables

The input event variables available to a Time Change event trigger are:

Event variable	Data type	Description
Delta in milliseconds	INT8	The difference between the Modified Timestamp and the Original Timestamp.
Modified Timestamp	TIMESTAMP	The TIMESTAMP after the system's date and time were changed.
Original Timestamp	TIMESTAMP	The TIMESTAMP before the system's date and time were changed.

The Time Change event trigger will fire when a time change via the IIoT runtime happens. This includes:

- A Set Date & Time trigger action. See Set Date & Time.
- A time synchronization with an NTP server. See Time Management

If a time change occurs during IIoT runtime startup, but before the Time Change event trigger is started, the Time Change event trigger will fire after it has been loaded.

The Time Change event will not fire when a time change occurs outside of the IIoT runtime. This includes:

- Setting the time using an operating system command line-based command.
- Setting the time using the Service Execute trigger action.
- Setting the time in a Lua script.

Related Topics

Set Date & Time

Time Management

## IIoTA industrial IoT Platform: Trigger Status Change

A **Trigger Status Change** event trigger executes when the status of a trigger changes.

### Defining a trigger status change event trigger

To define a trigger status change event trigger, follow these steps:

1. From the Workbench left pane, expand the node where you want to define the trigger status change event trigger.
2. Select the **Project** icon to display the **Projects** window, right-click a specific project tab to display its pop-up menu, and then select **New**.  
You also can select the **New** button at the bottom of the **Projects** window when a specific project tab has already been selected.
3. The new **Trigger** window appears.  
Name the trigger. The trigger name can be up to 64 characters and include letters, numbers, and the underscore character. Spaces are allowed.

- From the **Event** tab, select the **Trigger Event Type** down-arrow, expand the **Internal** category, and then select **Trigger Status Change**.

The screenshot shows a configuration window for an event trigger. At the top, there is a 'Name:' input field. Below it are tabs for 'Event', 'Local Variables', 'Static Variables', 'Settings', and 'Details'. The 'Event' tab is active. Under 'Trigger Event Type', a dropdown menu is set to 'Trigger Status Change'. Below this are four configuration fields: 'Project Name' (set to '== Current Project =='), 'Trigger Name' (set to '1\_Minute\_Event'), 'Fire on Load' (set to 'True'), and 'Fire on Unload' (set to 'True'). A refresh icon is visible next to the 'Project Name' field.

The **Event** tab becomes active with parameters that accommodate the trigger status change event.

Parameter	Description
<b>Project Name</b>	Displays a list of all projects defined on the current node. Select the project that contains the trigger you want to monitor for changes. The option == Current Project == can be used to indicate the project that this device status change event trigger is defined in.
<b>Trigger Name</b>	Displays a list of triggers defined within the project you selected in <b>Project Name</b> . Select the trigger that matches your data source.
<b>Fire on Load</b>	Options are True and False. If set to <b>True</b> , the trigger will execute whenever the status of the trigger specified in <b>Trigger Name</b> changes to <b>Loaded</b> . This status change might occur when the specified project or trigger are started depending on the order in which they are started.
<b>Fire on Unload</b>	Options are True and False. If set to <b>True</b> , the trigger will execute whenever the status of the trigger specified in <b>Trigger Name</b> changes to <b>Unloaded</b> . This status change might occur when the specified project or trigger is stopped depending on the order in which they are stopped or when the trigger becomes disabled.

## Trigger status change event type trigger event variables

The input event variables available to a trigger status change event trigger are:

Event variable	Data type	Description
Is Disabled	BOOL	The current state of the trigger, relative to being disabled. The values can be: True - the trigger is state is disabled (Is Loaded = False and State = Disabled). False - the trigger is not disabled.
Is Loaded	BOOL	The current status of the trigger, relative to being loaded. The values can be: True - the trigger is loaded (Is Disabled = False and State = Started). False - the trigger is not loaded (State could be Started, Stopped or Disabled).
Project Name	STRING	The name of the project where the trigger is defined.
State	STRING	The current state of the trigger project. The value can be:  Started Stopped Disabled.
Trigger Name	STRING	The name of the trigger whose status has changed.

## Project state, trigger status and trigger state

Projects are the "containers" that triggers are defined in and they have their own state: Started or Stopped. This allows the group of triggers that are defined in a project to be collectively controlled by the state of the project.

The trigger execution component maintains the state of each project (Started or Stopped), the state of each trigger (Started, Stopped, Disabled, and the transition states of Starting and Stopping) and the status of each trigger (Loaded and Unloaded). This can be seen on the project windows and an individual project's tab where the list of the triggers defined in the project are displayed.

The trigger status change event trigger event variables can be used to determine the specific state and status of the trigger.

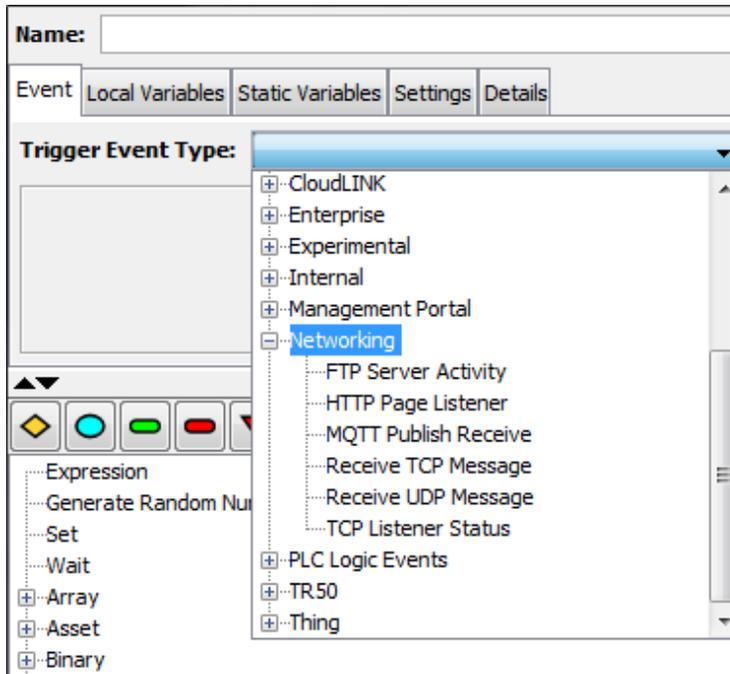
For a trigger to execute:

- Its status must be Loaded, which also means the project is Started
- Its state must be Started

Related Topics  
Device State Change

## IIoTA industrial IoT Platform: Networking events

The **Networking** category contains the event types related to network communication.



## IIoTA industrial IoT Platform: FTP Server Activity

An **FTP Server Activity** event trigger executes when a file is uploaded or downloaded using the FTP server.

Nodes have an FTP server that can be used to upload files to or download files from the staging browser area of the node.

To cause a trigger that uses this event to fire, you can do the following:

- Enable the FTP Server from the Administration FTP Server tab in the Workbench.
- Connect to the FTP server using an FTP client application and valid Workbench credentials.

- Put or get a file that matches the pattern you used in the **Path Contains** parameter in the trigger.

The FTP Server Activity event is part of the Advanced package

This event is part of the Advanced package.

## Defining an FTP server activity event trigger

To define an FTP Server Activity event trigger, follow these steps:

1. From the Workbench left pane, expand the node where you want to define the FTP Server Activity event trigger.
2. Select the **Project** icon to display the **Projects** window, right-click a specific project tab to display its pop-up menu, and then select **New**.  
You also can select the **New** button at the bottom of the **Projects** window when a specific project tab has already been selected.
3. The new **Trigger** window appears.  
Name the trigger. The trigger name can be up to 64 characters and include letters, numbers, and the underscore character. Spaces are allowed.
4. From the **Event** tab, select the **Trigger Event Type** down-arrow, expand the **Networking** category, and then select **FTP Server Activity**.

The screenshot shows a configuration window for a trigger. At the top, there is a 'Name:' field. Below it, there are four tabs: 'Event', 'Local Variables', 'Static Variables', 'Settings', and 'Details'. The 'Event' tab is selected. Under the 'Event' tab, there is a 'Trigger Event Type:' dropdown menu with 'FTP Server Activity' selected. Below that, there is a 'Fire When:' dropdown menu with 'File Uploaded' selected. At the bottom, there is a 'Path Contains:' text input field.

The **Event** tab becomes active with parameters that accommodate the FTP Server Activity event.

Parameter	Description
-----------	-------------

<b>Fire When</b>	Selects the type of FTP activity to cause the trigger event. The options are:  <b>File Uploaded</b> - to specify a file being transferred to the staging browser. <b>File Downloaded</b> - to specify a file being transferred from the staging browser.
<b>Path Contains</b>	Optional. A string can be specified to filter only FTP activity where the full path name of the file uploaded or downloaded matches the specified string. This does not support the use of wildcard characters or regular expressions. For example, the Path Contains values of "How", "Are", "AreYou" or ".txt" will match the file name "HowAreYou.txt".

## FTP Server Activity event type trigger event variables

The input event variables available to a FTP Server Activity event trigger are:

Event variable	Data type	Description
Command	STRING	The FTP command that was actually executed, typically this will be a "STOR" or "RETR" command string.
Directory	STRING	The directory on the node of the file being uploaded from or downloaded to. This directory is a relative path to the root of staging browser directory.
Filename	STRING	The name of the file being uploaded or downloaded. This file name will not contain the directory.
Full Path	STRING	The concatenated Directory and Filename.
Remote Address	STRING	The IP address and port number of the client that is interacting with the FTP server. The format will be similar to "192.168.2.65:43223".

## IIoTA industrial IoT Platform: HTTP Page Listener

A **HTTP Page Listener** event trigger executes when a request is received for the specified URL.

It is important to note that you cannot have more than one HTTP page listener event type trigger listening on the same page (using a given **Listener URL**), nor can you listen to a page that is served up by existing web applications on the gateway (such as the web management interface).

To cause a trigger that uses this event to fire, you can do the following:

- Enable the HTTP Server from the Administration > **HTTP Server** tab in the Workbench.
- Make an HTTP request to the URL specified in **Listener URL** parameter in the trigger.

## Defining a HTTP page listener event trigger

To define a HTTP page listener event trigger, follow these steps:

1. From the Workbench left pane, expand the node where you want to define the HTTP page listener event trigger.
2. Select the **Project** icon to display the **Projects** window, right-click a specific project tab to display its pop-up menu, and then select **New**.  
You also can select the **New** button at the bottom of the **Projects** window when a specific project tab has already been selected.
3. The new **Trigger** window appears.  
Name the trigger. The trigger name can be up to 64 characters and include letters, numbers, and the underscore character. Spaces are allowed.
4. From the **Event** tab, select the **Trigger Event Type** down-arrow, expand the **Networking** category, and then select **HTTP Page Listener**.

The screenshot shows the configuration window for a trigger. At the top, there is a text input field for the **Name**. Below this is a tabbed interface with four tabs: **Event**, **Local Variables**, **Static Variables**, **Settings**, and **Details**. The **Event** tab is active. Under the **Event** tab, there is a dropdown menu for **Trigger Event Type** set to **HTTP Page Listener**. Below the dropdown are four input fields, each with a corresponding '...' button to its right:

- Listener URL:** /trigger
- Query String:** searchKey&temperatureValue
- Request Header:** Host&From
- Response Header:** Warning

The **Event** tab becomes active with parameters that accommodate the HTTP page listener event.

Parameter	Description
<b>Listener URL</b>	<p>The URL that the calling application will request. The default is /trigger, which means that if you request http://&lt;ip address of your gateway&gt;/trigger the trigger will be executed. Note that you cannot have more than one HTTP page listener event type trigger listening on the same page.</p>
<b>Query String</b>	<p>The query string fields of the URL that the calling application will request. The URL can contain a query string, such as "http://host/trigger/searchKey=1334&amp;temperatureValue=72.3". The example screenshot above shows how to map the individual query string variables into trigger input event variables. Clicking the "..." button will bring up a list editor to edit this parameter. The "&amp;" character is reserved and is used as the string delimiter for this parameter.</p>
<b>Request Header</b>	<p>The HTTP header fields of the URL that the calling application will request. The HTTP request can contain key value pairs in the HTTP header, such as "Host" and "From". The example screenshot above shows how to map the individual request header key variables into trigger input event variables. Clicking the "..." button will bring up a list editor to edit this parameter. The "&amp;" character is reserved and is used as the string delimiter for this parameter.</p>
<b>Response Header</b>	<p>The HTTP header fields of the response that the trigger can supply with the response as key value pairs. The HTTP response can contain key pair values in the response header. The example screenshot above shows how the trigger will inject a "Warning" field into the response header. Clicking the "..." button will bring up a list editor to edit this parameter. The "&amp;" character is reserved and is used as the string delimiter for this parameter.</p>

## HTTP page listener event type trigger event variables

The input event variables available to a HTTP page listener event trigger are:

Event variable	Data type	Description
HTTP Content	STRING	If the incoming HTTP request is a POST, then this variable will hold the content that was POST'ed to the URL.

HTTP Content Length	INT4	If the incoming HTTP request is a POST, then this variable will hold the length of the content that was submitted.
HTTP Content-Type	STRING	If the incoming HTTP request is a POST, then this field will contain the MIME type of the data that was submitted.
HTTP Method	STRING	The value of this variable will be the HTTP method that was invoked, such as: “GET”, “POST”, “HEAD”, etc.
HTTP URL	STRING	The value of this variable will be the original URL that was captured by the HTTP Server.
HTTP Query String Variable(s)	STRING	Each query string field specified in the <b>Query String</b> parameter will map to a trigger input event variable. The trigger input event variable will contain a STRING value if the request contains the query string field name.
HTTP Request Header Variable(s)	STRING	Each HTTP request header field specified in the <b>Request Header</b> parameter will map to a trigger input event variable. The trigger input event variable will contain a STRING value if the request contains the request header field name.

The output event variables available to a HTTP page listener event trigger are:

Event variable	Data type	Description
HTTP Content	STRING	Whatever you set in this variable will be sent in the HTTP response body. The Content-Length is automatically set based on the length of the buffer that you write.
HTTP Content-Type	STRING	This variable should be set with the appropriate MIME type for the response data.
HTTP Status Code	INT4	This variable should be set with the HTTP response code, such as: 200, 500, etc.
HTTP Response Variable(s)	STRING	Each HTTP response header field specified in the <b>Response Header</b> parameter will map to a trigger output event variable. The value will be inserted into the response header if the trigger output event variable is set.

# IIoTA industrial IoT Platform: MQTT Publish Receive

The **MQTT Publish Receive** event subscribes to a topic for MQTT publishes being sent on the MQTT device's MQTT connection. The MQTT Publish Receive event will listen on a specific topic being published by the MQTT broker.

## Defining a MQTT Publish Receive event trigger

To define a **MQTT Publish Receive** event trigger, follow these steps:

1. From the Workbench left pane, expand the node where you want to define the **MQTT Publish Receive** event trigger.
2. Select the **Project** icon to display the **Projects** window, right-click a specific project tab to display its pop-up menu, and then select **New**.  
You also can select the **New** button at the bottom of the **Projects** window when a specific project tab has already been selected.
3. The new **Trigger** window appears.  
Name the trigger. The trigger name can be up to 64 characters and include letters, numbers, and the underscore character. Spaces are allowed.
4. From the **Event** tab, select the **Trigger Event Type** down-arrow, expand the **Networking** category, and then select **MQTT Publish Receive**.

The screenshot shows the configuration window for a MQTT Publish Receive event trigger. The window has a 'Name' field at the top. Below it are tabs for 'Event', 'Local Variables', 'Static Variables', 'Settings', and 'Details'. The 'Event' tab is active. Under 'Trigger Event Type', a dropdown menu is open, showing 'MQTT Publish Receive' selected. Below this are four fields: 'MQTT Client' with a dropdown menu showing 'client', 'Topic Filter' with a text input field containing '#', 'Quality of Service' with a text input field containing '0', and 'Payload Type' with a dropdown menu showing 'String'.

The **Event** tab becomes active with parameters that accommodate the **MQTT Publish Receive** event.

Parameter	Description
<b>MQTT Client</b>	The MQTT device that represents the MQTT connection to use for this subscription. The list contains the MQTT devices on this gateway that are in a <b>Started</b> state.
<b>Topic Filter</b>	<p>An expression is used to specify one or more topic names the subscription should filter on when subscribing to publishes from the MQTT broker. This expression can include wildcard characters to filter on. This field is required and supports a String up to 65535 bytes.</p> <p>Wildcard characters '+' and '#' are supported as follows:</p> <ol style="list-style-type: none"> <li>1. The '#' is a wildcard character that matches any number of levels within a topic, referred to as the multi-level wildcard. <ol style="list-style-type: none"> <li>1. The multi-level wildcard character must be specified either on its own or following a topic level separator, where a topic level separator is a '/' character.</li> <li>2. The multi-level wildcard must be the last character specified in the Topic Filter.</li> <li>3. The multi-level wildcard includes the parent level.</li> <li>4. The multi-level wildcard can be specified alone and will subscribe to all messages.</li> <li>5. Usage examples: <ol style="list-style-type: none"> <li>1. "sport/tennis/player1/#" matches "sport/tennis/player1", "sport/tennis/player1/ranking", "sport/tennis/player1/score/wimbledon"</li> <li>2. "sport/#" also matches the singular "sport", since # includes the parent level.</li> <li>3. "#" is valid and will receive every application message.</li> <li>4. "sport/tennis/#" is valid.</li> <li>5. "sport/tennis#" is not valid.</li> <li>6. "sport/tennis/#/ranking" is not valid.</li> </ol> </li> </ol> </li> <li>2. The '+' is a wildcard that matches only one topic level, referred to as the single-level wildcard. <ol style="list-style-type: none"> <li>1. The single-level wildcard can be used at any level in the Topic Filter, including first and last levels.</li> <li>2. The single-level wildcard must occupy the entire level of the filter.</li> <li>3. The single-level wildcard matches only a single level.</li> <li>4. Usage examples: <ol style="list-style-type: none"> <li>1. "sport/tennis/+" matches "sport/tennis/player1" and "sport/tennis/player2" but not "sport/tennis/player1/ranking".</li> <li>2. "sport/+" does not match "sport" but it does match "sport/" due to the single level matching restriction.</li> <li>3. "+" is valid.</li> <li>4. "+/tennis/#" is valid.</li> </ol> </li> </ol> </li> </ol>

	<ol style="list-style-type: none"> <li>5. "sport+" is not valid.</li> <li>6. "sport+/player1" is valid.</li> <li>7. "/finance" matches "+/" and "/+", but not "+".</li> </ol>
<b>Quality of Service</b>	<p>Quality of service (QoS) that is used to send the publish. Default is set to 0.</p> <ol style="list-style-type: none"> <li>1. QoS 0: The defined behavior for QoS 0 is that an acknowledge for a subscribed topic is not expected from the gateway when the subscribed topic data is sent from the MQTT broker.</li> <li>2. QoS 1: The defined behavior for QoS 1 is that an acknowledge for a subscribed topic is expected from the gateway when the subscribed topic data is sent from the MQTT broker.</li> </ol>
<b>Payload Type</b>	Type of payload to receive, either String or Binary. The default is String.

## MQTT Publish Receive event type trigger event variables

The input event variables available to a **MQTT Publish Receive** event trigger are:

Event variable	Data type	Description
<b>Topic</b>	STRING	The topic name of the incoming publish.
<b>Payload</b>	STRING or BINARY	The payload of the incoming publish. The data type depends on the option selected for the <b>Payload Type</b> parameter.
<b>Payload Length</b>	UINT4	The length of the Payload for the incoming publish.

## IIoTA industrial IoT Platform: Receive TCP Message

A **Receive TCP Message** event trigger executes when a TCP message is received.

### Defining a Receive TCP Message event trigger

To define a Receive TCP Message event trigger, follow these steps:

1. From the Workbench left pane, expand the node where you want to define the Receive TCP Message event trigger.
2. Select the **Project** icon to display the **Projects** window, right-click a specific project tab to display its pop-up menu, and then select **New**.  
You also can select the **New** button at the bottom of the **Projects** window when a specific project tab has already been selected.
3. The new **Trigger** window appears.  
Name the trigger. The trigger name can be up to 64 characters and include letters, numbers, and the underscore character. Spaces are allowed.
4. From the **Event** tab, select the **Trigger Event Type** down-arrow, expand the **Networking** category, and then select **Receive TCP Message**.

The screenshot shows a configuration window for a trigger. At the top, there is a text input field labeled 'Name:'. Below this is a tabbed interface with four tabs: 'Event', 'Local Variables', 'Static Variables', 'Settings', and 'Details'. The 'Event' tab is selected. Under the 'Event' tab, there is a dropdown menu labeled 'Trigger Event Type:' with 'Receive TCP Message' selected. Below the dropdown is a 'Device Name:' dropdown menu with a refresh icon to its right.

The **Event** tab becomes active with parameters that accommodate the Receive TCP Message event.

Parameter	Description
<b>Device Name</b>	The TCP Listener device that supports the TCP connection that receives TCP messages from partner applications.

## Receive TCP Message event type trigger event variables

The input event variables available to a Receive TCP Message event trigger are:

Event variable	Data type	Description
Data	Based on the option selected for the <b>Output Type</b> parameter in the TCP Listener device.	The data received by the TCP Listener device.
Data Length	UINT4	The size of the TCP data received by the TCP Listener device.
Device Name	STRING	The name of the TCP Listener device.
Source Address	STRING	The IP Address and port that the data was received from.

## Receive TCP Message event trigger considerations

- The selected TCP Listener device must be Started for the Receive TCP Message event trigger to receive data.
- The TCP message handling behavior (fixed message length, variable message length with terminating characters, defined message header) is defined in the TCP Listener device **Message Type** parameter.
- Multiple Receive TCP Message event triggers can be defined and started referencing the same TCP Listener device.  
Each trigger will receive a copy of the TCP message received by the TCP Listener device.
- If multiple sending partner applications are sending TCP messages to the same IP address and port, then each sent TCP message will be received as a separate and distinct Receive TCP Message event trigger execution. The trigger execution behavior is based on the trigger settings for **Max in Progress** and **Queue Size**.
- The TCP driver and the TCP Listener device does not support sending data back to the partner TCP application that sent the TCP message.

# IIoTA industrial IoT Platform: Receive UDP Message

A **Receive UDP Message** event trigger executes when a User Datagram Protocol (UDP) message is received.

## Defining a Receive UDP Message event trigger

To define a Receive UDP Message event trigger, follow these steps:

1. From the Workbench left pane, expand the node where you want to define the Receive UDP Message event trigger.
2. Select the **Project** icon to display the **Projects** window, right-click a specific project tab to display its pop-up menu, and then select **New**.  
You also can select the **New** button at the bottom of the **Projects** window when a specific project tab has already been selected.
3. The new **Trigger** window appears.  
Name the trigger. The trigger name can be up to 64 characters and include letters, numbers, and the underscore character. Spaces are allowed.
4. From the **Event** tab, select the **Trigger Event Type** down-arrow, expand the **Networking** category, and then select **Receive UDP Message**.

The screenshot shows a configuration window for a 'Receive UDP Message' event trigger. At the top, there is a 'Name:' field. Below it are tabs for 'Event', 'Local Variables', 'Static Variables', 'Settings', and 'Details'. The 'Event' tab is selected. Under 'Trigger Event Type', a dropdown menu shows 'Receive UDP Message'. Below this are three input fields: 'Port' with the value '5000', 'Maximum Packet Length' with the value '1024', and 'Output Type' with a dropdown menu set to 'Binary'.

The **Event** tab becomes active with parameters that accommodate the Receive UDP Message event.

Parameter	Description
<b>Port</b>	The UDP port to receive messages from. The default UDP port is 5000. The valid values are 1 - 65545. A port value of 0 is not valid.
<b>Maximum Packet Length</b>	The maximum size of the message to receive. Any data in the datagram beyond this size is discarded.
<b>Output Type</b>	The data type to convert the received message to. The options are <b>Binary</b> or <b>String</b> .

## Receive UDP Message event type trigger event variables

The input event variables available to a Receive UDP Message event trigger are:

Event variable	Data type	Description
Data	Based on the option selected for the <b>Output Type</b> parameter	The data received from the specified UDP port.
Source Address	STRING	The IP address and port of the sending application, in the format ip:port.

## Receive UDP Message event trigger considerations

- If the UDP datagram received is larger than the **Maximum Packet Length** parameter, the extra data has been received and discarded, so it is not available for a future Receive UDP Message event.
- If two or more Receive UDP Message triggers are defined and started with the same **Port**, then one or more trigger(s) will be Disabled. Meaning only one trigger can be started for the UDP port.
- If multiple sending applications are sending UDP datagrams to the same IP address and port, then each sent UDP message will be received as a separate and distinct Receive UDP Message event trigger execution. The trigger execution behavior is based on the trigger settings for **Max in Progress** and **Queue Size**.

- A **Port** value of 0 is not in the valid port range.

## IIoTA industrial IoT Platform: TCP Listener Status

A **TCP Listener Status** event trigger executes when a TCP Listener device's status changes.

### Defining a TCP Listener Status event trigger

To define a TCP Listener Status event trigger, follow these steps:

1. From the Workbench left pane, expand the node where you want to define the TCP Listener Status event trigger.
2. Select the **Project** icon to display the **Projects** window, right-click a specific project tab to display its pop-up menu, and then select **New**.  
You also can select the **New** button at the bottom of the **Projects** window when a specific project tab has already been selected.
3. The new **Trigger** window appears.  
Name the trigger. The trigger name can be up to 64 characters and include letters, numbers, and the underscore character. Spaces are allowed.
4. From the **Event** tab, select the **Trigger Event Type** down-arrow, expand the **Networking** category, and then select **TCP Listener Status**.

The screenshot shows a configuration window for a trigger. At the top, there is a text input field for the **Name**. Below this, there are four tabs: **Event**, **Local Variables**, **Static Variables**, **Settings**, and **Details**. The **Event** tab is active. Under the **Trigger Event Type** label, a dropdown menu is open, showing **TCP Listener Status** selected. Below this, there are four rows of configuration options, each with a label, a dropdown menu, and a refresh icon:

- Device Name:** TCPListener
- Fire when a connection is established:** True
- Fire when a connection is lost:** True
- Fire when a connection is rejected:** True

The **Event** tab becomes active with parameters that accommodate the TCP Listener Status event.

Parameter	Description
<b>Device Name</b>	The TCP Listener device that supports the TCP connection that receives TCP messages from partner applications.
Fire when a connection is established	Select True or False. Fire the event when a connection to the selected TCP Listener device is established.
Fire when a connection is lost	Select True or False. Fire the event when a connection to the selected TCP Listener device is lost.
Fire when a connection is rejected	Select True or False. Fire the event when a connection to the selected TCP Listener device is rejected.

## TCP Listener Status event type trigger event variables

The input event variables available to a TCP Listener Status event trigger are:

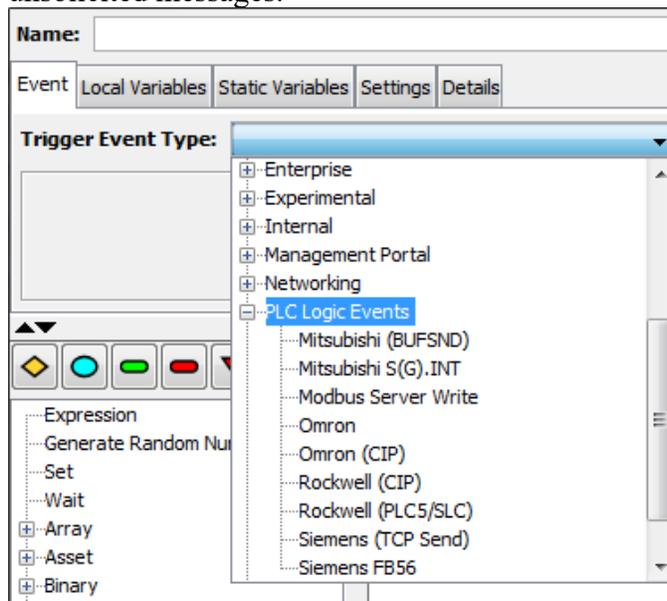
Event variable	Data type	Description
Device Name	STRING	The name of the TCP Listener device.
Source Address	STRING	The IP Address and port that the data was received from.
State	STRING	The status of the connection that fired the trigger. The values are:  Established Lost Rejected
Reason Code	INT4	The error code or status associated with the Reason String.
Reason String	STRING	The reason the connection was lost, established or rejected.

## TCP Listener Status event trigger considerations

- The selected TCP Listener device must be Started for the TCP Listener Status event trigger to be executed.

## IIoTA industrial IoT Platform: PLC Logic Events

The **PLC Logic Events** category contains the event types related to the device drivers' support of unsolicited messages.



The unsolicited message types supported is dependent on the device drivers that are installed on the node.

For information on each of the unsolicited message event types, see the specific driver information in Device types.

## IIoTA industrial IoT Platform: Trigger actions reference

### Overview

The main concepts of a trigger are:

- The trigger's event type
- The trigger's local variables, static variables, macros and event variables

- The trigger's settings
- The trigger's actions, including the success and failure routes between actions.

Every trigger identifies the actions, which are the trigger's application logic. These actions range in complexity from actions such as formatting strings, or setting values, to more complex actions such as executing enterprise transactions or executing remote triggers.

While the actions defined in a trigger identify the application logic, the trigger event type identifies when the trigger will execute. The trigger event types are covered in the Trigger event type reference.

## Assumptions

Before using information in this Trigger actions reference section, the following should have occurred:

- The Workbench was installed on a computer that has TCP connectivity to the node.
- You have a user ID and password to log on and use features of the Workbench.
- You understand how to create a trigger.

## General trigger action concepts

The general project and trigger concepts are covered in the first few sections of Projects and triggers.

Each action has a routing, which determines what action to go to next. The Canvas Editor and the List Editor have different mechanisms for defining each actions' routing. For more information, see Trigger actions.

Each trigger and each action have a **Details** tab where comments or information can be stored. For more information, see Trigger details.

## Entering action parameter values

Each action has the specific parameters needed for the action to provide its function.

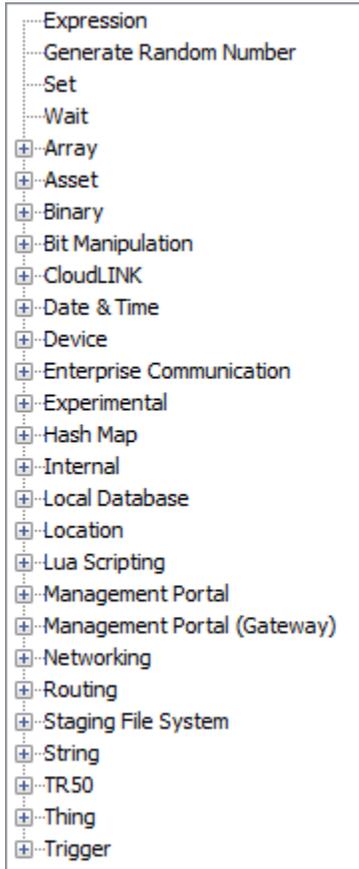
When a parameter can be specified by a device variable, trigger macro, constant, trigger local variable, trigger static variable or trigger event variable, the available options are displayed by clicking once in the **Value** cell for the parameter and then selecting the down arrow icon.

Only applicable options are displayed. For example:

- Macros and Constant do not apply for an output parameter
- Device variables are only available if the device is in a started state.

## How this reference is organized

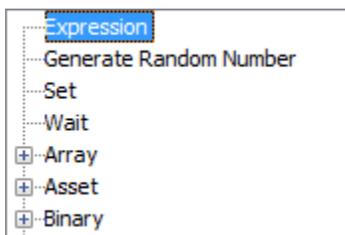
The organization of this Trigger actions reference section is based on the hierarchy of the available actions as displayed in the Workbench when defining a trigger, for example:



The information in this Trigger actions reference section covers the different trigger actions available across the different products. The specific trigger actions available on a node depends on the features that are part of the packages or extensions installed on the node.

## IIoTA industrial IoT Platform: Expression

The **Expression** action is a mathematical operation that takes a variable number of input arguments and returns a single output argument. An expression is any legal combination of symbols that represent a value. The system has rules for what symbols are legal and illegal.



## Overview

Every expression consists of at least two operands and can have one or more operators. Operands are values, whereas operators are symbols that represent particular actions.

In the expression  $X + Y - 10$

X, Y, and 10 are operands, and + and - are operators.

The following lists the valid operators.

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Exponentiation (for example: $2^3 = 8$ )
%	Modulus (for example: $5\%2 = 1$ )
()	Force precedence of evaluation
>>	Bitwise shift right
<<	Bitwise shift left
&	Not implemented as an operator, see the and function below.
	Not implemented as an operator, see the or function below.
^	Not implemented as an operator, see the xor function below.
~	Not implemented as an operator, see the not function below.

The following lists the valid logical operators for conditional operations:

Logical Operator	Description
==	Equal
!=	Not Equal
<	Less Than
<=	Less Than or Equal

>	Greater Than
>=	Greater Than or Equal
&&	And (for example: $x==5 \ \&\& \ y==10$ returns true if x is 5 and y is 10)
	Or (for example: $x==5 \    \ y==10$ returns true if x is 5 or y is 10)
!	Not (for example: $!(x==5)$ returns false if x is 5)

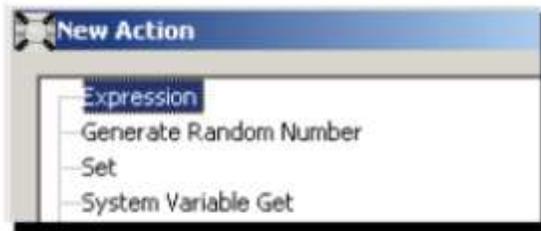
There are also several built-in functions that can be used in an expression, such as  $\sin(X)$  and  $\log(X)$ .

## Procedures

The **Expression** action is available as follows:

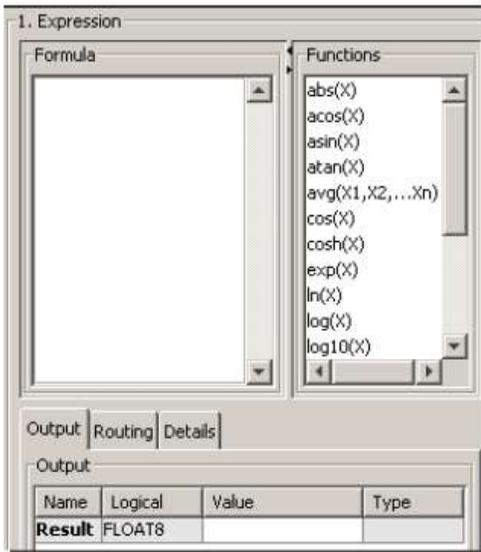
1. From the bottom the **Actions** tab, select **Add**.

The New Action window appears.



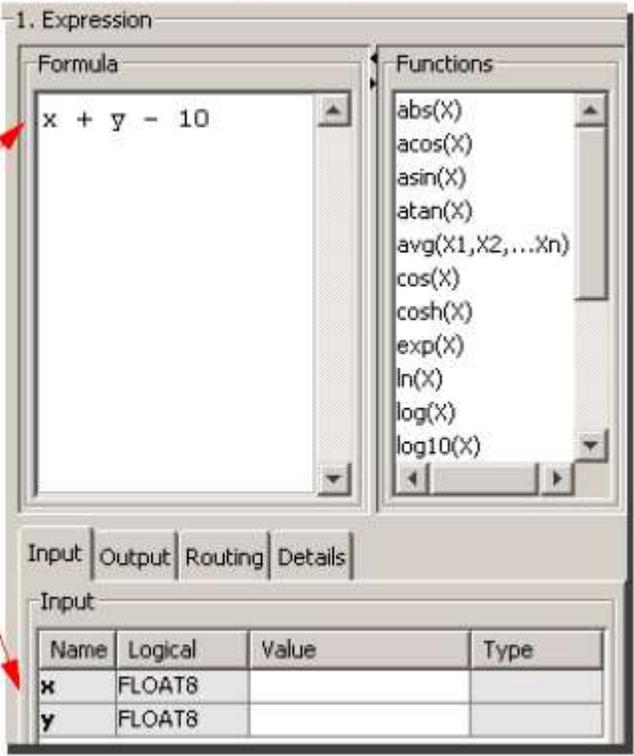
2. Select **Expression**, and then select **Add**.

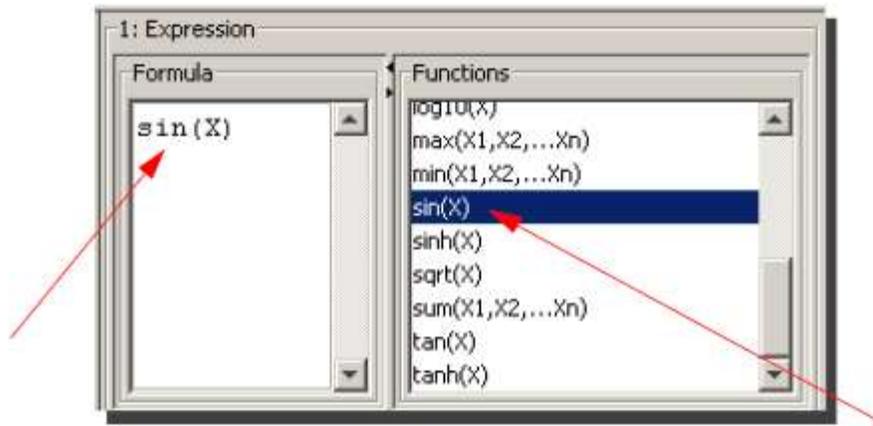
The right pane changes to accommodate an **Expression** action.



## Parameter description

The top of the Expression action provides the **Formula** and **Functions** parameters that accommodate creating an expression.

<p><b>Formula</b></p>	<p>Use the <b>Formula</b> pane to type an expression.</p>  <p>As you type in the <b>Formula</b> box, the logical variables used in the expression are automatically added to the <b>Input</b> tab. In the example expression, <math>x=y</math> (x is equal to y), the logical variables <b>x</b> and <b>y</b> are added to the input tab.</p>
<p><b>Functions</b></p>	<p>You can use built-in functions within an expression. To add a built-in function, double-click the function in the <b>Functions</b> list.</p>



The function is added to the **Formula** box. Note that the default input operand for the sin( ) function is **X**. You can change this to any value you want. When using the trig functions, the angle is specified in radians. For example, sin(1.0) = 0.841471. The 1.0 is equivalent to 1 radian: where 2 pi radians equals 360 degrees

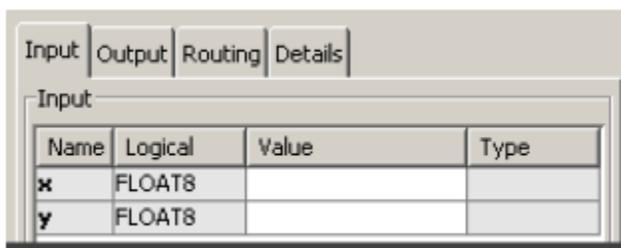
The following table lists the built-in functions that you can use within an Expression action.

Function	Description
abs(X)	Absolute value
acos(X)	Trigonometric arcsine
and(X,Y)	Bitwise AND "&"
asin(X)	Trigonometric arcsine
atan(X)	Trigonometric arctangent
avg(X1, X2...,Xn)	Average of a set of values
ceil(X)	Ceiling (round up)
cos(X)	Trigonometric cosine
cosh(X)	Hyperbolic cosine
exp(X)	e to the power X
floor(X)	Floor (round down)
ln(X)	Natural log (base e)
log(X)	Natural log (base e)

log10(X)	Log base 10
max(X1, X2...,Xn)	Maximum of a set of values
min(X1, X2...,Xn)	Minimum of a set of values
not(X)	Bitwise NOT "~"
or(X,Y)	Bitwise OR " "
sin(X)	Trigonometric sine
sinh(X)	Hyperbolic sine
sqrt(X)	Square root
sum(X1, X2...,Xn)	Sum of a set of values
tan(X)	Trigonometric tangent
tanh(X)	Hyperbolic tangent
xor(X,Y)	Bitwise XOR "^"

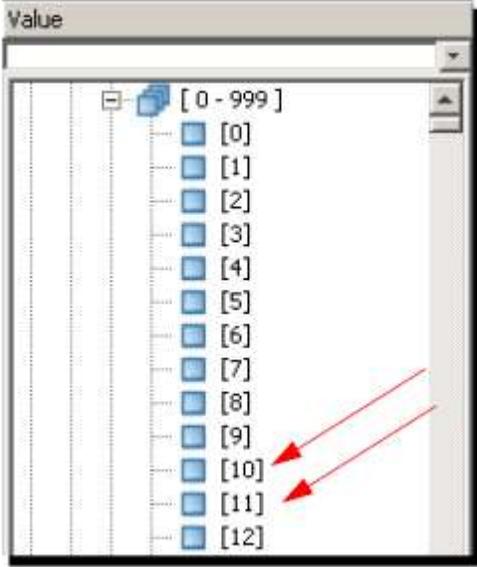
## Input tab

The following describes the **Input** tab available from the bottom of the **Expression** action pane. In the example expression,  $X + Y - 10$  will be evaluated. However, before the trigger can evaluate this expression, values need to be assigned to both **X** and **Y**. The **Input** tab lets you assign values to the expression using trigger variables.



The following describes the columns on the **Input** tab.

Column	Description
<b>Name</b>	This column provides the expression variables. The expression is automatically added when you type the expression in the <b>Formula</b> box or double-click a built-in expression function from the <b>Functions</b> list. The variables used in the expression are just placeholders. They are not local variables, static variables, or device variables.

<p><b>Logical</b></p>	<p>By default, all variables used in an expression have the data type <b>FLOAT8</b> which is an 8 byte floating point number.          Depending on your system, you might see <b>LREAL</b> as the default data type. LREAL is also an 8 byte floating point number.          When you assign the target value to hold the result of the expression, you can convert this FLOAT value to a different data type.</p>
<p><b>Value</b></p>	<p>The column lets you assign the trigger variable you want to use as input for the expression variable.          Select the <b>Value</b> column to select from a list of device variables, or a user defined local or static variable. For this example, the actual PLC device variable Local CPU 1.D[10] is assigned to <b>X</b>, and Local CPU 1.D[11] is assigned to <b>Y</b></p>  <p>The screenshot shows a window titled 'Value' with a list of variables from [0] to [12]. Each variable has a blue square icon to its left. Two red arrows point to the variables [10] and [11].</p>
<p><b>Type</b></p>	<p>When you specify <b>Value</b>, the default data type of the variable is automatically added to the <b>Type</b> column. You can override the default data type by selecting a different data type from the list.</p>

When the trigger executes and evaluates this expression, the following actions will occur:

- Get the value of the device variable data type INT2 Local CPU 1.D[10].
- Get the value of the device variable data type INT2 Local CPU 1.D[11].
- Add these 2 values together.
- Subtract 10 from the value.

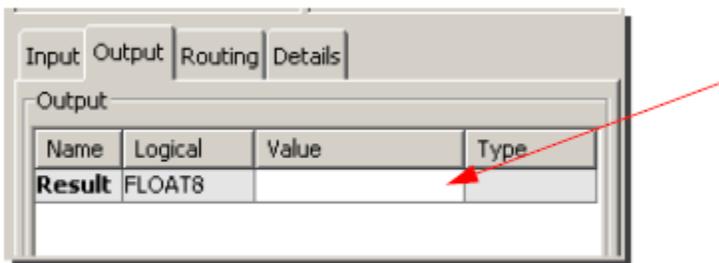
The last step is to specify the location to store the result of the expression.

## Output tab

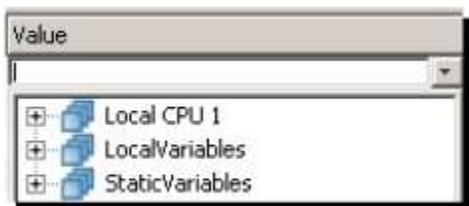
The **Output** tab for an **Expression** action lets you specify the location to store the result of the expression as follows:

### Variables

Use the **Variables** parameter to assign the variable that will store the result of the expression evaluation.



Select the **Value** column and then use the down-arrow to display a list of trigger variables.

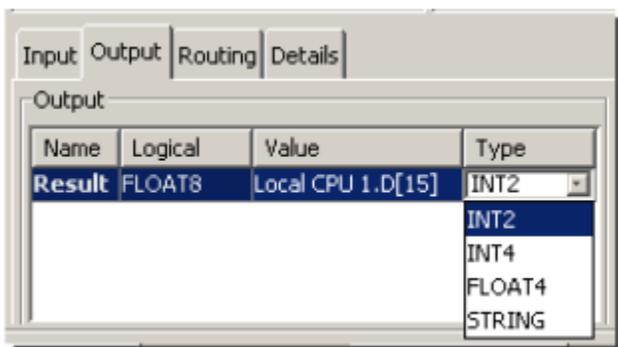


The list includes all started devices (for example Local CPU1), Local Variables, Static Variables, and Event Variables.

When you assign a device variable, the **Type** drop-down list becomes available with the associated data type of the variable.

### Type

Use the **Type** drop-down list if the data type of the device variable needs to be changed.



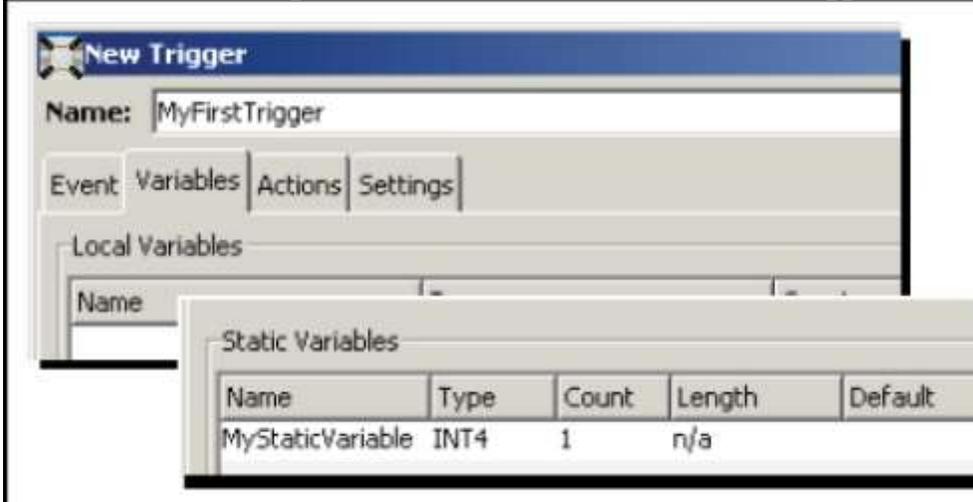
Only valid data types are listed. For this example, PLC device variable Local CPU1.D[15] is assigned to hold the result of the expression.

To review. When the trigger executes, the expression is evaluated and the following will occur:

- Add Local CPU1.D[10] to Local CPU1.D[11]
- Subtract 10 from the value
- Store the result in Local CPU1.D[15]

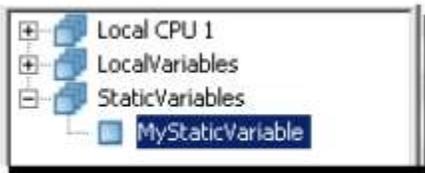
### Specifying a static variable to store the result of the expression

You can also specify a static variable as the recipient of an expression evaluation. A static variable retains its value while the trigger is in a started state. It is assumed that you have defined the static variable using the **Static Variables** tab on the New Trigger window.

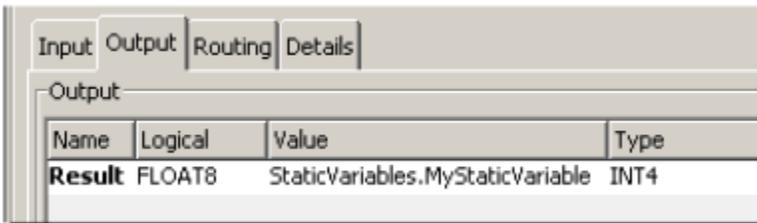


When the static variable is defined, its initial value is created when the trigger is started. The value (and possibly modified value) continues to exist while the trigger is in a started state and is lost when the trigger is stopped.

The steps to add a static variable to the **Output** tab for the expression are the same as a device variable, except you select a static variable from the list.



The **Value** column will contain the static variable.



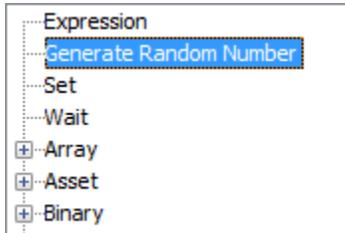
Notice the **Type** drop-down list is unavailable. You cannot change the data type of a static variables from this list.

Related topics

If

# IIoTA industrial IoT Platform: Generate Random Number

You can generate random numbers or strings for a single or array variable. The **Generate Random Number** action allows you to initialize or populate variables for any purpose such as simulating an environment and testing an application program.



## Adding a Generate Random Number action

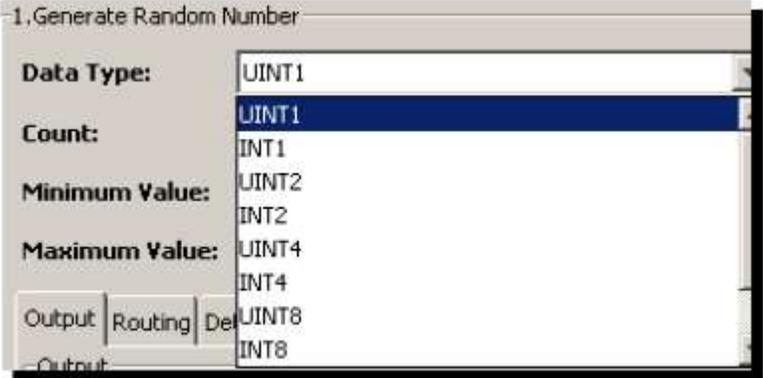
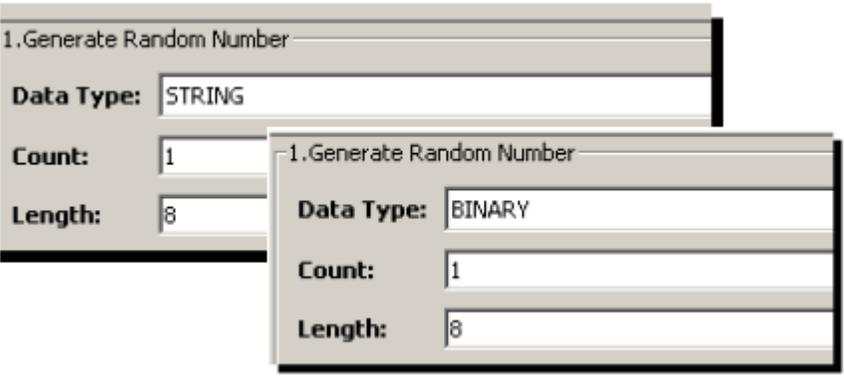
The **Generate Random Number** action is available from the Trigger window as follows:

 A screenshot of a configuration window titled '1. Generate Random Number'. It contains several input fields: 'Data Type' (UINT1), 'Count' (1), 'Minimum Value' (0), and 'Maximum Value' (0). Below these fields are three tabs: 'Output', 'Routing', and 'Details'. The 'Output' tab is active, showing a table with the following data:
 

Name	Logical	Count	Value	Type
<b>Value</b>	UINT1	1		

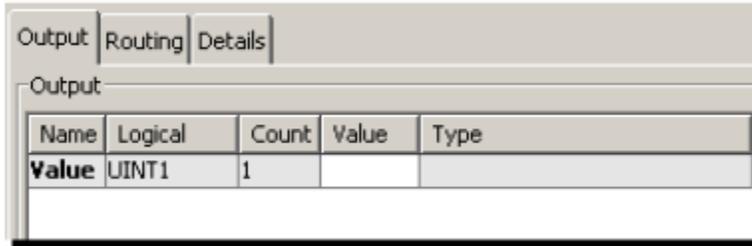
## Parameter description

The Generate Random Number pane provides these parameters:

Parameter	Description
<b>Data Type</b>	<p>A list of logical internal data types are provided:</p>  <p>This is the data type that the random number generator algorithm uses to determine the range of values within which to generate the random data. When the data type is selected, the value in the <b>Logical</b> column on the <b>Output</b> tab changes accordingly. Parameters on the Generate Random Number pane also change to support <b>String</b> and <b>Binary</b> data types.</p> 
<b>Count</b>	Indicates the number of values to return in the <b>Output</b> tab Value parameter. A value of 1 indicates a single value. A value greater than 1 indicates an array.
<b>Minimum Value</b>	Indicates the minimum value that can be generated from the action.
<b>Maximum Value</b>	Indicates the maximum value that can be generated from the action.
<b>Length</b>	Used in conjunction with <b>String</b> and <b>Binary</b> data types. Specifies the maximum characters (for the string) or maximum number of bytes (for the binary buffer) that can be generated from the action.

## Output tab

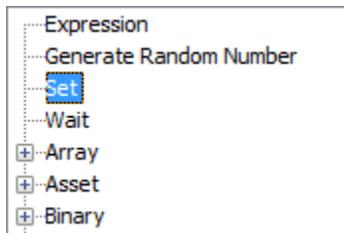
The **Output** tab is used to specify the destination of the randomized variable.



Parameter	Description
<b>Value</b>	The variable where the random number is returned. This could be a scalar (single) variable or an array, as specified by the <b>Count</b> parameter.

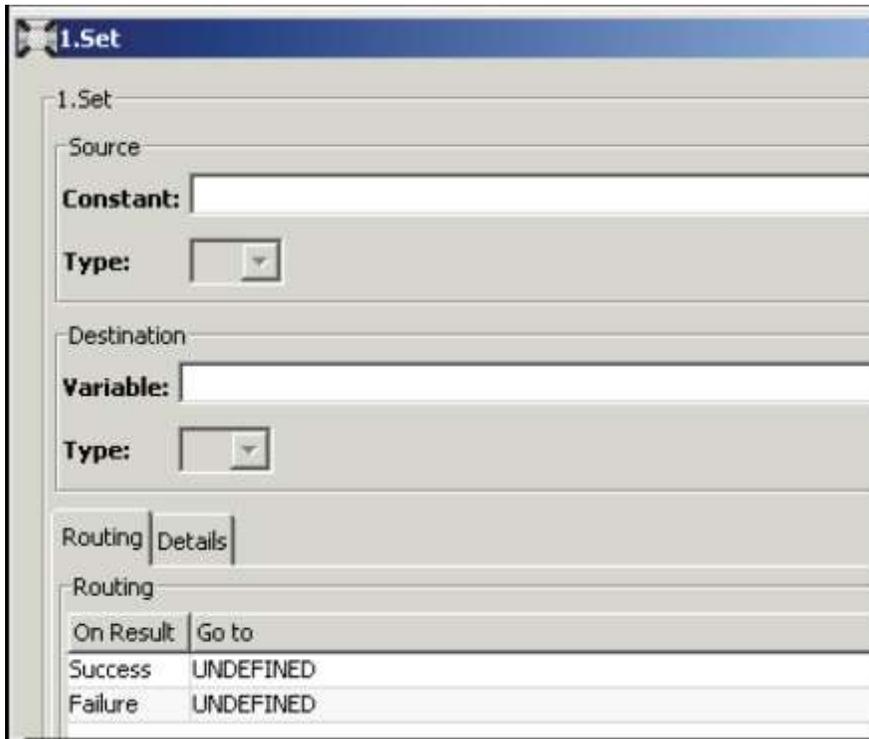
## IIoTA industrial IoT Platform: Set

The **Set** action is used to copy data from a source location (or variable) to a destination location (or variable).



### Parameter description

Within the **Set** pane, there are **Source** and **Destination** sections.



Parameter	Description
Source	<p>The <b>Source</b> section changes to accommodate the type of source. The first step is to specify a source variable, constant, or macro to use to copy a value from. This is the source whose value will be copied to the destination variable.</p> <ul style="list-style-type: none"> <li>• <b>Constant</b> Use the <b>Constant</b> parameter to enter the source variable.</li> <li>• <b>Type</b> The <b>Type</b> drop-down list is only available for some device variables. When available, use the <b>Type</b> down-arrow to change the data type of the source variable. This parameter also provides read-only data type information for other source variables, such as trigger macros.</li> <li>• <b>Length</b> This is a read-only parameter that shows the length of source variables of the STRING data type.</li> <li>• <b>Count</b> The <b>Count</b> box becomes available when both the source and destination variables are arrays. You enter the number of elements in the array in the <b>Count</b> parameter. When you enter a value for the source <b>Count</b> parameter, that value is also automatically entered into the destination <b>Count</b> parameter. When it is</li> </ul>

	available, the count parameter for both the source and the destination variables will be the same value. If no value is entered, a count of 1 is used.
<b>Destination</b>	<p>The second step is to specify a destination variable to copy the source value to.</p> <p>The <b>Destination</b> section changes based on the values entered into the source parameters.</p> <p>The value is supplied when the trigger executes at runtime.</p> <p>The value could be a constant value that doesn't change, such as the number 10.</p> <p>The value could be dynamic, such as a device variable that can be updated by other parts of the system.</p> <p>The value could be determined at runtime, such as the trigger macros.</p> <ul style="list-style-type: none"> <li>• <b>Variable</b> To specify a variable as the destination variable, under <b>Destination</b>, select the <b>Variable</b> down-arrow to display a list. The list is limited to started device variables, local variables, static variables and event variables. Expand the list of variables to locate the destination variable. Select the appropriate variable. For this example, <i>MyStaticVariable</i>. The variable is added to the <b>Variable</b> parameter.</li> <li>• <b>Type</b> To change the data type of some device variables, use the <b>Type</b> drop-down list.</li> <li>• <b>Length</b> This is a read-only parameter that shows the length of destination variables of the STRING data type.</li> <li>• <b>Count</b> The <b>Count</b> parameter becomes available when both the source and destination variables are arrays. You enter the number of elements in the array in the <b>Count</b> parameter. When you enter a value for the destination <b>Count</b> parameter, that value is also automatically entered into the source <b>Count</b> parameter. When it is available, the count parameter for both the source and the destination variables will be the same value. If no value is entered, a count of 1 is used.</li> </ul>

## IIoTA industrial IoT Platform: Examples of the Set action

This section describes examples of the **Set** action.

### Example 1 (INT2 source)

Example 1 shows the source as one INT2 of PLC data (2 bytes) which will be copied from D[5] in the PLC to the local variable called **LocalA**.

The screenshot shows a configuration window titled "1.Set". It is divided into two main sections: "Source" and "Destination".  
 In the "Source" section:  
 - "Variable:" is set to "Local CPU 1.D[5]"  
 - "Type:" is set to "INT4" (indicated by a dropdown arrow)  
 In the "Destination" section:  
 - "Variable:" is set to "LocalVariables.LocalA"  
 - "Type:" is set to "INT2" (indicated by a dropdown arrow)

Because it is an INT2, this will be an integer value between -32768 and 32767.

### Example 2 (INT4 source)

Example 2 shows the source as one INT4 of PLC data (4 bytes) which will be copied from D[8] in the PLC to the static variable called **StaticB**

The screenshot shows a configuration window titled "1.Set". It is divided into two main sections: "Source" and "Destination".  
 In the "Source" section:  
 - "Variable:" is set to "Local CPU 1.D[8]"  
 - "Type:" is set to "INT4" (indicated by a dropdown arrow)  
 In the "Destination" section:  
 - "Variable:" is set to "StaticVariables.StaticB"  
 - "Type:" is set to "INT4" (indicated by a dropdown arrow)

The PLC data points used will be D[8] and D[9]. Because it is an INT4, this could be a large value between - 2.147 billion and + 2.147 billion.

### Example 3 (STRING source)

Example 3 shows the source as 8 bytes of PLC data which will be copied from D[10] in the PLC to the static variable called **StaticD**.

The screenshot shows a configuration window titled "1.Set". It has two main sections: "Source" and "Destination".

- Source:** Variable: Local CPU 1.D[10], Type: STRING, Length: 8.
- Destination:** Variable: StaticVariables.StaticD, Type: STRING, Length: 8.

The PLC device variables used will be the 4 INT2s D[10], D[11], D[12], and D[13], since two (2) ASCII characters can be stored in one INT2. For example, if the ASCII string starting at D[10] was **ABCDEFGH**, then this data would be moved to StaticD. D[10] contains **AB**, D[11] contains **CD**, and so forth.

#### Example 4 (FLOAT4 source)

Example 4 shows one FLOAT4 of PLC data (4 bytes) which will be copied from D[20] in the PLC to the FLOAT4 static variable called **StaticC**.

The screenshot shows a configuration window titled "1.Set". It has two main sections: "Source" and "Destination".

- Source:** Variable: Local CPU 1.D[20], Type: FLOAT4.
- Destination:** Variable: StaticVariables.StaticC, Type: FLOAT4.

The PLC device variables used will be D[20] and D[21]. Because the data type is a FLOAT4, this will be a floating-point number.

## IIoTA industrial IoT Platform: Set action and data conversion considerations

The runtime can perform data conversions as part of the **Set** action. Because the system is designed to support devices from multiple vendors, and not all vendors support the same data types, there may be cases where the source data type and the destination data type are not compatible.

For example, you can copy data from an INT2 to an INT4. In this case, the 2 byte integer represented by the INT2 will be expanded to an INT4, or 4 bytes. Typically, this is permitted.

The range of values represented by an INT2 is -32768 to +32767. These values can also be represented within an INT4 as well.

However, you should be careful when moving data to other data types where possible conversion might distort the meaning of the data.

### Example 1 (INT2 to a static string)

The example will copy 2 bytes of data from D[20] in the PLC to a static string variable called **StaticE**.

The screenshot shows a configuration window titled "1. Set". It is divided into two main sections: "Source" and "Destination".

- Source:**
  - Variable: Local CPU 1.D[20]
  - Type: INT2
- Destination:**
  - Variable: StaticVariables.StaticE
  - Type: STRING
  - Length: 8

It is possible that the data in D[20] does not represent a valid pair of ASCII characters. Therefore, if the data is hexadecimal x00FF, which is numerically 255, this cannot be interpreted as valid ASCII characters. If the data is hexadecimal x4142, which is numerically 16,706, this could be interpreted as the ASCII string **AB**.

### Example 2 (INT4 to a FLOAT4)

The example will copy an INT4 (4 bytes) from D[20] to **StaticC** a FLOAT4.

The screenshot shows a configuration window titled "1. Set". It is divided into two main sections: "Source" and "Destination".

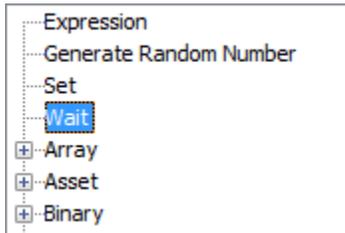
- Source:**
  - Variable: Local CPU 1.D[20]
  - Type: INT4
- Destination:**
  - Variable: StaticVariables.StaticC
  - Type: FLOAT4

The example might not be valid since FLOAT4 data is represented internally as an exponent and mantissa, both of which are represented in binary form within the 4 bytes.

As a general rule, when moving data, try to move data between similar data types to avoid possible errors.

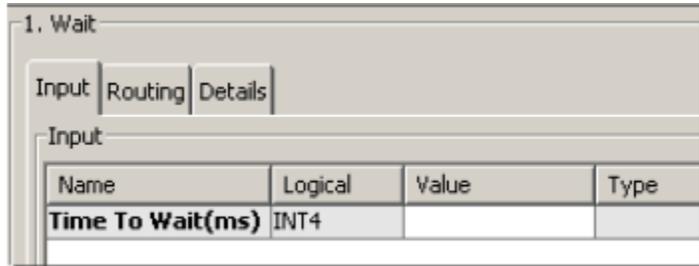
## IIoTA industrial IoT Platform: Wait

The **Wait** action provides a delay in trigger execution before advancing to the next action.



### Input tab

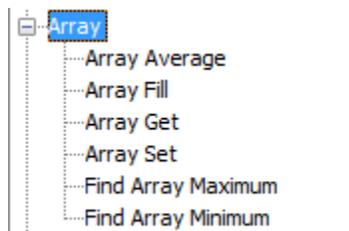
The Input tab is where you set the wait duration to pause before the next action is executed.



Parameter	Description
<b>Time To Wait(ms)</b>	The amount of time the trigger will wait in milliseconds. The input value's range is +/- 1.7976931348623157e+308. If the value is less than one, no wait occurs and execution proceeds immediately to the next action. Although you can use any value for the wait duration, 50 milliseconds is the minimum length of time a trigger can be paused.

## IIoTA industrial IoT Platform: Array

The **Array** category provides actions that operate on array variables.



An array is a data structure that contains several variables called the elements of the array. The array elements are accessed through computed indexes. For the system, the array indexes start at zero.

All the array elements must be of the same type (the element type of the array). Array elements can be of any type, including an array type. An array can be a single or a multi-dimensional array.

The array actions perform fast operations on arrays of data.

The **Array** category provides these actions:

- Array Average
- Array Fill
- Array Get
- Array Set
- Find Array Maximum
- Find Array Minimum

## IIoTA industrial IoT Platform: Array Average

The **Array Average** action calculates the average value for a range of array elements.

1. Array Average

**Data type:**

**Maximum number of elements:**

Input | Output | Routing | Details

Input

Name	Logical	Count	Value	Type
<b>Input Array</b>	INT1	8		
<b>Input Count</b>	INT2	1		

## Parameter descriptions

The action provides these parameters:

Parameter	Description
<b>Data type</b>	The list of the available data types for the node. For example, BYTE, WORD, DWORD, or INT1, INT2, INT4 and so forth.
<b>Maximum number of elements</b>	The value indicates the maximum number of array elements that can be passed into the array variable. The maximum number of elements can be any size.

## Input tab

Name	Logical	Count	Value	Type
<b>Input Array</b>	INT1	8		
<b>Input Count</b>	INT2	1		

Parameter	Description
<b>Input Array</b>	To calculate the average value from an array, the array must first be passed into the action. The starting element for the average calculation is entered in the <b>Value</b> column. The values for <b>Logical</b> and <b>Count</b> columns will change based on the <b>Data type</b> and <b>Maximum number of elements</b> parameters specified in the action definition.
<b>Input Count</b>	The <b>Input Count</b> holds the number of items actually passed into the action. This number must be less than or equal to the maximum number of elements in the array. If the count exceeds the value in <b>Maximum number of elements</b> , an error will occur at runtime.

## Output tab

Name	Logical	Count	Value	Type
<b>Average</b>	FLOAT8	1		

Parameter	Description
<b>Average</b>	The calculated average of the array elements, based on the <b>Input Array</b> and <b>Input Count</b> parameters.

## IIoTA industrial IoT Platform: Array Fill

The **Array Fill** action sets the values of an array's elements, based on an index and count, to a specified value.

The result of the array update, the entire array with an updated element value, is returned as a separate output variable.

1. Array Fill

**Array Data Type:**

**Array Size:**

Input Output Routing Details

Input

Name	Logical	Count	Value	Type
<b>Array</b>	UINT1	1		
<b>Index</b>	UINT4	1		
<b>Count</b>	UINT4	1		
<b>Value</b>	UINT1	1		

## Parameter descriptions

The action provides these parameters:

Parameter	Description
<b>Array Data Type</b>	The data type of the array elements. The list of the available data types for the node. For example, BYTE, WORD, DWORD, or INT1, INT2, INT4 and so forth.
<b>Array Size</b>	The total number of elements in the array.

## Input tab

Parameter	Description
<b>Array</b>	The source array (the first element) is entered in the <b>Value</b> column. The result of the <b>Array Fill</b> action is copied from the internal replica of the source array to the destination array. If you want to update a source array's elements directly with the value specified, then the <b>Output tab Array</b> parameter should be set to the same device variable as the <b>Input tab Array</b> parameter.
<b>Index</b>	The starting array index element to be set. The <b>Count</b> parameter specifies how many elements to fill. Array indexes start at zero.
<b>Count</b>	The number of array elements to fill (set) with the <b>Value</b> .
<b>Value</b>	The value to be copied into the array elements.

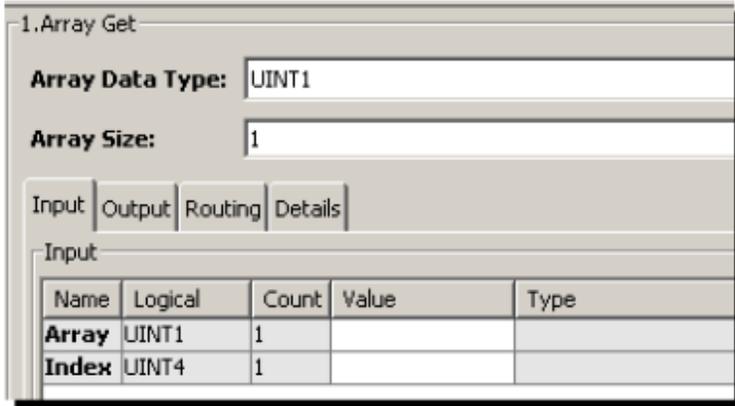
## Output tab

Parameter	Description
<b>Array</b>	Specifies the destination array that stores the result of the <b>Array Fill</b> action. The result of the <b>Array Fill</b> action is copied from the internal replica of the source array to the destination array. If you want to update a source array's elements directly with the value specified, then the <b>Output tab Array</b> parameter should be set to the same device variable as the <b>Input tab Array</b> parameter.

## IIoTA industrial IoT Platform: Array Get

The **Array Get** action returns the value of an array element, based on an index.

This can be used in a loop function to iterate through all the elements in the array.

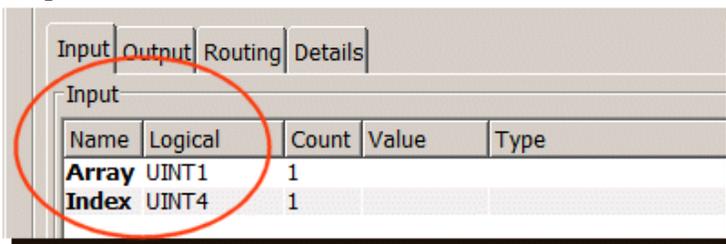


## Parameter descriptions

The action provides these parameters:

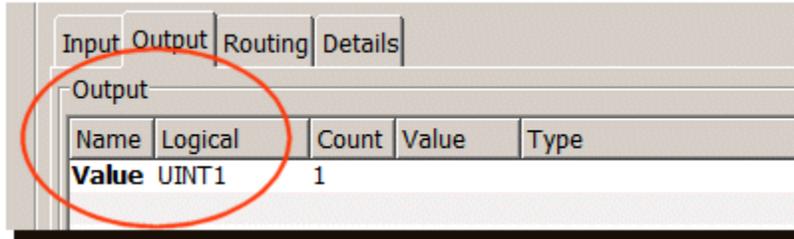
Parameter	Description
<b>Array Data Type</b>	The data type of the array elements. The list of the available data types for the node. For example, BYTE, WORD, DWORD, or INT1, INT2, INT4 and so forth.
<b>Array Size</b>	The number of elements in the array.
<b>Length</b>	If you select <b>BINARY</b> or <b>STRING</b> as the array data type, the <b>Length</b> parameter becomes available. <div data-bbox="418 1205 1029 1472" data-label="Image"> </div> <p>The value is the length to use for each string or binary data element.</p>

## Input tab



Parameter	Description
<b>Array</b>	The start of the array (the first element) is entered in the <b>Value</b> column. The values for <b>Logical</b> and <b>Count</b> columns will change based on the <b>Array Data type</b> and <b>Array Size</b> parameters specified in the action definition.
<b>Index</b>	The array index element value to be copied from. Array indexes start at zero.

## Output tab

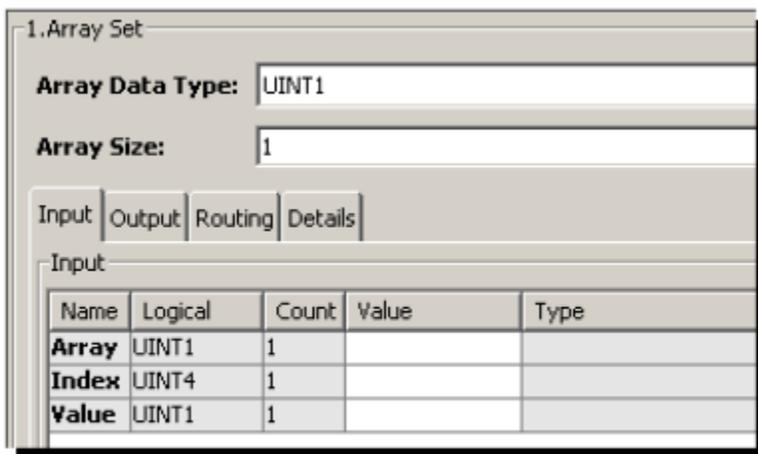


Parameter	Description
<b>Value</b>	The destination variable where the array element value is copied to.

## IIOA industrial IoT Platform: Array Set

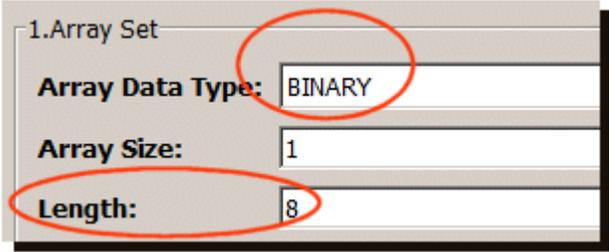
The **Array Set** action sets the value of an array element, based on an index, to a specified value.

The result of the array update, the entire array with an updated element value, is returned as a separate output variable.



## Parameter descriptions

The action provides these parameters:

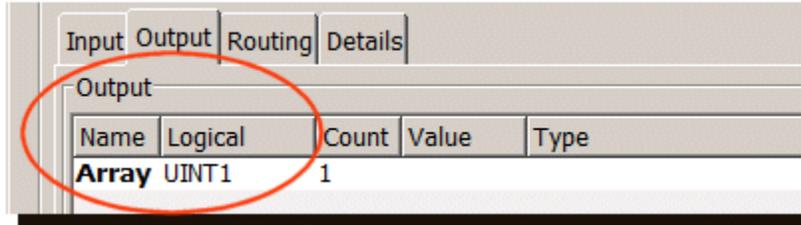
Parameter	Description
<b>Array Data Type</b>	The data type of the array elements. The list of the available data types for the node. For example, BYTE, WORD, DWORD, or INT1, INT2, INT4 and so forth.
<b>Array Size</b>	The number of elements in the array.
<b>Length</b>	If you select <b>BINARY</b> or <b>STRING</b> as the array data type, the <b>Length</b> parameter becomes available.  <p>The value is the length to use for each string or binary data element.</p>

## Input tab

Input	Output	Routing	Details	
Input				
Name	Logical	Count	Value	Type
<b>Array</b>	UINT1	1		
<b>Index</b>	UINT4	1		
<b>Value</b>	UINT1	1		

Parameter	Description
<b>Array</b>	The source array (the first element) is entered in the <b>Value</b> column. The result of the <b>Array Set</b> action is copied from the internal replica of the source array to the destination array. If you want to update a source array directly with the value specified, then the <b>Output tab Array</b> parameter should be set to the same device variable as the <b>Input tab Array</b> parameter.
<b>Index</b>	The array index element value to be set. Array indexes start at zero.
<b>Value</b>	The value to be copied into the array element.

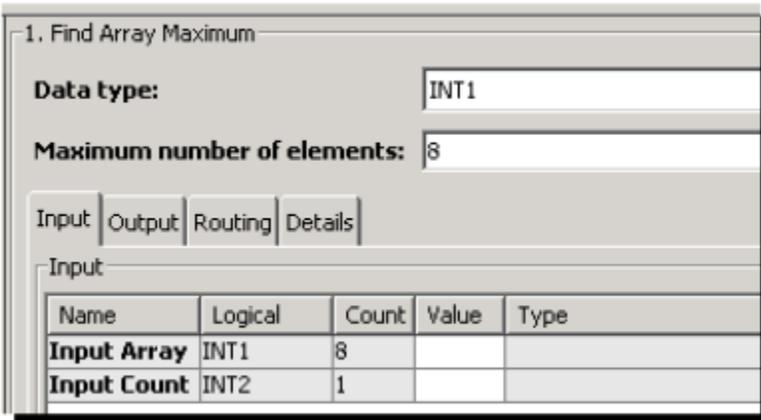
## Output tab



Parameter	Description
<b>Array</b>	Specifies the destination array that stores the result of the <b>Array Set</b> action. The result of the <b>Array Set</b> action is copied from the internal replica of the source array to the destination array. If you want to update a source array directly with the value specified, then the <b>Output tab Array</b> parameter should be set to the same device variable as the <b>Input tab Array</b> parameter.

## IIOA industrial IoT Platform: Find Array Maximum

The **Find Array Maximum** action returns the maximum value for a range of array elements.

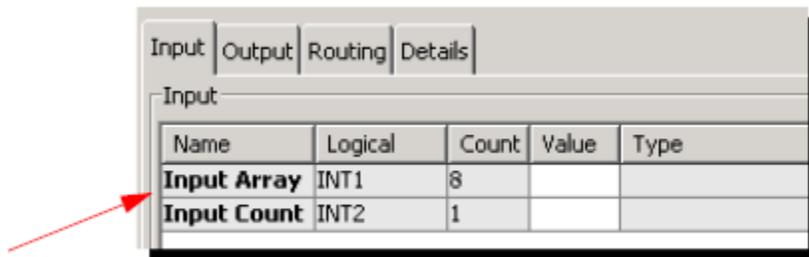


## Parameter descriptions

The action provides these parameters:

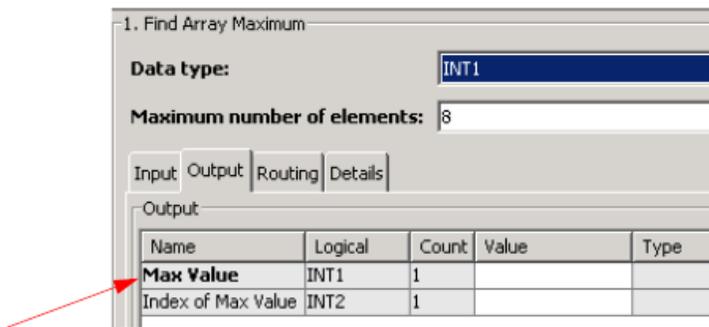
Parameter	Description
<b>Data type</b>	The data type of the array elements. The list of the available data types for the node. For example, BYTE, WORD, DWORD, or INT1, INT2, INT4 and so forth.
<b>Maximum number of elements</b>	The value indicates the maximum number of array elements that can be passed into the array variable. The maximum number of elements can be any size.

## Input tab



Parameter	Description
<b>Input Array</b>	To find the maximum value within an array, the array must first be passed into the action. The starting element for the maximum calculation is entered in the <b>Value</b> column. The values for <b>Logical</b> and <b>Count</b> columns will change based on the <b>Data type</b> and <b>Maximum number of elements</b> parameters specified in the action definition.
<b>Input Count</b>	The <b>Input Count</b> is the number of items actually passed into the action. This number must be less than or equal to the maximum number of elements in the array. If the count exceeds the <b>Maximum number of elements</b> , an error will occur at runtime.

## Output tab



Parameter	Description
<b>Max Value</b>	The maximum value of the array elements, based on the <b>Input Array</b> and <b>Input Count</b> parameters.
<b>Index of Max Value</b>	The index in the array for the maximum value of the array elements, based on the <b>Input Array</b> and <b>Input Count</b> parameters. The index starts at zero for the first item in the array; thus, the maximum number is N-1 where N is the number of items in the array. The index is optional as you do not have to map the result.

## IIoTA industrial IoT Platform: Find Array Minimum

The **Find Array Minimum** action returns the minimum value for a range of array elements.

1. Find Array Minimum

**Data type:** INT1

**Maximum number of elements:** 8

Input | Output | Routing | Details

Input

Name	Logical	Count	Value	Type
<b>Input Array</b>	INT1	8		
<b>Input Count</b>	INT2	1		

### Parameter descriptions

The action provides these parameters:

Parameter	Description
<b>Data type</b>	The data type of the array elements. The list of the available data types for the node. For example, BYTE, WORD, DWORD, or INT1, INT2, INT4 and so forth.
<b>Maximum number of elements</b>	The value indicates the maximum number of array elements that can be passed into the array variable. The maximum number of elements can be any size.

## Input tab

Name	Logical	Count	Value	Type
<b>Input Array</b>	INT1	8		
<b>Input Count</b>	INT2	1		

Parameter	Description
<b>Input Array</b>	To find the minimum value within an array, the array must first be passed into the action. The starting element for the minimum calculation is entered in the <b>Value</b> column. The values for <b>Logical</b> and <b>Count</b> columns will change based on the <b>Data type</b> and <b>Maximum number of elements</b> parameters specified in the action definition.
<b>Input Count</b>	The <b>Input Count</b> is the number of items actually passed into the action. This number must be less than or equal to the maximum number of elements in the array. If the count exceeds the <b>Maximum number of elements</b> , an error will occur at runtime.

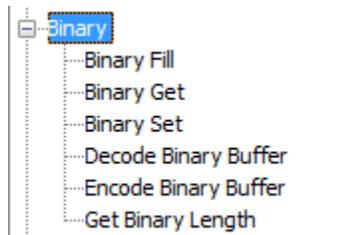
## Output tab

Name	Logical	Count	Value	Type
<b>Min Value</b>	INT1	1		
<b>Index of Min Value</b>	INT2	1		

Parameter	Description
<b>Min Value</b>	The minimum value of the array elements, based on the <b>Input Array</b> and <b>Input Count</b> parameters.
<b>Index of Min Value</b>	The index in the array for the minimum value of the array elements, based on the <b>Input Array</b> and <b>Input Count</b> parameters. The index starts at zero for the first item in the array; thus, the maximum number is N-1 where N is the number of items in the array. The index is optional as you do not have to map the result.

## IIoTA industrial IoT Platform: Binary

The **Binary** category provides actions on binary variables.

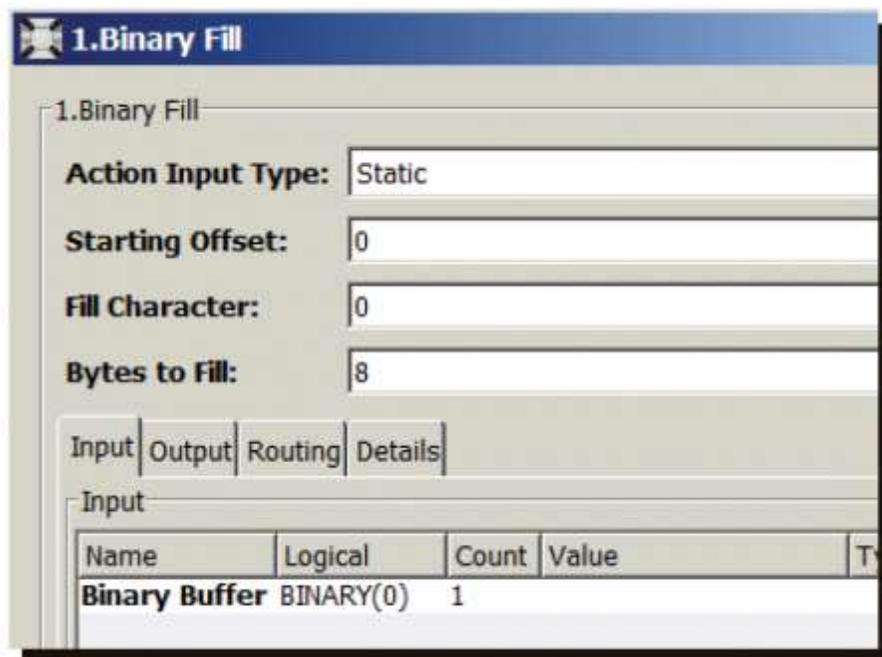


The Binary category provides these actions:

- Binary Fill
- Binary Get
- Binary Set
- Decode Binary Buffer
- Encode Binary Buffer

## IIoTA industrial IoT Platform: Binary Fill

The **Binary Fill** action fills a BINARY variable with a specified one-byte value. You can use the action to initialize or clear a BINARY variable. In addition, you can choose to fill only a portion of the variable. The fill character is a one-byte value in the range of 0 to FF (the digits A - F can be entered in upper or lower case).



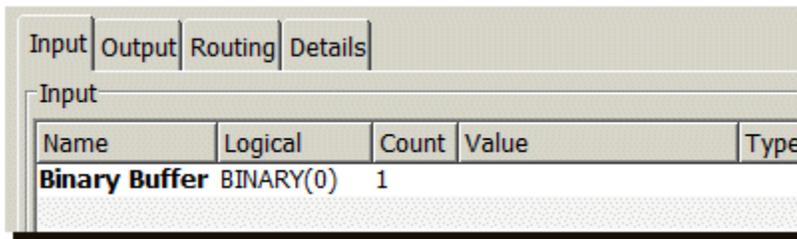
## Parameter descriptions

The action provides these parameters:

Parameter	Description
<b>Action Input Type</b>	The options are: <b>Static</b> - The specified <b>Starting Offset</b> cannot be changed unless you edit the trigger. <b>Dynamic</b> - The <b>Starting Offset</b> parameter becomes available from the <b>Input</b> tab.
<b>Starting Offset</b>	The first byte of the variable to fill with the <b>Fill Character</b> . The offset is calculated by the number of bytes counted from the beginning of the variable. If you select <b>Dynamic</b> as the <b>Action Input Type</b> , <b>Starting Offset</b> becomes available from the <b>Input</b> tab where you can specify a variable whose value can change at runtime.
<b>Fill Character</b>	The single byte value used to fill into the variable. The range value can be anywhere from 0 to FF (the digits A - F can be entered in upper or lower case). This one-byte value is entered as 2 digits using 0 - 9 and A - F. There is no need to prefix the one-byte value with "0x".
<b>Bytes to Fill</b>	The number of the bytes to fill. The default is 8. The range value can be anywhere from 1 to 2147483647 but cannot be larger than the length of the BINARY variable.

## Input tab (Static)

If you select **Static** as the value for **Action Input Type**, the Input tab appears as follows:

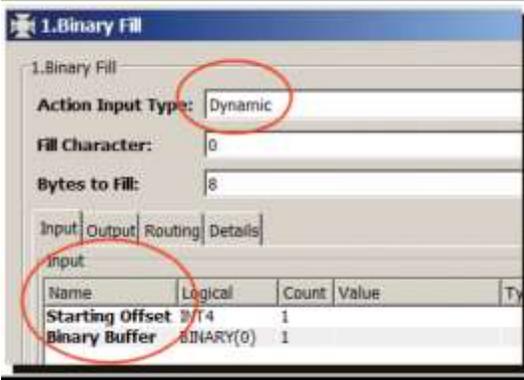


Parameter	Description
<b>Binary Buffer</b>	The source binary buffer is entered in the <b>Value</b> column. The result of the <b>Binary Fill</b> action is copied from the internal replica of the source binary buffer to the destination binary buffer. If you want to fill a source binary buffer directly with the fill character value

specified, then the **Output tab Binary Buffer** parameter should be set to the same variable as the **Input tab Binary Buffer** parameter.

## Input tab (Dynamic)

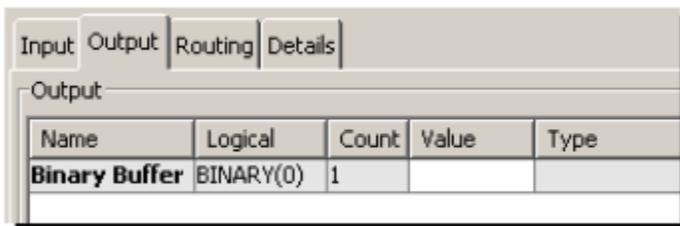
If you select **Dynamic** as the value for **Action Input Type**, the **Input** tab appears as follows:



Notice the **Input** tab provides an extra parameter:

Parameter	Description
<b>Binary Buffer</b>	The source binary buffer is entered in the <b>Value</b> column. The result of the <b>Binary Fill</b> action is copied from the internal replica of the source binary buffer to the destination binary buffer. If you want to fill a source binary buffer directly with the fill character value specified, then the <b>Output tab Binary Buffer</b> parameter should be set to the same variable as the <b>Input tab Binary Buffer</b> parameter.
<b>Starting Offset</b>	The first byte of the variable to fill with the <b>Fill Character</b> . The offset is calculated by the number of bytes counted from the beginning of the variable.

## Output tab



Parameter	Description
<b>Binary Buffer</b>	Specifies the destination binary buffer that stores the result of the <b>Binary Fill</b> action.

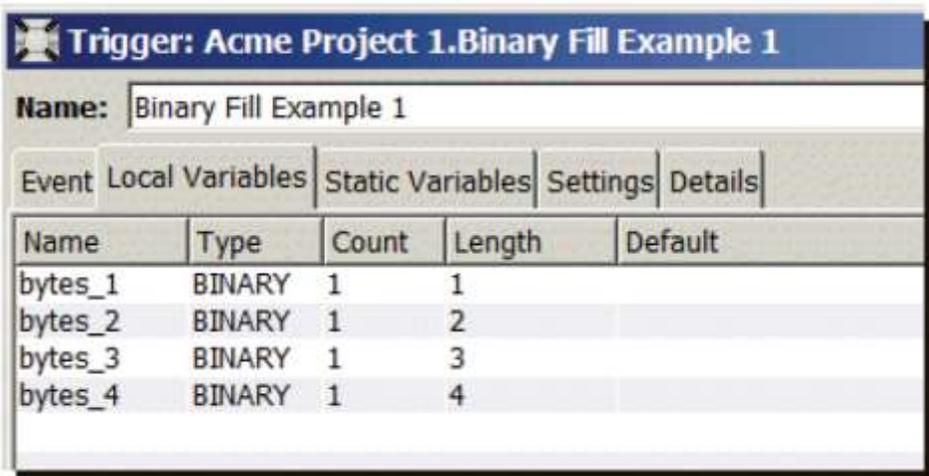
The result of the **Binary Fill** action is copied from the internal replica of the source binary buffer to the destination binary buffer.  
 If you want to fill a source binary buffer directly with the fill character value specified, then the **Output tab Binary Buffer** parameter should be set to the same variable as the **Input tab Binary Buffer** parameter.

## Binary Fill example 1

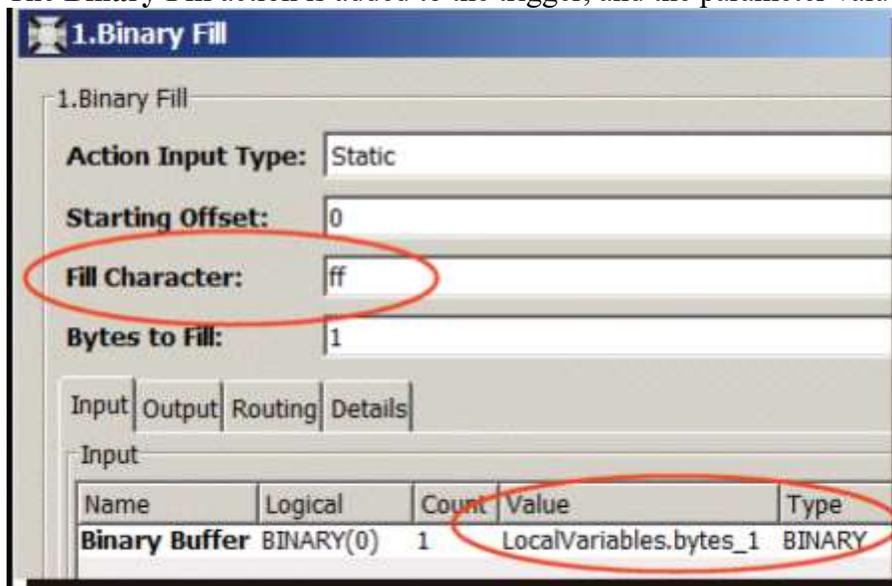
This example describes a trigger that uses the **Binary Fill** action to move a 1-byte value of 0xFF into the binary variable LocalVariables.bytes\_1.

The process follows:

1. A new trigger is named, and an event specified.

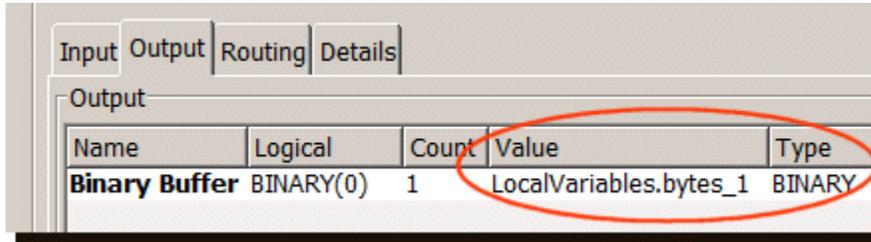


2. Local variables are available for the trigger to use.
3. The **Binary Fill** action is added to the trigger, and the parameter values filled in.



Notice that the value for the **Fill Character** parameter is specified as 2 hexadecimal digits.

4. From the **Input** tab, under **Value**, the local variable **bytes\_1** is assigned to the action.



Notice the **Output** tab uses the same local variable **bytes\_1**.

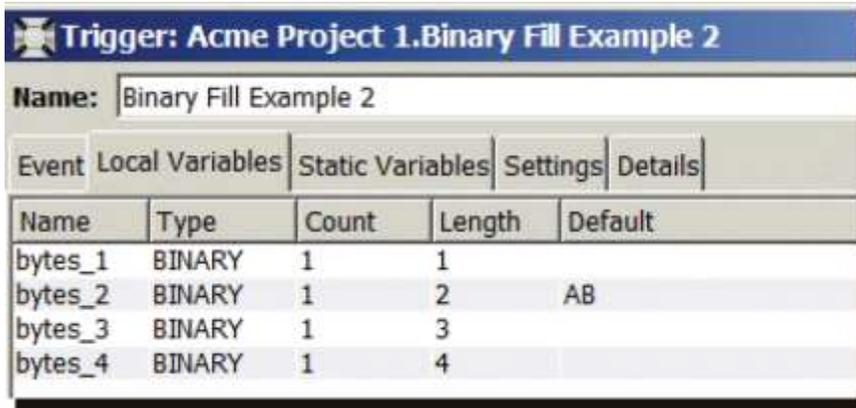
## Binary Fill example 2

This example provides a trigger that uses the **Binary Fill** action to move a 1-byte value of 0x0C into the binary variable **LocalVariables.bytes\_2**, offset 1. Before the move, the value of **bytes\_2** is 0x4142 which is ASCII **AB**.

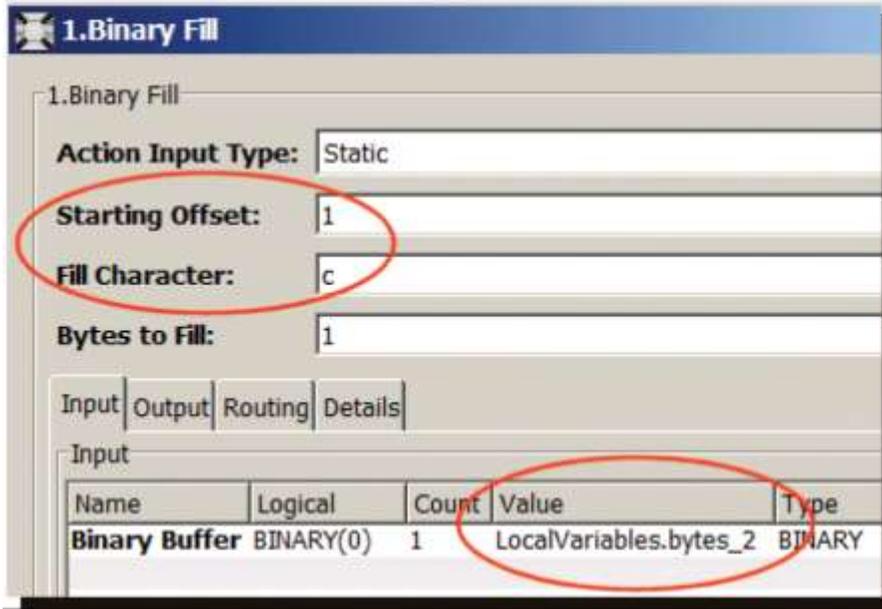
After the move, the value of **bytes\_2** is 0x410C, since x0C was moved into the position at offset 1.

The process follows:

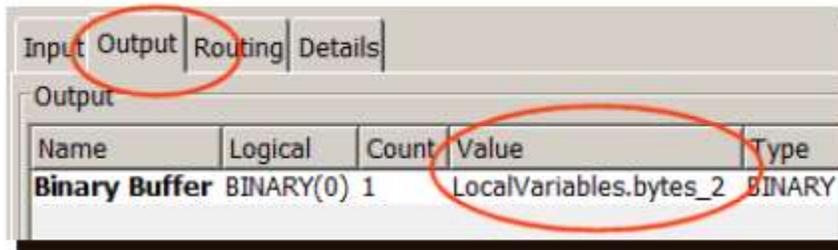
1. A new trigger is named, and an event specified.



2. Local variables are available for the trigger to use.
3. The **Binary Fill** action is added to the trigger, and the parameter values filled in.



4. From the **Input** tab, under **Value**, the local variable **bytes\_2** was assigned to the action.



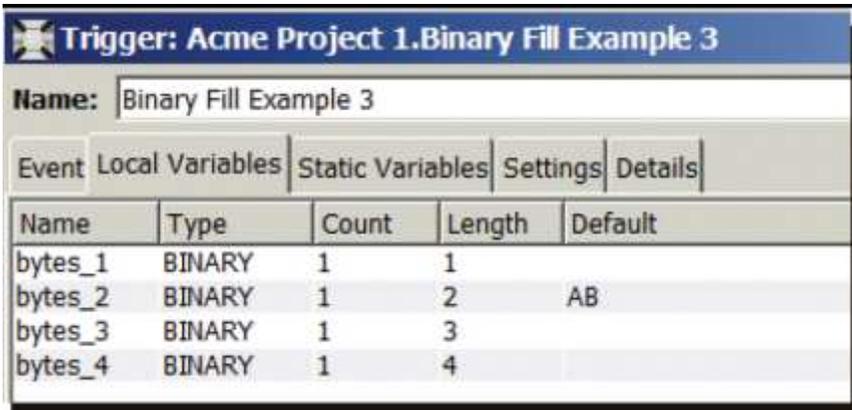
The input variable and output variable are assigned the same local variable **bytes\_2**.

### Binary Fill example 3

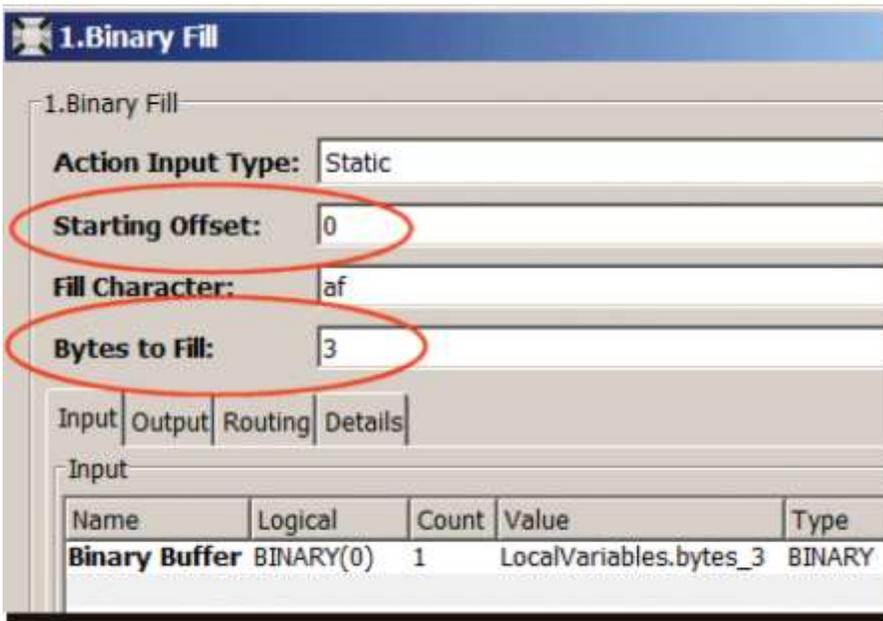
This example provides a trigger that uses the **Binary Fill** action to set the binary variable **LocalVariables.bytes\_3** to 0xAFAFAF.

The process follows:

1. A new trigger is named, and an event specified.



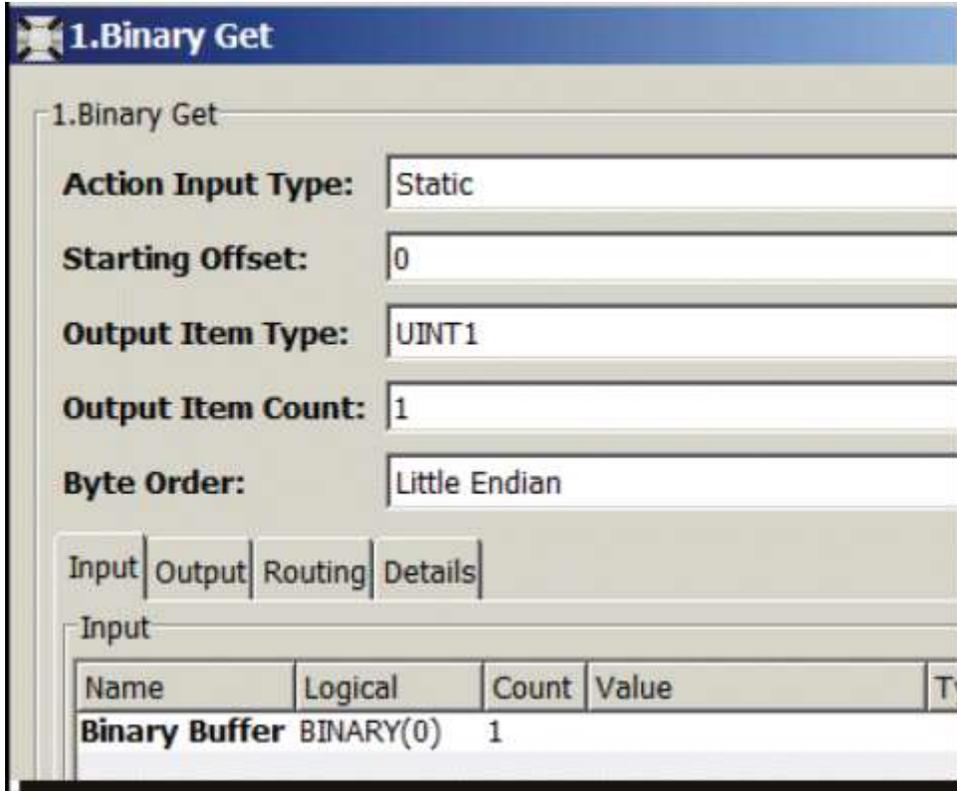
- Local variables are available for the trigger to use.
- The **Binary Fill** action is added to the trigger, and the parameter values filled in.



Notice that **Bytes to Fill** is set to 3, and the starting offset is 0. This sets all three bytes of the variable to the same value.

## IIoTA industrial IoT Platform: Binary Get

The **Binary Get** action copies a value from a BINARY data type variable into another variable of any data type.



## Parameter descriptions

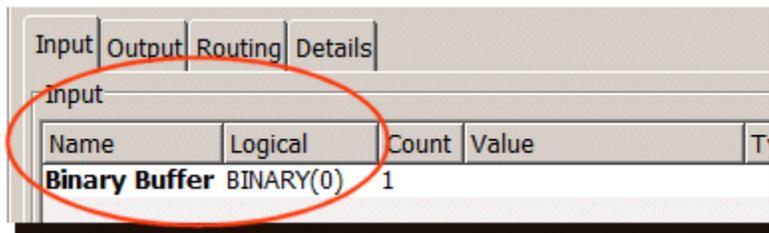
The action provides these parameters:

Parameter	Description
<b>Action Input Type</b>	Can be:  <b>Static</b> - The specified <b>Starting Offset</b> cannot be changed unless you edit the trigger. <b>Dynamic</b> - The <b>Starting Offset</b> parameter becomes available from the <b>Input</b> tab.
<b>Starting Offset</b>	The first byte of the data that the <b>get</b> begins with. The offset is calculated by the number of bytes counted from the beginning of the variable. If you select <b>Dynamic</b> as the <b>Action Input Type</b> , <b>Starting Offset</b> becomes available from the <b>Input</b> tab where you can set a variable whose value can vary at runtime.
<b>Output Item Type</b>	The data type of the output variable. The list of the available data types for the node. For example, BYTE, WORD, DWORD, or INT1, INT2, INT4 and so forth.

<p><b>Output Item Count</b></p>	<p>A value that specifies a single integer or an array. For an array, the value specifies the number of elements in the array.  <b>Output Item Count</b> defaults to one, a single value.                  When you change the value of <b>Output Item Count</b>, the value under the <b>Count</b> column on the <b>Value</b> row of the <b>Output</b> tab also changes.</p>
<p><b>Byte Order</b></p>	<p>Can be:</p> <p><b>Little Endian</b> - Stores the low-order byte of the value into the destination variable at the highest address.  <b>Big Endian</b> - Stores the high-order byte of the value into the destination variable at the lowest address.</p>

### Input tab (Static)

If you select **Static** as the action input type, the **Input** tab appears as follows:

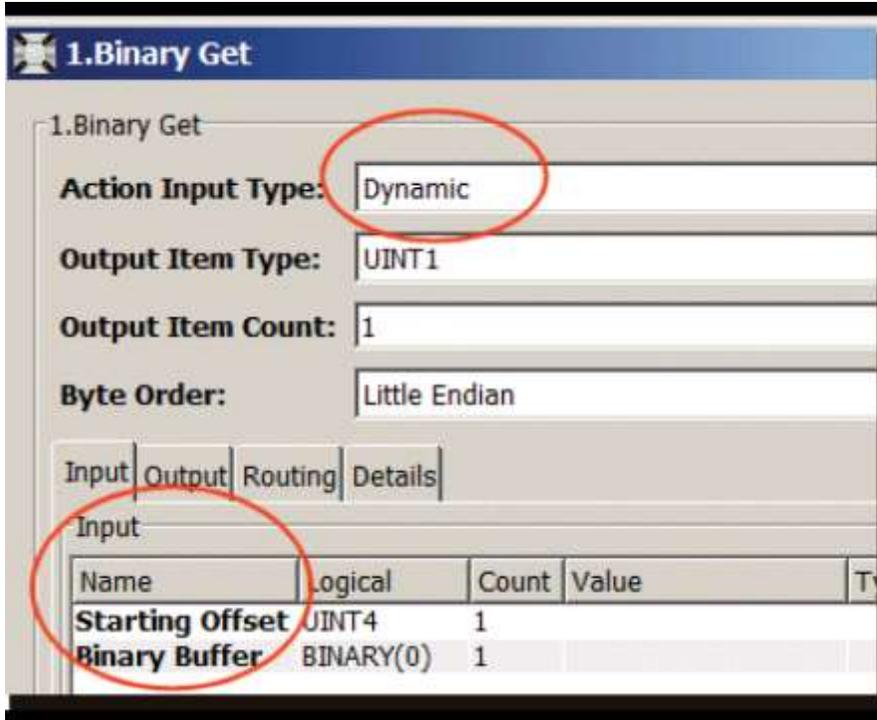


The following describes the **Input** tab for **Binary Get** when **Static** is selected for the **Action Input Type**:

Parameter	Description
<b>Binary Buffer</b>	Specifies the source BINARY buffer.

### Input tab (Dynamic)

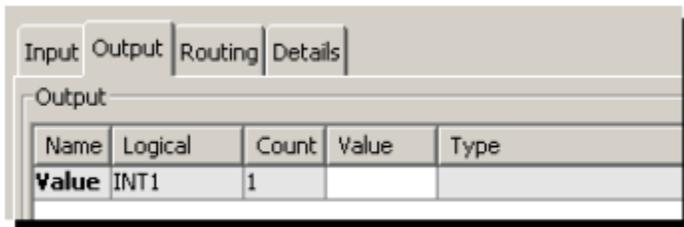
If you select **Dynamic** as the value for **Action Input Type**, the **Input** tab appears as follows:



Notice that **Starting Offset** appears on the **Input** tab where you can set a variable whose value can vary at runtime.

## Output tab

The **Output** tab is used to specify the destination of the result variable for the action. The **Output** tab remains the same regardless of whether you select **Static** or **Dynamic** for the **Action Input Type**.



Parameter	Description
<b>Value</b>	Specifies the destination of the Binary Get. The results of taking the source Binary buffer, starting at the specified offset and getting values for the specified item count.

## Binary Fill and Binary Get example 4

This example describes a trigger that incorporates actions that insert a Carriage Return Line Feed sequence x0D0A into the middle of a string.

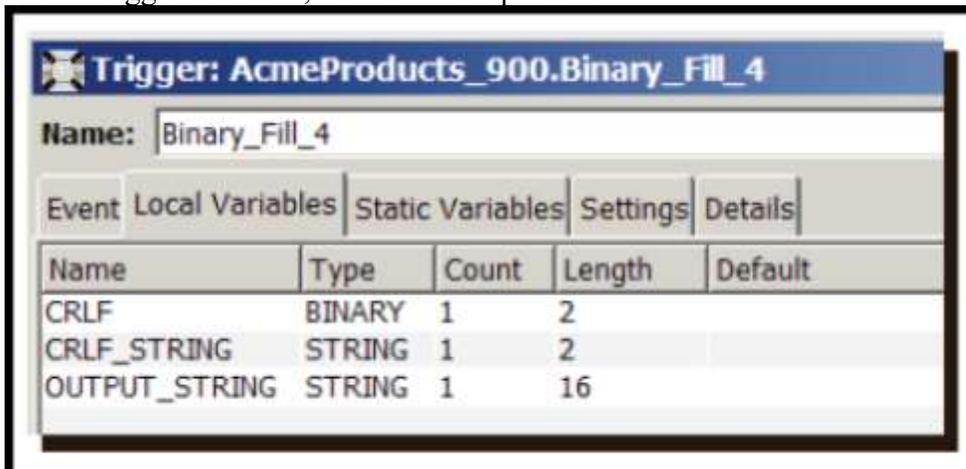
The example trigger contains these actions:

- **Binary Fill** (action 1) puts 0x0D into the first byte of the local variable CRLF.
- **Binary Fill** (action 2) puts 0x0A into the second byte of the local variable CRLF.
- **Binary Get** (action 3) moves the binary variable CRLF to the string variable CRLF\_STRING.
- **String Builder** (action 4) inserts the variable CRLF\_STRING into the middle of the variable OUTPUT\_STRING.

The process follows:

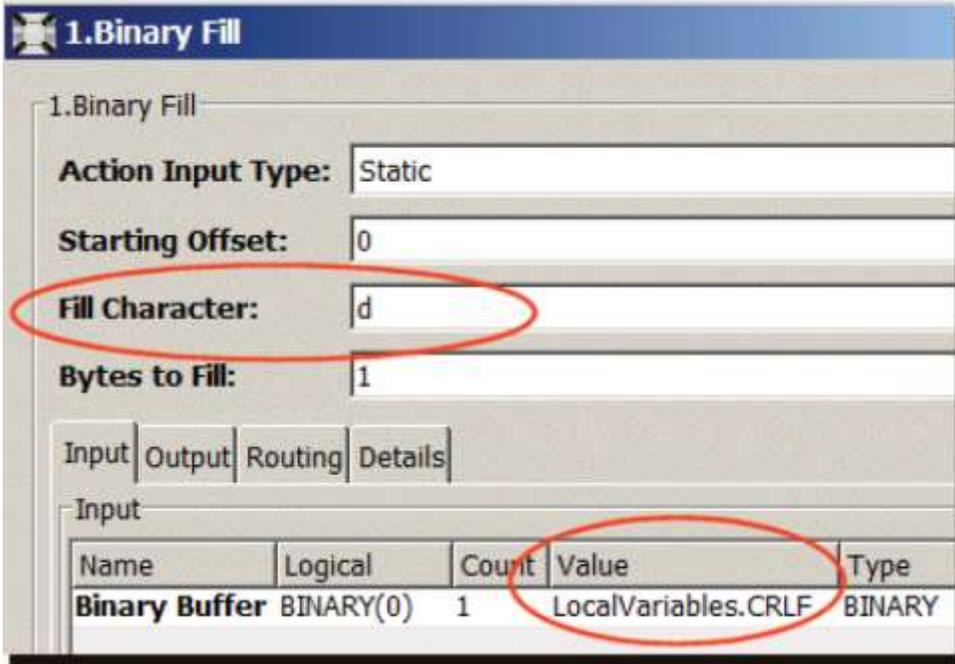
Binary Fill action 1

1. A new trigger is named, and an event specified.



Local variables are available for the trigger to use.

2. Using the Canvas Editor, the first **Binary Fill** action is added to the trigger and the parameter values filled in.

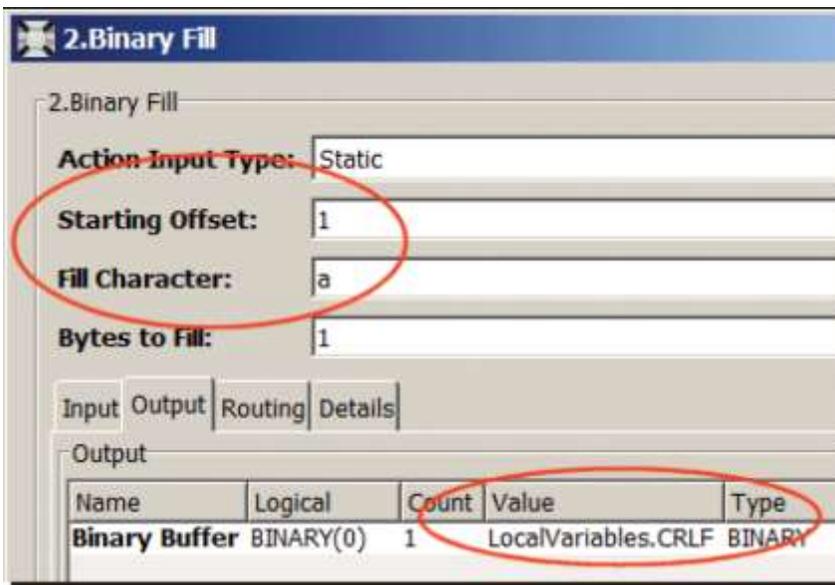


This **Binary Fill** action will put 0x0D into the first byte of the local variable **CRLF**.

3. The input and output variables are assigned the same local variable **CRLF**.

Binary Fill action 2

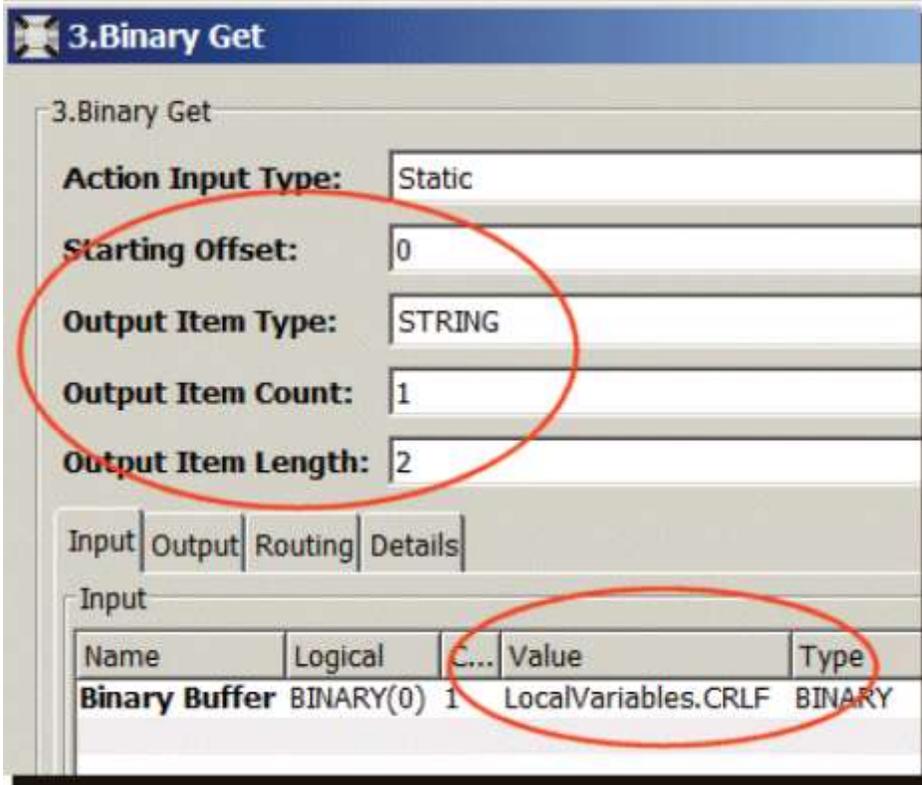
The second **Binary Fill** action is added to the trigger and the parameter values filled in.



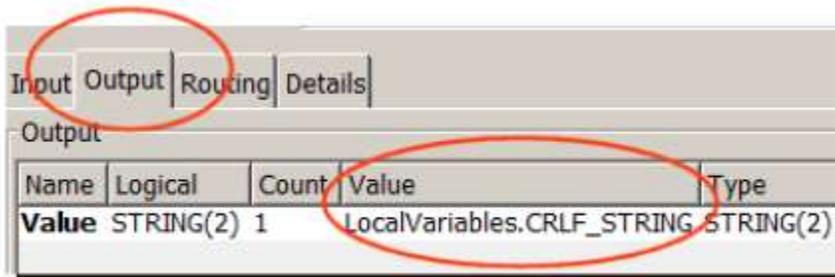
This action will put 0x0A into the second byte of the local variable **CRLF**.  
The input and output variables are assigned the same local variable **CRLF**.

Binary Get action 3

**Binary Get** is added as the third action to the trigger and its parameter values filled in.



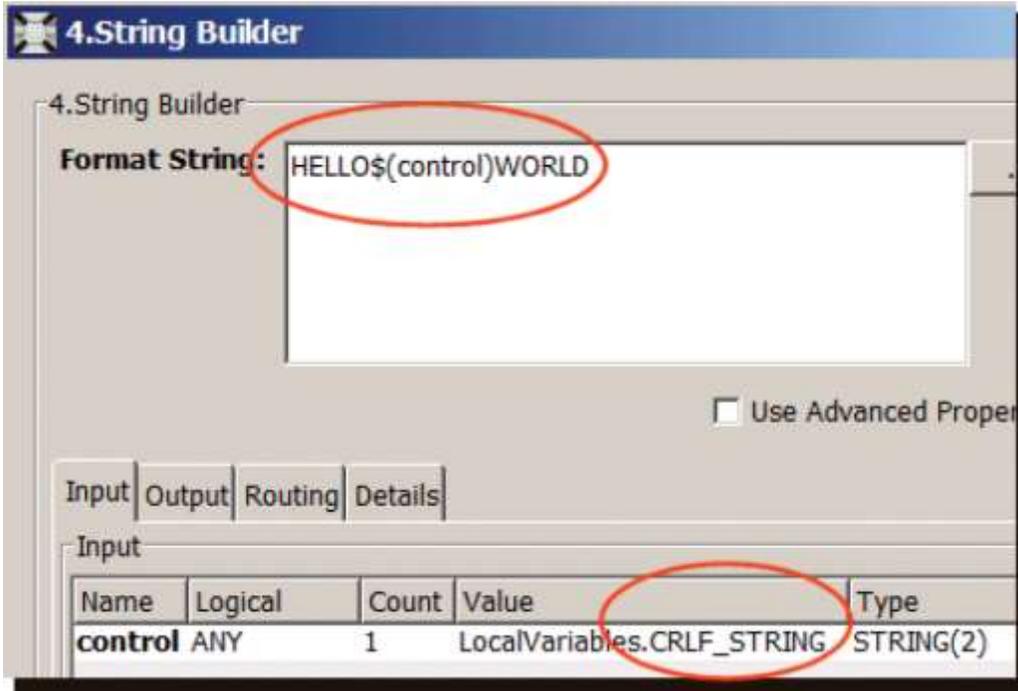
This action moves the binary variable **CRLF** to the string variable **CRLF\_STRING**, and the content of this string variable is now 0x0D0A



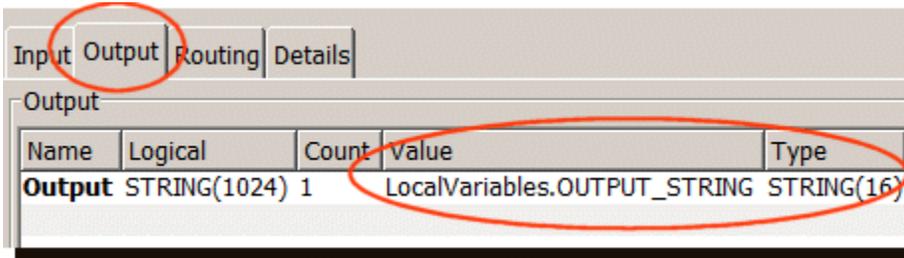
Notice that the output variable is assigned **CRLF\_STRING**.

String Builder action 4

The **String Builder** action is added to the trigger and its parameter values filled in.



When the trigger executes, **String Builder** will insert the variable **CRLF\_STRING** into the middle of the local variable **OUTPUT\_STRING**. In the format string, \$(control) is placed between HELLO and WORLD. This will be replaced by the variable **CRLF\_STRING**.



After the trigger is saved, started and executed, a report is generated.

```

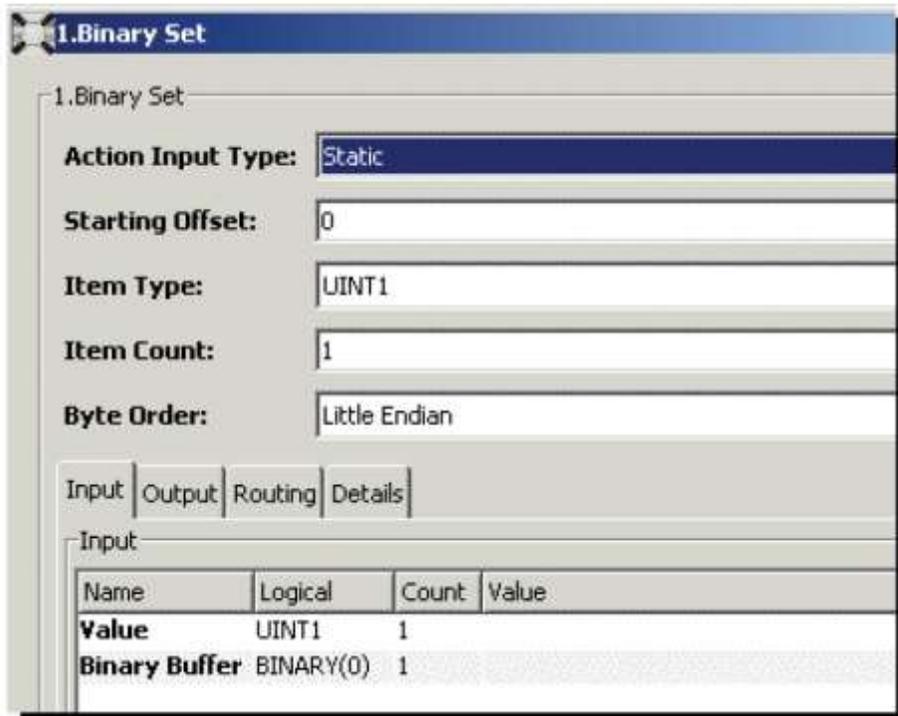
Binary Get
String Builder
  Seq: 4
  ID: 4
  Route: Success
  Ext. Status: 0
  Execution Time: 0 ms
  Input
  Local Variable Change
    OUTPUT_STRING
      Status: 0
      Data: HELLOWORLD
      0000: 0c 48 45 4c 4c 4e 0d 0a 57 4f 52 4c 44 .HELLO..
      Type: STRING
      Count: 1
    Output
      Status: 0
      Output
        Data: HELLOWORLD
        0000: 0c 48 45 4c 4c 4e 0d 0a 57 4f 52 4c 44 .HELLO..
        Type: STRING
        Count: 1

```

Notice in the trigger report, the value of the local variable **OUTPUT\_STRING** has the value 0x0D0A embedded between the words HELLO and WORLD.

## IIoTA industrial IoT Platform: Binary Set

The **Binary Set** action copies the value of a non-binary data type variable to a BINARY data type variable.



## Parameter descriptions

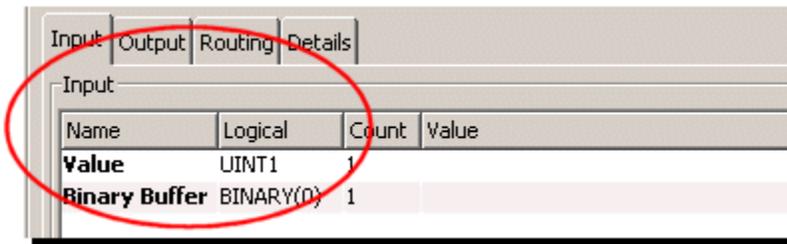
The action provides these parameters:

Parameter	Description
<b>Action Input Type</b>	<p>Can be:</p> <p><b>Static</b> - The default. Using <b>Static</b>, the specified <b>Starting Offset</b> cannot be changed unless you edit the trigger.</p> <p><b>Dynamic</b> - When you select <b>Dynamic</b>, the <b>Starting Offset</b> becomes available from the Input tab.</p>
<b>Starting Offset</b>	<p>The first byte of the data where the set begins. The offset is calculated by counting the number of bytes from the beginning of the source binary buffer. The source binary buffer is first copied into an internal replica. Then, starting at the specified offset, the value is copied into the internal replica according to the <b>Item Type</b>, <b>Item Count</b>, and <b>Byte Order</b> parameters. The original source binary buffer is not changed as part of this internal work.</p> <p>If you select <b>Dynamic</b> as the action input type, <b>Starting Offset</b> appears on the <b>Input</b> tab where you can set a variable whose value can vary at runtime.</p>
<b>Item Type</b>	<p>The data type of the source variable. The list of the available data types for the node. For example, BYTE, WORD, DWORD, or INT1, INT2, INT4 and so forth.</p>

	When you change the value of <b>Item Type</b> , the value under the <b>Logical</b> column on the <b>Name</b> row of the <b>Input</b> tab also changes.
<b>Item Count</b>	A value that specifies a single value or an array. For an array, the value specifies the number of elements in the array. <b>Item Count</b> defaults to one, a single value. When you change the value of <b>Item Count</b> , the value under the <b>Count</b> column on the <b>Value</b> row of the <b>Input</b> tab also changes.
<b>Byte Order</b>	Can be:  <b>Little Endian</b> - Stores the low-order byte of the value into the destination variable at the highest address. <b>Big Endian</b> - Stores the high-order byte of the value into the destination variable at the lowest address.

## Input tab (Static)

If you select **Static** as the action input type, the **Input** tab appears as follows:

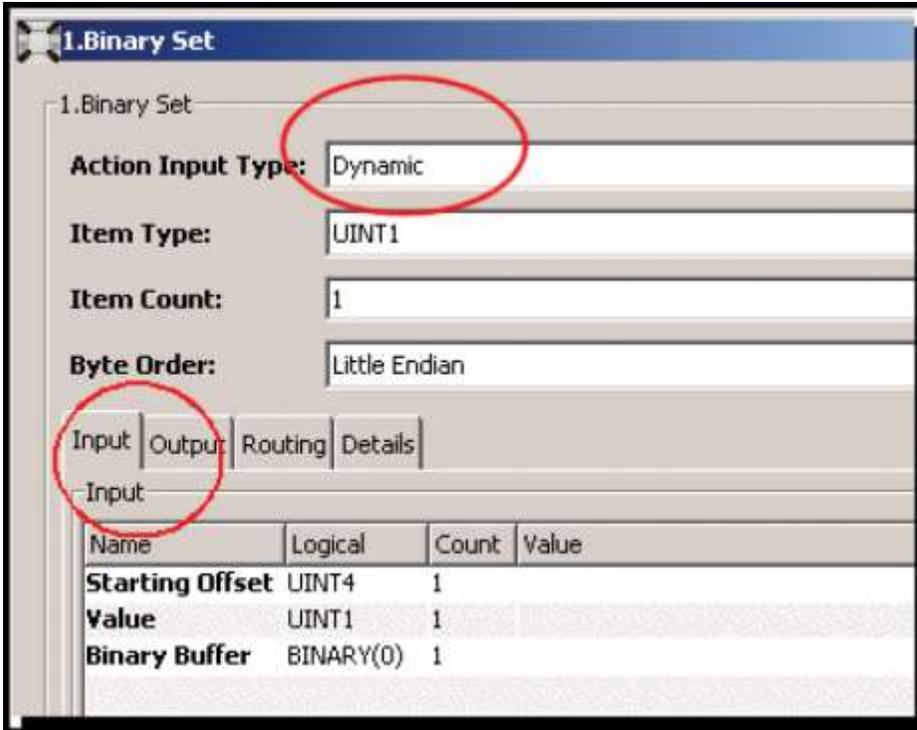


The **Input** tab is used to specify the source value of the initial binary buffer. The following describes the **Input** tab for **Binary Set**:

Parameter	Description
<b>Value</b>	Specifies the value you want to set into the source binary buffer.
<b>Binary Buffer</b>	Specifies the source binary buffer.

## Input tab (Dynamic)

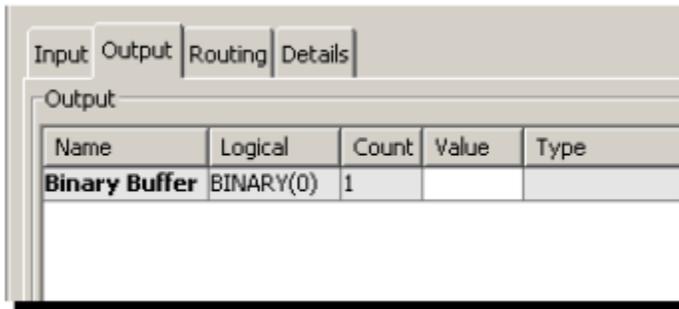
If you select **Dynamic** as the action input type, the **Input** tab appears as follows:



Notice that **Starting Offset** appears on the **Input** tab where you can set a variable whose value can vary at runtime.

## Output tab

The **Output** tab is used to specify the variable to store the result of the action.



The following describes the **Output** tab for **Binary Set**:

Parameter	Description
<b>Binary Buffer</b>	Specifies the destination binary buffer that stores the result of the Binary Set action. The result of the Binary Set action is copied from the internal replica of the source binary buffer to the destination binary buffer.

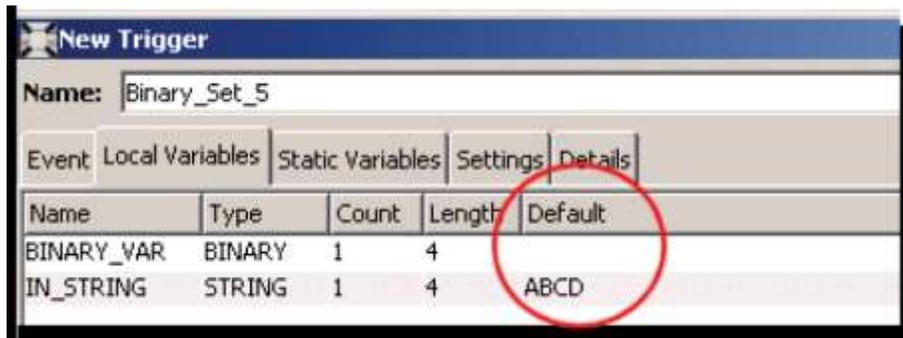
If you want to update a source binary buffer directly with the value specified, then the **Output tab Binary Buffer** parameter should be set to the same device variable as the **Input tab Binary Buffer** parameter.

## Binary Set example 5

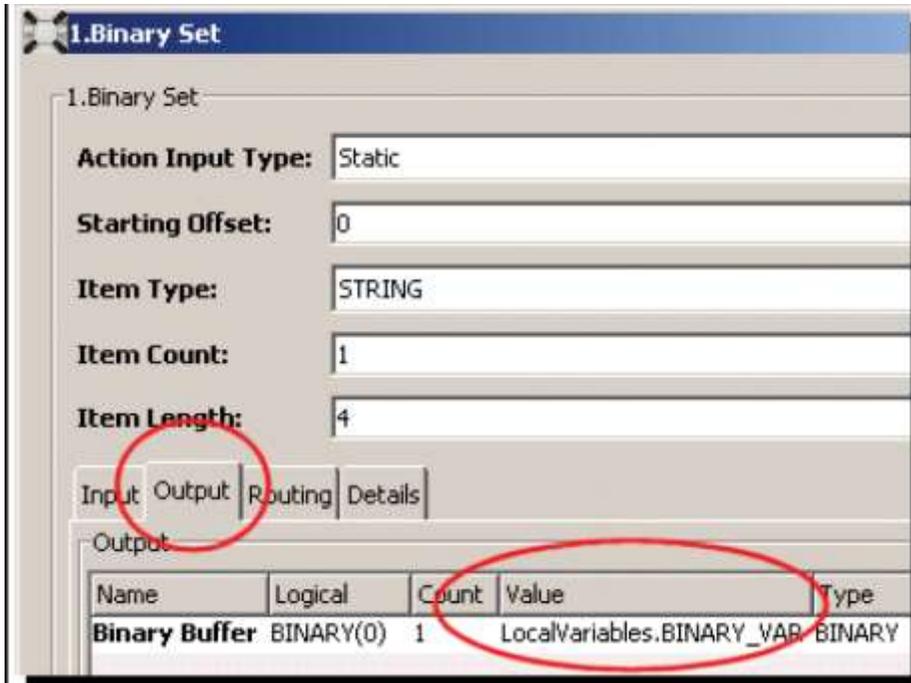
This example describes a trigger that uses the **Binary Set** action to move the 4 bytes in the string variable **IN\_STRING** to the binary variable **BINARY\_VAR**.

The process follows:

1. A new trigger is named and an event specified.

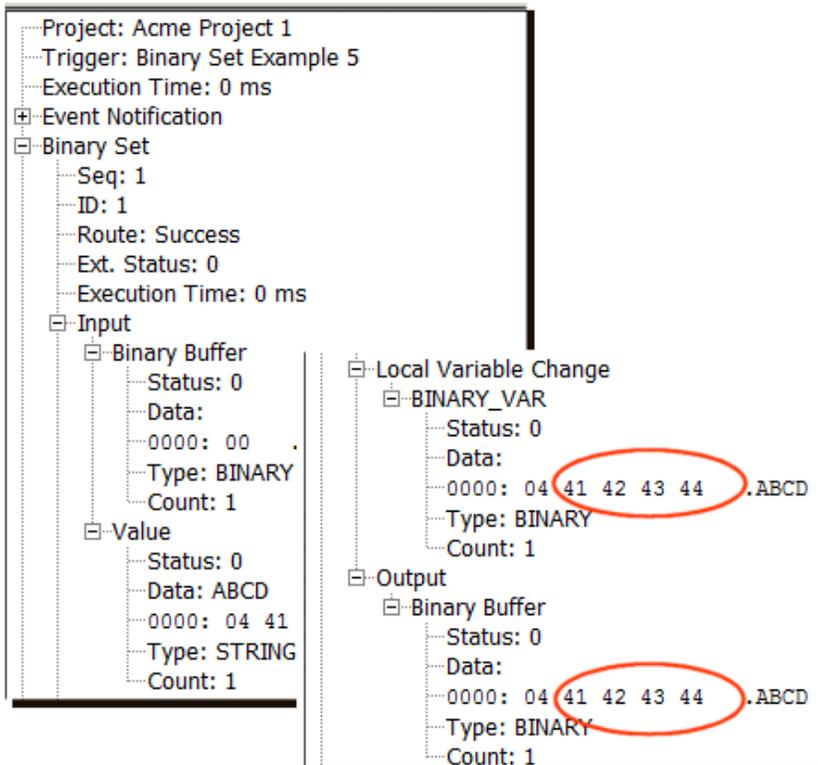


2. Local variables are available for the trigger to use. Notice the default value for **IN\_STRING** is ASCII **ABCD**.
3. Using the Canvas Editor, the **Binary Set** action is added to the trigger, and its parameter values filled in.
4. From the **Input** tab, under **Value**, the local variable **IN\_STRING** and **BINARY\_VAR** are assigned to the action.
5. From the **Input** tab, under **Value**, the local variable **BINARY\_VAR** is assigned to the action.



The destination for the move is **BINARY\_VAR**.

- When the trigger is saved and started, a report is generated. After trigger execution, **BINARY\_VAR** contains 0x41424344, as is shown in the trigger report



# IIoTA industrial IoT Platform: Decode Binary Buffer

The **Decode Binary Buffer** action uses a single input binary buffer to retrieve data values and then writes the values to multiple output variables. The source binary buffer variables, data types, offsets in the input buffer, counts, intervals, lengths and special handling options can be specified.

4.Decode Binary Buffer

Rules Input Type:

Rules In JSON:

```

1 {
2   "First" : {"offset":8, "type":"FLOAT4", "count":1 },
3   "Second" : {"offset": 12, "type":"STRING", "count":2, "length":4 },
4   "Third" : {"offset":30, "type":"INT4", "count":4, "interval":6 }
5 }

```

Byte Order:

Input Output Routing Details

Name	Logical	Count	Value	Type
Binary Buffer	BINARY	1	LocalVariables.LocalBinary	BINARY
Offset	UINT4	1	0	CONSTANT

## Parameter descriptions

The action provides these parameters:

Parameter	Description
<b>Rules Input Type</b>	<p>The options are:</p> <p><b>Manual</b> - To indicate the variables to decode from a binary buffer will be defined in the <b>Rules in JSON</b> parameter.</p> <p><b>Staging File</b> - To indicate the variables to decode from a binary buffer will be defined in the <b>File</b> parameter.</p>
<b>Rules in JSON</b>	<p>When <b>Rules Input Type</b> is <b>Manual</b>, this parameter is used to specify the JSON description of the variables to decode from a binary buffer. Each variable is specified as shown in the example with:</p>

Map variable name - The name of the map variable, which will be added as a row on the Output tab.

offset - The location where the data will start in the binary buffer. This is a required field.

type - The type of the source variable in the binary buffer.

The supported types are:

BOOL, INT1, INT2, INT4, INT8, UINT1, UINT2, UINT4, UINT8, FLOAT4, FLOAT8, STRING, and BINARY.

The special types listed below are also supported.

count - The number of variable elements. Specify a 1 for a single (scalar) element or a value greater than 1 for an array.

interval - The distance between the starting of each element of an array. This field is only used when an array is specified.

For example: if the offset is 30, the interval is 6, and the count is 4, the data will be read from the binary buffer at offsets 30, 36, 42, and 48 regardless of the data type.

If the interval is not specified, or specified as a zero, the distance will be the length of the element. Each element will start immediately after the end of the previous element.

length - The length of data. This is only specified if the type is STRING or BINARY.

swap - A boolean value to indicate whether the data byte/word order should be swapped before writing into the destination variable.

rockwell.logix.string - A 4 byte length value, followed by the string data.

rockwell.logix.INT4bool - A single or multi-byte integer that represents a stand-alone boolean tag or a group of stand-alone boolean tags. Individual boolean values are stored in the byte. The bit\_offset value indicates which bit in the integer is associated with the boolean. These items can be written individually as boolean data types.

rockwell.logix.packedbool - A boolean array tag, represented as a 4 byte integer. Each element in the array of booleans is represented as a bit within the 4-byte integer. The 4-byte integer must be kept to maintain integrity with the representation of the boolean array on the Rockwell PLC.

siemens.string - The 1st byte is the full string length, the 2nd byte is the current string length, followed by the string data.

string.end\_of\_buffer - A variable length string that reads data from the "offset" to the end of the binary buffer. The "type" must be "STRING" or "BINARY", "count" must be 1, and "length" must be 0.

Examples of using the custom data types:

```
{ "melco_timestamp_var" : {"offset":12, "type":"INT8", "count":1,
"special":"melco.timestamp" } }
```

```
{ "rockwell_string_var" : {"offset":12, "type":"STRING", "count":1, "length":50,
"special":"rockwell.logix.string" } }
```

	<pre>{ "bool1" : { "offset":0, "count":1, "type":"BOOL", "bit_offset":0, "special":"rockwell.logix.INT4bool" }} { "bool2" : { "offset":0, "count":1, "type":"BOOL", "bit_offset":1, "special":"rockwell.logix.INT4bool" }} { "boolArray" : { "offset":0, "count":1, "type":"INT4", "special":"rockwell.logix.packedbool" }} { "siemens_string_var" : { "offset":12, "type":"STRING", "count":1, "length":50, "special":"siemens.string" }} { "end_of_buffer_var" : { "offset": 12, "type":"STRING", "count":1, "length":0 , "special":"string.end_of_buffer" }}</pre> <p>The multi-line input icon  can be used to display a larger input area for the JSON variable description.</p>
<b>File</b>	<p>When <b>Rules Input Type</b> is <b>Staging File</b>, this parameter is used to specify a file in the Staging Browser area that contains the JSON description of the variables to decode from a binary buffer.</p> <p>This file is read when the trigger is edited. The description of the variables in the file is handled in the same manner as the <b>Rules in JSON</b> parameter. The file is only read when the trigger is edited, so changes to the file's content do not impact a started trigger's definition or function. To change a trigger's function, the trigger must be stopped, edited (the file is read) to make any necessary changes, and then restarted.</p>
<b>Byte Order</b>	<p>The options are:</p> <p><b>Little Endian</b> - The low-order byte/word of the value in the binary buffer is stored at the highest address.</p> <p><b>Big Endian</b> - The high-order byte/word of the value in the binary buffer is stored at the highest address.</p>

## Input tab

Parameter	Description
<b>Binary Buffer</b>	Specifies the source binary buffer.
Offset	An optional offset used in addition to each variable's offset value. This can be used, for example, when a variable length header in the source buffer needs to be accounted for.

## Output tab

4.Decode Binary Buffer

Rules Input Type: Manual

Rules In JSON:

```

1 {
2   "First" : {"offset":8, "type":"FLOAT4", "count":1 },
3   "Second" : {"offset": 12, "type":"STRING", "count":2, "length":4 },
4   "Third" : {"offset":30, "type":"INT4", "count":4, "interval":6 }
5 }
    
```

Byte Order: Little Endian

Input Output Routing Details

Output

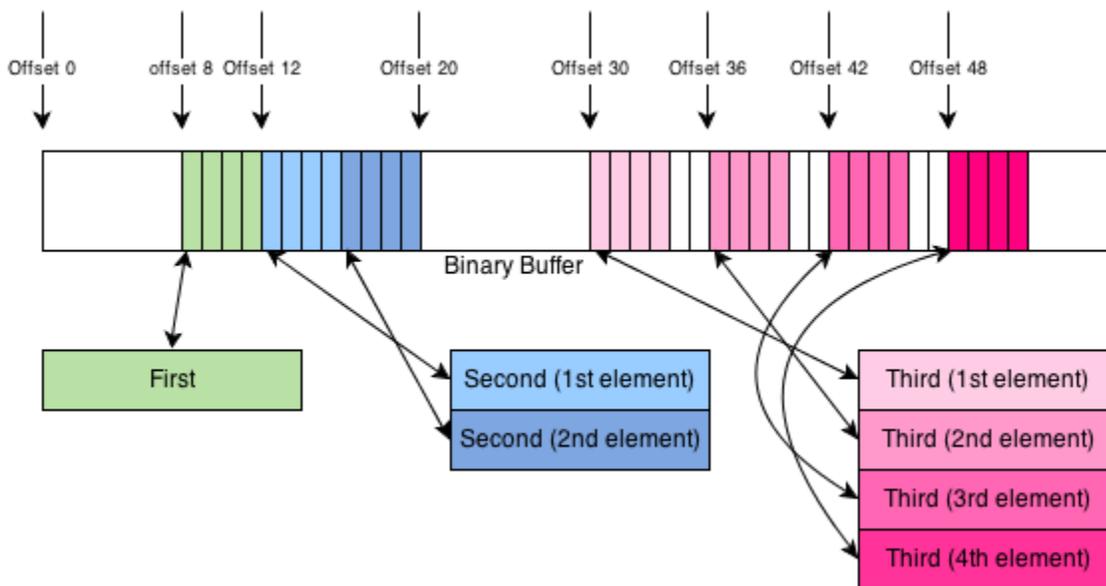
Name	Logical	Count	Value	Type
First	FLOAT4	1	Logix_1_67.REALarray1Da[0]	FLOAT4
Second	STRING(4)	2	Logix_1_67.STRarray1Da[0]	STRING(82)
Third	INT4	4	Logix_1_67.DINTarray1Da[0]	INT4

## Decode Binary Buffer example

This example shows the JSON description of the variables and the placement of the data as it is read from the source binary buffer.

```

{
  "First" : {"offset":8, "type":"FLOAT4", "count":1 },
  "Second" : {"offset": 12, "type":"STRING", "count":2, "length":4 },
  "Third" : {"offset":30, "type":"INT4", "count":4, "interval":6 }
}
    
```



# IIoTA industrial IoT Platform: Encode Binary Buffer

The **Encode Binary Buffer** action writes multiple variable values into a single output binary buffer. The source variables, data types, offsets into the destination buffer, counts, intervals, lengths and special handling options can be specified.

1.Encode Binary Buffer

Rules Input Type: Manual

Rules In JSON:

```
1 {
2   "First" : {"offset":8, "type":"FLOAT4", "count":1 },
3   "Second" : {"offset": 12, "type":"STRING", "count":2, "length":4 },
4   "Third" : {"offset":30, "type":"INT4", "count":4, "interval":6 }
5 }
```

Byte Order: Little Endian

Input Output Routing Details

Input

Name	Logical	Count	Value	Type
Binary Buffer	BINARY	1	LocalVariables.LocalBinary	BINARY
Offset	UINT4	1	0	CONSTANT
First	FLOAT4	1	Logix_1_67.REALarray1Da[0]	FLOAT4
Second	STRING(4)	2	Logix_1_67.STRarray1Da[0]	STRING(82)
Third	INT4	4	Logix_1_67.DINTarray1Da[0]	INT4

## Parameter descriptions

The action provides these parameters:

Parameter	Description
<b>Rules Input Type</b>	<p>The options are:</p> <p><b>Manual</b> - To indicate the variables to encode into a binary buffer will be defined in the <b>Rules in JSON</b> parameter.</p> <p><b>Staging File</b> - To indicate the variables to encode into a binary buffer will be defined in the <b>File</b> parameter.</p>
<b>Rules in JSON</b>	<p>When <b>Rules Input Type</b> is <b>Manual</b>, this parameter is used to specify the JSON description of the variables to encode into a binary buffer. Each variable is specified as shown in the example with:</p> <p>Map variable name - The name of the map variable, which will be added as a row on the Input tab.</p>

offset - The location where the data will start in the binary buffer. This is a required field.

type - The type of the source variable.

The supported types are:

BOOL, INT1, INT2, INT4, INT8, UINT1, UINT2, UINT4, UINT8, FLOAT4, FLOAT8, STRING, and BINARY.

The special types listed below are also supported.

count - The number of variable elements. Specify a 1 for a single (scalar) element or a value greater than 1 for an array.

interval - The distance between the starting of each element of an array. This field is only used when an array is specified.

For example: if the offset is 30, the interval is 6, and the count is 4, the data will be written into the binary buffer at offsets 30, 36, 42, and 48 regardless of the data type.

If the interval is not specified or specified as a zero, the distance will be the length of the element. Each element will start immediately after the end of the previous element.

length - The length of data. This is only specifying this if the type is STRING or BINARY.

swap - A boolean value to indicate whether the data byte/word order should be swapped before writing into the binary buffer.

special - Custom handling of the data types:

rockwell.logix.string - A 4 byte length value, followed by the string data.

rockwell.logix.INT4bool - A single or multi-byte integer that represents a stand-alone boolean tag or a group of stand-alone boolean tags. Individual boolean values are stored in the byte. The bit\_offset value indicates which bit in the integer is associated with the boolean. These items can be written individually as boolean data types.

rockwell.logix.packedbool - A boolean array tag, represented as a 4 byte integer. Each element in the array of booleans is represented as a bit within the 4-byte integer. The 4-byte integer must be kept to maintain integrity with the representation of the boolean array on the Rockwell PLC.

siemens.string - The 1st byte is the full string length, the 2nd byte is the current string length, followed by the string data.

Examples of using the three custom data types:

```
{ "melco_timestamp_var" : { "offset":12, "type":"INT8", "count":1,
"special":"melco.timestamp" } }
{ "rockwell_string_var" : { "offset":12, "type":"STRING", "count":1, "length":50,
"special":"rockwell.logix.string" } }
{ "bool1" : { "offset":0, "count":1, "type":"BOOL", "bit_offset":0,
"special":"rockwell.logix.INT4bool" } }
{ "bool2" : { "offset":0, "count":1, "type":"BOOL", "bit_offset":1,
"special":"rockwell.logix.INT4bool" } }
```

	<pre>{ "boolArray" : {"offset":0, "count":1, "type":"INT4", "special":"rockwell.logix.packedbool" }} { "siemens_string_var" : {"offset":12, "type":"STRING", "count":1, "length":50, "special":"siemens.string" } }</pre> <p>The multi-line input icon  can be used to display a larger input area for the JSON variable description.</p>
<b>File</b>	<p>When <b>Rules Input Type</b> is <b>Staging File</b>, this parameter is used to specify a file in the Staging Browser area that contains the JSON description of the variables to encode into a binary buffer.</p> <p>This file is read when the trigger is edited. The description of the variables in the file is handled in the same manner as the <b>Rules in JSON</b> parameter. The file is only read when the trigger is edited, so changes to the file's content do not impact a started trigger's definition or function. To change a trigger's function, the trigger must be stopped, edited (the file is read) to make any necessary changes, and then restarted.</p>
<b>Byte Order</b>	<p>The options are:</p> <p><b>Little Endian</b> - Stores the low-order byte/word of the value into the destination binary buffer variable at the highest address.</p> <p><b>Big Endian</b> - Stores the high-order byte/word of the value into the destination binary buffer variable at the highest address.</p>

## Input tab

Parameter	Description
<b>Binary Buffer</b>	Specifies the source binary buffer. This source binary buffer is copied to an internal buffer before the encoding (writing) of the individual source variables.
Offset	An optional offset used in addition to each variable's offset value. This can be used, for example, when a variable length header in the destination buffer needs to be accounted for.
Input map variables	<p>The map variables identified in the JSON variable description. Each variable in the JSON variable description will be added as a map variable row in the Input tab.</p> <p>In the <b>Value</b> column, specify the source variable. The variable can be a constant, trigger macro, event variable, trigger variable or device variable.</p>

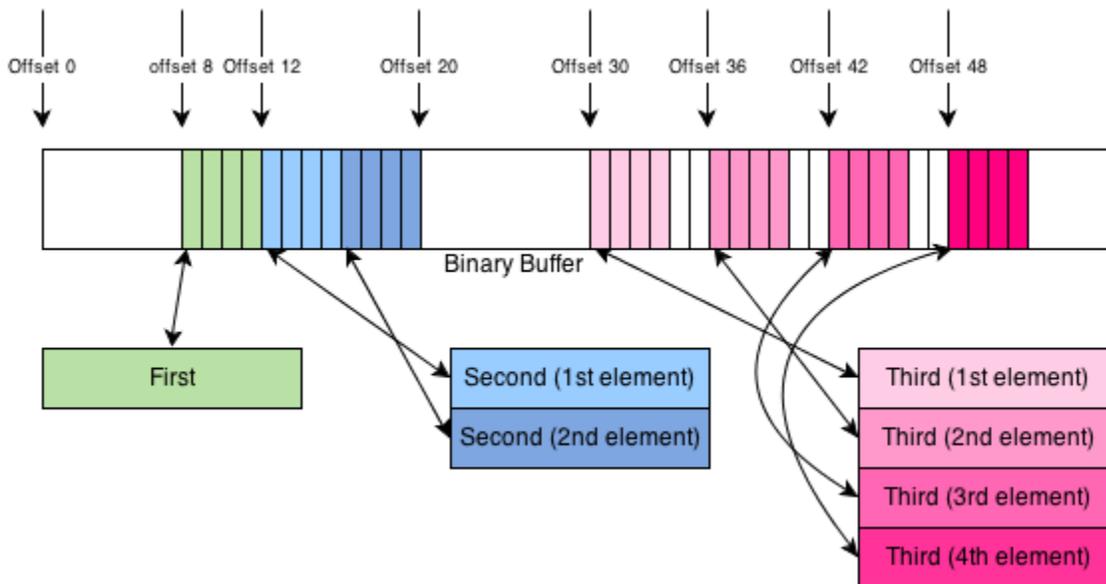
## Output tab

Parameter	Description
<b>Binary Buffer</b>	Specifies the destination binary buffer that stores the result of the Encode Binary Buffer action. The result of the Encode Binary Buffer action is copied from the internal replica of the source binary buffer to the destination binary buffer. If you want to update a source binary buffer directly with the variable values specified, then the Output tab <b>Binary Buffer</b> parameter should be set to the same variable as the Input tab <b>Binary Buffer</b> parameter.

## Encode Binary Buffer example

This example shows the JSON description of the variables and the placement of the data as it is written into the output binary buffer.

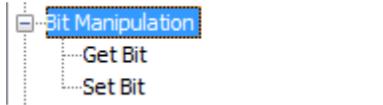
```
{
  "First" : {"offset":8, "type":"FLOAT4", "count":1 },
  "Second" : {"offset": 12, "type":"STRING", "count":2, "length":4 },
  "Third" : {"offset":30, "type":"INT4", "count":4, "interval":6 }
}
```



## IIoTA industrial IoT Platform: Bit Manipulation

The **Bit Manipulation** category provides actions that perform get and set functions on variables. The bit manipulation actions algorithmically manipulate bits and other data shorter than a byte.

Tasks that require bit manipulation include low-level device control and error detection and correction algorithms.

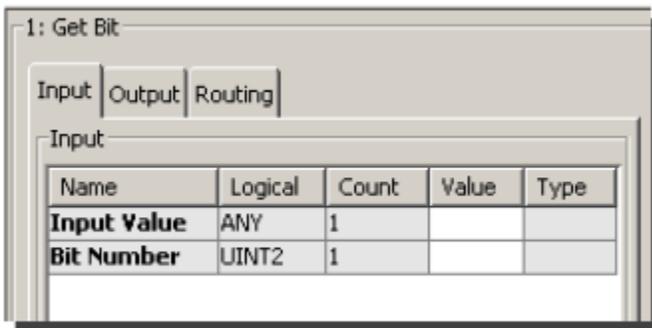


The **Bit Manipulation** category provides these actions:

- Get Bit
- Set Bit
- Bit numbering when using strings

## IIoTA industrial IoT Platform: Get Bit

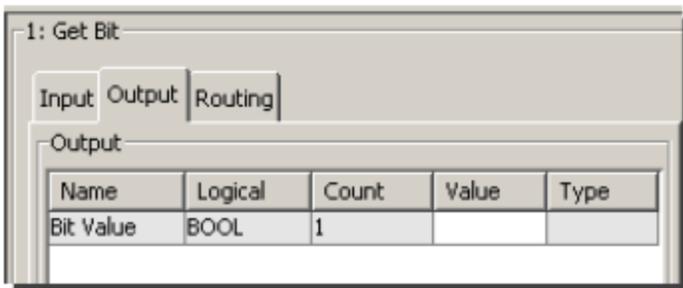
The **Get Bit** action copies the value of a bit from a source variable.



### Input tab

Parameter	Description
<b>Input Value</b>	The variable from which to get a bit value. The input variable can be of any data type. However, a STRING variable should not be used as the length of the string is embedded in the first 4 bytes.
<b>Bit Number</b>	The position of the bit in the input variable, where 0 is the low-order bit position.

## Output tab



Parameter	Description
<b>Bit Value</b>	Specifies the destination variable for the <b>Bit Get</b> action. The destination variable can be any type, but the value returned by the <b>Get Bit</b> action will always be a 0 or a 1.

## Routing tab

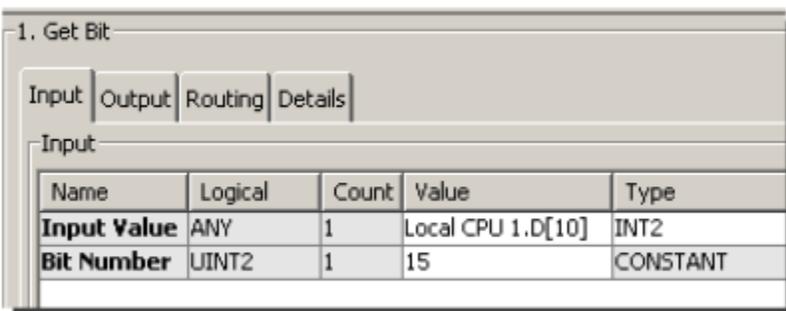
The Routing tab has **On Result** options different from the standard Success and Failure routes:

Parameter	Description
<b>True</b>	The route to take when the returned bit value is 1.
<b>False</b>	The route to take when the returned bit value is 0.
<b>Failure</b>	The route to take if the action fails.

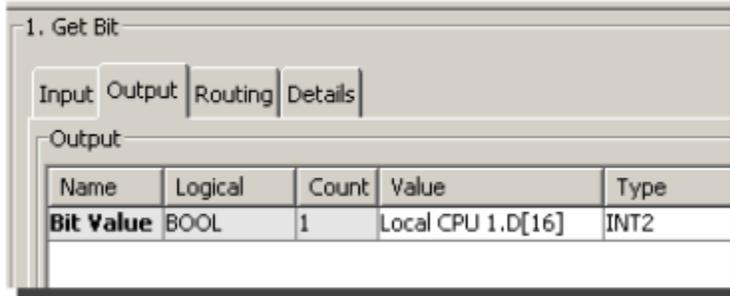
## Example Get Bit

You can use the **Get Bit** action to retrieve the value of a bit in an integer variable (such as an INT2). Bit 0 always represents the least significant bit, and bit 15 represents the most significant bit.

The example **Input** tab shows how to get the value of bit 15 from D[10].

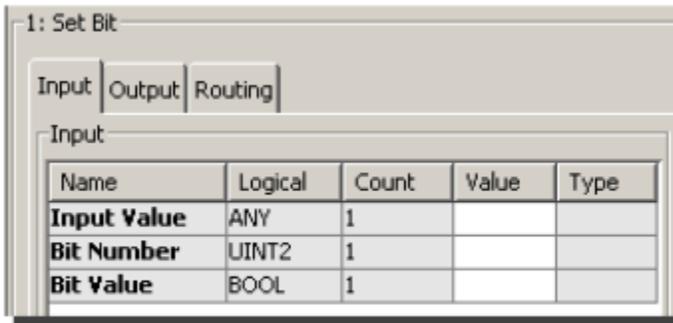


The **Output** tab is used to store the result in D[16]



## IIoTA industrial IoT Platform: Set Bit

The **Set Bit** action sets the value of a bit in a variable.



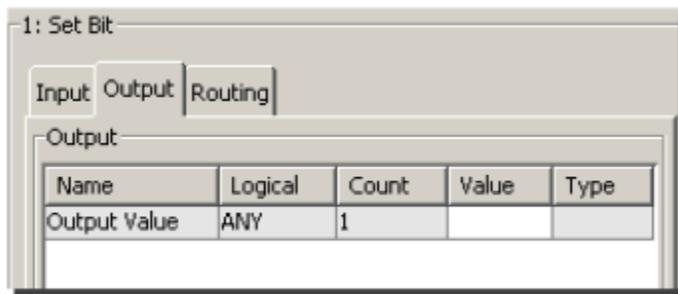
### Input tab

Parameter	Description
<b>Input Value</b>	<p>The name of the source variable whose bit you want to set. The input variable can be of any data type. However, a STRING variable should not be used as the length of the string is embedded in the first 4 bytes. The result of the <b>Set Bit</b> action is copied from the internal replica of the source variable to the destination array.</p> <p>If you want to update a source variable directly with the bit value specified, then the <b>Output tab Output Value</b> parameter should be set to the same device variable as the <b>Input tab Input Value</b> parameter.</p>
<b>Bit Number</b>	The position of the bit in the input variable, where 0 is the low-order bit position.

<b>Bit Value</b>	<p>The value to set into the bit at position <b>Bit Number</b>. This can be a any type of variable.</p> <p>For a Constant:</p> <p>True or 1 = Set the bit to 1 (on) . False or 0 = Set the bit to 0 (off).</p>
------------------	--

## Output tab

The **Output** tab provides the means to set one bit in an integer. You can only set one bit per **Set Bit** action.



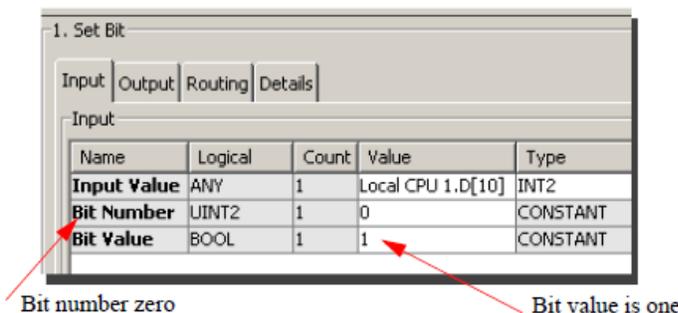
The **Output** tab provides three rows to set values for **Set Bit**.

Parameter	Description
<b>Output Value</b>	<p>Specifies the destination variable to place the result of the <b>Set Bit</b> action. The result of the <b>Set Bit</b> action is copied from the internal replica of the source variable to the destination array.</p> <p>If you want to update a source variable directly with the bit value specified, then the <b>Output tab Output Value</b> parameter should be set to the same device variable as the <b>Input tab Input Value</b> parameter.</p>

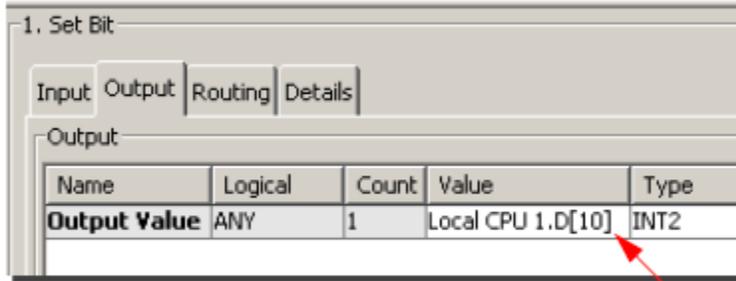
## Example Set Bit

You can use the **Set Bit** action in a trigger to set a bit in an integer variable (such as INT2). Since an INT2 (or WORD) represents a 2-byte data field, bit 0 always represents the least significant bit, and bit 15 represents the most significant bit.

The example **Input** tab shows how to set bit number zero in D[10] to the value one.



The **Output** tab is used to store the result of this action.

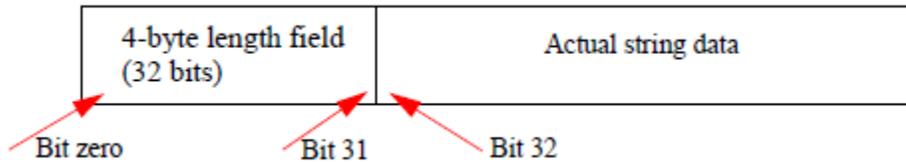


The result of this action is stored back in D[10].

Note that the other bits in D[10] are not affected by this operation. They retain their values from before the action. In other words, bits 1 through 15 remain unchanged.

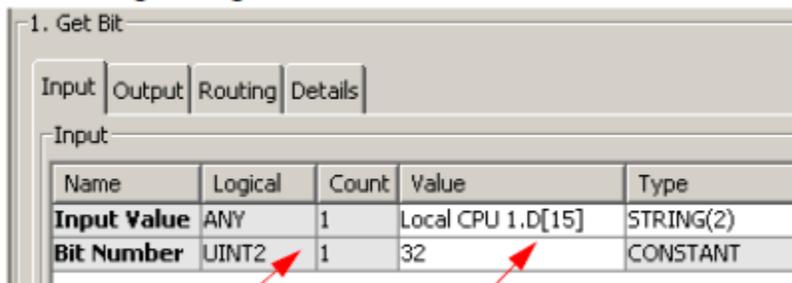
## I IOTA industrial IoT Platform: Bit numbering when using strings

Strings are arranged in memory as a 4-byte length field, followed by the actual string data. Therefore, when reading or writing bits in a string, you must account for the length field.



Notice bits zero through 31 represent the 4-byte length field for the string. The first bit of the actual string data is bit 32. The second data bit would be number 33 and so forth.

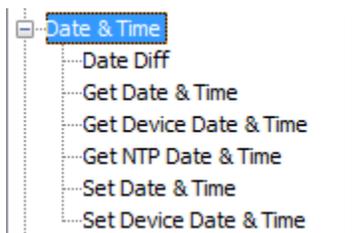
The following **Input** tab illustrates the **Get Bit** action for bit numbers when using a string.



Notice the **Type** column for the input value D[15] is STRING(2).

## IIoTA industrial IoT Platform: Date and Time

The **Date and Time** category provides actions for Get, Set, and Difference functions.



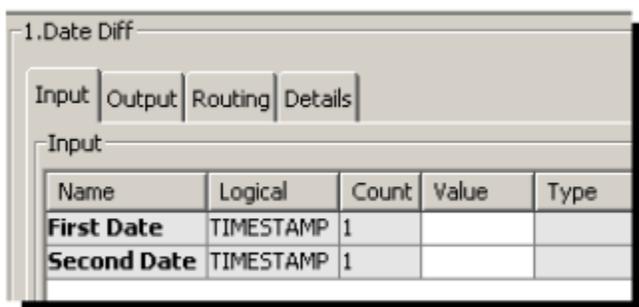
In addition to these actions, some products have date and time functions available from the Administration Time Management tab.

The **Date and Time** category provides these actions:

- Date Diff
- Get Date & Time
- Get Device Date & Time
- Set Date & Time
- Set Device Date & Time
- Example Get Device Date & Time
- Example Set Device Date & Time

## IIoTA industrial IoT Platform: Date Diff

The **Date Diff** action calculates the difference between two **TIMESTAMP** variables. This is useful for time processing when you want to determine the difference between two timestamps; for example, you might want to skip an update if the operation was recently performed. The results of the calculated difference are always positive (unsigned) numbers.



## Input tab

Parameter	Description
<b>First Date</b>	The input variable to use as the first <b>TIMESTAMP</b> .
<b>Second Date</b>	The input variable to use as the second <b>TIMESTAMP</b> .

## Output tab

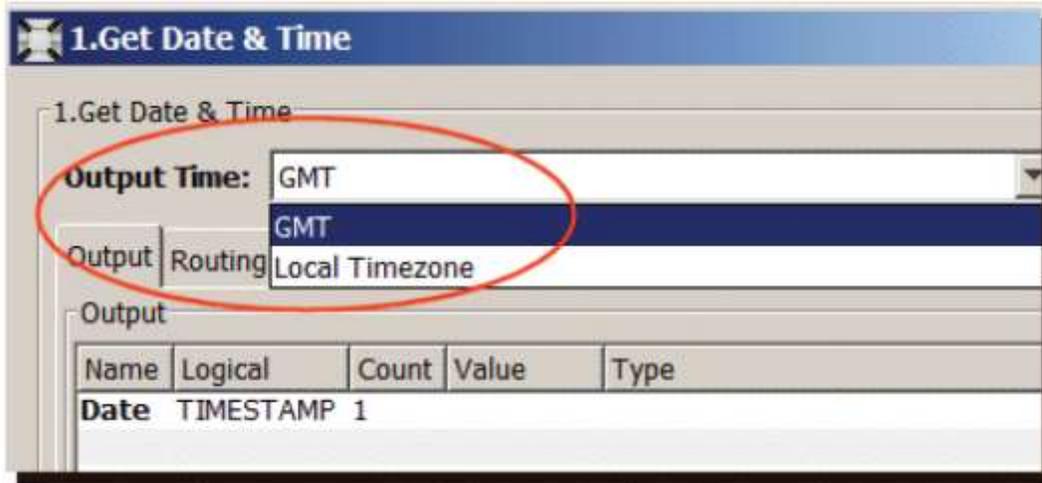
The **Output** tab provides the variables to hold the result values for the **Date Diff** action.

Name	Logical	Count	Value	Type
Total Seconds	UINT4	1		
Total Milliseconds	UINT4	1		
Days	UINT4	1		
Hours	UINT4	1		
Minutes	UINT4	1		
Seconds	UINT4	1		
Milliseconds	UINT4	1		

Parameter	Description
<b>Total Seconds</b>	The output variable that will hold the total number of seconds.
<b>Total Milliseconds</b>	The output variable that will hold the total number of milliseconds.
<b>Days</b>	The output variable that will hold the number of days.
<b>Hours</b>	The output variable that will hold the number of hours.
<b>Seconds</b>	The output variable that will hold the number of seconds.
<b>Milliseconds</b>	The output variable that will hold the number of milliseconds.

# IIoTA industrial IoT Platform: Get Date & Time

The **Get Date & Time** action retrieves the date and time from the node.



## Parameters

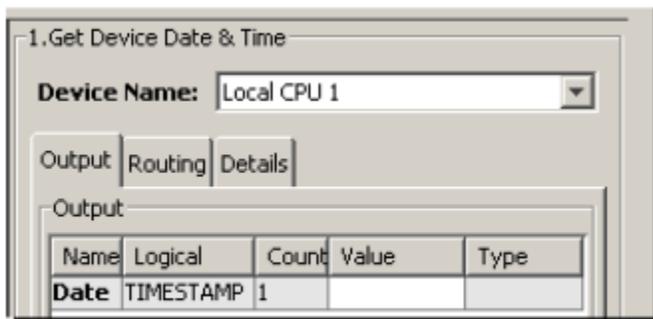
Parameter	Description
<b>Output Time</b>	<p>Specifies the format of the date and time as it is stored internally in the node. This relates to the internal format and not the format of the <b>Date</b> output variable on the output tab.</p> <p>Options are:</p> <p><b>GMT</b> — Greenwich Mean Time (GMT). This is the default format. The internal representation of the date and time in the node is in a Unix time format.</p> <p><b>Local Time zone</b> — The internal representation of the date and time in the node is a local time representation, considering the current configured time zone for the node. This is not the common node configuration and this option should only be used when the node uses the local time internal representation. For either option, the time stamp will be adjusted to represent the local time in the time zone configured for the node before being written to the output variable specified in <b>Date</b> on the <b>Output</b> tab.</p>

## Output tab

Parameter	Description
<b>Date</b>	The output variable that will be set to the current local date and time for the node. Any adjustments for the currently configured time zone and daylight savings time are made when converting to the local time output to the <b>Date</b> output variable.

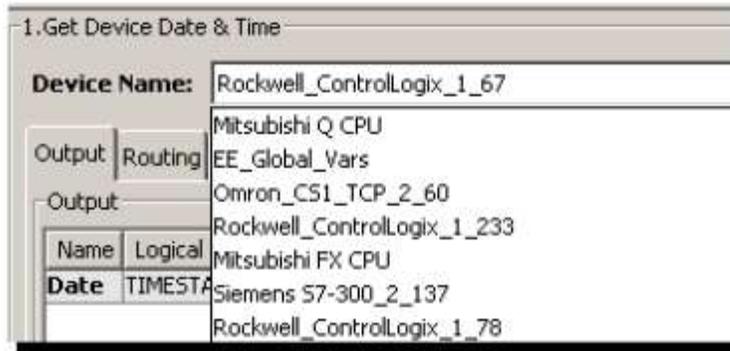
## IIoTA industrial IoT Platform: Get Device Date & Time

The **Get Device Date & Time** action retrieves the date and time from a device.



## Parameter Description

Parameter	Description
<b>Device Name</b>	The name of the device whose date and time you want to retrieve. The <b>Device Name</b> drop-down list provides the available devices that are in a started state on the node.



Not all devices or device drivers support the **Get Device Date & Time** action. You will receive a *command not supported* error message from a device that does not support the action.

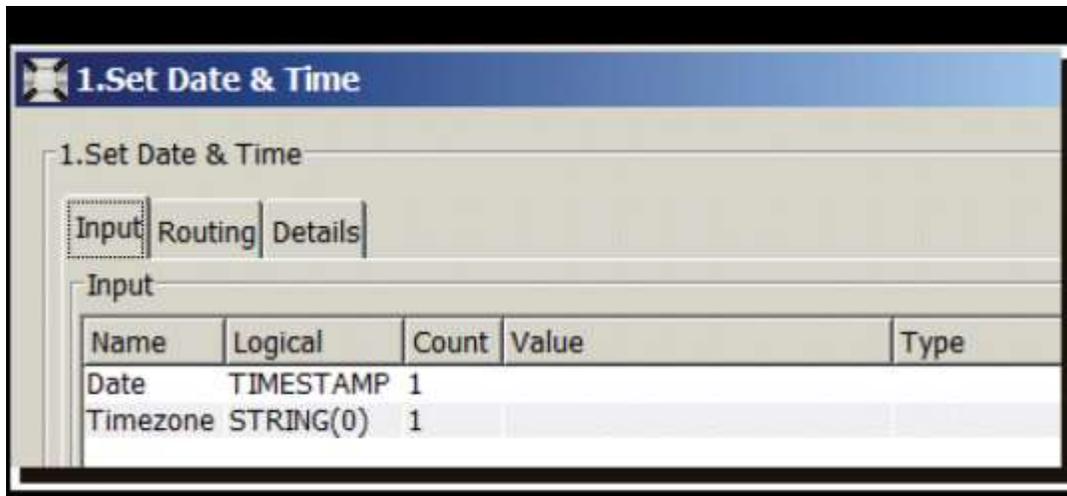
## Output tab

Parameter	Description
Date	The output variable that will receive the time stamp value from the device. For some devices, the millisecond portion of the time is not available and will be returned as 000.

## IIOA industrial IoT Platform: Set Date & Time

The **Set Date & Time** action sets the date and time, as well as the time zone of the node.

Depending on the features and capabilities of your node, the ability to set the time zone may not be available.



## Notes about specifying the time zone

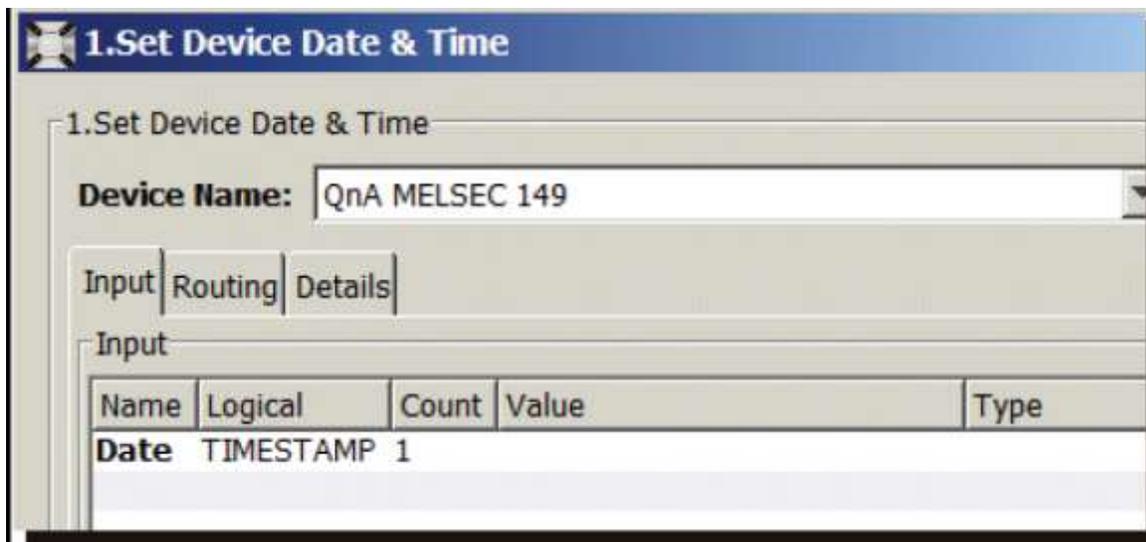
To find the available time zones and formats go to the Workbench > **Administration** > Time Management tab and view the options available using the **Device Time Zone** drop-down list. The time zone strings visible in the drop-down list are the exact formats that must be specified in the **Time zone** parameter on the **Input** tab.

## Input tab

Parameter	Description
<b>Date</b>	The date and time on the node will be set to the value supplied in this variable.
<b>Time zone</b>	The time zone on the node will be set to the value supplied in this variable. See the note above for details about specifying the time zone correctly.

## IIoTA industrial IoT Platform: Set Device Date & Time

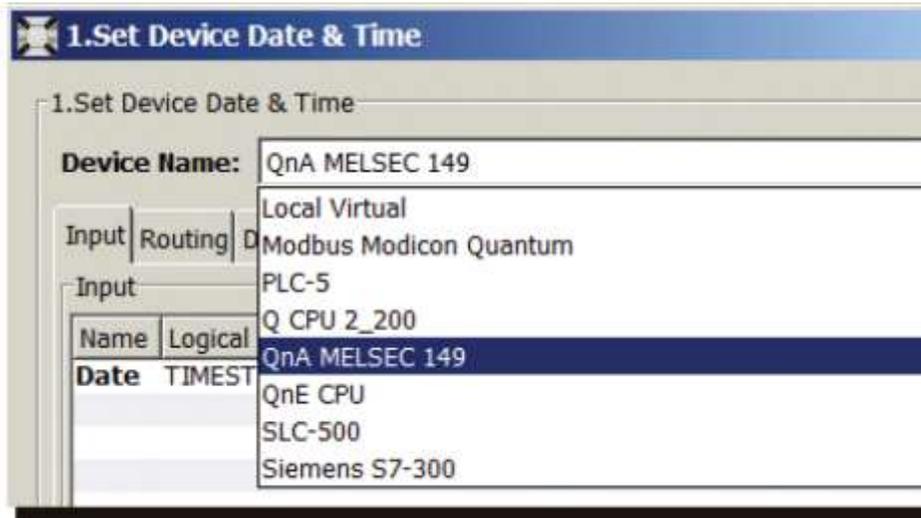
The **Set Device Date & Time** action sets the date and time on a device.



## Parameter description

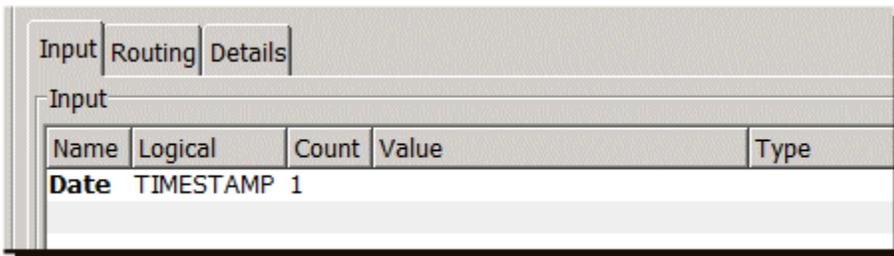
Parameter	Description
<b>Device Name</b>	The name of the device whose date and time you want to set. The <b>Device Name</b> drop-down list provides the available devices that are in a started state

on the node.



Not all devices or device drivers support the **Set Device Date & Time** action. You will receive a *command not supported* error message from a device that does not support the action.

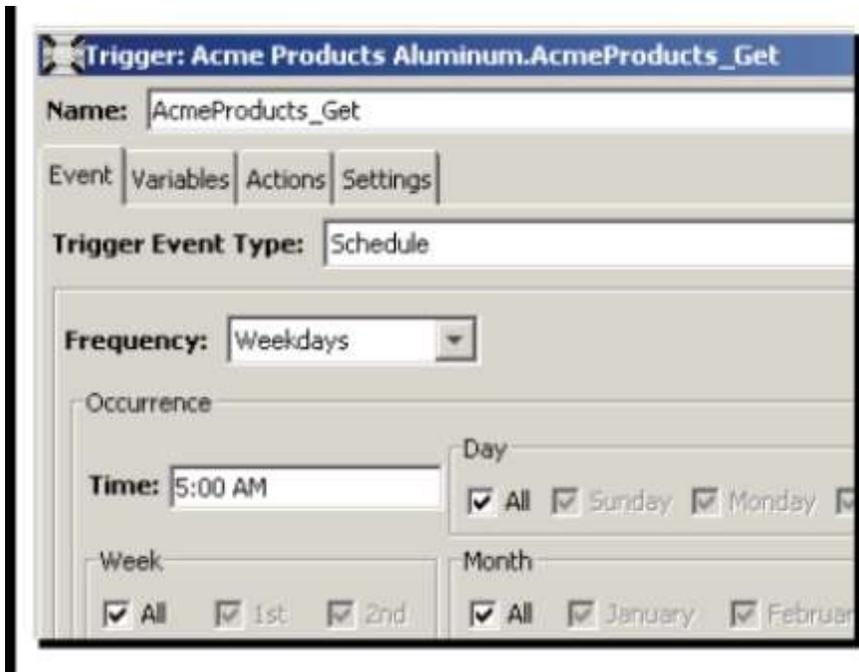
## Input tab



Parameter	Description
Date	The input variable with the time stamp to use as the input

## IIOA industrial IoT Platform: Example Get Device Date & Time

The following shows a schedule trigger that uses a **Get Device Date & Time** and a **Set Date & Time** action.



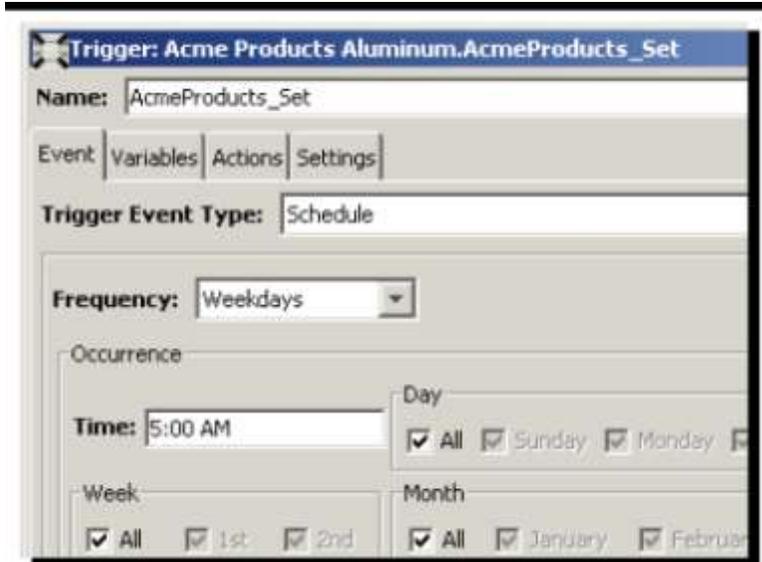
The trigger will execute every day at 5:00 A.M. to synchronize the time between a device and the node.

The **Get Device Date & Time** action reads the system time from a device named local CPU 1. The **Set Date & Time** action sets time on the node.

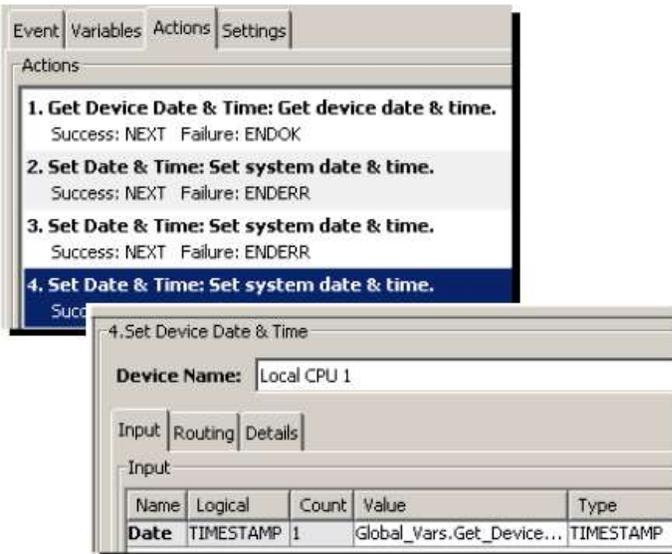


## IIoTA industrial IoT Platform: Example Set Device Date & Time

The following shows a schedule trigger that uses a **Get Date & Time** and three **Set Device Date & Time** actions. The trigger will execute every day at 5:00 A.M.



For this example, the **Get Date & Time** action reads the system time from the node and then sets the time on three different devices.

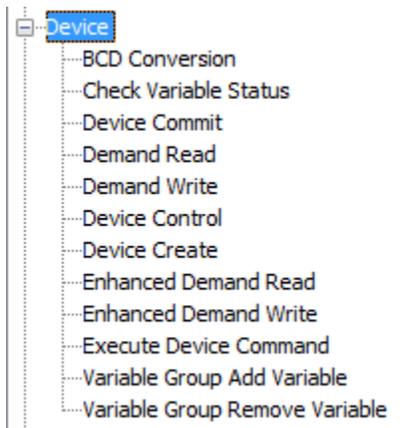


**Note:**

In the case where several devices will have their time set, you might experience a time delay to process each request due to network latency. For that situation, consider executing a sequence where a **Get Date & Time** action is immediately followed by a **Set Device Date & Time** action for each device. That way both actions (get and set) are repeated for each device.

## IIOA industrial IoT Platform: Device

The **Device** category provides actions that access and control variables, devices, and variable groups.



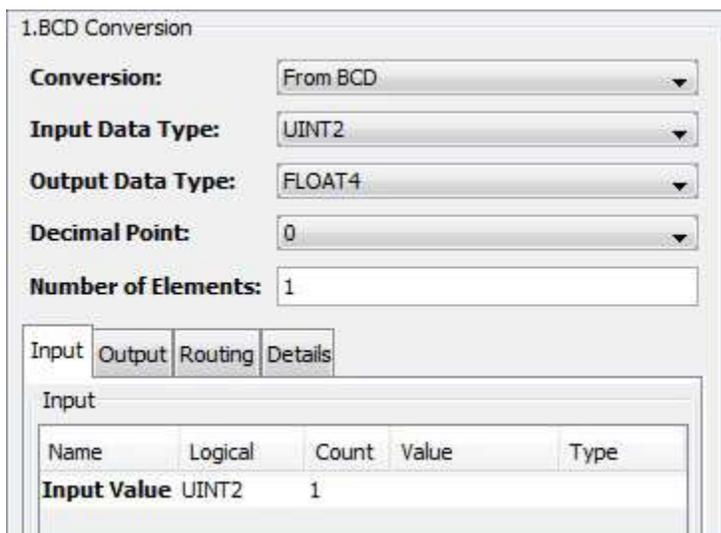
The **Device** category provides these actions:

- BCD Conversion
- Check Variable Status
- Demand Read
- Demand Write
- Device Commit
- Device Control
- Enhanced Demand Read
- Enhanced Demand Write
- Execute Device Command
- Get Device State
- Variable Group Add Variable
- Variable Group Remove Variable

## IIoTA industrial IoT Platform: BCD Conversion

The **BCD Conversion** action converts data from a BCD format into a numeric format. The value that was converted from a BCD format can be stored in either an integer, an unsigned integer, or a floating-point numeric format. The action can also convert integer, unsigned integer, or floating-point data into a BCD format.

A BCD Conversion can be done on an individual data element or an array of data elements.



## Parameter Descriptions

Parameter	Description
<b>Conversion</b>	The type of conversion to be performed, either <b>From BCD</b> or <b>To BCD</b> . The <b>From BCD</b> option will convert BCD data into a numeric format, while the <b>To BCD</b> option will convert numeric data into a BCD format.
<b>Input Data Type</b>	<p>The data type of the input data. This is the format of the variables that contain the BCD values, if the <b>From BCD</b> conversion option is selected. The following options are available if the <b>Conversion</b> parameter is <b>From BCD</b>.</p> <p>UINT1 UINT2 UINT4 UINT8</p> <p>This is the format of the variables that contain the numeric values, if the <b>To BCD</b> conversion option is selected. The following options are available if the <b>Conversion</b> parameter is <b>To BCD</b>.</p> <p>UINT1 UINT2 UINT4 FLOAT4 UINT8 FLOAT8</p>

<p><b>Output Data Type</b></p>	<p>The data type of the output data.</p> <p>This is the format of the variables that contain the numeric values, if the <b>From BCD</b> conversion option is selected. The following options are available if the <b>Conversion</b> parameter is <b>From BCD</b>.</p> <p>UINT1          UINT2          UINT4          FLOAT4          UINT8          FLOAT8</p> <p>This is the format of the variables that contain the BCD values, if the <b>To BCD</b> conversion option is selected. The following options are available if the <b>Conversion</b> parameter is <b>To BCD</b>.</p> <p>UINT1          UINT2          UINT4          UINT8</p>
<p><b>Decimal Point</b></p>	<p>This option will indicate how the conversion will handle the decimal points when either the FLOAT4 or FLOAT8 data types are selected.</p> <p>FLOAT4 data types will contain the following options:</p> <p>0, 1, 2, 3 or 4.</p> <p>FLOAT8 data types will contain the following options:</p> <p>0, 1, 2, 3, 4, 5, 6, 7 or 8.</p> <p>If the <b>Conversion</b> parameter is <b>From BCD</b>, the <b>Decimal Point</b> parameter dictates how many decimal places the value should be shifted to the right after it has been converted from its BCD format. For example, if a value of 1234 is returned from the <b>From BCD</b> process and the <b>Decimal Point</b> parameter is 3, the value of 1.234 will be written to the FLOAT4 or FLOAT8 variable.</p> <p>If the <b>Conversion</b> parameter is <b>To BCD</b>, the <b>Decimal Point</b> parameter dictates how many decimal places the value should be shifted to the left before it is converted to a BCD format. For example, if a FLOAT4 variable has a value of 13.64316 and the <b>Decimal Point</b> parameter is 2, the value of 1364 will be converted into a BCD value.</p>
<p><b>Number of Elements</b></p>	<p>This is the number of elements that will be processed during the conversion. A value of 1 indicates the conversion should process one value. A value</p>

greater than 1 indicates that an array of values will be converted when the action is executed.

Name	Logical	Count	Value	Type
<b>Input Value</b>	FLOAT4	1		

## Input tab

Parameter	Description
<b>Input Value</b>	The element that contains the data that will be the source of the BCD conversion. The value in the Logical column is set based on the value selected in the <b>Input Data Type</b> parameter. The value in the Count column is set based on the value entered in the <b>Number of Elements</b> parameter.

Name	Logical	Count	Value	Type
<b>Output Value</b>	UINT2	1		
resultStatus	INT4	1		
Element In Error	UINT2	1		

## Output tab

Parameter	Description
<b>Output Value</b>	The destination variable where the converted data is copied. The value in the Logical column is set based on the value selected in the <b>Output Data Type</b> parameter. The value in the Count column is set based on the value entered in the <b>Number of Elements</b> parameter.
resultStatus	A numeric value that will be 0 (zero), if the conversion was successful. A value less than 0 will indicate that an error occurred during the conversion. Possible errors include.  -5214: The variable defined in the Input Value parameter is missing. -5215: The variable defined in the Output Value parameter is missing.

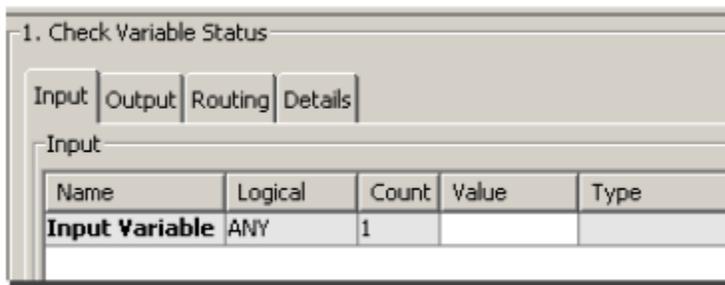
	<p>-6254: In the case where the conversion is from a numeric value <b>To BCD</b>, the value being converted is not in a valid BCD format.</p> <p>-6254: In the case where the conversion is <b>From BCD</b> the value converted from a BCD is too large to fit in the output variable. For example, if the BCD value converted to a value &gt; 256 will not fit in an Unsigned Int. The largest value that can be written to a FLOAT4 variable is 4294967000. Attempting to write a value larger than this will result in an error.</p>
Element in Error	This will indicate the first array element that contained a value that could not be converted because it was not in a valid BCD format, or contained a value that could not be converted into a valid BCD format.

## I IOTA industrial IoT Platform: Check Variable Status

The **Check Variable Status** action checks the status, or availability to access, a device variable.

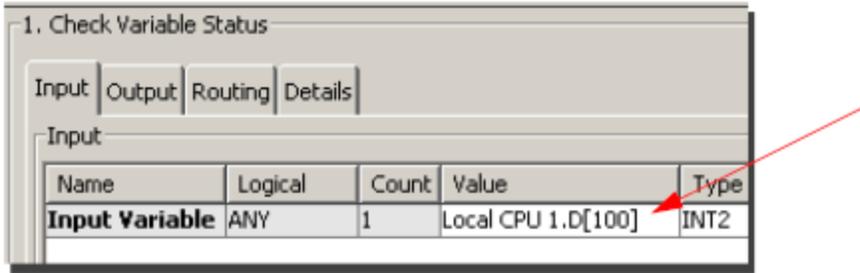
This can be used to determine if a device and its variables is available (meaning the device is in a Started state) before continuing with a series of actions or before execution portions of the application logic (before executing triggers).

In the normal flow of a trigger's logic, an action would reference a device variable and would have the failure route defined accordingly if the action failed. There are cases where a decision about the ability to access a device is made at a higher level before continuing with the application logic.



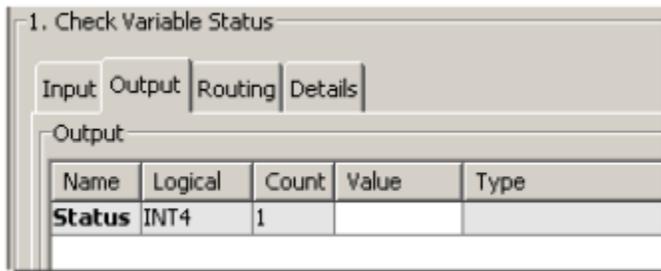
### Input tab

The **Input** tab allows you to specify the device variable whose status needs to be checked. For this example D[100].



Parameter	Description
<b>Input Variable</b>	The device variable to check for ability to access.

## Output tab



Parameter	Description
<b>Status</b>	The output status of the device variable. If the variable was successfully accessed, the status will be zero. If there was an error accessing the <b>Input Variable</b> , the status will have an error code that was returned from the device driver.

## Routing tab



Parameter	Description
<b>Good Value</b>	The route to take when the variable was accessed successfully.
<b>Bad Value</b>	The route to take when the device driver had an error accessing the variable.

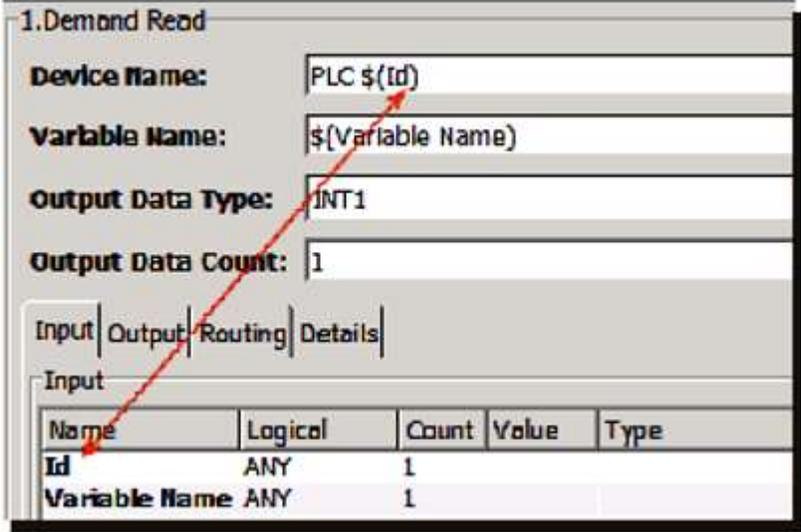
## IIoTA industrial IoT Platform: Demand Read

The **Demand Read** action reads a device variable at that point in a trigger's execution, bypassing the initial trigger buffer read at the start of the trigger's execution.

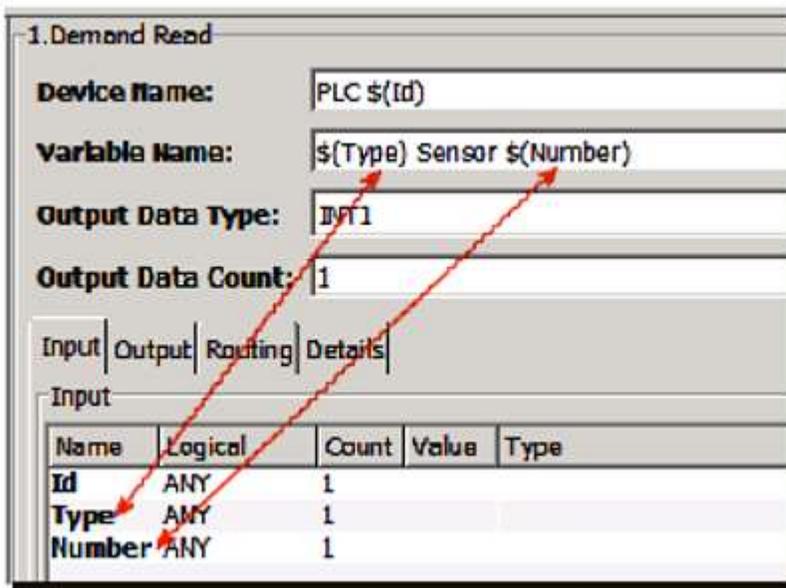
When a trigger starts its execution, all variables referenced by the trigger's actions are read into internal buffers. The variables' values are read from or written to this internal buffer when the variables are referenced by any of the trigger's actions. The internal buffer is written to the devices when the trigger completes its execution.

To bypass this internal buffering, the **Demand Read** action forces a read of a device variable through the device manager and device driver to the device.

### Parameter descriptions

Parameter	Description
<b>Device Name</b>	<p>The device where the variable resides. The name can be a constant (entered directly in the parameter), or the name can be dynamically specified. The default value is the substitution variable, \$(Device Name). This can be used along the <b>Input</b> tab parameter Device Name, or it can be changed. The parameter lets you specify the name of a device using a compound string. The following shows an example <b>Device Name</b> parameter and the row that holds its <b>Id</b> variable in the <b>Input</b> tab.</p> 
<b>Variable Name</b>	<p>The variable to read. The name can be a constant (entered directly in the parameter), or the name can be dynamically specified.</p>

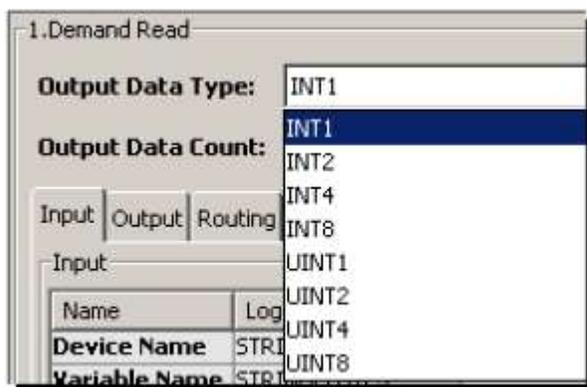
The default value is the substitution variable, \$(Variable Name). This can be used along the **Input** tab parameter Variable Name, or it can be changed. The parameter lets you specify the name of a device using a compound string. The following shows an example **Variable Name** parameter and the rows that hold its **Type** and **Number** variables in the **Input** tab.



The substitution variables \$(Type) and \$(Number) are mapped in the **Input** tab.

**Output Data Type**

The parameter specifies the data type of the output variable.



**Output Data Count**

The value specifies the number of elements of the output variable. For a single device variable, enter 1. For an array device variable, enter the number of array elements to read.

## Input tab

The **Input** tab is used to read a device variable. The device name and the variable name must first be passed into the action.

Input				
Name	Logical	Count	Value	Type
<b>Device Name</b>	STRING(128)	1		
<b>Variable Name</b>	STRING(128)	1		

Parameter	Description
<b>Device Name</b>	The name of the device containing the variable whose value you want to read.
<b>Variable Name</b>	The name of the variable whose value you want to read.

## Output tab

Output				
Name	Logical	Count	Value	Type
<b>Value</b>	INT1	1		

Parameter	Description
<b>Value</b>	The value read from the device variable. The <b>Logical</b> data type value is set based on the <b>Output Data Type</b> parameter. The <b>Count</b> value is set based on the <b>Output Data Count</b> parameter.

## Demand Read considerations

Device variables are normally "read" when they are the source variable in an action. For example, a **Set** action has a source variable and a destination variable.

When a trigger starts its execution, all variables referenced by the trigger's actions are read into internal buffers. The variables' values are read from or written to this internal buffer when the variables are referenced by any of the trigger's actions. The internal buffer is written to the devices when the trigger completes its execution. This process maximizes performance of the trigger execution.

In cases where the device variable needs to be read directly from the device and not from the previously read internal trigger buffers, the **Demand Read** action is used. These cases need to

consider the overall application logic in the triggers and in the devices to understand the dynamic interaction of all parts of the application. The **Demand Read** action does take additional resources in terms of processing time (compared to for example a **Set** action), since the trigger engine and device driver must communicate down to the physical device and get a response. This includes the communication across the network to the device.

Related topics

Enhanced Demand Read

Device Commit

## IIoTA industrial IoT Platform: Demand Write

The **Demand Write** action writes a device variable through to the device at that point in a trigger's execution, bypassing the internal trigger buffer used for device variables that is normally written when the trigger ends execution.

When a trigger starts its execution, all variables referenced by the trigger's actions are read into internal buffers. The variables' values are read from or written to this internal buffer when the variables are referenced by any of the trigger's actions. The internal buffer is written to the devices when the trigger completes its execution.

To bypass this internal buffering, the **Demand Write** action forces a write of a device variable through the device manager and device driver to the device.

### Parameter descriptions

Parameter	Description
<b>Device Name</b>	<p>The device where the variable resides.</p> <p>The name can be a constant (entered directly in the parameter), or the name can be dynamically specified.</p> <p>The default value is the substitution variable, \$(Device Name). This can be used along the <b>Input</b> tab parameter Device Name, or it can be changed.</p> <p>The parameter lets you specify the name of a device using a compound string. The following shows an example <b>Device Name</b> parameter and the row that holds its <b>Id</b> variable in the <b>Input</b> tab.</p>

1.Demand Write

**Device Name:** PLC \$(Id)

**Variable Name:** \$(Variable Name)

**Input Data Type:** INT1

**Input Data Count:** 1

Input | Routing | Details

Input

Name	Logical	Count	Value	Type
Id	ANY	1		
Variable Name	ANY	1		
Value	INT1	1		

The substitution variable \$(Id) is mapped on the **Input** tab.

**Variable Name**

The variable to write.  
 The name can be a constant (entered directly in the parameter), or the name can be dynamically specified.  
 The default value is the substitution variable, \$(Variable Name). This can be used along the **Input** tab parameter Variable Name, or it can be changed.  
 The parameter lets you specify the name of a device using a compound string. The following shows an example **Variable Name** parameter and the rows that hold its **Type** and **Number** variables in the **Input** tab.

1.Demand Write

**Device Name:** PLC \$(Id)

**Variable Name:** \$(Type) Sensor \$(Number)

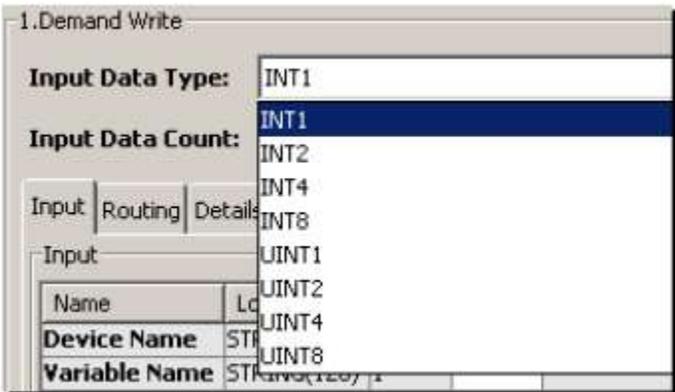
**Input Data Type:** INT1

**Input Data Count:** 1

Input | Routing | Details

Input

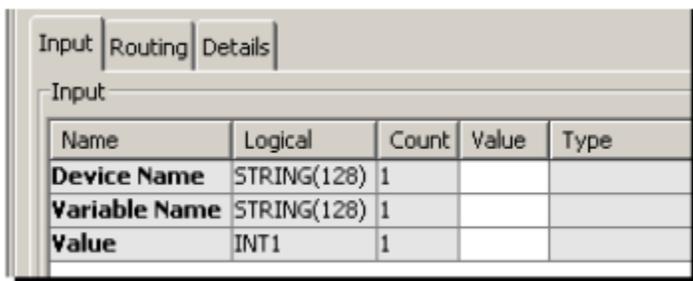
Name	Logical	Count	Value	Type
Id	ANY	1		
Type	ANY	1		
Number	ANY	1		
Value	INT1	1		

	The substitution variables \$(Type) and \$(Number) are mapped in the <b>Input</b> tab.
<b>Input Data Type</b>	<p>The parameter specifies the data type of the input variable.</p> 
<b>Input Data Count</b>	<p>The value specifies the number of elements of the input variable.                  For a single variable, enter 1.                  For an array variable, enter the number of array elements to write.</p>

## Input tab

The parameters in the Input tab are based on the substitution variables used in the Device Name and Variable Name parameters.

The default substitution variables are \$(Device Name) and \$(Variable Name), which results in an Input tab as show below.



Parameter	Description
<b>Device Name</b>	The name of the device that contains the variable that you want to write to.
<b>Variable Name</b>	The name of the variable whose value you want to write.
<b>Value</b>	The value to write to the target variable based on the <b>Device Name</b> and <b>Variable Name</b> parameters.

For the example substitution variables \$(Id), \$(Type) and \$(Number), they would result in a Input tab with those parameters to map to input variables.

## Demand Write considerations

Device variables are normally "written to" when they are the destination variable in an action. For example, a **Set** action has a source variable and a destination variable.

When a trigger starts its execution, all variables referenced by the trigger's actions are read into internal buffers. The variables' values are read from or written to this internal buffer when the variables are referenced by any of the trigger's actions. The internal buffer is written to the devices when the trigger completes its execution. This process maximizes performance of the trigger execution.

In cases where the device variable needs to be written directly to the device and not just into the internal trigger buffers, the **Demand Write** action is used. These cases need to take into account the overall application logic in the triggers and in the devices to understand the dynamic interaction of all parts of the application. The **Demand Write** action does take additional resources in terms of processing time (compared to for example a **Set** action), since the trigger engine and device driver must communicate down to the physical device and get a response. This includes the communication across the network to the device.

The internal trigger buffers (for example the destination variable of a **Set** action) are written to the devices when the trigger completes its execution.

## IIoTA industrial IoT Platform: Device Commit

The **Device Commit** action writes the device variables held in the trigger's internal buffers through the device manager to the device driver to the device.

When a trigger starts its execution, all variables referenced by the trigger's actions are read into internal buffers. The variables' values are read from or written to this internal buffer when the variables are referenced by any of the trigger's actions. The internal buffer is written to the devices when the trigger completes its execution. This process maximizes performance of the trigger execution.

To bypass this internal buffering, the **Device Commit** action forces a write of the device variables from the internal buffers through the device manager and device drivers to the devices.

## Background

You can use the **Device Commit** action in a trigger to flush the device variable write buffer to the attached devices. As the trigger executes, all writes to a device variable are written to a

buffer, and when the trigger execution ends, the buffer is flushed to the devices. This is done to maximize the performance of variable writes to the devices.

As an example, suppose you have a trigger that has two **Set** actions and a **Wait** action.

1. Set D[1000] to 1
2. Wait 2000 ms
3. Set D[1000] to 0

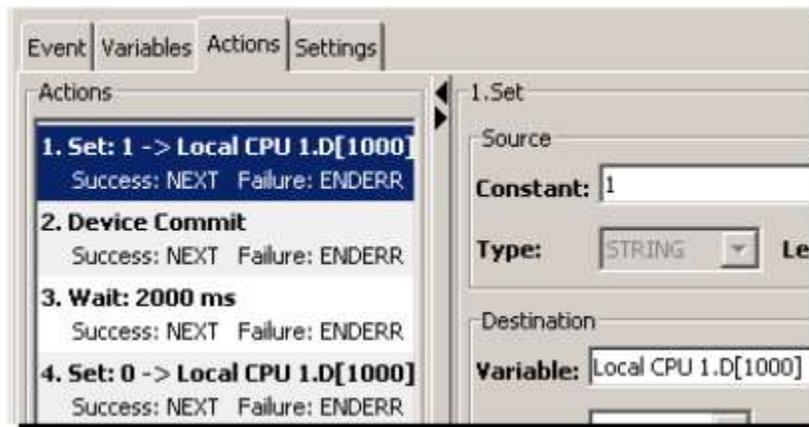
Each **Set** action setting a value for a device variable. When the trigger executes, D[1000] is set to 1 in the write buffer, and the **Wait** action pauses the trigger for 2000 milliseconds.

When the trigger execution resumes, the second **Set** action sets D[1000] to zero in the write buffer, and then overwrites the previous value of 1 that is already in the buffer.

When trigger execution ends, the buffer is flushed and D[1000] is set to zero in the attached device.

The trigger never actually writes 1 to D[1000] at any point during execution.

Adding a **Device Commit** action between the two **Set** actions, would enable the value of the first **Set** action to be written to the PLC.



When this trigger executes, its behavior will be different. The trigger will set D[1000] to 1 in the write buffer.



Then **Device Commit** will cause the write buffer to be flushed and D[1000] will be set to 1 in the attached device. The trigger will then wait for 2000 milliseconds; after which time, the second **Set** action will set D[1000] to zero in the write buffer. When trigger execution ends, the

trigger will flush the write buffer and then set D[1000] to zero in the attached device. The different behavior is that D[1000] gets set to 1 for 2 seconds, and then set back to zero.

## Device Commit considerations

Device variables are normally "written to" when they are the destination variable in an action. For example, a **Set** action has a source variable and a destination variable.

When a trigger starts its execution, all variables referenced by the trigger's actions are read into internal buffers. The variables' values are read from or written to this internal buffer when the variables are referenced by any of the trigger's actions. The internal buffer is written to the devices when the trigger completes its execution. This process maximizes performance of the trigger execution.

In cases where all device variables need to be written directly to the devices and not just into the internal trigger buffers, the **Device Commit** action is used. These cases need to take into account the overall application logic in the triggers and in the devices to understand the dynamic interaction of all parts of the application. The **Device Commit** action does take additional resources in terms of processing time (compared to for example a **Set** action), since the trigger engine and device driver must communicate down to the physical devices. This includes the communication across the network to the device.

The internal trigger buffers (for example the destination variable of a **Set** action) are written to the devices when the trigger completes its execution.

## IIoTA industrial IoT Platform: Device Control

The **Device Control** action starts, stops or deletes a device.

1. Device Control

**Action Input Type:** Static

**Operation:** Start Device

**Device Name:** Local CPU 1

Routing | Details

Routing

On Result	Go to
Success	NEXT
Failure	UNDEFINED

## Parameter descriptions

Parameter	Description
<b>Action Input Type</b>	<p>There are two options:</p> <p><b>Static</b> — The specified device name cannot be changed unless you edit the trigger.</p> <p><b>Dynamic</b> — When you select <b>Dynamic</b>, an <b>Input</b> tab parameter lets you specify the device name in a string variable.</p>
<b>Operation</b>	<p>The control operation option:</p> <p><b>Start Device</b> - This operation starts all the activity associated with a specific device. Keep the following in mind: Before the action executes, the state of the device (on the Workbench Devices tab) must be <b>Stopped</b>; otherwise, the trigger will fail.</p> <p><b>Stop Device</b> - This operation stops all the activity associated with a specific device. Keep the following in mind: Before the action executes, the state of the device (on the Workbench Devices tab) must be <b>Started</b>; otherwise, the trigger will fail.</p> <p>The <b>Device Start</b> and <b>Device Stop</b> actions are asynchronous. This means the action initiates a request for the device to start or stop. If you wish to start and stop a device within a single trigger, you should add a <b>Wait</b> action between the <b>Device Start</b> and the <b>Device Stop</b> actions with a duration suitable to allow the device to correctly change states. You can obtain an approximate value to use for this time by referring to the device start and device stop times available from the <b>Status</b> tab on the Devices tab in the Workbench.</p> <p><b>Delete Device</b> — When the trigger executes, the operation deletes a specific device. Before the action executes, the state of the device (on the Workbench Devices tab) must be <b>Stopped</b>; otherwise, the trigger will fail.</p>
<b>Device Name</b>	<p>Available when the <b>Action Input Type</b> is <b>Static</b>. This is the name of the device that will be started, stopped, or deleted. The list displays the devices that have been defined on the node.</p>

## Input tab

1. Device Control

**Action Input Type:** Dynamic

**Operation:** Start Device

Input | Routing | Details

Input

Name	Logical	Count	Value	Type
Device Name			STRING(128)	1

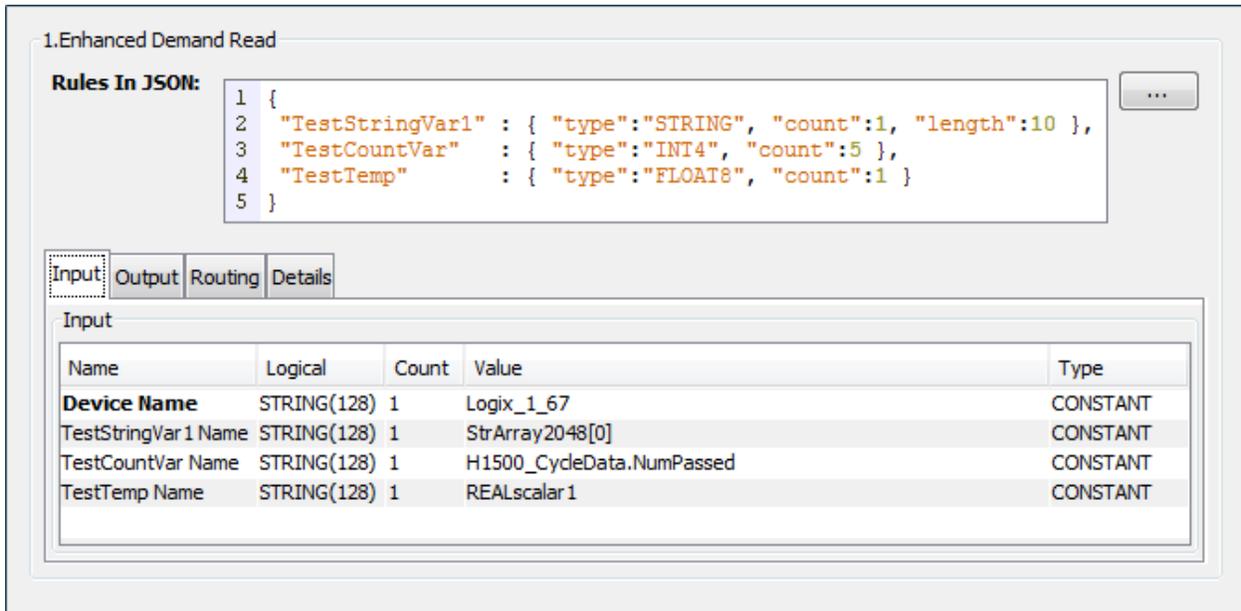
Parameter	Description
<b>Device Name</b>	Available when the <b>Action Input Type</b> is <b>Dynamic</b> . This is the name of the device that will be started, stopped, or deleted.

## IIoTA industrial IoT Platform: Enhanced Demand Read

The **Enhanced Demand Read** action reads a group of device variables at that point in a trigger's execution, bypassing the initial trigger buffer read at the start of the trigger's execution.

When a trigger starts its execution, all variables referenced by the trigger's actions are read into internal buffers. The variables' values are read from or written to this internal buffer when the variables are referenced by any of the trigger's actions. The internal buffer is written to the devices when the trigger completes its execution.

To bypass this internal buffering, the **Enhanced Demand Read** action forces a read of the device variables through the device manager and device driver to the device.



The Enhanced Demand Read action is part of the Advanced Features package.

This action is part of the Advanced Features package.

## Parameter descriptions

Parameter	Description
<b>Rules in JSON</b>	<p>The description of the device variables to read. Each variable is specified as shown in the example with:</p> <p>Map variable name - The name of the map variable, which will be added as a row on the Input tab.</p> <p>type - The data type of the variable to read. This type must match the data type of the variable.</p> <p>The supported types are:                      BOOL, INT1, INT2, INT4, INT8, UINT1, UINT2, UINT4, UINT8, FLOAT4, FLOAT8, STRING, TIMESTAMP, and BINARY.</p> <p>count - The number of variable elements. Specify a 1 for a single (scalar) element or a value greater than 1 for an array.</p> <p>length - For types STRING and BINARY, the length of the variable element.</p> <p>The multi-line input icon  can be used to display a larger input area for the JSON variable description.</p>

## Input tab

The **Input** tab has the device name parameter and an input map variable for each variable identified in the JSON variable description.

Parameter	Description
<b>Device Name</b>	The name of the device containing the variables whose values you want to read.
Input map variables	The map variables identified in the JSON variable description. Each variable in the JSON variable description will be added as a map variable row in the Input tab. In the <b>Value</b> column, specify the device variable name. The name can be a constant (entered directly in the parameter), or the name can be dynamically specified in a variable. In the example input tab above, the variables names are specified as constants.

## Output tab

The **Output** tab has an output variable for each variable identified in the JSON variable description.

1.Enhanced Demand Read

**Rules In JSON:**

```

1 {
2   "TestStringVar1" : { "type":"STRING", "count":1, "length":10 },
3   "TestCountVar"   : { "type":"INT4", "count":5 },
4   "TestTemp"       : { "type":"FLOAT8", "count":1 }
5 }

```

Input **Output** Routing Details

Name	Logical	Count	Value	Type
TestStringVar1	STRING(10)	1	LocalVariables.StringVar1	STRING(10)
TestCountVar	INT4	5	LocalVariables.CountVar[0]	INT4
TestTemp	FLOAT8	1	LocalVariables.Temp	FLOAT8

Parameter	Description
Output variables	Each variable in the JSON variable description will be added as a row in the Output tab. In the <b>Value</b> column, specify the variable where the value of the read input

<p>variable will be placed. The output variables can be trigger variables (local or static) or device variables. In the example output tab above, the variables are read into trigger local variables.</p>
--

## Enhanced Demand Read considerations

Device variables are normally "read" when they are the source variable in an action. For example, a **Set** action has a source variable and a destination variable.

When a trigger starts its execution, all variables referenced by the trigger's actions are read into internal buffers. The variables' values are read from or written to this internal buffer when the variables are referenced by any of the trigger's actions. The internal buffer is written to the devices when the trigger completes its execution. This process maximizes performance of the trigger execution.

In cases where the device variables need to be read directly from the device at a certain point in the trigger's execution and not from the previously read internal trigger buffers, the **Enhanced Demand Read** action is used. These cases need to take into account the overall application logic in the triggers and in the devices to understand the dynamic interaction of all parts of the application. The **Enhanced Demand Read** action does take additional resources in terms of processing time (compared to for example a **Set** action), since the trigger engine and device driver must communicate down to the physical device and get a response. This includes the communication across the network to the device.

The data type specified for each variable must match the data type of the device variable. If it does not match, that variable will not be successfully read.

The action will fail if there is a problem with the device specified (not defined or not started). The action will succeed if there are individual problems with the variables specified. The variables that had individual problems (variable not defined, data type mismatch, data overflow writing to the output variable, data cast problem writing to the output variable) will not update the corresponding output variable. The variables that did not have a problem will update the corresponding output variables.

Related topics  
[Demand Read](#)  
[Device Commit](#)

# IIoTA industrial IoT Platform: Enhanced Demand Write

The **Enhanced Demand Write** action writes a group of device variables at that point in a trigger's execution, bypassing the internal trigger buffer used for device variables that is normally written when the trigger ends execution.

When a trigger starts its execution, all variables referenced by the trigger's actions are read into internal buffers. The variables' values are read from or written to this internal buffer when the variables are referenced by any of the trigger's actions. The internal buffer is written to the devices when the trigger completes its execution.

To bypass this internal buffering, the **Enhanced Demand Write** action forces a write of the device variables through the device manager and device driver to the device.

4.Enhanced Demand Write

**Rules In JSON:**

```

1 |{
2 |  "TestStringVar1" : { "type":"STRING", "count":1, "length":10 },
3 |  "TestCountVar"   : { "type":"INT4", "count":5 },
4 |  "TestTemp"       : { "type":"FLOAT8", "count":1 }
5 |}

```

Input Routing Details

Input

Name	Logical	Count	Value	Type
<b>Device Name</b>	STRING(128)	1	Logix_1_67	CONSTANT
TestStringVar1 Name	STRING(128)	1	StrArray2048[0]	CONSTANT
TestStringVar1	STRING(10)	1	LocalVariables.StringVar1	STRING(10)
TestCountVar Name	STRING(128)	1	H1500_CycleData.NumPassed	CONSTANT
TestCountVar	INT4	5	LocalVariables.CountVar[0]	INT4
TestTemp Name	STRING(128)	1	REALscalar1	CONSTANT
TestTemp	FLOAT8	1	LocalVariables.Temp	FLOAT8

The Enhanced Demand Write action is part of the Advanced Features package.

This action is part of the Advanced Features package.

## Parameter descriptions

Parameter	Description
<b>Rules in JSON</b>	The description of the device variables to write. Each variable is specified as shown in the example with:

Map variable name - The name of the map variable, which will be added as two rows on the Input tab.  
 The first row on the Input tab with the "Name" suffix is used to specify the destination device variable.  
 The second row on the Input tab is used to specify the source variable who's value will be written to the destination variable.  
 type - The data type of the variable to write. This type must match the data type of the destination variable.  
 The supported types are:  
 BOOL, INT1, INT2, INT4, INT8, UINT1, UINT2, UINT4, UINT8, FLOAT4, FLOAT8, STRING, TIMESTAMP, and BINARY.  
 count - The number of variable elements. Specify a 1 for a single (scalar) element or a value greater than 1 for an array.  
 length - For types STRING and BINARY, the length of the variable element.

The multi-line input icon  can be used to display a larger input area for the JSON variable description.

## Input tab

The **Input** tab has the device name parameter and two rows for each variable identified in the JSON variable description.

Parameter	Description
<b>Device Name</b>	The name of the device containing the variables whose values you want to write.
Variables	<p>Each variable in the JSON variable description will be added as two rows on the Input tab.</p> <p>The first row with the "Name" suffix is used to specify the destination device variable.            The second row is used to specify the source variable who's value will be written to the destination variable.            The source variables can be trigger variables (local or static) or device variables.            In the example input tab above, the destination device variables names are specified with constants.            The source variables are trigger local variables.</p>

## Enhanced Demand Write considerations

When a trigger action writes to a device variable, the updated value is held in an internal buffer until the trigger completes its execution.

When a trigger starts its execution, all variables referenced by the trigger's actions are read into internal buffers. The variables' values are read from or written to this internal buffer when the variables are referenced by any of the trigger's actions. The internal buffer is written to the devices when the trigger completes its execution. This process maximizes performance of the trigger execution.

In cases where the device variables need to be written directly to the device at a certain point in the trigger's execution and not just to the internal trigger buffers, the **Enhanced Demand Write** action is used. These cases need to take into account the overall application logic in the triggers and in the devices to understand the dynamic interaction of all parts of the application. The **Enhanced Demand Write** action does take additional resources in terms of processing time (compared to for example a **Set** action), since the trigger engine and device driver must communicate down to the physical device and get a response. This includes the communication across the network to the device.

The data type specified for each variable must match the data type of the destination device variable. If it does not match, that variable will not be successfully written.

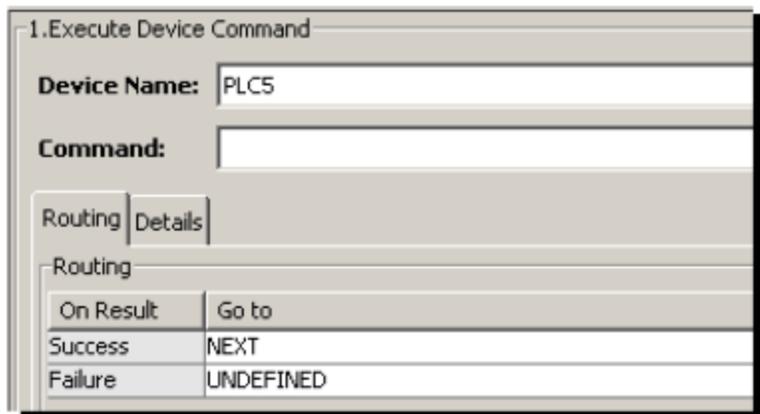
The action will fail if there is a problem with the device specified (not defined or not started). The action will succeed if there are individual problems with the variables specified. The variables that had individual problems (variable not defined, data type mismatch, data overflow, data cast problem) will not update the corresponding device variable. The variables that did not have a problem will update the corresponding device variables.

Related topics  
Demand Write  
Device Commit

## IIoTA industrial IoT Platform: Execute Device Command

The **Execute Device Command** action executes a command supported by the device driver and the device.

The available device commands is dependent on the device drivers installed on the node and the support within the device driver and the device models.



## Parameter descriptions

Parameter	Description
<b>Device Name</b>	<p>The name of the device. The list displays the devices that have been defined on the node.</p>
<b>Command</b>	<p>The Command drop-down list provides the available commands for the specific device. As mentioned, depending on the device driver installed and the device model, you might not see a list of commands as not all device types support device commands.</p>

## IIOA industrial IoT Platform: Get Device State

The **Get Device State** action returns the current state of a device. The device must be defined on the same node as the trigger.

1. Get Device State

**Action Input Type:** Static

**Device Name:** Data Mapping Global Vars

Output Routing Details

Output

Name	Logical	Count	Value	Type
Device State	UINT1	1		

The Get Device State action is part of the Technology Preview Extension

This action is part of the Technology Preview Extension, which is also referred to as the sandbox package. The action will be found in the **Devices** trigger action category upon successful deployment of the sandbox package.

See Technology Preview Extension for information on obtaining and installing the extension.

## Parameter description

Parameter	Description
<b>Action Input Type</b>	<p>The manner in which the device name parameter is defined. There are two options:</p> <p><b>Static</b> - The name of the device will be selected from the list shown in the <b>Device Name</b> parameter. The image above shows the Get Device State action when the Static option is selected.</p> <p><b>Dynamic</b> - The name of the device is defined in the <b>Device Name</b> parameter on the Input tab.</p>
<b>Device Name</b>	<p>Available when the <b>Action Input Type</b> is <b>Static</b>. The name of the device selected from the drop-down list of devices.</p>

## Input tab

Parameter	Description
<b>Device Name</b>	Available when the <b>Action Input Type</b> is <b>Dynamic</b> . The variable that contains the name of the device. This can be a variable or be a constant string.

## Output tab

Parameter	Description
<b>Device State</b>	An unsigned, one-byte integer that contains the current state of the device. The values are:  1 = Started 2 = Stopped 3 = Disabled 4 = Starting 5 = Stopping

# IIoTA industrial IoT Platform: Variable Group Add Variable

The **Variable Group Add Variable** action adds a device variable to an existing variable group.

The variable group used in the **Variable Group Add Variable** action must have previously been defined. A variable group is a collection of variables that are logically related and can be monitored collectively.

1. Variable Group Add Variable

**Variable Group:** \$(Variable Group)

**Device Name:** \$(Device Name)

**Variable Name:** \$(Variable Name)

**Key:** \$(Key)

Input Routing Details

Input

Name	Logical	Count	Value	Type
<b>Variable Group</b>	ANY	1		
<b>Device Name</b>	ANY	1		
<b>Variable Name</b>	ANY	1		
<b>Key</b>	ANY	1		

?

A **Variable Group** event type trigger is used to monitor the variables in a variable group. Starting the variable group will cause the variables to be monitored. When the value of any variable within the group changes, the trigger will execute.

## Parameters descriptions

The parameters support compound strings, they can be entered as direct constants or compound strings to build the parameter strings.

Parameter	Description
<b>Variable Group</b>	The variable group to which the device variable will be added. The variable group must have previously been defined.
<b>Device Name</b>	The device name. For example, Local CPU 1.
<b>Variable Name</b>	The variable name. For example, D[1]. The concatenation of the device name and device variable will be added to the variable group.
<b>Key</b>	A string value that will be available to the <b>Variable Group</b> event trigger. This variable key can be used as a reference ID, or correlation ID, to help identify which variable's value in the group has changed.

## Input tab

Parameter	Description
<b>other</b>	Depending on the values for the <b>Variable Group</b> , <b>Device Name</b> , <b>Variable Name</b> and key parameters, other input items might be available to correctly build the parameter strings. For more information, see Using compound strings.

Related topics

Variable Group

Defining, viewing, and controlling variable groups

## IIoTA industrial IoT Platform: Variable Group Remove Variable

The **Variable Group Remove Variable** action removes a device variable from a variable group.

2.Variable Group Remove Variable

**Variable Group:**

**Device Name:**

**Variable Name:**

Input Routing Details

Input				
Name	Logical	Count	Value	Type
<b>Variable Group</b>	ANY	1		
<b>Device Name</b>	ANY	1		
<b>Variable Name</b>	ANY	1		

?

## Parameters descriptions

The parameters support compound strings, they can be entered as direct constants or compound strings to build the parameter strings.

Parameter	Description
<b>Variable Group</b>	The variable group from which the device variable will be removed.
<b>Device Name</b>	The device name. For example, Local CPU 1.
<b>Variable Name</b>	The variable name. For example, D[1]. The concatenation of the device name and device variable will be removed from the variable group.

## Input tab

Parameter	Description
<b>other</b>	Depending on the values for the <b>Variable Group</b> , <b>Device Name</b> and <b>Variable Name</b> parameters, other input items might be available to correctly build the parameter strings. For more information, see <a href="#">Using compound strings</a> .

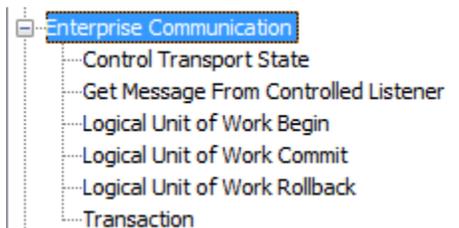
Related topics

Variable Group

Defining, viewing, and controlling variable groups

# IIoTA industrial IoT Platform: Enterprise Communication

The **Enterprise Communication** category provide actions that communicate with endpoint enterprise application programs.



The **Enterprise Communication** category provides these actions:

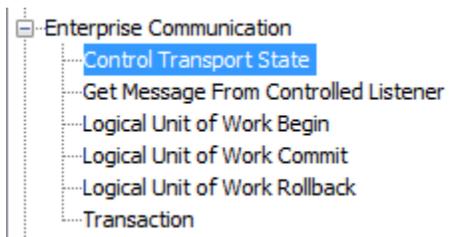
- Control Transport State
- Get Message From Controlled Listener
- Logical Unit of Work Begin
- Logical Unit of Work Commit
- Logical Unit of Work Rollback
- Transaction

## IIoTA industrial IoT Platform: Control Transport State

The **Control Transport State** action puts a transport in the suspended state or resumes a previously suspended transport.

When a transport in the **Up** state is suspended, the transport will disconnect from the host. The status of the transport as it appears on the **Transports** tab is changed to **Suspended**. Once suspended, transactions that come through the transport will be saved into a store and forward queue, if the transport has store and forward enable. Otherwise the transactions will be failed.

When a suspended transport is resumed, the transport will connect to the host. If it has transactions in its store and forward queue, they will be forwarded to the host.



### Parameter description

Parameter	Description
<b>Transport Name</b>	Provides a list of transports for the node.
<b>Operation</b>	Options are:  <b>Suspend</b> - Puts the transport in a suspended state. <b>Resume</b> - Removes the transport from the suspended state.

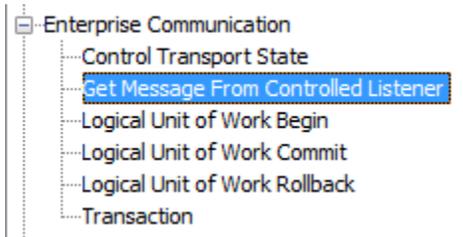
Related Topics

Suspending a transport

## IIoTA industrial IoT Platform: Get Message From Controlled Listener

The **Get Message From Controlled Listener** action initiates the retrieval of a message from a remote queue for a controlled listener. A controlled listener will not retrieve a message from its

associated queue system until it receives notification from a trigger in the form of a **Get Message From Controlled Listener** action.



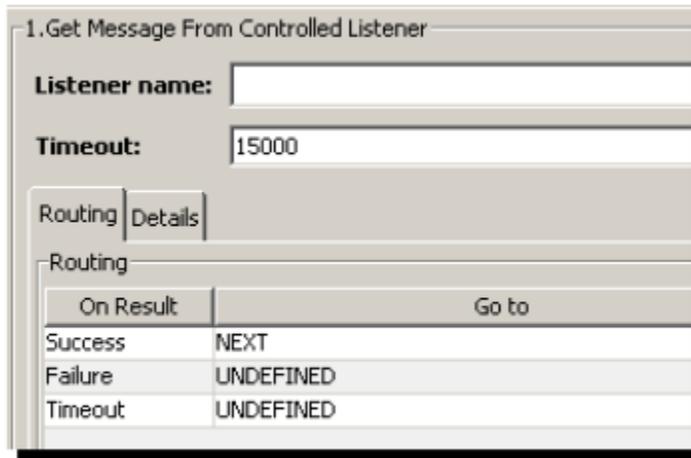
## Controlled listener

In order to use a **Get Message From Controlled Listener** action, you must have previously defined a controlled listener. A controlled listener is configured so that a **Get Message From Controlled Listener** action determines when the listener should get the command request from the remote queue. The result of getting a request from the queue is communicated to the action step.

**Note:** this action is not used in a Listener event type listener trigger.

For information about controlled listeners, see Listeners.

## Parameter descriptions



Parameter	Description
<b>Listener Name</b>	Use the <b>Listener Name</b> down-arrow to display a list of controlled listeners for the node. It is assumed that you have previously created a controlled listener.

<b>Timeout</b>	The <b>Timeout</b> parameter lets you set the maximum amount of time in milliseconds the action will wait for a listener request. The default timeout period is 15000 milliseconds or 15 seconds.
----------------	--

Relate topics

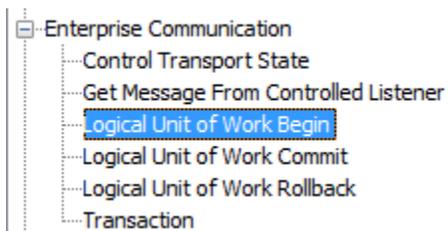
Creating a controlled listener

Listener

## IIoTA industrial IoT Platform: Logical Unit of Work Begin

The **Logical Unit of Work Begin** action indicates the beginning of a logical unit of work for a group of transactions. This action returns a logical unit of work identifier (LUWid) that is used to indicate that a **Transaction** action is part of this unit of work.

The corresponding Logical Unit of Work Commit and Logical Unit of Work Rollback actions are used to indicate the termination point of the unit of work.



### Assumptions

The following is assumed:

- You are familiar with the concept of a *logical unit of work* as it relates to a relational database.

### Parameter descriptions

Parameter	Description
<b>Transport Name</b>	This parameter provides a list of transports for the node. Select a transport from the list to indicate the database with which you want to start the logical unit of work.

### Database transport definition connection pool size

When the **Logical Unit of Work Begin** action executes, the Transaction Server reserves a connection from the transport's connection pool in order to process transactions associated with

this logical unit of work. To allow database transactions outside of a logical unit of work to be processed, the Transaction Server does not allow the last available connection to be reserved. In this condition, the **Logical Unit of Work Begin** action will fail. Make sure that the connection pool size of the selected transport is set to 2 (at a minimum). However, depending on your application design you should set this value higher based on how many logical unit of work and non-logical unit of work transactions you anticipate executing concurrently.

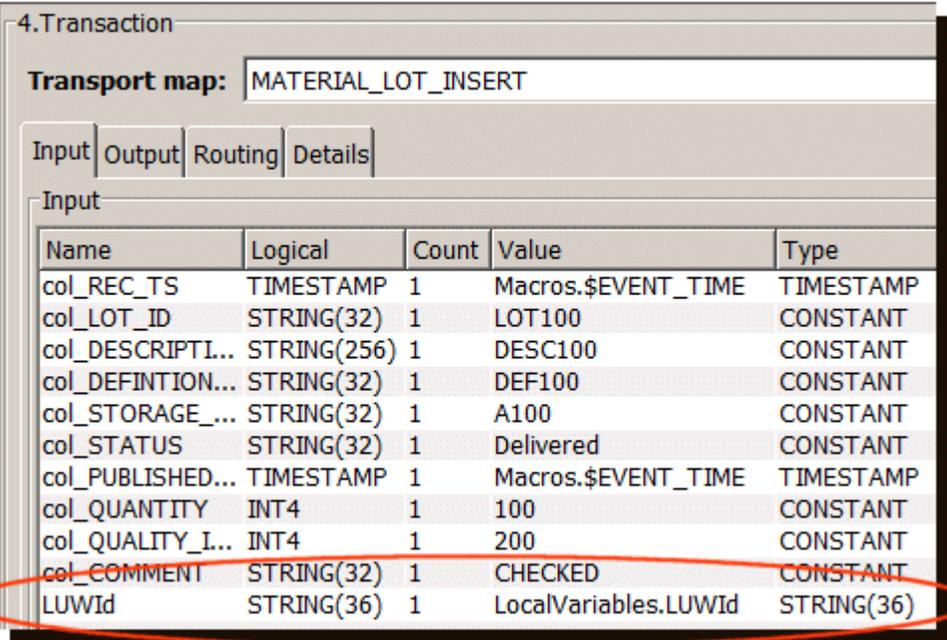
### Slow database connection times

When a transport is loaded at initialization, it establishes the number of connections defined by the transport pool size. If the time to connect to the database is relatively slow, then a **Logical Unit of Work Begin** action in a trigger may execute before all the transport's connections have been established. In this case, the **Logical Unit of Work Begin** action may timeout and need to be retried.

### LOCALDB

The Logical Unit of Work feature cannot be used with the LOCALDB transports including LOCALDB through LOCALDB5.

### Output tab

Parameter	Description																																																																											
<b>Identifier</b>	<p>The 36 character globally unique identifier (GUID) that is the unique identifier for this logical unit of work.</p> <p>The string is used in the <b>LUWId</b> input parameter of a <b>Transaction</b> action to have it execute as part of this logical unit of work.</p>  <table border="1" data-bbox="430 1218 1377 1858"> <caption>4.Transaction</caption> <tr> <td colspan="2"><b>Transport map:</b></td> <td colspan="3">MATERIAL_LOT_INSERT</td> </tr> <tr> <td colspan="5"> <div style="display: flex; border-bottom: 1px solid black;"> <span style="border-right: 1px solid black; padding: 2px 5px;">Input</span> <span style="padding: 2px 5px;">Output</span> <span style="padding: 2px 5px;">Routing</span> <span style="padding: 2px 5px;">Details</span> </div> </td> </tr> <tr> <td colspan="5">Input</td> </tr> <tr> <th>Name</th> <th>Logical</th> <th>Count</th> <th>Value</th> <th>Type</th> </tr> <tr> <td>col_REC_TS</td> <td>TIMESTAMP</td> <td>1</td> <td>Macros.\$EVENT_TIME</td> <td>TIMESTAMP</td> </tr> <tr> <td>col_LOT_ID</td> <td>STRING(32)</td> <td>1</td> <td>LOT100</td> <td>CONSTANT</td> </tr> <tr> <td>col_DESCRIPTI...</td> <td>STRING(256)</td> <td>1</td> <td>DESC100</td> <td>CONSTANT</td> </tr> <tr> <td>col_DEFINITION...</td> <td>STRING(32)</td> <td>1</td> <td>DEF100</td> <td>CONSTANT</td> </tr> <tr> <td>col_STORAGE_...</td> <td>STRING(32)</td> <td>1</td> <td>A100</td> <td>CONSTANT</td> </tr> <tr> <td>col_STATUS</td> <td>STRING(32)</td> <td>1</td> <td>Delivered</td> <td>CONSTANT</td> </tr> <tr> <td>col_PUBLISHED...</td> <td>TIMESTAMP</td> <td>1</td> <td>Macros.\$EVENT_TIME</td> <td>TIMESTAMP</td> </tr> <tr> <td>col_QUANTITY</td> <td>INT4</td> <td>1</td> <td>100</td> <td>CONSTANT</td> </tr> <tr> <td>col_QUALITY_I...</td> <td>INT4</td> <td>1</td> <td>200</td> <td>CONSTANT</td> </tr> <tr> <td>col_COMMENT</td> <td>STRING(32)</td> <td>1</td> <td>CHECKED</td> <td>CONSTANT</td> </tr> <tr> <td>LUWId</td> <td>STRING(36)</td> <td>1</td> <td>LocalVariables.LUWId</td> <td>STRING(36)</td> </tr> </table>	<b>Transport map:</b>		MATERIAL_LOT_INSERT			<div style="display: flex; border-bottom: 1px solid black;"> <span style="border-right: 1px solid black; padding: 2px 5px;">Input</span> <span style="padding: 2px 5px;">Output</span> <span style="padding: 2px 5px;">Routing</span> <span style="padding: 2px 5px;">Details</span> </div>					Input					Name	Logical	Count	Value	Type	col_REC_TS	TIMESTAMP	1	Macros.\$EVENT_TIME	TIMESTAMP	col_LOT_ID	STRING(32)	1	LOT100	CONSTANT	col_DESCRIPTI...	STRING(256)	1	DESC100	CONSTANT	col_DEFINITION...	STRING(32)	1	DEF100	CONSTANT	col_STORAGE_...	STRING(32)	1	A100	CONSTANT	col_STATUS	STRING(32)	1	Delivered	CONSTANT	col_PUBLISHED...	TIMESTAMP	1	Macros.\$EVENT_TIME	TIMESTAMP	col_QUANTITY	INT4	1	100	CONSTANT	col_QUALITY_I...	INT4	1	200	CONSTANT	col_COMMENT	STRING(32)	1	CHECKED	CONSTANT	LUWId	STRING(36)	1	LocalVariables.LUWId	STRING(36)
<b>Transport map:</b>		MATERIAL_LOT_INSERT																																																																										
<div style="display: flex; border-bottom: 1px solid black;"> <span style="border-right: 1px solid black; padding: 2px 5px;">Input</span> <span style="padding: 2px 5px;">Output</span> <span style="padding: 2px 5px;">Routing</span> <span style="padding: 2px 5px;">Details</span> </div>																																																																												
Input																																																																												
Name	Logical	Count	Value	Type																																																																								
col_REC_TS	TIMESTAMP	1	Macros.\$EVENT_TIME	TIMESTAMP																																																																								
col_LOT_ID	STRING(32)	1	LOT100	CONSTANT																																																																								
col_DESCRIPTI...	STRING(256)	1	DESC100	CONSTANT																																																																								
col_DEFINITION...	STRING(32)	1	DEF100	CONSTANT																																																																								
col_STORAGE_...	STRING(32)	1	A100	CONSTANT																																																																								
col_STATUS	STRING(32)	1	Delivered	CONSTANT																																																																								
col_PUBLISHED...	TIMESTAMP	1	Macros.\$EVENT_TIME	TIMESTAMP																																																																								
col_QUANTITY	INT4	1	100	CONSTANT																																																																								
col_QUALITY_I...	INT4	1	200	CONSTANT																																																																								
col_COMMENT	STRING(32)	1	CHECKED	CONSTANT																																																																								
LUWId	STRING(36)	1	LocalVariables.LUWId	STRING(36)																																																																								

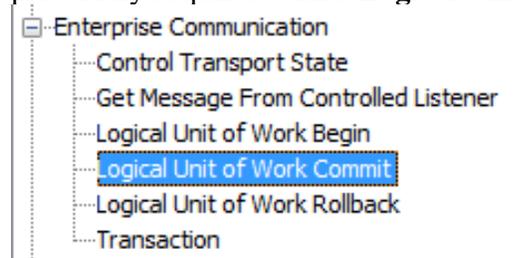
Multiple logical units of work can be in process concurrently. The **Transaction** actions indicate if they are of a logical unit of work with their **LUWid** parameter.  
If a **Transaction** action is not part of a logical unit of work, the **LUWid** parameter is left blank.

#### Related topics

Using a logical unit of work  
Logical Unit of Work Commit  
Logical Unit of Work Rollback

## IIoTA industrial IoT Platform: Logical Unit of Work Commit

The **Logical Unit of Work Commit** action completes a logical unit of work with the intent of committing all the changes made by the transactions that were part of the logical unit of work. This action requires that you specify a logical unit of work identifier (LUWid) that was previously acquired with a **Logical Unit of Work Begin** action.



### Input tab

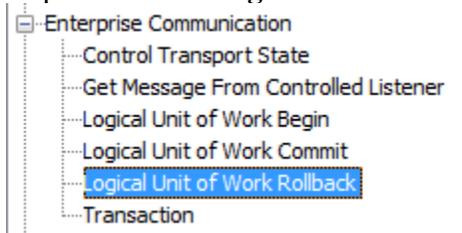
Parameter	Description
<b>Identifier</b>	The logical unit of work identifier that needs to be committed.

#### Related topics

Using a logical unit of work  
Logical Unit of Work Begin  
Logical Unit of Work Rollback

## IIoTA industrial IoT Platform: Logical Unit of Work Rollback

The **Logical Unit of Work Rollback** action is used to complete a logical unit of work with the intent of rolling back or discarding all changes made by the transactions that were part of the logical unit of work. This action requires the user to specify a **LUWId** that was previously acquired with a **Logical Unit of Work Begin** action.



### Input tab

Parameter	Description
<b>Identifier</b>	The logical unit of work identifier that needs to be rolled back.

Related topics

Using a logical unit of work

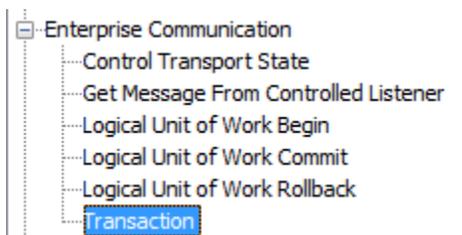
Logical Unit of Work Begin

Logical Unit of Work Commit

## IIoTA industrial IoT Platform: Transaction

The **Transaction** action sends data to or receives data from an endpoint enterprise application program.

For transactions that result in data flowing from the enterprise application (for example, a database select operation or stored procedure), the trigger's **Output** tab will the returned output variables.



1.Transaction

**Transport map:** MATERIAL\_LOT\_INSERT

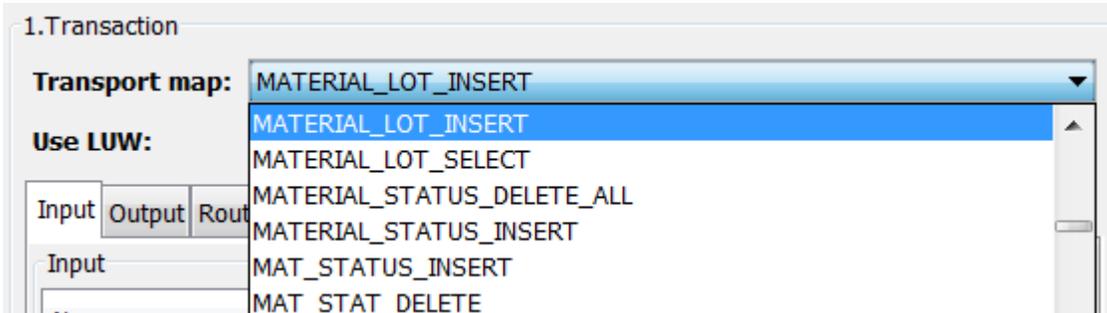
**Use LUW:** False

Input Output Routing Details

Input

Name	Logical	Count	Value	Type
col_REC_TS	TIMESTAMP	1		
col_LOT_ID	STRING(32)	1		
col_DESCRIPTION	STRING(256)	1		
col_DEFINITION_ID	STRING(32)	1		
col_STORAGE_LOCATION	STRING(32)	1		
col_STATUS	STRING(32)	1		
col_PUBLISHED_DATE	TIMESTAMP	1		
col_QUANTITY	INT4	1		
col_QUALITY_INSPECTION	INT4	1		
col_COMMENT	STRING(32)	1		

## Parameter description

Parameter	Description
<b>Transport Map</b>	<p>Use the <b>Transport Map</b> drop-down list to select a transport map defined on the node.</p>  <p>For example, <b>MATERIAL_LOT_INSERT</b> supports an SQL Insert operation.</p> <p>The <b>Input</b> tab becomes populated with the names of the map variables from the transport map <b>MATERIAL_LOT_INSERT</b></p>

Input				
Name	Logical	Count	Value	Type
col_REC_TS	TIMESTAMP	1		
col_LOT_ID	STRING(32)	1		
col_DESCRIPTION	STRING(256)	1		
col_DEFINITION_ID	STRING(32)	1		
col_STORAGE_LOCATION	STRING(32)	1		
col_STATUS	STRING(32)	1		
col_PUBLISHED_DATE	TIMESTAMP	1		
col_QUANTITY	INT4	1		
col_QUALITY_INSPECTION	INT4	1		
col_COMMENT	STRING(32)	1		

Notice the names **col\_LOT\_ID**, **col\_DESCRIPTION**, and so forth. These are the map variables that were defined when the transport map was defined.

**Use LUW**

This field is only displayed when a database Transport Map is selected in the **Transport Map** pick list. It allows you to indicate that the **Transaction** is part of a logical unit of work. The 36 character **LUWId** is obtained by executing the Logical Unit of Work Begin action.

The options are **True** or **False**

**False** - This is the default value. When selected, the **Input** tab does not display the **LUWId** parameter.

**True** - When selected, the **Input** tab displays the **LUWId** parameter

1.Transaction

**Transport map:** MATERIAL\_LOT\_INSERT

**Use LUW:** True

Input				
Name	Logical	Count	Value	Type
col_REC_TS	TIMESTAMP	1		
col_LOT_ID	STRING(32)	1		
col_DESCRIPTION	STRING(256)	1		
col_DEFINITION_ID	STRING(32)	1		
col_STORAGE_LOCATION	STRING(32)	1		
col_STATUS	STRING(32)	1		
col_PUBLISHED_DATE	TIMESTAMP	1		
col_QUANTITY	INT4	1		
col_QUALITY_INSPECTION	INT4	1		
col_COMMENT	STRING(32)	1		
LUWId	STRING(36)	1		

## Input tab

The **Input** tab is where you associate the variables from the trigger with the map variables defined in the transport map. You do so by selecting local, static, event, device or macro variables, or entering a constant in the **Value** column.

1.Transaction

**Transport map:** MATERIAL\_LOT\_INSERT

**Use LUW:** False

Input Output Routing Details

Input

Name	Logical	Count	Value	Type
col_REC_TS	TIMESTAMP	1		
col_LOT_ID	STRING(32)	1		
col_DESCRIPTION	STRING(256)	1		
col_DEFINITION_ID	STRING(32)	1		
col_STORAGE_LOCATION	STRING(32)	1		
col_STATUS	STRING(32)	1		
col_PUBLISHED_DATE	TIMESTAMP	1		
col_QUANTITY	INT4	1		
col_QUALITY_INSPECTION	INT4	1		
col_COMMENT	STRING(255)	1		

Parameter	Description
<b>Input Variables</b>	The parameters on the <b>Input</b> tab appear based on the definition of the transport map. For example, a transport map for a database insert operation would have map variables for each of the database table columns.

## Example Input tab

A completed **Input** tab with selected device variables, macros and constants might look like this:

1.Transaction

**Transport map:** MATERIAL\_LOT\_INSERT

**Use LUW:** False

Input Output Routing Details

Input

Name	Logical	Count	Value	Type
col_REC_TS	TIMESTAMP	1	Macros.\$EVENT_TIME	TIMESTAMP
col_LOT_ID	STRING(32)	1	Local CPU 1.D[100]	STRING(32)
col_DESCRIPTION	STRING(256)	1	Local CPU 1.D[150]	STRING(256)
col_DEFINITION_ID	STRING(32)	1	DEF100	CONSTANT
col_STORAGE_LOCATION	STRING(32)	1	Local CPU 1.D[500]	STRING(32)
col_STATUS	STRING(32)	1	Delivered	CONSTANT
col_PUBLISHED_DATE	TIMESTAMP	1	Macros.\$EVENT_TIME	TIMESTAMP
col_QUANTITY	INT4	1	100	CONSTANT
col_QUALITY_INSPECTION	INT4	1	200	CONSTANT
col_COMMENT	STRING(32)	1	CHECKED	CONSTANT

For this example, Transaction action, each time the trigger executes, the variables from the trigger are inserted into the database table identified in the MATERIAL\_LOT\_INSERT transport map.

## Output tab

The **Output** tab can contain the map variables that will be used as output from the transaction and additional output variables that are updated after the transport map executes. The additional output variables depend on the type of transport map that is being used.

For transactions that use database operations, the Transaction action's **Output** tab provides the **resultCount** , **resultStatus** and optionally **resultKey** output variables.

1.Transaction

**Transport map:** MATERIAL\_LOT\_INSERT

**Use LUW:** False

Input Output Routing Details

Output

Name	Logical	Count	Value	Type
resultStatus	INT4	1		
resultCount	INT4	1		

Output	Description
<b>Output variables</b>	
<b>resultStatus</b>	Provides a return value that indicates the success of the action. A 0 indicates the action was successful.
<b>resultCount</b>	Provides a return value that indicates the number of rows that were processed from the result of a database insert, update, select, and stored procedure operation.
<b>resultKey</b>	This INT8 field is only available for a database INSERT transaction action. It will contain the value of an autogenerated column after the transaction that uses the \$GENERATED macro has executed successfully.

### Example Output tab with output variables

The **Output** tab will also contain variables, as defined in the transport map:

1. Transaction

Transport Map: DB2sel01a

Input Output Routing Details

Output

Name	Logical	Count	Value	Type
log1	INT2	5	Local CPU 1.D[10]	INT2
log2	INT4	5		
log3	FLOAT4	5		
log4	FLOAT4	5		
log5	STRING(16)	5		
resultStatus	INT4	1		
resultCount	INT4	1		

Output map variables

If you had selected a transport map that was defined for a database select or stored procedure operation, the **Output** tab will contain the output variables that were defined on the transport map's **Output** tab. This means that the database data values will be output from the Transaction action and can be mapped to variables by selecting the variable in the **Value** column.

Another example shows an **Output** tab that contains an output variable and the **resultStatus** and **resultCount** variables associated with PLC device variables.

Output				
Name	Logical	Count	Value	Type
log1	INT2	5	Local CPU 1.D[10]	INT2
log2	INT4	5		
log3	FLOAT4	5		
log4	FLOAT4	5		
log5	STRING(16)	5		
resultStatus	INT4	1	Local CPU 1.D[7]	INT4
resultCount	INT4	1	Local CPU 1.D[8]	INT4

## Routing tab

The **Transaction** action will have a **On Result** row for **Store and Forward** that can be used for a **Transaction** action that references a transport map that references a transport that supports the store and forward function.

Routing	
On Result	Go to
Success	NEXT
Failure	UNDEFINED
Store and Forward	NEXT

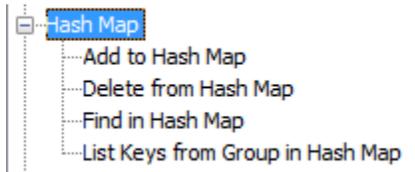
On Result	Description
Success	The route to take upon a successful completion of the action.
Failure	The route to take upon a failed completion of the action.
Store and Forward	The route to take if the result of the <b>Transaction</b> action was the writing of the transaction data to a store and forward queue. The transport associated with the Transaction action's transport map, must have store and forward capabilities and that capability must be enabled. For information about transports and how to

enable Store and Forward, see Transports.  
When using the Canvas Editor, the **Transaction** action will have an output port (route) for the Store and Forward result.

## IIoTA industrial IoT Platform: Hash Map

### Overview

The **Hash Map** category of actions allow you to store and retrieve the value of a variable based on groups and keys that are represented as strings. Optimized for performance, the hash map actions enable you to quickly create an internal object model for dynamic metadata that might be required by your application program.



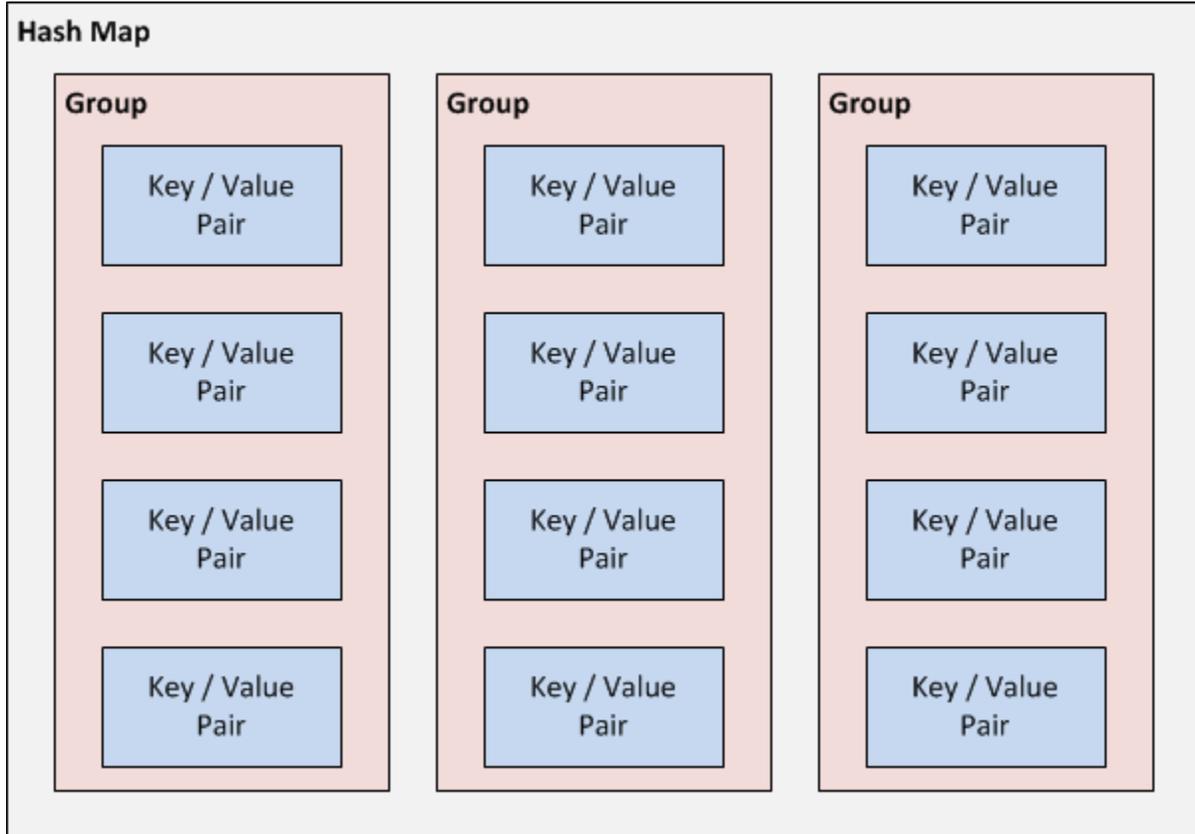
It is important to understand that the hash map was designed to be very fast and to use in-memory persistence that is lost when the node is restarted. For information regarding a storage solution that will persist across node restarts, see Local Database.

### Key features of the hash map

There is one global hash map within a given node. The hash map provides a group and key level as follows:

- **Group**  
There can be one or more groups within the node. The group is the first level of organization and can be any string.  
Use a group when you want to iterate through a list of entries on the hash map. Because groups do not contain values, they cannot be directly searched for or listed.
- **Key**  
The key is the second level of organization. Each key must be unique within its group.  
There can be one or more keys within each group.  
You can search for a single key, but the search will look for an exact match.

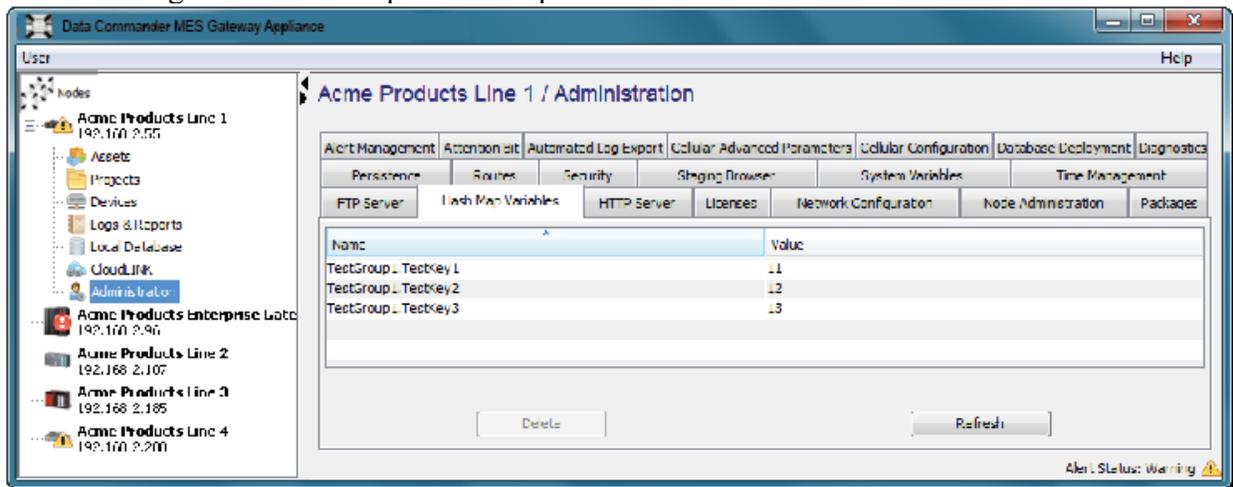
The following illustrates the components of a hash map:



## Viewing the hash map

Once data is generated by the trigger, you can view or delete the data contained in the hash map. The data becomes available from the Hash Map Variables tab.

The following shows an example hash map:



The **Hash Map** category provides these actions:

- Add to Hash Map
- Delete from Hash Map
- Find in Hash Map
- List Keys from Group in Hash Map

Related topics

Hash Map Variables

Local Database

## IIoTA industrial IoT Platform: Add to Hash Map

Adds a new value to the hash map based on a group and key.

The **Add to Hash Map** action adds a new value to the hash map based on a group and key. Once added, the variable can be retrieved with either the **Find in Hash Map** or **List Keys in Hash Map** actions.

Name	Logical	Count	Value	Type
Item	ANY	1		
group	ANY	1		
key	ANY	1		

### Parameter description

Parameter	Description
<b>Group</b>	A compound string used for the group that the key and value pair will be added to. With each \$ (variable) name you type, a corresponding row is added to the <b>Input</b> tab. For formatting options, see Using compound strings.
<b>Key</b>	A compound string used for the key that identifies the value in the hash map. With each \$ (variable) name you type, a corresponding row is added to the <b>Input</b> tab. For formatting options, see Using compound strings.

<b>Replace Existing</b>	When <b>True</b> , replace the current value of a key with the value supplied by the <b>Item</b> input parameter.
-------------------------	---

## Input tab

Parameter	Description
<b>Item</b>	The item to add to the hash map. This value can be any type of variable in the system but is stored internally as a string.
<b>other</b>	Depending on the values for the <b>Group</b> and <b>Key</b> parameters, additional input items may be available to correctly build the group and key strings. For more information, see Using compound strings.

## Routing tab

On Result	Description
<b>Success</b>	The action completed successfully.
<b>Failure</b>	The action encountered a failure.
<b>Already Exists</b>	The key is already present in the hash map, and the <b>Replace Existing</b> parameter is set to <b>False</b> .

Related topics  
Hash Map Variables

# IIoTA industrial IoT Platform: Delete from Hash Map

The **Delete from Hash Map** action removes a single item or an entire group from the hash map.

1.Delete from Hash Map

**Scope:** Key

**Group:** \$(group)

**Key:** \$(key)

Input Routing Details

Input

Name	Logical	Count	Value	Type
<b>group</b>	ANY	1		
<b>key</b>	ANY	1		

## Parameter description

Parameter	Description
<b>Scope</b>	The available options are Key and Group. When the <b>Scope</b> is set to <b>Group</b> , all keys in that group are deleted. When the <b>Scope</b> is set to <b>Key</b> , only the specific key is deleted from the specific group.
<b>Group</b>	A compound string for the group that the key and value pair will be deleted from. With each \$ (variable) name you type, a corresponding row is added to the <b>Input</b> tab. For formatting options, see Using compound strings.
<b>Key</b>	Required if shown. This parameter only appears when <b>Scope</b> is set to <b>Key</b> . The compound string for the key that identifies the item in the hash map. With each \$ (variable) name you type, a corresponding row is added to the <b>Input</b> tab. For formatting options, see Using compound strings.

## Input tab

Parameter	Description
<b>}other</b>	Depending on the values for the <b>Group</b> and <b>Key</b> parameters, additional input items may be available to correctly build the group and key strings. For more information, see Using compound strings.

## Routing tab

On Result	Description
-----------	-------------

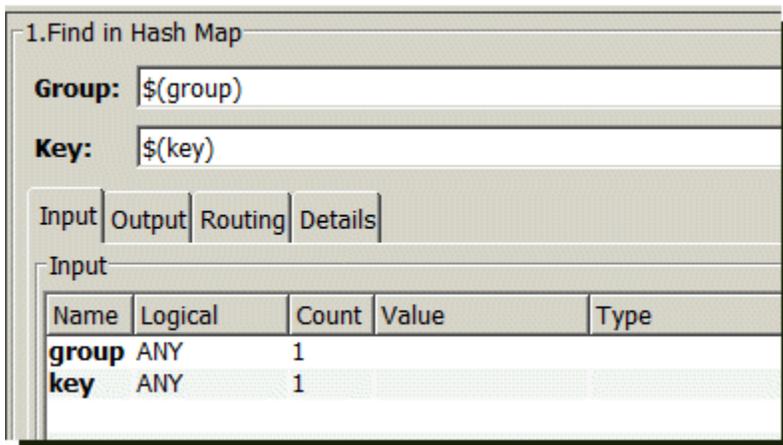
<b>Success</b>	The action completed successfully.
<b>Failure</b>	The action encountered a failure.
<b>Not Found</b>	The group was not found, if the scope is "Group"; or the key was not found, if the scope is "Key".

Related topics

Hash Map Variables

## IIoTA industrial IoT Platform: Find in Hash Map

The **Find in Hash Map** action searches the hash map for a particular group and key.



### Parameters descriptions

Parameter	Description
<b>Group</b>	A compound string used for the group. With each \$ (variable) name you type, a corresponding row is added to the <b>Input</b> tab. For formatting options, see Using compound strings.
<b>Key</b>	A compound string used for the key to identify the item in the hash map. With each \$ (variable) name you type, a corresponding row is added to the <b>Input</b> tab. For formatting options, see Using compound strings.

### Input tab

Parameter	Description
<b>other</b>	Depending on the values for the <b>Group</b> and <b>Key</b> parameters, additional input items may be available to correctly build the group and key strings. For more information, see Using compound strings.

## Output tab

Parameter	Description
<b>Item</b>	The value of the retrieved item. All items in the hash map are stored in String format, using a variable that has a different data type than the original data could result in incorrect data.

## Routing tab

On Result	Description
<b>Success</b>	The action completed successfully.
<b>Failure</b>	The action encountered a failure.
<b>Not Found</b>	The key is not found in the hash map.

Related topics

Hash Map Variables

# IIoTA industrial IoT Platform: List Keys from Group in Hash Map

The **List Keys from Group in Hash Map** action returns an array of keys that are present in a group within the hash map.

1.List Keys from Group in Hash Map

**Group:**

**Max Keys to List:**

Input | Output | Routing | Details

Input

Name	Logical	Count	Value	Type
<b>group</b>	ANY	1		

## Parameter description

Parameter	Description
<b>Group</b>	A compound string used for the group. With each \$ (variable) name you type, a corresponding row is added to the <b>Input</b> tab. For formatting options, see Using compound strings.
<b>Max Keys to List</b>	The maximum number of keys to return by the action.

## Input tab

Parameter	Description
<b>other</b>	Depending on the values for the <b>Key</b> parameter, additional input items may be available to correctly build the group and key strings. For more information, see Using compound strings.

## Output tab

Parameter	Description
<b>Keys</b>	Up to a number of keys in the group are returned in an Array of Strings. The max length of the string is 128.
<b>Count</b>	The number of keys returned by the action. The count should never exceed the value of the <b>Max Keys to List</b> parameter.

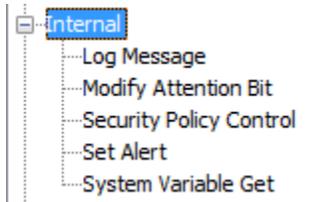
## Routing tab

On Result	Description
Success	The action completed successfully.
Failure	The action encountered a failure.
Not Found	No items were found for that group in the hash map.

Related topics  
Hash Map Variables

## IIoTA industrial IoT Platform: Internal

The **Internal** category provides actions for several features of the node.



The **Internal** category provides these actions:

- Log Message
- Modify Attention Bit
- Operating System
- Security Policy Control
- Set Alert
- System Variable Get

## IIoTA industrial IoT Platform: Log Message

The **Log Message** action writes a custom message to the **Exceptions** log with a specified message level.

System components also record messages in the **Exceptions** log.

1. Log Message

**Message Type:** EXCEPTION

**Message Level:** INFO

**Component:** User Trigger

**Message:** \$(Message)

Input   Routing   Details

Name	Logical	Count	Value	Type
Message	ANY	1		

## Parameter description

Parameter	Description
<b>Message Type</b>	Currently, this action only supports system exception messages logged to the <b>Exceptions</b> log.
<b>Message Level</b>	The severity of the message. The message level for user messages written to the Exceptions log can be used as appropriate. This is one of the parameters that can be used when filtering the display of messages using the Workbench. The options are:  <b>FATAL</b> <b>ERROR</b> <b>WARN</b> <b>INFO</b>
<b>Component</b>	The application defined component for this log message. This value will be shown under the <b>Component</b> column when the <b>Exceptions Log</b> is displayed. This is one of the parameters that can be used when filtering the display of messages using the Workbench.
<b>Message</b>	A compound string substitution variable for the log message. The substitution variables entered in this parameter are then defined in the corresponding parameters in the <b>Input</b> tab.

## Input tab

Parameter	Description
<b>Message</b>	The value for the substitution variables entered for the Message parameter. You can define them to reference any started device, constant, trigger macro, trigger local variable, trigger static variable, or event variable available from the pull-down list displayed for the <b>Value</b> cell for that row.

## Example Exceptions Log messages

An example of the Exceptions log with messages created using the **Log Message** action is shown below.

Note the Filter options for the Message Level (Type) and Component parameters:

Acme Products Line 3 / Logs & Reports

Overview Reports Mapping Logs Exceptions Log Audit Log

Filter (500 entries found)

From Date: All To Date: All

Message Type: INFO Component: [CloudLink Remote Trigger Testin  
[TCP Send Ascii]  
[User Trigger]

Message Contains:

Clear Filter Apply Filter

Date	Time	Type	Component	Message
2013-10-14	18:09:35,333	INFO	[CloudLink Remote Trigger Testing]	Response message back from remote trigger from 2.212
2013-10-14	18:09:17,233	INFO	[CloudLink Remote Trigger Testing]	Response message back from remote trigger on 2.96
2013-10-14	18:08:48,099	INFO	[TCP Send Ascii]	From TCP_SendASCII_Connnectionless_YesResp_NoHeader

## IIoTA industrial IoT Platform: Modify Attention Bit

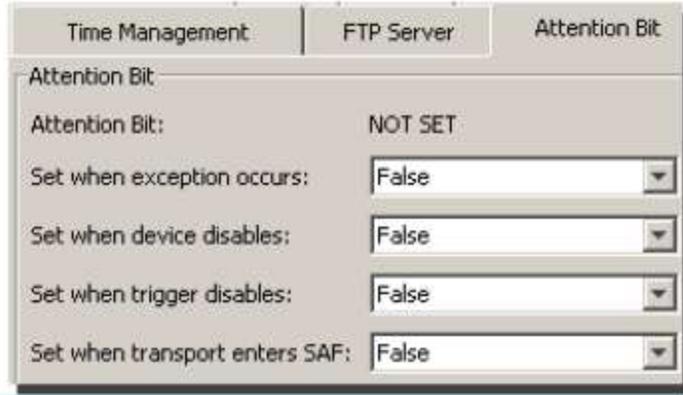
The **Modify Attention Bit** action works in conjunction with the attention bit feature. The attention bit feature provides an attention mechanism that indicates the node is in an exception state. The state is reflected as follows:

- The list of nodes in the left pane of the Workbench.



- In the audit logs of the node.

You can set the attention bit manually or programmatically by using the **Modify Attention Bit** action or the **Attention Bit** tab from the Workbench Administration window. The following shows the **Attention Bit** tab available from the Workbench Administration window.

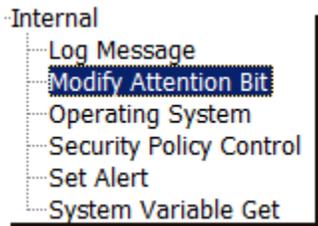


When a system condition is set to **True** from the **Attention Bit** tab, any trigger on the node that encounters the condition will cause the attention bit to be set. From the **Attention Bit** tab, you can also manually set or clear the attention bit.

Using the **Modify Attention Bit** action, you can also set or clear the attention bit when a condition is met for a specific trigger. This action is useful for tighter control over the type of application conditions that cause the bit to be set.

The **Modify Attention Bit** is available from the Trigger window as follows:

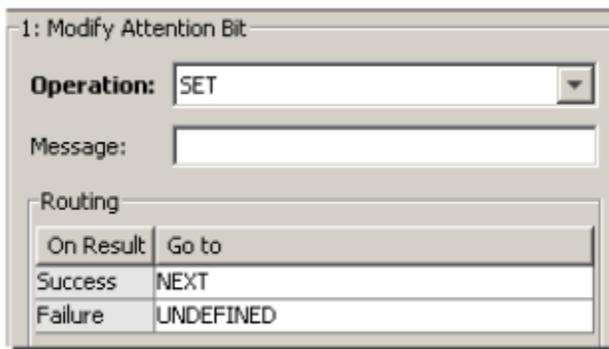
1. From the **Actions** tab, click **Add**.



The New Action window appears.

2. Expand **Internal**, select **Modify Attention Bit**, and then click **Add**.

The right pane changes to accommodate a **Modify Attention Bit** action.



## Parameter description

This section describes the parameter values that can be set for a **Modify Attention Bit** action. Parameter names in **bold** are required.

### Operation

There are two options available from the **Operation** drop-down list.

1: Modify Attention Bit

**Operation:** SET

Message: SET  
CLEAR

<b>Set</b>	Sets the attention bit whenever the trigger's predefined condition occurs.
<b>Clear</b>	Clears the attention bit on the node.

### Message

You can write descriptive text that will appear in the audit log when the attention bit is set or cleared.

1: Modify Attention Bit

**Operation:** SET

Message: Register exceeded temperature range

The message could be used to identify the cause of the exception or problem.

## Example trigger - set alert

You can create a trigger to alert a user that a transport has lost its connection to the enterprise system and any new transactions will be sent to a store and forward queue. This example describes how to create the trigger and set actions that alert the user of a problem.

### Transport map and store and forward

The example assumes that a transport and transport map were created. The transport associated with the transport map has store and forward enabled as shown.

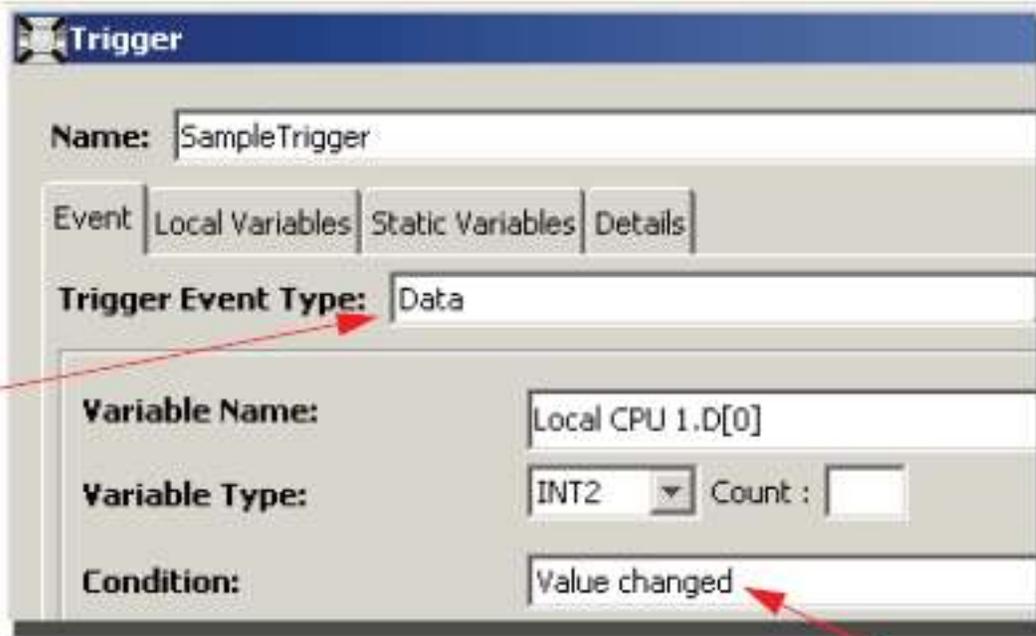
Store & Forward

TTL (sec): 86400 Max Storage (MB): 20

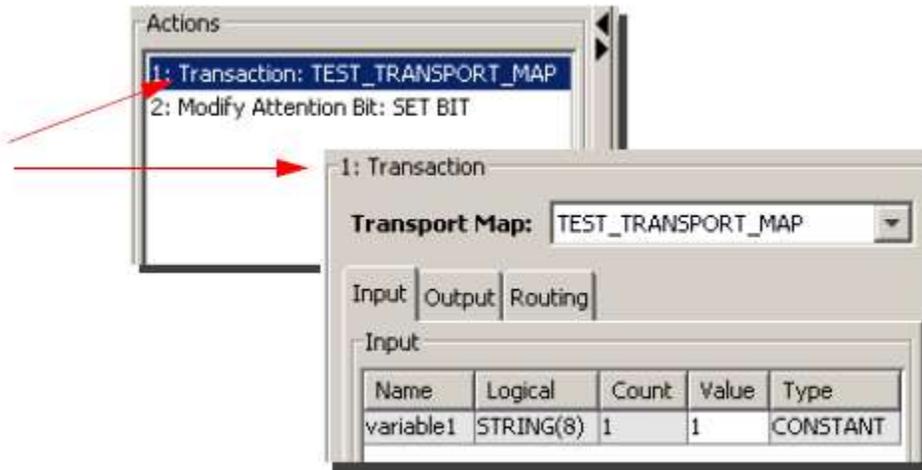
On overflow: Discard new message

### Sample trigger

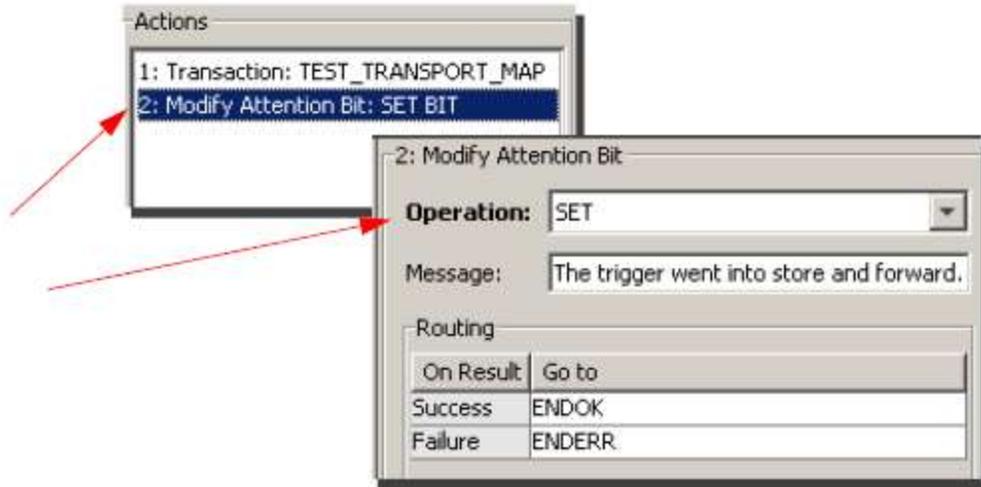
The following shows the upper portion of the sample Trigger window. A data event is selected as the trigger type. The event is based on a PLC device variable condition.



This trigger will execute whenever a PLC device variable's value changes, and then execute the first action within the trigger which is a **Transaction** action.

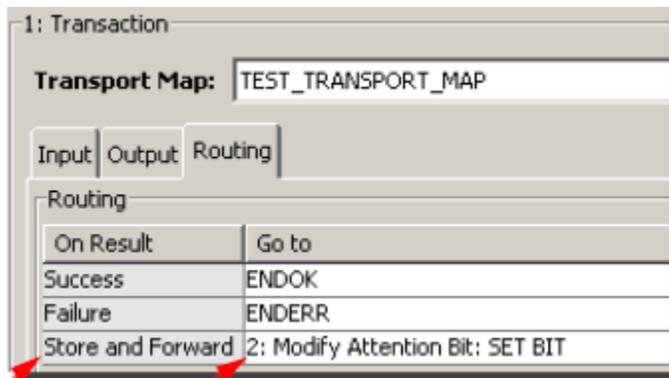


The second action is a **Modify Attention Bit**, with the **Operation** parameter set to **Set**.



### Sample Transaction action Routing tab

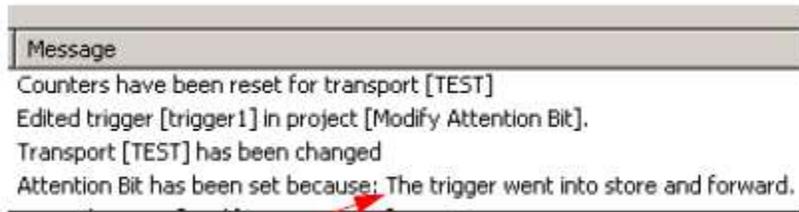
When the trigger executes, the transaction will go into store and forward, and then continue to the next route which will trigger the **Modify Attention Bit Set** operation.



The **Modify Attention Bit Set** operation turns on the attention bit on the node.



In addition, a message is recorded in the audit log that the trigger is in a store and forward mode.

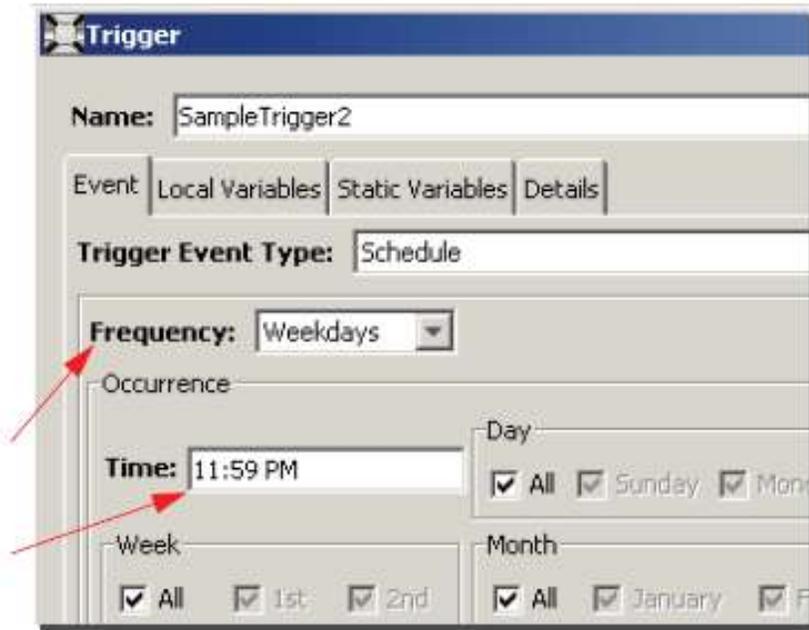


When the **Modify Attention Bit Set** operation completes, the action is considered successful and trigger execution ends.

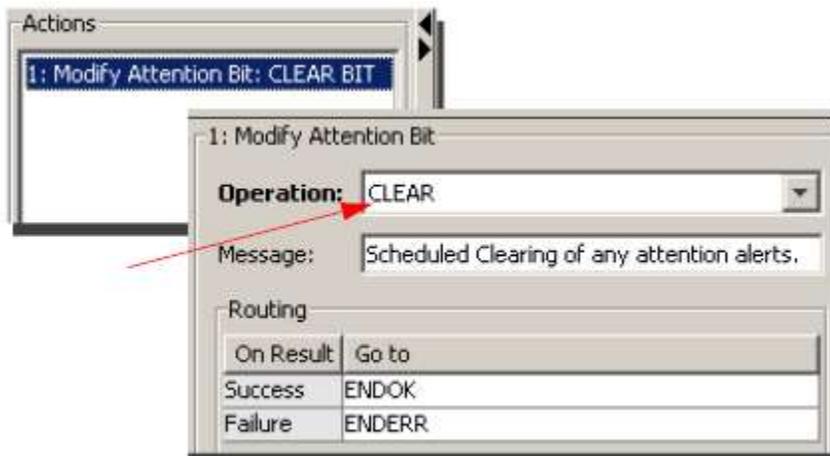
## Example trigger - clear alert

This example describes a trigger that will clear any alert that might have occurred during the day. The trigger is scheduled to execute every day at 11:59 P.M.

The following shows the upper portion of the sample Trigger window. A schedule event is selected as the trigger type.



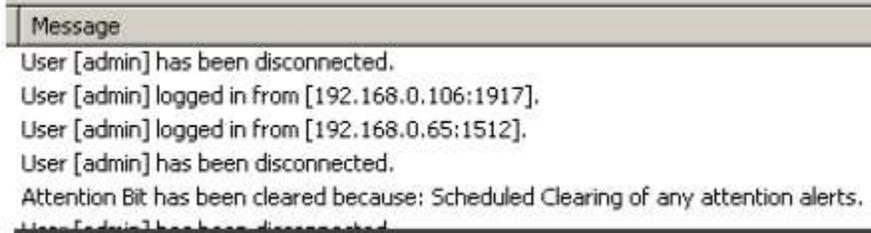
The trigger will execute every day at 11:59 P.M. At that time, the **Modify Attention Bit Clear** operation will execute.



The **Modify Attention Bit Clear** operation turns off the attention bit on the node.



In addition, a message is recorded in the audit log that the attention bit has been cleared.



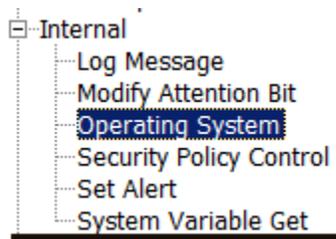
When the **Clear** operation completes, the action is considered successful and trigger execution ends.

## IIoTA industrial IoT Platform: Operating System

The **Operating System** action initiates a shutdown or restart of the runtime on the node.

**Warning:** Care should be taken when this action is selected as the execution of the **Operating System** action will immediately shutdown or restart the runtime on the node where this trigger and possibly other triggers are executing.

**You might not have this action:** You might not see this **Operating System** action on your Workbench as its availability is determined by the installed product.



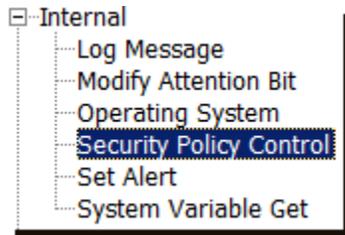
### Parameter description



Parameter	Description
Operation	The options are: <b>RESTART</b> - The node will shut down and then restart. <b>SHUTDOWN</b> - The node will be stopped.

## IIOA industrial IoT Platform: Security Policy Control

The **Security Policy Control** action enables or disables a policy defined within the current node.



You must have previously defined policies in order to use this action.

Suppose your organization has enabled a set of policies that control the way resources are used. For example, there might be a policy named *FloorDevices101* that allows certain users read access to all device variables on the node. Using the Workbench, you can create security policies from the Administration's **Security** tab.

Node Administration				
Licenses		Packages		Security
Notifications				
Policies		Users	Roles	
Name	State	Priority	Last State Change	
Default Policy	Enabled	50	2008-05-12 14:00	
FloorDevices101	Enabled	1	2008-05-12 14:00	
FloorDevices102	Disabled	50	2008-05-12 14:00	

Once the policy is created, you can assign a user access to that policy.

Name	Status	AAATech
Administrator	Enabled	No
admin	Enabled	No
it_user	Enabled	No
plant_user	Enabled	No
ZoeySmith	Enabled	Yes

The AAATech role has access to resources for FloorDevices101.

## Parameter description

1: Security Policy Control

**Operation:** Enable Policy

**Policy Name:** Default Policy

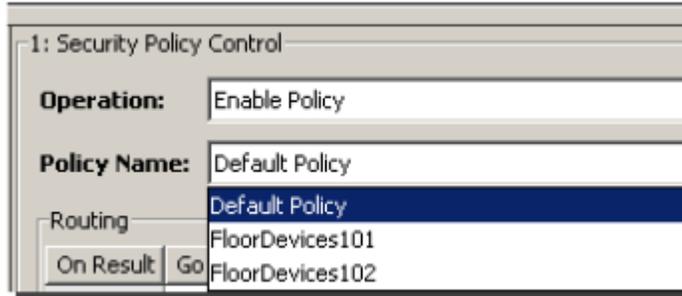
Routing

On Result	Go to
Success	NEXT
Failure	UNDEFINED

Parameter	Description																
<b>Operation</b>	<p>Options are <b>Enable Policy</b> or <b>Disable Policy</b>.</p> <p><b>Enable Policy</b> — Activates the selected policy when the trigger executes. It is assumed that the state of the policy is set to <b>Disable</b> on the Administration's <b>Security</b> tab.</p> <table border="1"> <thead> <tr> <th>Name</th> <th>State</th> <th>Priority</th> <th>Last State Change</th> </tr> </thead> <tbody> <tr> <td>Default Policy</td> <td>Enabled</td> <td>50</td> <td>2008-05-12 14:07</td> </tr> <tr> <td>FloorDevices101</td> <td>Enabled</td> <td>1</td> <td>2008-05-12 14:07</td> </tr> <tr> <td>FloorDevices102</td> <td>Disabled</td> <td>50</td> <td>2008-05-12 14:06</td> </tr> </tbody> </table> <p>If you set <b>Operation</b> to Enable Policy and the state of the policy is set to Enabled on the Security tab, the trigger will fail.</p> <p><b>Disable Policy</b> — Disables the selected policy when the trigger executes. It is assumed that the state of the policy was set to <b>Enable</b> on the Administration's <b>Security</b> tab. If you set <b>Operation</b> to Disable Policy and the state of the policy is set to Disabled on the Security tab, the trigger will fail.</p>	Name	State	Priority	Last State Change	Default Policy	Enabled	50	2008-05-12 14:07	FloorDevices101	Enabled	1	2008-05-12 14:07	FloorDevices102	Disabled	50	2008-05-12 14:06
Name	State	Priority	Last State Change														
Default Policy	Enabled	50	2008-05-12 14:07														
FloorDevices101	Enabled	1	2008-05-12 14:07														
FloorDevices102	Disabled	50	2008-05-12 14:06														

**Policy Name**

Use the **Policy** down-arrow to display a list of policies available for the current node.



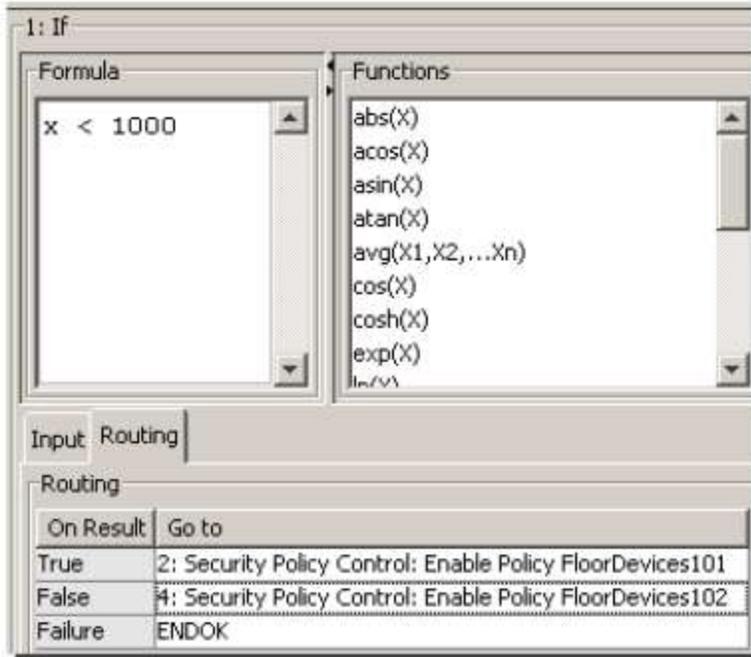
For this example, *FloorDevices101* and *FloorDevices102* were defined on the Administration's **Security** tab and became available for the **Security Policy Control** action. Because the **Enable Policy** is selected, upon execution of the trigger, enforcement of one of the selected policies will be enabled when the security action is performed.

**Example trigger - Security Policy Control**

This example describes a trigger that will enable the read/write access between two devices. For this example, two policies have been defined that control different devices on the floor. These policies are named *FloorDevices101* and *FloorDevices102*.

Node Administration   Licenses   Packages   Security   Notifications					
Policies   Users   Roles					
Name	State	Priority	Last State Change	Last Modif	
Default Policy	✓ Enabled	50	2008-06-30 13:...	2008-05-30	
FloorDevices101	✓ Enabled	1	2008-06-30 13:...	2008-06-30	
FloorDevices102	✗ Disabled	50	2008-06-30 13:...	2008-06-30	

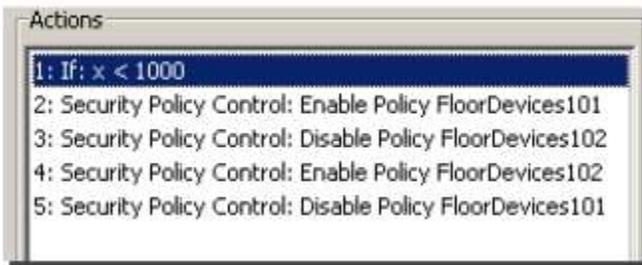
The trigger is configured to execute whenever the value of a device variable defined on a third device changes. The value of this variable will determine which of the two policies will be enabled and which is disabled. For the example, an **If** action is added to the trigger that will branch to enable the policies of either the **FloorDevices101** policy or the **FloorDevices102** policy. The following shows the **Routing** tab for the **If** action.



In addition to enabling one policy or another policy, the trigger will also disable the other policy. The following shows the **Routing** tab associated with Action 2: Security Policy Control: Enable Policy FloorDevices101.



Notice that if the enablement of the FloorDevices101 policy is successful, the next action performed will disable the FloorDevices102 policy. The following shows the full list of actions associated with this example trigger.

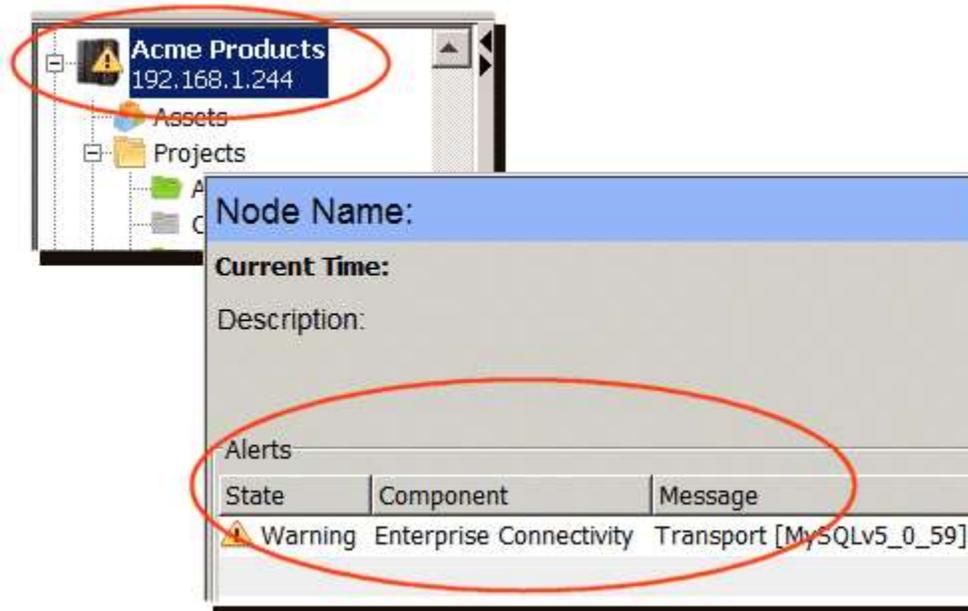


## IIoTA industrial IoT Platform: Set Alert

Use the **Set Alert** action to create a textual and graphical alert notification for the node. You can also use the **Set Alert** action to clear alert notifications.

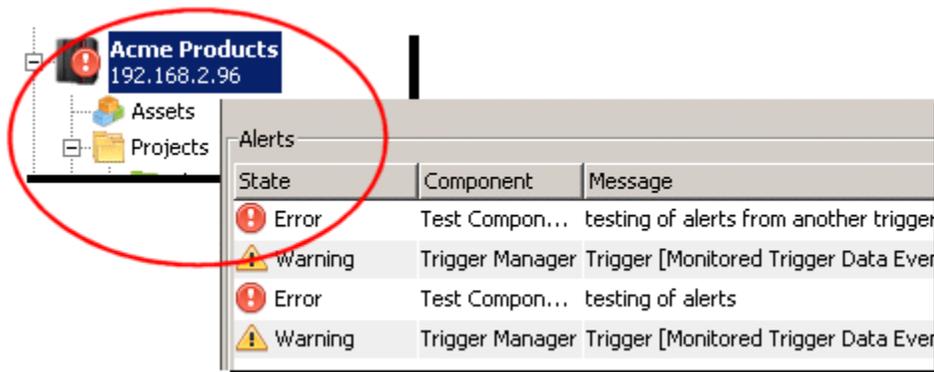
In addition to the user defined alerts created with the **Set Alert** action, the system will also set and clear system level alerts.

When an alert is set, it is displayed on the right pane of the main window for the node as follows:



Notice the alert severity level graphic also appears on the left pane on the node icon. When multiple alerts are set at the same time, the icon for the highest-level alert is displayed on the node icon. Error is the highest level and Debug is the lowest level.

In the example below, the Error level alert is the highest level and its icon is displayed in the left pane on the node icon.



## Creating an alert

The **Set Alert** action is available from the Trigger window as follows:

1. From the **Actions** tab, click **Add**.  
The New Action window appears.



2. Expand **Internal**, select **Set Alert**, and then click **Add**.  
The right pane changes to accommodate a **Set Alert** action.

Name	Logical	Count	Value
<b>Key</b>	ANY	1	
<b>Component</b>	ANY	1	

The **Status** parameter determines whether or not the **Message** parameter becomes available.

1.Set Alert

1.Set Alert

**Key:** \$(Key)

**Status:** Error

**Component:** Clear Status  
Error

**Message:** Warning  
Info  
Debug  
Dynamic

Input Routing

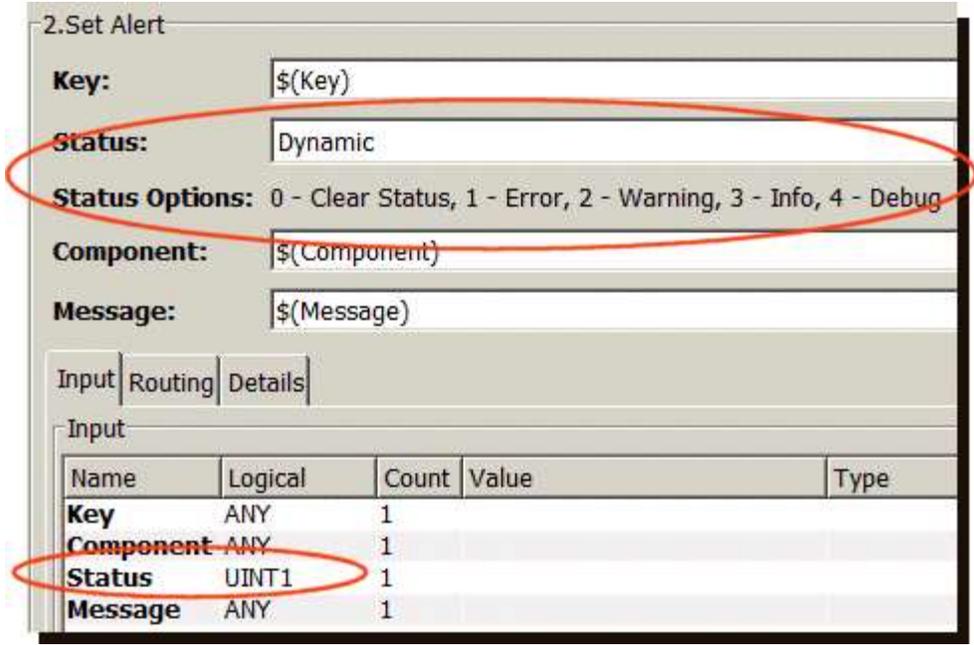
Input

Name	Logical	Count	Value	Type
<b>Key</b>	ANY	1		
<b>Component</b>	ANY	1		
<b>Message</b>	ANY	1		

## Property descriptions

This section describes the parameters that are available for **Set Alert**.

Parameter	Description
<b>Key</b>	Required. A compound string used to create a key that identifies the alert for the node.
<b>Status</b>	<p>The following are the severity levels you can set for the alert:</p> <p><b>Clear Status</b> — The alert indicated by the <b>Key</b> parameter is removed from the node.</p> <p><b>Error</b> — The alert will display an error icon.</p> <p><b>Warning</b> — The alert will display a warning icon.</p> <p><b>Info</b> — The alert will display an info icon.</p> <p><b>Debug</b> — The alert will display a debug icon.</p> <p><b>Dynamic</b> — The status value will be taken from the <b>Status</b> row of the <b>Input</b> tab. The following shows the <b>Set Alert</b> properties when <b>Dynamic</b> is selected:</p>



The **Status Options** parameter becomes available and provides a numeric value for each severity level. Use one of the values for the value within the **Status** variable.

The **Input** tab provides a **Status** variable where you can set either a constant (0, 1, 2, 3, or 4), or a device variable with a value of 0, 1, 2, 3, or 4).

<b>Component</b>	Required. A compound string used to identify a component that can be used to group alerts into categories for the node. With each \$ (variable) name you type, a corresponding row is added to the <b>Input</b> tab.
<b>Message</b>	Required. For all <b>Status</b> values (other than <b>Clear Status</b> ), a compound string used to define the alert message that will be displayed with the alert.

### Input tab

The following shows an **Input** tab when **Dynamic** is selected as the **Status**.

Name	Logical	Count	Value	Type
<b>Key</b>	ANY	1	RedCell	CONSTANT
<b>Component</b>	ANY	1	Cell5	CONSTANT
<b>Status</b>	UINT1	1	StaticVariables.dynamicstatus	INT2
<b>Message</b>	ANY	1	RedCellFailed	CONSTANT

For this example, the **Input** tab has the values in the Value column set.

The following describes the rows and column values on the **Input** tab for **Set Alert**.

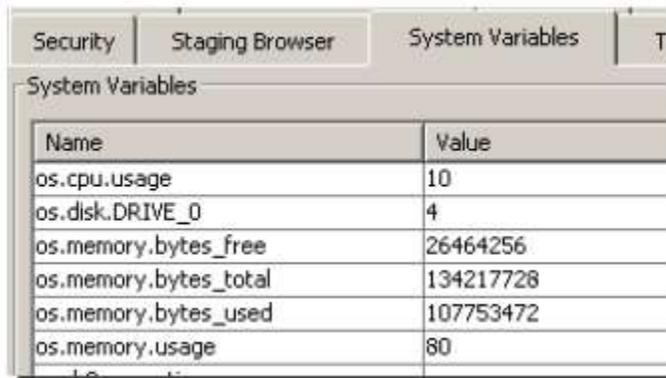
Parameter	Description
<b>Key</b>	The compound string for the key.
<b>Component</b>	The compound string for the component.
<b>Status</b>	When the Status parameter is set to <b>Dynamic</b> , the <b>Status</b> input row is available to set the value from a variable.
<b>Message</b>	The compound string for the alert message. For all <b>Status</b> values (other than <b>Clear Status</b> ), a compound string used to define the alert message that will be displayed with the alert.

Related topics

Using compound strings

## IIoTA industrial IoT Platform: System Variable Get

A system variable contains runtime information about the node. The Workbench provides a view to these internal variables via the **Administration** feature and the **System Variables** tab.



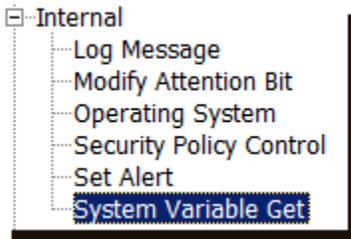
Name	Value
os.cpu.usage	10
os.disk.DRIVE_0	4
os.memory.bytes_free	26464256
os.memory.bytes_total	134217728
os.memory.bytes_used	107753472
os.memory.usage	80

When troubleshooting an issue, you might be asked to record the values of specific system variables. You can also use the **System Variable Get** action to obtain the value of these internal variables. You can use any trigger variable including static, local, and device variables.

### Adding a System Variable Get action

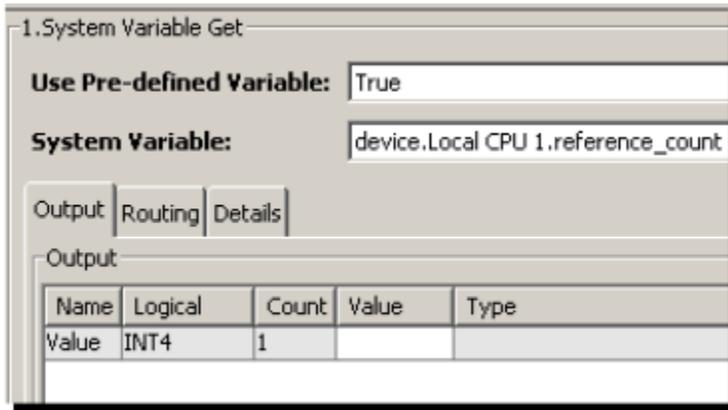
The **System Variable Get** action is available from the Trigger window.

1. From the **Actions** tab, click **Add**.



The New Action window appears.

2. Expand **Internal**, select **System Variable Get**, and then click **Add**.  
The right pane changes to accommodate a **System Variable Get** action.



## Parameter description

Parameter	Description
<b>Use Pre-defined Variable</b>	Options are:  <b>True</b> — Provides a list of pre-defined system variables in the <b>System Variable</b> parameter. <b>False</b> — Provides a single system variable name that you can change.
<b>System Variable</b>	The name of the system variable whose value you want to get.  When you select <b>True</b> from the <b>Use Pre-defined Variable</b> option, a list of system variables that match the internal variables available from the <b>System Variables</b> tab appears. When you select a variable from this list, its corresponding data type appears under the <b>Logical</b> column on the <b>Output</b> tab

When you select **False** from the **Use Pre-defined Variable** option, the name of a single internal variable appears. You change the name of the variable. When the name is changed or a new name is specified, the data type under the **Logical** column changes to **ANY**. This means the data type of the variable can be any type.

## Output tab

The **Output** tab is used to store the value of a system variable.

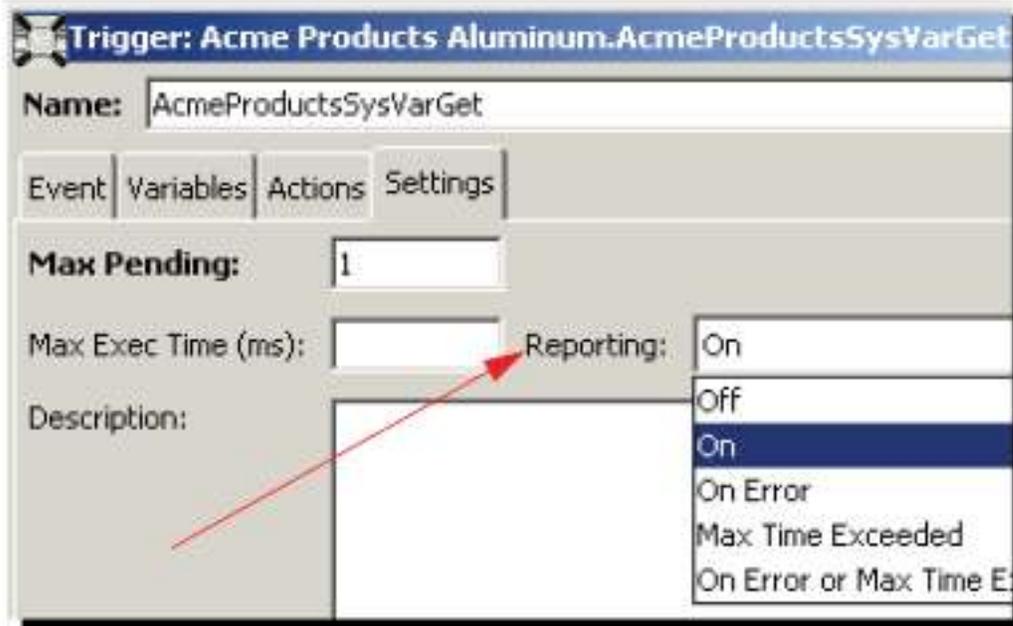
Parameter	Description
<b>Name</b>	This row is used to identify the name of the variable to place the result of the action. The name will change based on the system variable selected.
<b>Logical</b>	The column specifies the data type for the variable. Depending on the system variable selected, the result could be an INT4, INT8 or a String. For a variable selected using the <b>False</b> option, the result will be ANY.
<b>Value</b>	The variable where the value of the system variable is returned. You can select from a list of device variables, or a user defined local or static variable.
<b>Type</b>	When you specify <b>Value</b> , the data type of variable is automatically added to the <b>Type</b> column.

For more information about system variables, see System Variables.

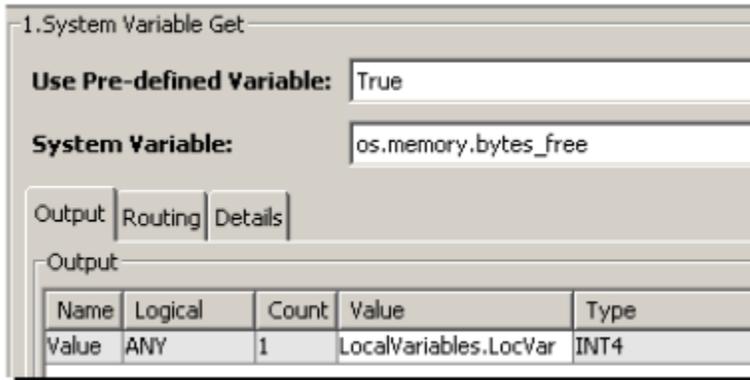
## Obtaining the data values for the system variable

You can create a trigger and then set the reporting option to on to obtain the input and output data values of the **System Variable Get** action in a report. You can export the report as a file to your local computer.

The following assumes that you have created a trigger and set the **Reporting** option to **On**.



For this example, an **On-Demand** trigger type is specified and the *LocVar* local variable was created. This local variable is where the value of the system variable will be returned. You have also added a **System Variable Get** action similar to the following:



Once the trigger is saved, follow these steps to obtain the input and output values for the system variable:

1. From the project tab, start the appropriate trigger.



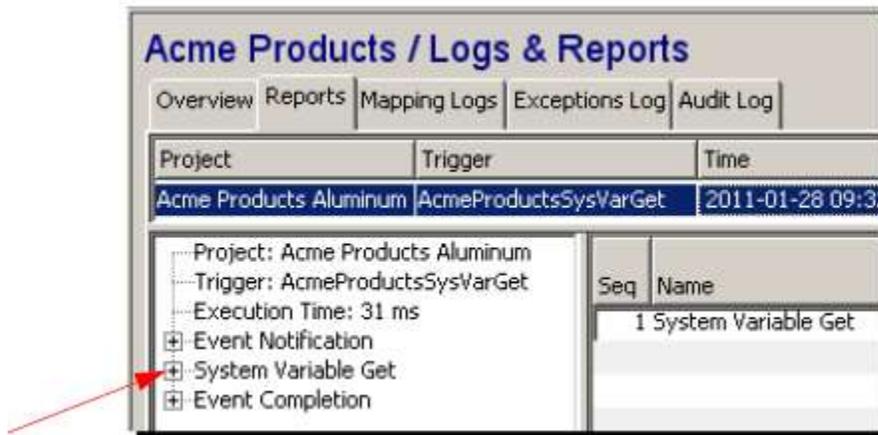
- When the state of the trigger is **Started**, right-click to display its pop-up, and then click **Fire Trigger**.

The **Successes** column will be incremented by one. The next step is to view the report that was generated when the trigger executed.

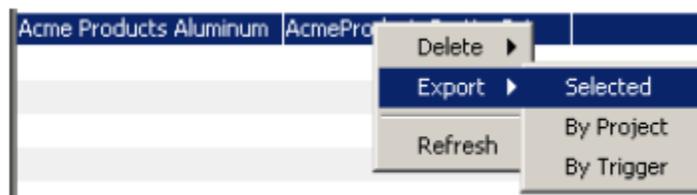
- From the Workbench left pane, expand **Logs & Reports**, and then click **Reports**. The Reports tab appears.



- Select the appropriate report. For this example, *AcmeProductsSysVarGet* trigger. Information about the trigger appears on the lower portion of the **Reports** tab.



- Expand the **System Variable Get** category to view the report. The next step is to export the report so that you have a record.

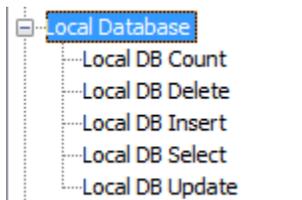


6. From the **Reports** tab, select the report you want to export, right-click to display its pop-up menu, select **Export**, and then select **Selected** (the report will contain information for the project and trigger on the selected row).  
The Export File Location window appears.
7. From the **Look in** box, change to the location you want to export the report to, and then click **Open**.  
The export location is added to the **Look in** box.
8. In the **File Name** box, enter a name for the report.
9. Select **Export**.
10. A message appears telling you the report was successfully export. Select **OK**.

The report is saved with a .txt extension. You can open the report with any text editor.

## IIoTA industrial IoT Platform: Local Database actions

The **Local Database** category provides actions to access the local database feature in the node.



The Local Database feature on a node can be used for many purposes, including:

- Temporarily storing data before forwarding to an enterprise database
- Look up table information.

The Local Database actions and the **Transaction** action provide two paths to the feature.

**Note:** The Transaction Server's support of Local Database tables is restricted to disk-based tables. Local Database tables created in memory are not supported by transport maps and the Transaction action.

The node's Local Database feature icon provides the functions to define and manage the Local Database tables.

The **Local Database** category provides these actions:

- Local DB Count
- Local DB Delete
- Local DB Insert
- Local DB Select
  - Example using Local DB Select action
- Local DB Update

Related topics

Local Database

Transaction

## IIoTA industrial IoT Platform: Local DB Count

The **Local DB Count** action provides a count of rows for a Local Database table. An optional Where clause can be used.

4. Local DB Count

**Table:** LocalDBTable

**WHERE:** c\_integer > \$(min) or c\_real < 0

Input Output Routing Details

Input

Name	Logical	Count	Value	Type
min	ANY	1		

## Parameter descriptions

Parameter	Description
<b>Table</b>	The name of the local database table for the count operation. The table name will be available from a drop-down list.
<b>Where</b>	<p>A Where clause can be used to restrict the rows counted.</p> <p>Both constants and substitution variables can be used in the Where clause.</p> <p>To construct a Where clause, use an operator (=, !=, &gt;, &gt;=, &lt;, &lt;=, like, is null, is not null ) to relate the column to either a constant or a substitution variable. Each of these operators can be combined with other operators using an <b>And</b> or <b>Or</b> statement.</p> <p>To use a substitution variable, insert \$(x) where x is a variable that will be associated to a variable on the <b>Input</b> tab (see the <b>Input</b> tab below).</p> <p><b>Note:</b> For columns of type TEXT, the constant or substitution variable must be enclosed in double quotes. For example:</p> <p>C01 = "JohnDoe" The constant is enclosed in double quotes.</p> <p>C01 = "\$(test)" The substitution variable is enclosed in double quotes. The variable (test in this example) will be added to the Input tab and can then be mapped to a STRING variable (or a constant).</p> <p>The Where clause builder, accessed by selecting the icon , can be used to assist in building the Where clause.</p> <p>For more information on the Where clause builder, see Local DB Select.</p>

## Input tab

Optional. The **Input** tab will only appear if a Where clause is specified and the Where clause contains a logical variable (for example, \$(VariableName)).

There will be as many rows in the **Input** tab as there are logical variables in the Where clause.

Parameter	Description
<b>Logical Variables from the Where clause</b>	<p>Required. The variable whose value is to be substituted in the Where clause logical variable.</p> <p>This variable can be any type of variable in the system.</p>

## Output tab

Parameter	Description
<b>Result Count</b>	The number of rows return by the count, based on the Where clause.
<b>Error Message</b>	Optional. Used to provide information if the count fails.

Related topics

Local Database

Transaction

## IIoTA industrial IoT Platform: Local DB Delete

The **Local DB Delete** action deletes rows from a local database table. An optional Where clause can be used.

2. Local DB Delete

**Table:** LocalDBTable

**WHERE:** c\_integer > \$(min) or c\_real < 0

Input Output Routing Details

Input

Name	Logical	Count	Value	Type
min	ANY	1		

## Parameter descriptions

Parameter	Description
<b>Table</b>	The name of the local database table for the delete operation. The table name will be available from a drop-down list.
<b>Where</b>	<p>A Where clause can be used to restrict the rows to be deleted.</p> <p>Both constants and substitution variables can be used in the Where clause.</p> <p>To construct a Where clause, use an operator (=, !=, &gt;, &gt;=, &lt;, &lt;=, like, is null, is not null ) to relate the column to either a constant or a substitution variable. Each of these operators can be combined with other operators using an <b>And</b> or <b>Or</b> statement.</p> <p>To use a substitution variable, insert \$(x) where x is a variable that will be associated to a variable on the <b>Input</b> tab (see the <b>Input</b> tab below).</p> <p><b>Note:</b> For columns of type TEXT, the constant or substitution variable must be enclosed in double quotes. For example:</p> <p>C01 = "JohnDoe" The constant is enclosed in double quotes.</p> <p>C01 = "\$(test)" The substitution variable is enclosed in double quotes. The variable (test in this example) will be added to the Input tab and can then be mapped to a STRING variable (or a constant).</p> <p>The Where clause builder, accessed by selecting the icon , can be used to assist in building the Where clause.</p> <p>For more information on the Where clause builder, see Local DB Select</p>

## Input tab

Optional. The **Input** tab will only appear when a Where clause is specified and the Where clause contains a logical variable (for example, \$(Variablename)).

There will be as many rows on the **Input** tab as there are logical variables in the Where clause.

Parameter	Description
<b>Logical Variables from the Where clause</b>	The variable whose value is to be substituted in the Where clause logical variable. This variable can be any type of variable in the system.

## Output tab

Parameter	Description
<b>Rows Deleted</b>	The actual number of rows deleted from the table.
<b>Error Message</b>	Optional. Used to provide information if the delete fails.

Related topics  
 Local Database  
 Transaction

## IIoTA industrial IoT Platform: Local DB Insert

The **Local DB Insert** action inserts a row into a local database table.

3. Local DB Insert

**Table Name:** LocalDBTable

Input Output Routing Details

Input

Name	Logical	Count	Value	Type
c_text	STRING(32)	1		
c_integer	INT8	1		
c_real	FLOAT8	1		
c_numeric	INT8	1		

## Parameter description

Parameter	Description
<b>Table</b>	The name of the local database table selected to insert rows. The table name will be available from a drop-down list.

## Input tab

The **Input** tab will have one row per column in the database table, where each row is identified by the table column name.

Parameter	Description
<b>Table Column Name</b>	Optional. The table column name: The column can be assigned any variable.

## Output tab

Parameter	Description
<b>Error Message</b>	Optional. Used to provide information if the query fails.

Related topics

Local Database

Transaction

## IIoTA industrial IoT Platform: Local DB Select

The **Local DB Select** action retrieves rows from a Local Database table. An optional Where clause and Group By clause can be used.

The **SELECT** parameter can be an asterisk ( \* ), which is the default, to indicate select all columns, or the **SELECT** parameter can be any valid SQL select statement.

When all columns are selected, the Logical type of the Output variables will be the data type from the table definition.

When the **Return Result as Array** parameter is set to **True**, the Output tab variables are treated as arrays, the Output **Row Index** variable is not used, and the **SELECT** parameter is not available (it is handled as a **SELECT \***).

1. Local DB Select

**SELECT:**  ...

**FROM:**

**WHERE:**  ...

**GROUP BY:**  ...

**Max Rows:**

**Return Result as Array:**

Output Routing Details

Output

Name	Logical	Count	Value	Type
C04	FLOAT8	1		
C03	INT8	1		
C02	INT8	1		
C01	STRING(32)	1		
Error Message	STRING(128)	1		
Rows Selected	INT4	1		
Row Index	INT4	1		

?

When a select statement other than the default SELECT \* is used, the Logical type of the Output variables will be ANY.

1. Local DB Select

**SELECT:**

**FROM:**

**WHERE:**

**GROUP BY:**

**Max Rows:**

**Return Result as Array:**

Output

Name	Logical	Count	Value	Type
C01	ANY	1		
Column_2	ANY	1		
Error Message	STRING(128)	1		
Rows Selected	INT4	1		
Row Index	INT4	1		

## Parameter description

You must supply valid SQL statements within the **Select** and **Where** clause. The use of String Format Specifiers may cause runtime problems.

Parameter	Description
SELECT	<p>The Select clause is used to determine the Action's <b>Output</b>. The <b>FROM</b> parameter (the Local Database table) must be selected and contain data for validation to occur.</p> <p>This can be an asterisk ( * ), to indicate all columns. An empty (blank) select parameter is treated the same as an asterisk.</p> <p>A valid SQL statement, including alias names for columns</p>

	<p>All SQLite aggregate functions are supported, see the <a href="#">SQLite Aggregate Functions List</a>.</p> <p>The select clause builder, accessed by selecting the icon , can be used to assist in building the select clause.</p>
<b>FROM</b>	<p>The name of the Local Database table to select the rows from. The table name will be available from a drop-down list.</p>
<b>WHERE</b>	<p>A Where clause can be used to restrict the rows selected.</p> <p>Both constants and substitution variables can be used in the Where clause.</p> <p>To construct a Where clause, use an operator (=, !=, &gt;, &gt;=, &lt;, &lt;=, like, is null, is not null ) to relate the column to either a constant or a substitution variable.</p> <p>Each of these operators can be combined with other operators using an <b>And</b> or <b>Or</b> statement.</p> <p>To use a substitution variable, insert \$(x) where x is a variable that will be associated to a variable on the <b>Input</b> tab (see the <b>Input</b> tab below).</p> <p><b>Note:</b> For columns of type TEXT, the constant or substitution variable must be enclosed in double quotes. For example:</p> <p>C01 = "JohnDoe"          The constant is enclosed in double quotes.          C01 = "\$(test)"          The substitution variable is enclosed in double quotes. The variable (test in this example) will be added to the Input tab and can then be mapped to a STRING variable (or a constant).</p> <p>The where clause builder, accessed by selecting the icon , can be used to assist in building the where clause.</p>
<b>GROUP BY</b>	<p>A Group by clause can be used to to collect data across multiple rows and group the results by one or more columns.</p> <p>The group by clause builder, accessed by selecting the icon , can be used to assist in building the group by clause.</p>
<b>Max Rows</b>	<p>The maximum number of rows the query is expected to return. The query can return less rows than the <b>Max Rows</b> specified.</p> <p>However, if the query returns more rows than specified in <b>Max Rows</b>, only the number of rows specified by <b>Max Rows</b> will be actually written to the output variables.</p> <p>The other rows will be discarded.</p>

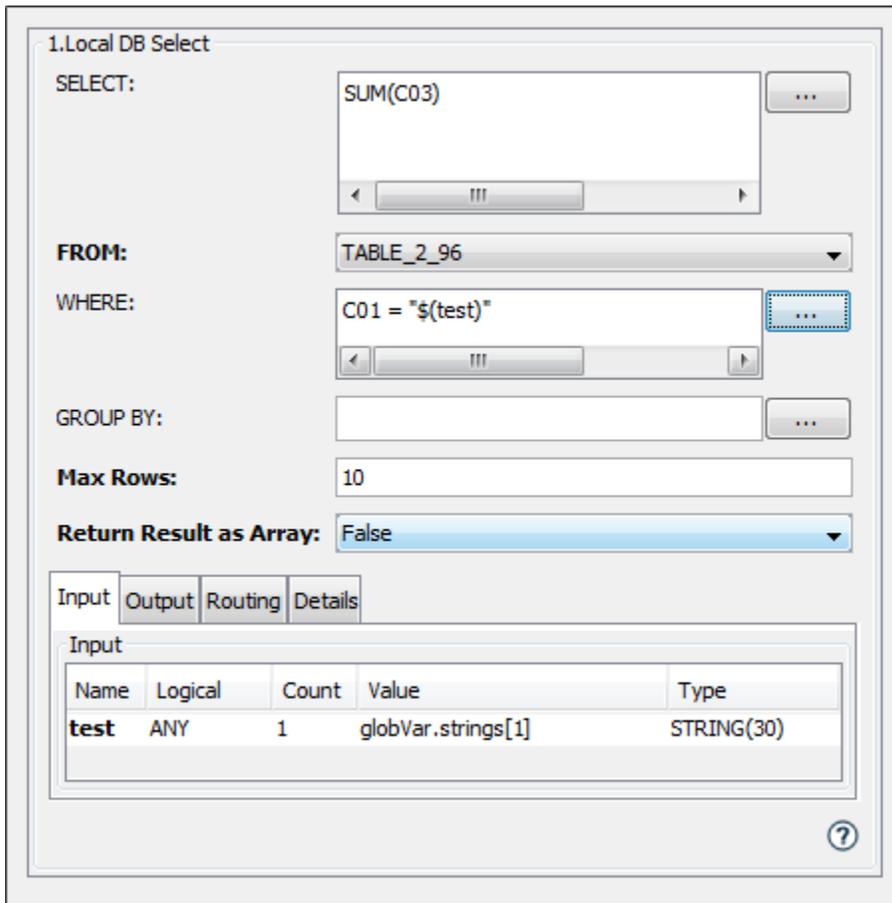
**Return Result as Array**

An option to return the results as an array.  
**True** - Returns the results as an array, up to the **Max Rows** value. The Output tab variables will have a **Count** parameter equal to the **Max Rows** and the output variables will be treated as arrays. The **Row Index** variable is not used.  
**False** - Returns the results one row at a time. The Output tab variable **Row Index** is returned as an index variable. Routing is controlled by the Routing tab **Next Row** parameter.

**Select clause builder**

The SELECT parameter can be an asterisk ( \* ), which is the default, to indicate select all columns.

An example of a Local DB Select action with the SELECT and WHERE parameters specified is:



The select clause builder can be used to assist in building the select clause.

When the **Return Result as Array** parameter is set to **True**, the SELECT parameter is not available (it is handled as a SELECT \*).

The select clause builder allows selections of columns, aggregate functions and alias names. For example:

Table: TABLE\_2\_96  
SELECT SUM(C03)

Column	Function	Alias
C03	SUM	

Buttons: Add All, Add, Delete, Up, Down, OK, Cancel

- The output of the select clause builder is placed in the SELECT parameter and can be manually modified.
- The Add All, Add, Delete, Up and Down buttons handle the placement of the table column names in the select clause.
- The optional aggregate functions (none, AVG, Count, MAX, MIN, SUM) are selected in the **Function** pull down column.
- An alias name can be assigned to each output of the select clause in the **Alias** column, and will be used as the variable name on the Output tab.
- The where clause (see next section) can be used to restrict the rows selected.
- *Example select clauses:*
- Examples of the select clause as it is displayed in the select clause builder include:
  - SELECT \*  
Select all rows, return the value of all columns.
  - SELECT C03  
Select all rows, return the value in the column named C03.
  - SELECT SUM(C03)  
Select all rows, return the Sum of the values of the column named C03.
  - SELECT SUM(C03) AS C03\_sum  
Select all rows, return the Sum of the values of the column named C03 in an output variable named C03\_sum.

- `SELECT SUM(C03) AS C03_sum, AVG(C03) AS C03_avg, MAX(C03) AS C03_max`  
Similar to the previous example, returns the Sum, Avg and Max of the values of the column named C03 in the output variables: C03\_sum, C03\_avg, C03\_max.

## Where clause builder

The where clause builder can be used to assist in building the where clause. For example:

The output of the where clause builder is placed in the WHERE parameter and can be manually modified.

- The where clause builder displays all columns from the Local DB table specified in the **FROM** parameter.
- To build a where clause: Select a column name from the **Database Columns** list, select an operator from the **Operators** list, and select **And** or **Or** (used when adding multiple columns to the where clause). Then select the **Add** button.
- The Move Up, Move Down and Remove buttons handle the placement of the column names in the where clause.
- Substitution variables can be used to indicate a variable should be specified in the input tab
- *Example where clauses:*
- `WHERE C01 = "$(test)"`  
Restrict the select rows to those with a value for the column named C01 to be equal to the value of the substitution variable **test**.  
The substitution variable **test** will be inserted as a row in the input tab. It can then

reference any available variable in the system (device variable, trigger variable, constant, etc.).

- WHERE C02 >= 10

Restrict the selected rows to those with a value for the column named C02 to be greater than or equal to 10.

- Where C01 = "\$ (test)" AND C02 >= 10

Restrict the select rows to those that meet the criteria for C01 and C02.

- **Note:** For columns of type TEXT, the constant or substitution variable must be enclosed in double quotes. For example:

- C01 = "JohnDoe"

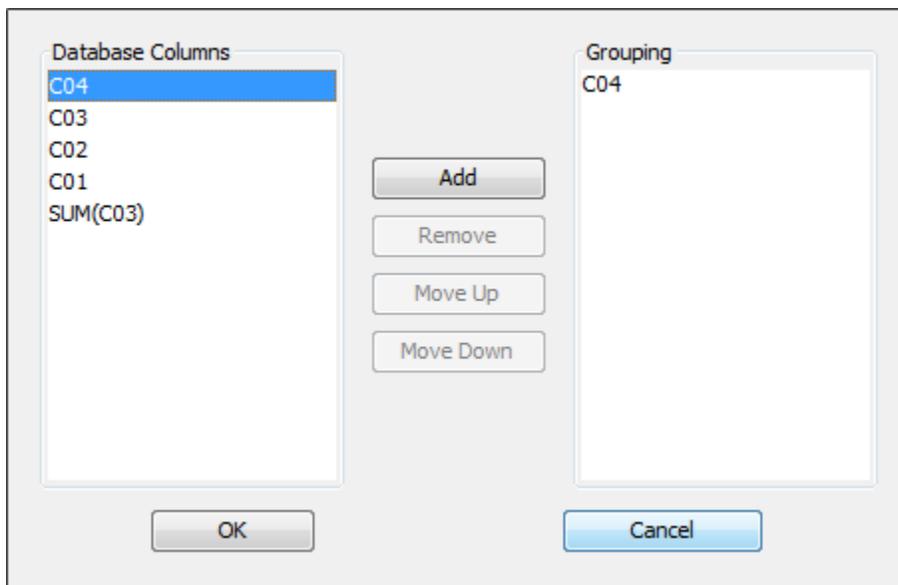
The constant is enclosed in double quotes.

- C01 = "\$ (test)"

The substitution variable is enclosed in double quotes. The variable (test in this example) will be added to the Input tab and can then be mapped to a STRING variable (or a constant).

## Group by clause builder

The group by clause builder can be used to assist in building the group by clause. For example:



The output of the group by clause builder is placed in the GROUP BY parameter and can be manually modified.

## Select and Where clause changes

After using the select and where clause builders, manual changes to the SELECT and WHERE parameters may result in invalid SQL statements. Validation of the parameters will be attempted and a warning displayed if there is a problem parsing the statement.

## Input tab

Optional. The **Input** tab will only appear when a Where clause is specified and the Where clause contains a substitution variable (for example, \$(test)).

There will be as many rows in the **Input** tab as there are substitution variables in the Where clause.

Parameter	Description
<b>Logical Variables from the Where clause</b>	Required. The variable whose value is to be substituted in the Where clause statement. This variable can then reference any available variable in the system (device variable, trigger variable, constant, etc.)

## Output tab

The **Output** tab will have one row per returned variable (such as a column name), based on the select clause.

Parameter	Description
<b>Output Name</b>	The returned variable. This could be a column name, the result of an aggregate function, or an alias name. The value of the returned variable is written to the variable specified in the Value cell.  <b>Important:</b> if a returned value is a null and that value is mapped to a numeric variable, the trigger will fail because a null cannot be written to a numeric field.
<b>Error Message</b>	Optional. Used to provide information if the SQL query fails.
<b>Rows Selected</b>	The number of rows returned by a SQL query operation.
<b>Row Index</b>	The current row number when iterating through the rows returned by the Select operation, when the <b>Return Result as Array</b> parameter is set to <b>False</b> . When iterating through the rows returned by the Local DB Select action, <b>Row Index</b> is incremented by one for each row processed. Using the <b>Next Row</b> route from the <b>Routing</b> tab, it is possible to iterate through each row returned by the Local DB Select action. For each row processed, <b>Row Index</b> is incremented by 1. For more information, see Example using Local DB Select action.

### Next Row Routing

When the **Return Result as Array** is set to **False**, the returned rows must be processed in a loop. The actions in the row processing loop are indicated by the execution path starting at the **Next Row** routing option. For more information, see Example using Local DB Select action.

## Routing tab

On Result	Description
Success	The action completed successfully.
Failure	The action encountered a failure.
Next Row	Fetch the next row of values retrieved from the select statement when the <b>Return Result as Array</b> parameter is set to <b>False</b> .

### Related topics

Local Database

Transaction

Local DB Export

# IIoTA industrial IoT Platform: Example using Local DB Select action

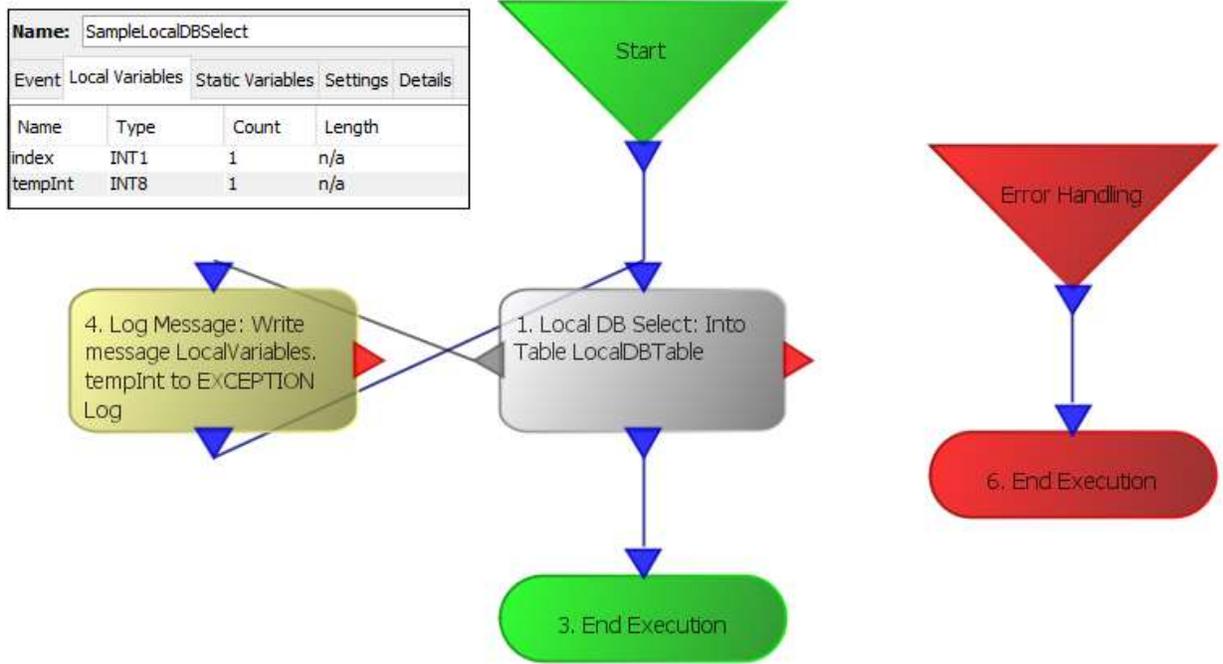
## Assumptions

The following is assumed:

- You know how to create a trigger and add actions to that trigger using the Canvas Editor. For more information on the Canvas Editor, see Using the Canvas Editor.
- You are familiar with Structured Query Language (SQL) and relational databases.

## Retrieving data from a local database table using Local DB Select

This example describes a trigger that when executed retrieves rows from a local database table using the **Local DB Select** action. The following shows a portion of the sample trigger:



The sample trigger has a **Local DB Select** action and **Log Message** action.

Using the **Local DB Select** action, each column can be mapped to a variable (including trigger variables and device variables).

1. Local DB Select

SELECT: \*

FROM: LocalDBTable

WHERE: C\_INTERGER < 10000 or C\_INTERGER > \$(max)

GROUP BY:

Max Rows: 10

Return Result as Array: False

Input Output Routing Details

Output

Name	Logical	Count	Value	Type
C_REAL	FLOAT8	1		
C_INTERGER	INT8	1	LocalVariables.index	INT8
C_NUMERIC	INT8	1		
C_TEXT	STRING(32)	1		
Error Message	STRING(128)	1		
Rows Selected	INT4	1		
Row Index	INT4	1		

In order to process the data retrieved, the **Local DB Select** action must iterate through the rows returned by the SQL Select operation. You can process the column values for each row using the **Next Row** option on the **Local DB Select** action **Routing** tab. The last action on the **Next Row** route must be routed to the **Local DB Select** action.

1. Local DB Select

SELECT: \*

FROM: LocalDBTable

WHERE: C\_INTERGER < 10000 or C\_INTERGER > \$(max)

GROUP BY:

Max Rows: 10

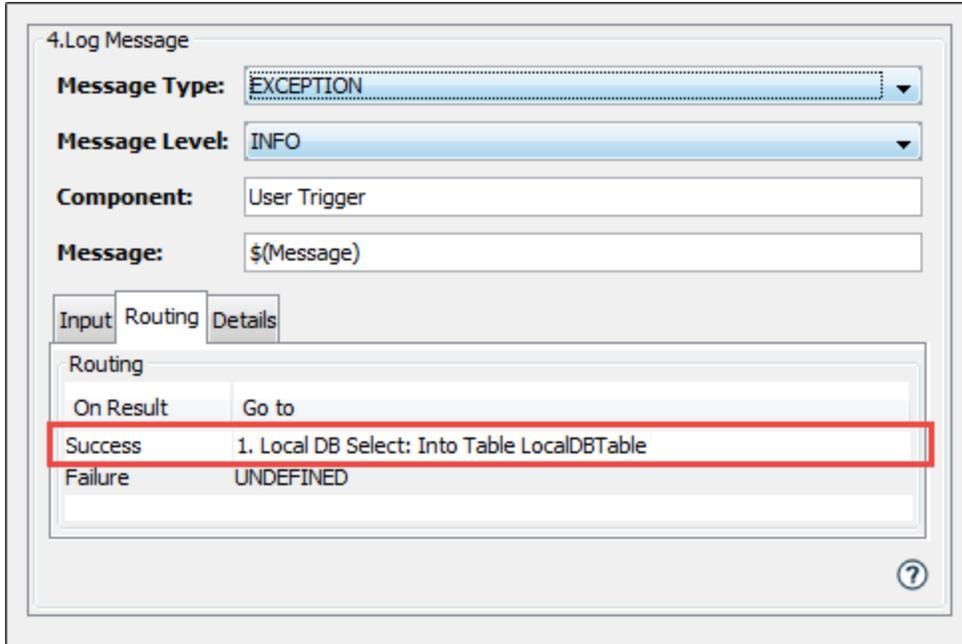
Return Result as Array: False

Input Output Routing Details

Routing	
On Result	Go to
Success	3. End Execution (Success)
Failure	UNDEFINED
Next Row	4. Log Message: Write message LocalVariables.index to EXCEPTION Log

?

For this example, there is only one action to execute from the **Next Row** route. The **Log Message** action **Success** route is returning to the **Local DB Select** action.

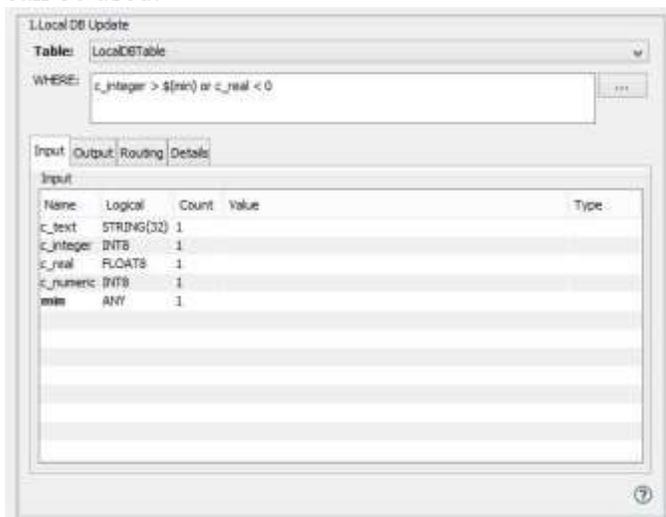


When there are no more rows to process and no errors encountered, the logic flow from the **Local DB Select** action will follow the **Success** route. The number of times the **Next Row** route will be executed is given by the values returned in **Rows Selected**. The **Row Index** keeps track of the current row being processed.

Related topics  
 Local Database  
 Transaction

## IIOA industrial IoT Platform: Local DB Update

The **Local DB Update** action updates rows in a local database table. An optional Where clause can be used.



## Parameter description

Parameter	Description
<b>Table</b>	The name of the local database table for the update operation. The table name will be available from a drop-down list.
<b>Where</b>	<p>A Where clause can be used to restrict the rows to be updated.</p> <p>Both constants and substitution variables can be used in the Where clause.</p> <p>To construct a Where clause, use an operator (=, !=, &gt;, &gt;=, &lt;, &lt;=, like, is null, is not null ) to relate the column to either a constant or a substitution variable. Each of these operators can be combined with other operators using an <b>And</b> or <b>Or</b> statement.</p> <p>To use a substitution variable, insert \$(x) where x is a variable that will be associated to a variable on the <b>Input</b> tab (see the <b>Input</b> tab below).</p> <p><b>Note:</b> For columns of type TEXT, the constant or substitution variable must be enclosed in double quotes. For example:</p> <p>C01 = "JohnDoe" The constant is enclosed in double quotes.</p> <p>C01 = "\$(test)" The substitution variable is enclosed in double quotes. The variable (test in this example) will be added to the Input tab and can then be mapped to a STRING variable (or a constant).</p> <p>The Where clause builder, accessed by selecting the icon , can be used to assist in building the Where clause.</p> <p>For more information on the Where clause builder, see Local DB Select.</p>

## Input tab

The **Input** tab will have one row per column in the local database table, where each row is identified by the table column name.

In addition, the **Input** tab will have one row for each logical variable used in the Where clause (for example, variable referenced as \$(VariableName)).

Parameter	Description
<b>Table Column Names</b>	Optional. The columns that will be updated by the value from the variable mapped to it. One or more columns can be updated with values, in each of the rows returned by the Where clause.

	This variable can be any type of variable in the system or a constant.
<b>Logical Variables from the Where clause</b>	The variable whose value is to be substituted in the Where clause logical variable. This variable can be any type of variable in the system or a constant.

## Output tab

Parameter	Description
<b>Rows Updated</b>	The actual number of rows updated.
<b>Error Message</b>	Optional. Used to provide information if the update fails.

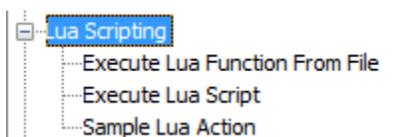
Related topics

Local Database

Transaction

# IIoTA industrial IoT Platform: Lua Scripting

The **Lua Scripting** category provides actions used to execute a Lua script. Lua is a scripting language that is supported by the runtime to perform custom functions that would not be possible with the standard provided trigger actions.



The Lua scripting language can be used to:

- Create Lua functions that can be executed from a trigger action.
- Create Lua functions in a text file that can be executed from a trigger action.
- Create a new trigger action that can become available to triggers in the same manner as the standard provided actions.

Information available from the **Lua Scripting** category pages assumes that you are familiar with the Lua language and syntax, and that you have reviewed the information on **Extending the system using Lua scripting**.

The **Lua Scripting** category provides these actions:

In addition to the provided actions, custom Lua actions that have been defined and added to the node will also be listed under the Lua Scripting category.

## IIoTA industrial IoT Platform: Execute Lua Function From File

The **Execute Lua Function From File** action executes a custom Lua function defined in a file in the Staging Browser. The Lua function can be simple or complex. Since the Lua function is defined in a separate file, the function can be easily updated without modifying the trigger.

A different action, the **Execute Lua Script** action executes a custom Lua script defined directly in the action.

2. Execute Lua Function From File

**Script File:**

**Function Name:**

**Input Variables:**

**Output Variables:**

Output

Output

Name	Logical	Count	Value	Type
ReturnMessage	STRING(0)	1		
ErrorMessage	STRING(0)	1		

### Parameter descriptions

Parameter	Description
<b>Script File</b>	The name of your Lua script file in the Staging Browser. Scripts are usually placed in the “/scripts” directory. For example, “/scripts/sample.lua”.
<b>Function Name</b>	The name of the function you want to execute inside the script file. For example, if the function is declared as “function stringsplit()”, you would use “stringsplit”.
<b>Input Variables</b>	Variables that will be passed into the Lua script when execution of the script begins. When an input variable is added using the <b>Configure...</b> Variables window, it is also added to the Input tab.

<b>Output Variables</b>	Variables that will be returned by the Lua script when execution of the script ends. When an output variable is added using the <b>Configure...</b> Variables window, it is also added to the Output tab.
-------------------------	---

## Input tab

Parameter	Description
<b>Input Variables</b>	Input variables will appear and can be mapped to variables when <b>Input Variables</b> parameters are added using the <b>Configure...</b> Variables window.

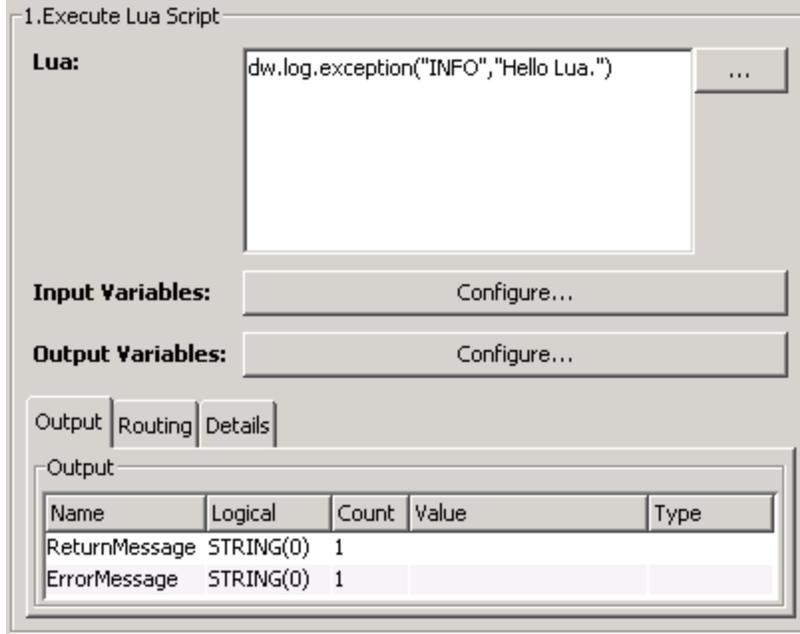
## Output tab

Parameter	Description
<b>Output Variables</b>	Output variables will appear and can be mapped to variables when <b>Output Variables</b> parameters are added using the <b>Configure...</b> Variables window.
<b>ReturnMessage</b>	A return message set by the Lua script. The script can return a numeric code or string.
<b>ErrorMessage</b>	An error message set by the Lua script.

# IIoTA industrial IoT Platform: Execute Lua Script

The **Execute Lua Script** action executes a custom Lua script defined in the action. The Lua script can contain scripting code that normally executes inside a Lua function or it can contain a mix of scripting code and functions themselves.

A different action, the **Execute Lua Function From File** action executes a Lua function that is defined in a file that has been uploaded to the staging directory.



## Parameter descriptions

Parameter	Description
<b>Lua</b>	The custom Lua script is entered directly in this parameter. This can contain lines of Lua scripting code or entire Lua functions.
<b>Input Variables</b>	Variables that will be passed into the Lua script when execution of the script begins. When an input variable is added using the <b>Configure...</b> Variables window, it is also added to the Input tab.
<b>Output Variables</b>	Variables that will be returned by the Lua script when execution of the script ends. When an output variable is added using the <b>Configure...</b> Variables window, it is also added to the Output tab.

### LUA Example

```
function logsimple()
  dw.log.debug("INFO","Log Message")
end
```

```
logsimple();
dw.log.debug("INFO","Log Message2");
```

## Input tab

Parameter	Description
-----------	-------------

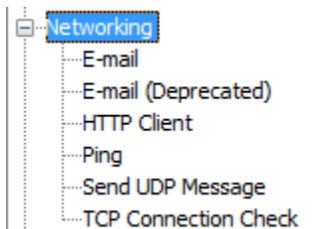
<b>Input Variables</b>	Input variables will appear and can be mapped to variables when <b>Input Variables</b> parameters are added using the <b>Configure... Variables</b> window.
------------------------	---

## Output tab

Parameter	Description
<b>Output Variables</b>	Output variables will appear and can be mapped to variables when <b>Output Variables</b> parameters are added using the <b>Configure... Variables</b> window.
<b>ReturnMessage</b>	A return message set by the Lua script. The script can return a numeric code or string.
<b>ErrorMessage</b>	An error message set by the Lua script.

## IIoTA industrial IoT Platform: Networking

The **Networking** category provides actions that relate to your network and e-mail capabilities.



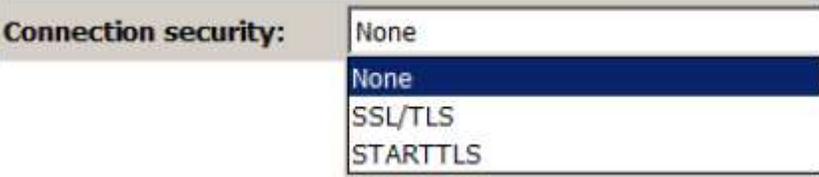
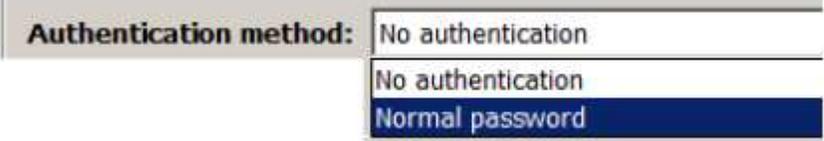
The **Networking** category provides these actions:

- E-mail
- MQTT Publish
- Ping
- Send UDP Message
- RCP Connection Check

## IIoTA industrial IoT Platform: E-mail

The **E-mail** action sends an e-mail message to a dynamically created list of users. The e-mail message can optionally include file attachments from the Staging Browser area.

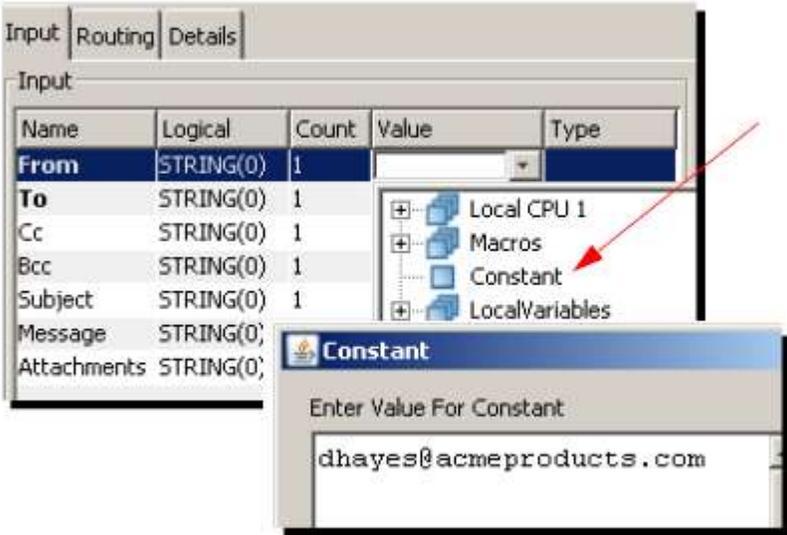
## Parameter description

Parameter	Description
<b>Server Address</b>	The IP address of the mail server. You can also use a host name as long as the DNS can resolve it.
<b>Server Port</b>	The port number where the mail server is installed (usually 25).
<b>Connection security</b>	<p>The <b>E-mail</b> action optionally supports Secure Sockets Layer (SSL) and Transport Layer Security (TLS) and STARTTLS security connections.</p> 
<b>Authentication method</b>	<p>Use the <b>Authentication method</b> down-arrow to select <b>No authentication</b> or <b>Normal password</b>.</p>  <p>When you select <b>Normal password</b>, the <b>Username</b> and <b>Password</b> parameters become available. <b>Username</b> and <b>Password</b> are required in order to log on to the mail server when the authentication method is normal password.</p>
<b>Username</b>	The user name to use to log on to the mail server.
<b>Password</b>	The password associated with <b>Username</b> to log on to the mail server.

## Input tab

The **Input** tab is used to specify the sender of the e-mail and one or more recipients. You can also specify one or more files to attach to the e-mail.

Input				
Name	Logical	Count	Value	Type
From	STRING(0)	1		
To	STRING(0)	1		
Cc	STRING(0)	1		
Bcc	STRING(0)	1		
Subject	STRING(0)	1		
Message	STRING(0)	1		
Attachments	STRING(0)	1		

Parameter	Description
<b>From</b>	<p>The e-mail address of the sender of the message. The following describes how to specify an e-mail address of the sender as a constant value.</p> <p>From the <b>Input</b> tab, on the <b>From</b> row, select the <b>Value</b> column to display variables for the node, and then select <b>Constant</b>.</p>  <p>When the Constant window appears, type the e-mail address for the sender of the e-mail, and then select <b>OK</b>. The e-mail address is added to the <b>Value</b> column.</p>

Name	Logical	Value	Type
From	STRING(0)	dhayes@acmeproducts.com	CONSTANT

You can also specify a local variable. The following describes how to specify the from e-mail address as a local variable: \* Create the local variable. From the **Variables** tab of the current trigger, under **Local Variables**, select **Add**.

**New Variable**

Name:

Type:

Length:

Count:

Default Value:

From the New Variable window, fill in the parameters as appropriate, and then select **Add**. The **Default Value** will be the from e-mail address.\* From the **Input** tab, on the **From** row, select the **Value** column, and then select the local variable.

The from e-mail address is added to the **Value** column.

Name	Logical	Value	Type
From	STRING(0)	LocalVariables.FromAddress	STRING(50)

The steps are similar when using a static variable or a device variable.

**To**

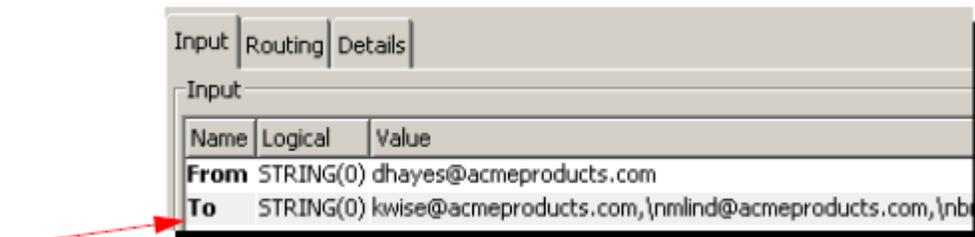
This is one or more e-mail addresses for the recipients of the message. To send the message to more than one person, separate each e-mail address with a comma.  
The following describes how to specify multiple e-mail addresses as a constant value:

From the **Input** tab, on the **To** row, select the **Value** column to display variables for the node, and then select **Constant**.

When the Constant window appears, type the e-mail address of each person to send the message, and then select **OK**.



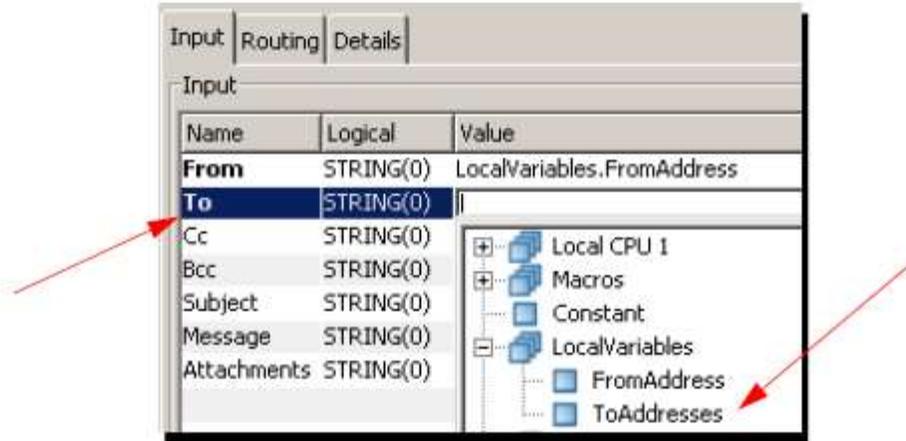
Be sure to separate each e-mail address with a comma. The e-mail addresses are added to the **Value** column.



You can also specify a local variable to hold the e-mail addresses of the recipients of the message. The following describes how to specify the more than one e-mail address as a local variable: \* Create the local variable. From the **Variables** tab of the current trigger, under **Local Variables**, select **Add**.



From the New Variable window, fill in the parameters as appropriate, and then select **Add**. Make sure the length of the string can accommodate all of the e-mail addresses. The **Default Value** will be the e-mail addresses to use to send the message. Separate each e-mail address with a comma.\* From the **Input** tab, on the **To** row, select the **Value** column, and then select the appropriate local variable.

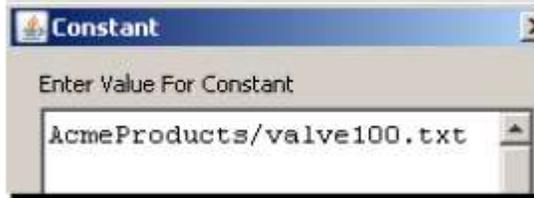


The local variable is added to the **Value** column.  
 The steps are similar when using a static variable or a device variable.

<b>Cc</b>	<p>This is one or more e-mail address to use to send the message as a Carbon Copy. To send the message as a Carbon Copy to more than one person, separate each e-mail address with a comma.</p> <p>The value can be a constant, local variable, static variable, or device variable. The steps to use a constant or variable to hold the e-mail addresses are similar to the <i>To</i> parameter.</p>
<b>Bcc</b>	<p>This is one or more e-mail address to use to send the message as a Blind Carbon Copy. To send the message as a Blind Carbon Copy to more than one person, separate each e-mail address with a comma.</p> <p>The value can be a constant, local variable, static variable, or device variable. The steps to use a constant or variable to hold the e-mail addresses are similar to the <i>To</i> parameter.</p>
<b>Subject</b>	<p>The subject of the e-mail. The value can be a constant, local variable, static variable, or device variable. The steps to use a constant or variable to hold the subject are similar to the <i>From</i> parameter.</p>
<b>Message</b>	<p>The text string that you want to include in the e-mail. The value can be a multi-line constant, local variable, static variable, or device variable. The steps to use a constant or variable to hold the message are similar to the <i>From</i> parameter.</p>
<b>Attachments</b>	<p>The files that you want to include in the e-mail. Separate each attachment with a comma to send more than one attachment. The value can be a constant, local variable, static variable, or device variable. The file must reside on the Staging Browser area.</p>

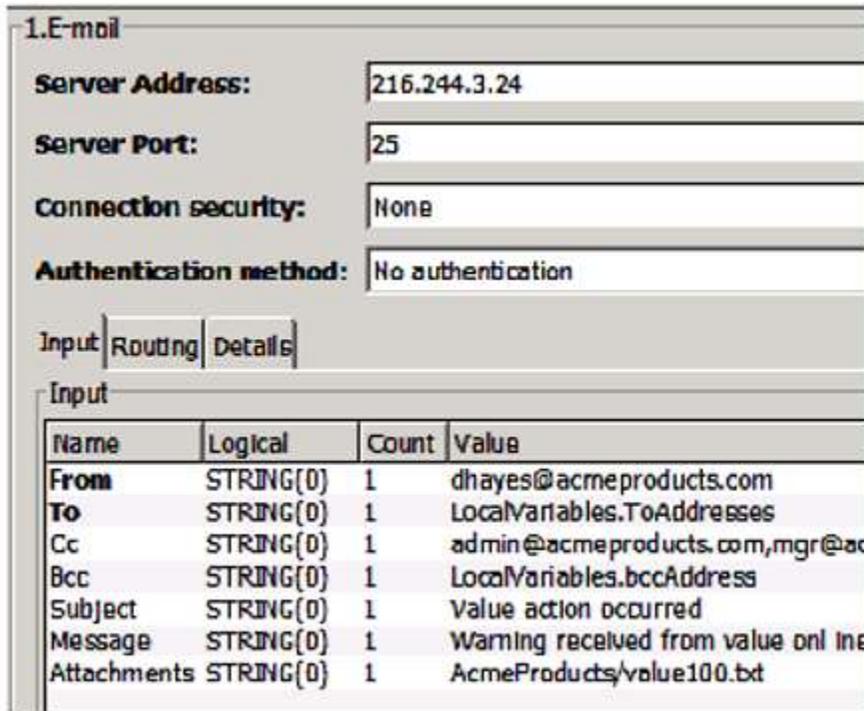


You must type the complete path and file name to attach the e-mail. For example:



### Example E-mail action

A completed E-mail action might look like this:



When the **E-mail** action executes, an e-mail is sent to the addresses defined in the Local Variable **ToAddresses**, the addresses defined in the **Cc** parameter, and the addresses defined in the Local Variables **bccAddress**. The recipients will receive the e-mail with the message shown in the **Message** parameter and one attachment, the valve100.txt file. The valve100.txt file is located in the **Staging Browser**, AcmeProducts folder.

## IIoTA industrial IoT Platform: MQTT Publish

The **MQTT Publish** action sends raw MQTT publishes on the MQTT device's MQTT connection. The MQTT Publish action allows custom data publishing to the MQTT device's MQTT broker.

### Parameter description

Parameter	Description
<b>MQTT Client</b>	The MQTT device that represents the MQTT connection to use for this MQTT Publish. The list contains the MQTT devices on this gateway that are in a <b>Started</b> state.
<b>Topic Name</b>	The MQTT topic to use. This is a required field of type String. The topic name can be a user defined topic or a topic that is specified by the MQTT broker.

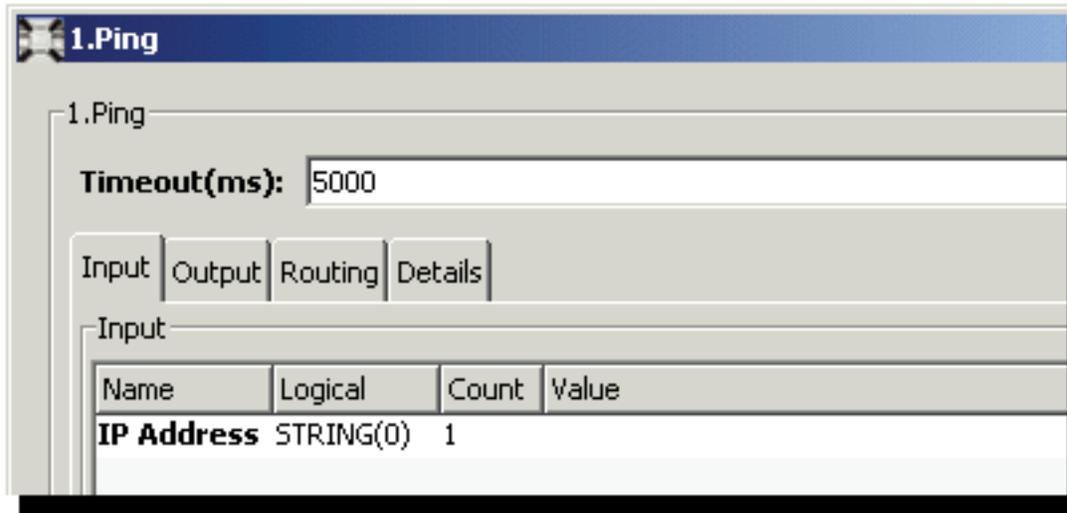
	The command can use the substitution string feature indicated by using \$(string). The substitution variables entered in this parameter are then defined in the corresponding parameters in the <b>Input</b> tab.
<b>Quality of Service (QoS)</b>	<p>Quality of service that is used to send the publish. Default is set to 0.</p> <p>QoS 0: The defined behavior for QoS 0 is that an acknowledge for the publish is not required for a successful action.</p> <p>QoS 1: The defined behavior for QoS 1 is that an acknowledge for the publish is returned from the server for the action to be successful. In the case of a failure, the MQTT connection will be disconnected and the MQTT device will be disabled.</p>
<b>Payload Type</b>	Type of payload to send, either String or Binary. The default is String.
<b>Payload</b>	<p>The payload or content of the publish to send. This field is only enabled when the Payload Type is set to String. This field is required when the payload Type is set to String and defaults to \$(payload).</p> <p>This field can use the substitution string feature indicated by using \$(payload). The substitution variables entered in this parameter are then defined in the corresponding parameters in the <b>Input</b> tab. If the substitution string feature is not used, this field is simply treated as a String field and the substitution parameter in the input tab is disabled.</p>

## Input tab

Input	Description
<b>Topic</b>	The value for the substitution variables entered for the Topic Name parameter. You can define them to reference any started device, constant, trigger macro, trigger local variable, trigger static variable, or event variable available from the pull-down list displayed for the <b>Value</b> cell for that row.
<b>Payload</b>	<p>When the Payload Type is set to String, this field can be used to set the value for the substitution variables entered for the Payload parameter. You can define them to reference any started device, constant, trigger macro, trigger local variable, trigger static variable, or event variable available from the pull-down list displayed for the <b>Value</b> cell for that row. When the substitution string feature is not used in the Payload parameter, this field is disabled.</p> <p>When the Payload Type is set to Binary, this field is a required field to set the binary content of the payload to send.</p>

# IIoTA industrial IoT Platform: Ping

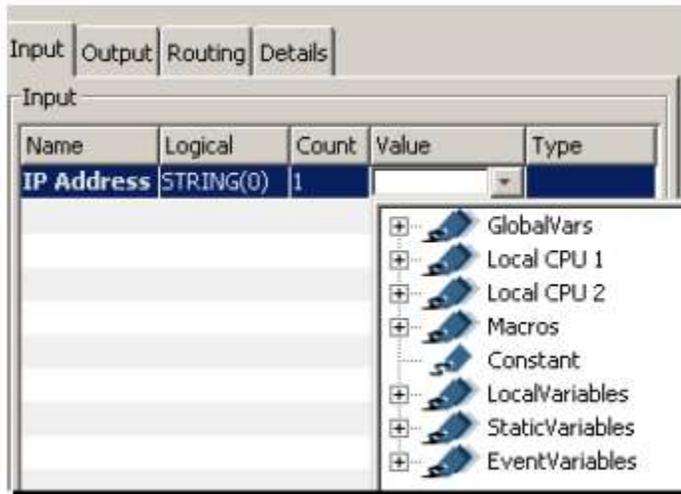
The **Ping** action sends a ping echo request to test reachability of a host.



## Parameter description

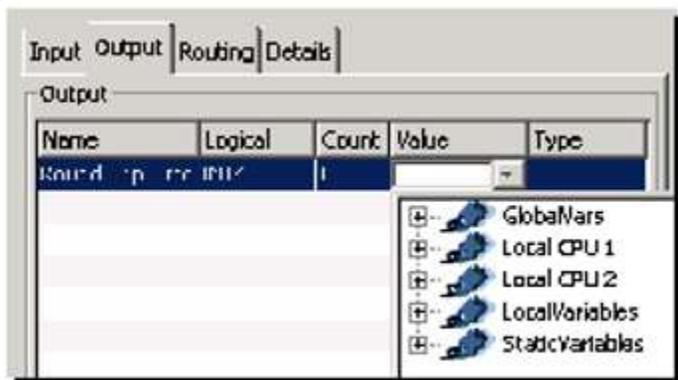
Parameter	Description
Timeout(ms)	The timeout to use for the ping echo request.

## Input tab



Parameter	Description
IP Address	The destination IP address or host name for the ping echo request.

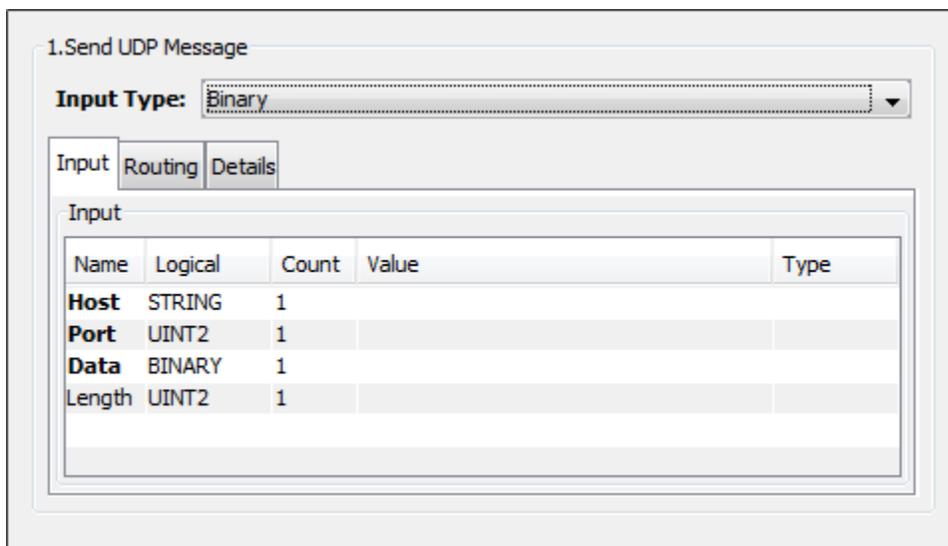
## Output tab



Parameter	Description
<b>Round Trip Time</b>	The round-trip time, in milliseconds, for the ping echo request and the corresponding ping echo reply to complete.

## IIOA industrial IoT Platform: Send UDP Message

The **Send UDP Message** action sends a User Datagram Protocol (UDP) message to a destination hostname or IP address, and port.



## Parameters

Parameter	Description
<b>Input type</b>	Specifies the data type, <b>Binary</b> or <b>String</b> of the message to send. The data is taken from the <b>Data</b> parameter and any data conversion is done into the message sent in the UDP message.

## Input tab

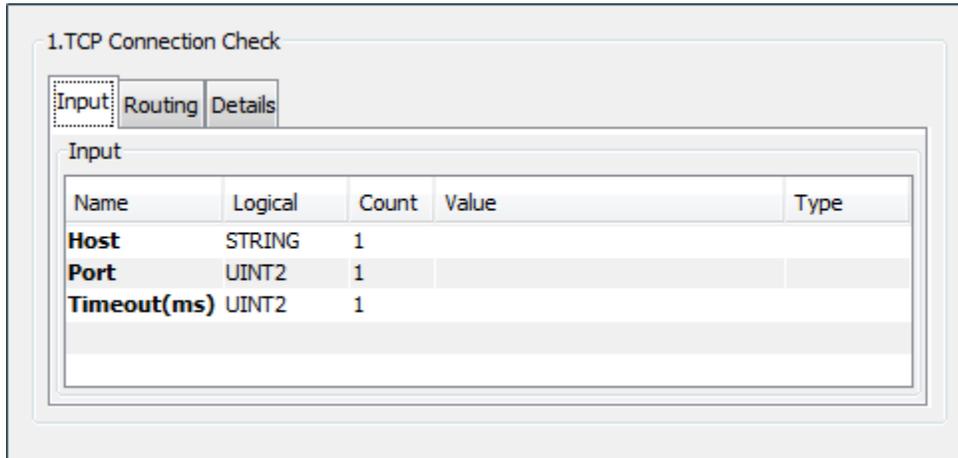
Parameter	Description
<b>Host</b>	The hostname or IP address of the remote computer to send the UDP message to.
<b>Port</b>	The UDP port on the remote computer to send the UDP message to. The valid values are 1 - 65545. A port value of 0 is not valid.
<b>Data</b>	The message to send to the remote computer. The logical data type is set based on the option selected in the <b>Input Type</b> parameter. The data type conversions supported are:  INT1, INT2, INT4, INT8, UINT1, UINT2, UINT4, UINT8, STRING, FLOAT4, FLOAT8, TIMESTAMP to STRING BINARY to BINARY.
<b>Length</b>	An option length that can be used to send less than the full amount of the message in the <b>Data</b> parameter. The length of the data in the <b>Data</b> parameter is known for both <b>Binary</b> and <b>String</b> types. If this optional Length parameter is not specified, then the full message based on the known size is sent.

## Send UDP Message action considerations

- The UDP datagram protocol specifies that the data is sent in a single network packet.
- If two or more triggers are defined and started with a Send UDP Message action to the same **Host** (hostname or IP address) and **Port**, then the platform's UDP support will handle the sending of the data as two distinct UDP messages. A target Receive UDP Message event trigger (or external application) will receive the data as two distinct UDP messages (the Receive UDP Message trigger will receive the two messages in two distinct executions of the trigger).
- A **Port** value of 0 is not in the valid port range.

# IIoTA industrial IoT Platform: TCP Connection Check

The **TCP Connection Check** action checks the ability to do a TCP connection to a hostname or IP address, and port.

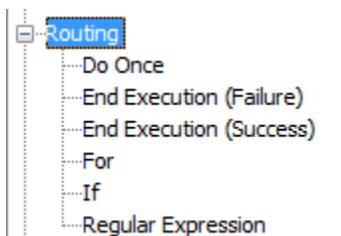


## Input tab

Parameter	Description
<b>Host</b>	The hostname or IP address of the remote computer.
<b>Port</b>	The port to use for the TCP connect.
<b>Timeout(ms)</b>	The timeout, in milliseconds, to wait for a successful completion of the TCP connect.

# IIoTA industrial IoT Platform: Routing actions

The **Routing** category provides actions that control the executing path of a trigger.



The **Routing** category provides these actions:

- Do Once
- End Execution (Failure)
- End Execution (Success)
- Error Handling
- For
- If
- Regular Expression
- Jump
- Connector

## IIoTA industrial IoT Platform: Do Once

The **Do Once** action provides a First Execution route for the first time a trigger is executed. All trigger executions after the first time follow a Normal Execution route.

Normally, the **Do Once** action would be the first action in a trigger.

The First Execution route would be used to execution initialization or other actions that should only be executed once.

The Normal Execution route would be used for all other executions of the trigger.

### Routing tab

The **On Result** column provides two routes for the action:

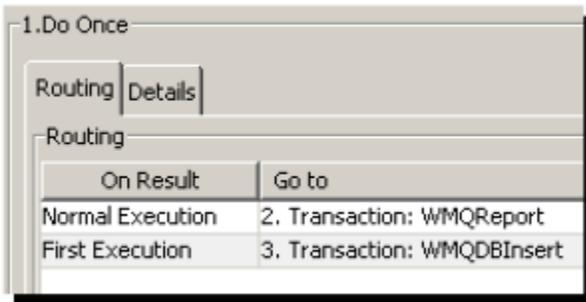
Parameter	Description
<b>Normal Execution</b>	The route to take for the second and all subsequent executions of the trigger.
<b>First Execution</b>	The route to take for the first execution of the trigger. Note that stopping the trigger or its project will cause the triggers status to be <b>Unloaded</b> , which resets the indication of whether the trigger has been executed.

### Example Do Once

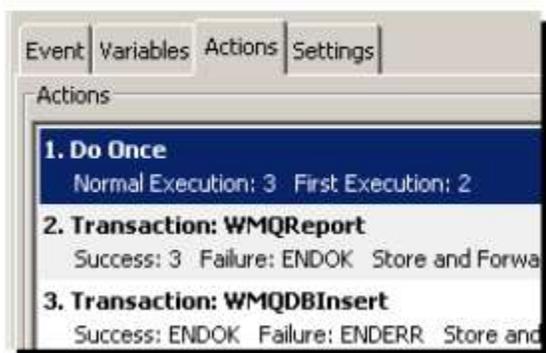
This example describes a trigger that will use a **Do Once** action. In this scenario, the trigger is set up to send transactions to a WebSphere MQ queue. The trigger will be started at 8:00 AM every day and shut down at 11:59 PM. The purpose of the trigger is to transmit a report that is written to a WebSphere MQ queue, the first time the trigger is executed. The content of the transaction is a run notice that is formatted to indicate the initial status of the device variables. The trigger is then run every 30 minutes to report on the general health of the system being monitored. This is

accomplished by executing a database insert operation, in which system data is pushed up to the enterprise Oracle database. The formats of the two transactions are different, as is the requirements for each transaction. The first transaction is an initialization report sent once a day when the trigger is first started, the other is a monitoring transaction that populates a database table with device data.

A **Do Once** action is defined in this trigger to handle the dual requirements. Two transport maps were created: **WMQReport**, which contains the format for the initialization report and **WMQDBInsert**, which defines a database insert operation, mapping device variables to Oracle database table columns. The following shows the routing associated with the **Do Once** action:



The routing has been set for the trigger to execute Action 2: Transaction WMQReport one time, the first time the Trigger is executed after it is started. The routing associated with Action 2: is to exit this trigger upon successful completion of the action. Subsequent executions of this trigger will bypass this action and move immediately to the Action 3: Transaction: WMQDBInsert action. The routing associated with Action 3: is set to exit the trigger upon the successful completion of the action. The corresponding **Actions** pane shows that the **Do Once** action will be the first action to execute when the Trigger is fired. This action will make the determination as to which action to perform.



The trigger is started and executed five times. The following shows the **Transport Maps** tab:

Name	Transport	Successes	Failures
WMQReport	wmq	1	
WMQDBInsert	ora216_dwuser	4	

The **Successes** column indicates that Action 2: Transaction WMQReport is executed once, while the Action 2: WMQDBInsert is performed the other four times the trigger executes. Consequently, there will be one message delivered to the WebSphere MQ queue associated with the WMQReport transport map, while there will be four new rows added to the Oracle database table associated with the WMQDBInsert transport map.

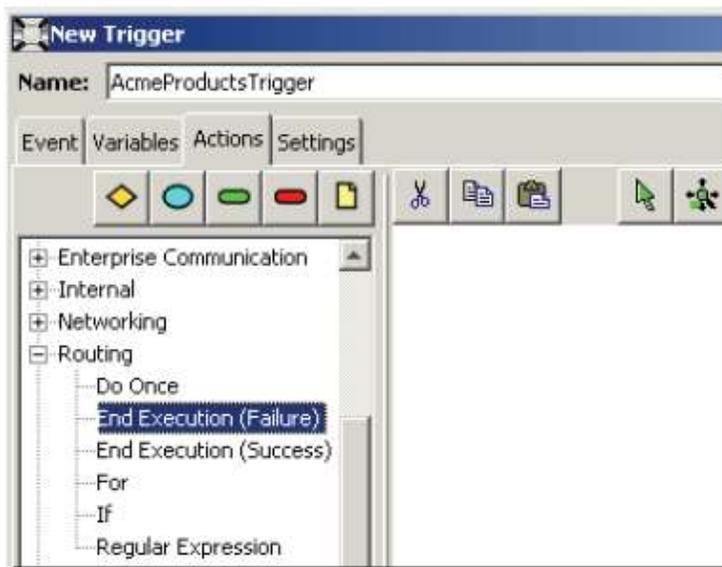
## IIoTA industrial IoT Platform: End Execution (Failure)

The **End Execution (Failure)** action provides an end execution failure route for triggers defined with the Canvas Editor.

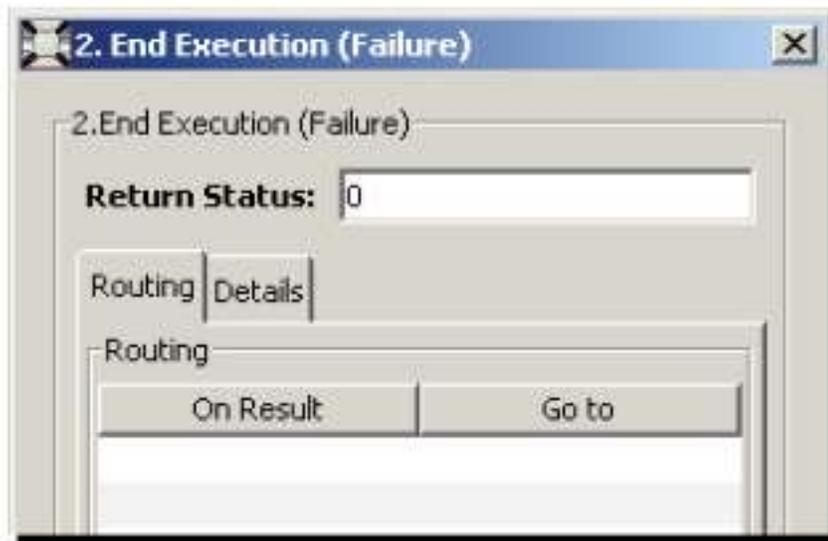
This action should only be used when defining triggers with the Canvas Editor.

The **End Execution (Failure)** action is available from the Canvas Editor **Actions** pane:

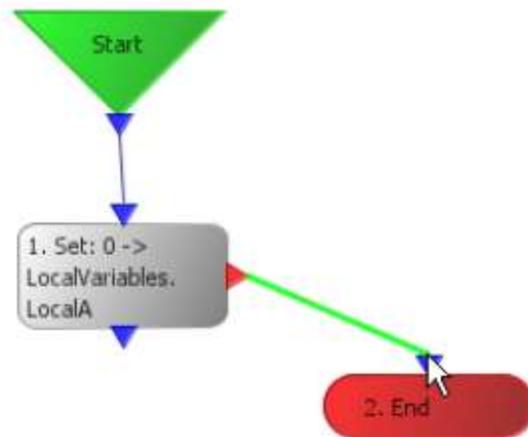
1. From the **Actions** left pane, locate and then expand **Routing**.  
Alternatively, the **End Execution (Failure)** action is available on the Canvas Editor toolbar.



2. Select (click once and hold) the **End Execution (Failure)** action and then drag to where you want the action to appear. The mouse pointer changes and adds a crossbar.
3. Release the mouse button.  
The **End Execution (Failure)** action appears on the right pane of the **Actions** tab.
4. Double-click the action.
5. The End Execution (Failure) window appears.



6. You can accept the default value of zero for the Return Status parameter **Return Status** parameter or type an application specific error code to identify the error. A custom error code enables you to identify where the error occurred in the trigger if there are numerous end error exits.
7. Close the window.  
The following assumes that you have specified an action from which to connect the **End Execution (Failure)** action (for this example, the **Set** action).



8. Select (click once and hold) the desired output connector, and then drag the line to the input connector of the **End Execution (Failure)** action.
9. Release the mouse button. The connection is made.  
Once the trigger is saved and started, you can track the success or failure of the trigger using the project tab associated with the trigger. When the trigger has reporting turned on, you can view the report (from the Reports tab) and then check the return status of the **End Execution (Failure)** action.

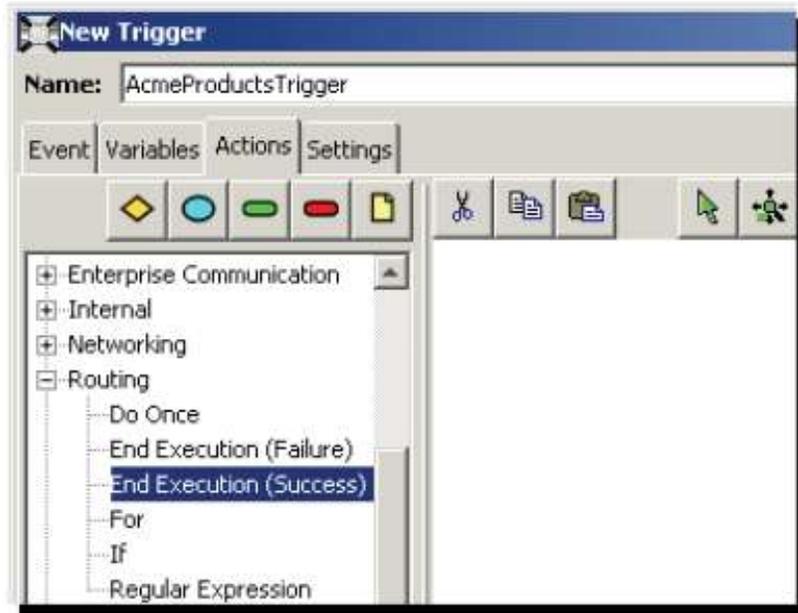
## IIoTA industrial IoT Platform: End Execution (Success)

The **End Execution (Success)** action provides an end execution success route for triggers defined with the Canvas Editor.

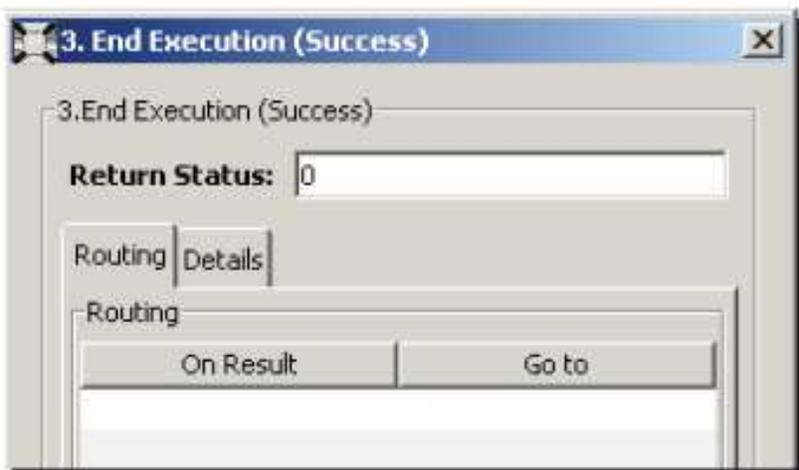
This action should only be used when defining triggers with the Canvas Editor.

The **End Execution (Success)** action is available from the Canvas Editor **Actions** pane:

1. From the **Actions** left pane, locate and then expand **Routing**.  
Alternatively, the **End Execution (Failure)** action is available on the Canvas Editor toolbar.



2. Select (click and hold) the **End Execution (Success)** action and then drag to where you want the action to appear. The mouse pointer changes to a crossbar.
3. Release the mouse button.  
The **End Execution (Success)** action appears on the right pane of the **Actions** tab.
4. Double-click the action.
5. The End Execution (Success) window appears.



6. You can accept the default value of zero for the **Return Status** parameter or type an application specific code to identify the successful completion of the action. A custom code enables you to identify where the success occurred in the trigger if there are numerous end success exits.
7. Close the window.
8. Once the trigger is saved and started, you can track its success using the project tab associated with the trigger. When the trigger has reporting turned on, you can view the report (from the Reports tab) and then check the return status of the **End Execution (Success)** action.

The following shows the **End Execution (Success)** action section available from a report. The extended status reflects the custom value set in the trigger.

Seq	Name	ID	Status / Route	Extended Status	Execution Time (ms)
1	Wait	2	Success		0
2	End Execution (Success)	3		888	

## IIoTA industrial IoT Platform: Error Handling

The **Error Handling** action provides an execution route for any unconnected failure route in a trigger. It is only available via the The Canvas Editor toolbar. Only one instance of it may be placed in a trigger.

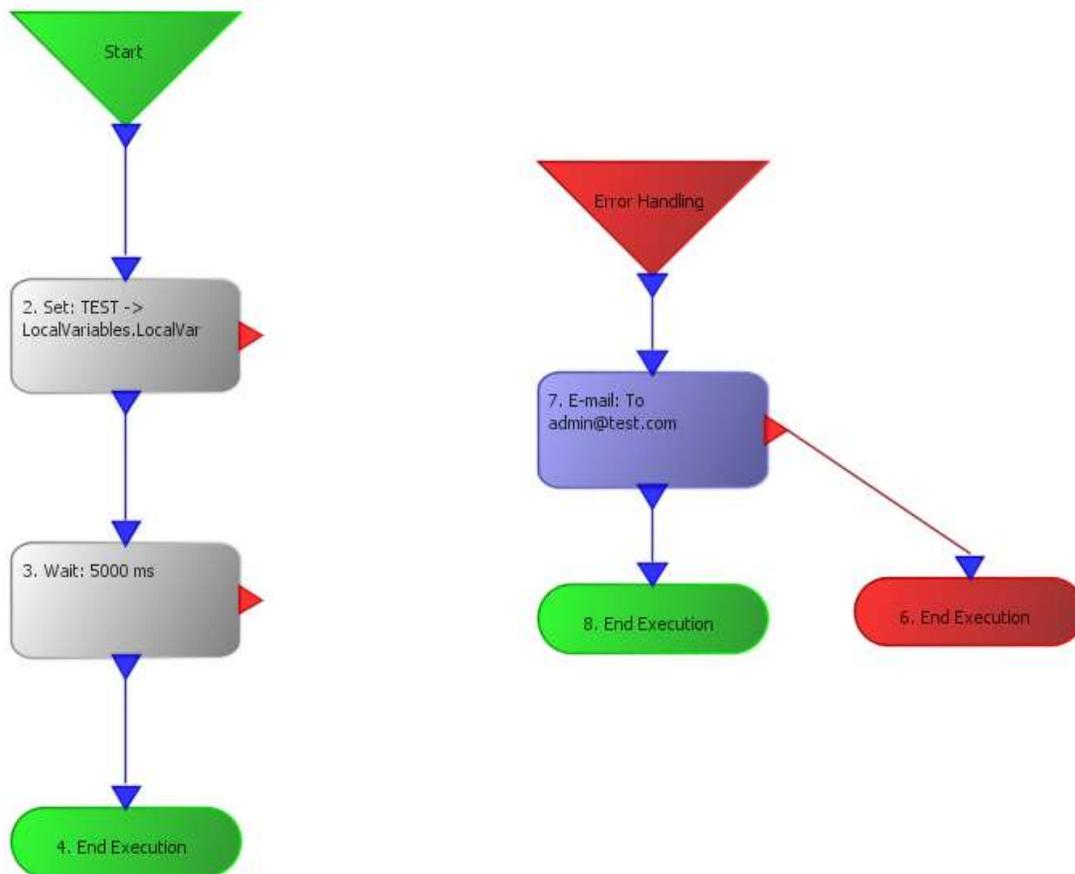
### Routing tab

The **On Result** column provides one route for the action:

Parameter	Description
Success	The route to take for when an unconnected failure route is taken.

### Example Error Handling

Below is an example trigger with an Error Handling action defined.



Without an Error Handling action defined, if the unconnected failure route is taken from any action the trigger will end with a failure status.

In the above example trigger, if the unconnected failure route is taken, the E-mail action will be executed.

**Note:** The failure route on the email action is connected to an End Execution action. If it was unconnected, it would be routed to the Error Handling action and the potential for an infinite loop would exist.

Any actions can be in the execution path after the Error Handling action, including routing back into the main body of the trigger.

As with all triggers, success and failure execution paths should eventually lead to the trigger ending its execution. Defining a trigger that gets into an infinite loop will cause execution problems for the system.

## IIoTA industrial IoT Platform: For

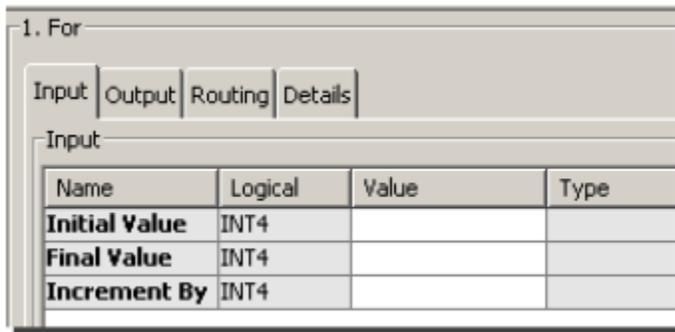
The **For** action provides an execution control route for loop sequences of actions. The loop sequence of actions is repeated until an end condition is met.

**Note:** The current value is an output only variable. Changing it within the loop will not affect the counter. If the loop is from 0 to 10 with an increment of 1 and the counter is currently at 5, a **Set** action used to change the variable used for the current counter to 20 will not end the loop. In the next iteration of the loop the counter will be 6 and the variable referenced will be set to 6.

A **For** action might be useful when you have an iteration requirement such as advancing an index and so forth.

### Input tab

The **Input** tab is where you set values for the **For** action.



Name	Logical	Value	Type
<b>Initial Value</b>	INT4		
<b>Final Value</b>	INT4		
<b>Increment By</b>	INT4		

Parameter	Description
<b>Initial Value</b>	The starting number value. In the example, the number is set to 1 using a constant.
<b>Final Value</b>	End the loop when the output variable <b>Current Value</b> reaches this number. In the example, the number is set to 5 using a constant.
<b>Increment By</b>	For each successful loop increase the number in <b>Current Value</b> by this number. In the example, the number is set to 1 using a constant.

### Example Input tab

The following shows an example **Input** tab.

1. For

Input Output Routing Details

Input

Name	Logical	Value	Type
<b>Initial Value</b>	INT4	1	CONSTANT
<b>Final Value</b>	INT4	5	CONSTANT
<b>Increment By</b>	INT4	1	CONSTANT

## Output tab

1. For

Input Output Routing Details

Output

Name	Logical	Value	Type
<b>Current Value</b>	INT4		

Parameter	Description
<b>Current Value</b>	<p>Holds the current output value of the loop iteration. For this example, a local variable for the trigger was previously defined.</p> 

## Routing tab

1. For

Input Output Routing Details

Routing

On Result	Go to
Success	NEXT
Failure	UNDEFINED
Continue	UNDEFINED

Parameter	Description
<b>Success</b>	The action to take upon a successful completion of the <b>For</b> action. The action specified for the <b>Success</b> route will execute once the <b>Current Value</b> output variable reaches the value specified in the <b>Final Value</b> input variable.
<b>Failure</b>	The action to take is there is a failure with the <b>For</b> action.
<b>Continue</b>	The action that is the first action within the loop. Routing can be done as normal. When an action routes back to the <b>For</b> action, the loop will continue and increment the <b>Current Value</b> output variable. Upon completion of the <b>For</b> action, the <b>Success</b> route will be taken.

## IIoTA industrial IoT Platform: If

The **IF** action provides an expression evaluation to either True or False. This can be used for an **If then Else** routing control.

The **IF** action can use any number of logical variables as input on either side of the equation.

### Overview

Every expression consists of at least two operands and can have one or more operators. Operands are values, whereas operators are symbols that represent particular actions.

In the expression  $X + Y - 10$   
X, Y, and 10 are operands, and + and - are operators.

The following lists the valid operators.

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Exponentiation (for example: $2^3 = 8$ )
%	Modulus (for example: $5\%2 = 1$ )

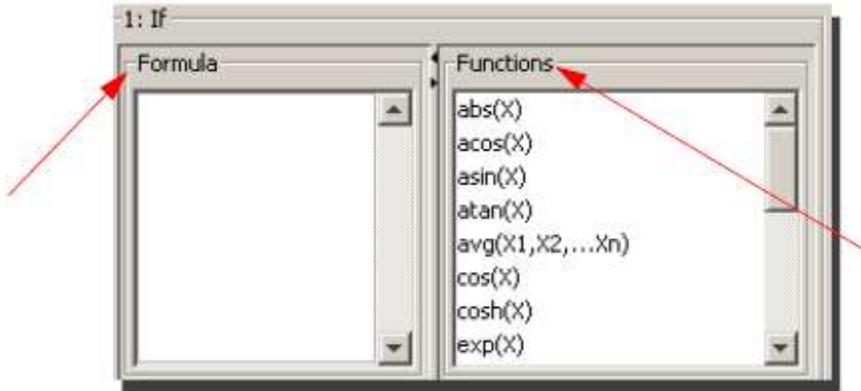
()	Force precedence of evaluation
>>	Bitwise shift right
<<	Bitwise shift left
&	Not implemented as an operator, see the and function below.
	Not implemented as an operator, see the or function below.
^	Not implemented as an operator, see the xor function below.
~	Not implemented as an operator, see the not function below.

The following lists the valid logical operators for conditional operations:

Logical Operator	Description
==	Equal
!=	Not Equal
<	Less Than
<=	Less Than or Equal
>	Greater Than
>=	Greater Than or Equal
&&	And (for example: <code>x==5&amp;&amp; y==10</code> returns true if x is 5 and y is 10)
	Or (for example: <code>x==5    y==10</code> returns true if x is 5 or y is 10)
!	Not (for example: <code>!(x==5)</code> returns false if x is 5)

There are also several built-in functions that can be used in an expression, such as `sin(X)` and `log(X)`.

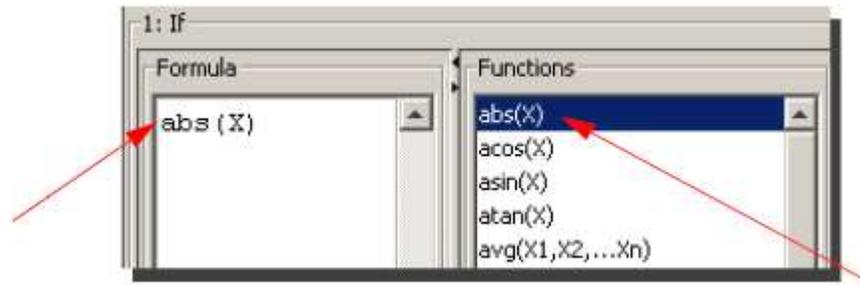
## Parameter description



Input area	Description
<b>Formula</b>	Use the <b>Formula</b> box to type an expression.
<p>As you type in the <b>Formula</b> box, the logical variables used in the expression are automatically added to the <b>Input</b> tab.</p> <p>In the example expression, <math>x=y</math> (x is equal to y), the logical variables <b>x</b> and <b>y</b> are added to the input tab.</p>	

**Functions**

Use the **Functions** list to add a built-in function to the formula.



To add a built-in function, double-click the function. The function is added to the **Formula** box.

Note that the default input operand to the abs( ) function is **X**. You can change this to any value you want.

When using the trig functions, the angle is specified in radians. For example,  $\sin(1.0) = 0.841471$ . The 1.0 is equivalent to 1 radian: where 2 pi radians equals 360 degrees

The following table lists the built-in functions that you can use within an If action.

Function	Description
abs(X)	Absolute value
acos(X)	Trigonometric arc cosine
and(X,Y)	Bitwise AND "&"
asin(X)	Trigonometric arcsine
atan(X)	Trigonometric arctangent
avg(X1, X2...,Xn)	Average of a set of values
ceil(X)	Ceiling (round up)
cos(X)	Trigonometric cosine
cosh(X)	Hyperbolic cosine
exp(X)	e to the power X
floor(X)	Floor (round down)
ln(X)	Natural log (base e)
log(X)	Natural log (base e)

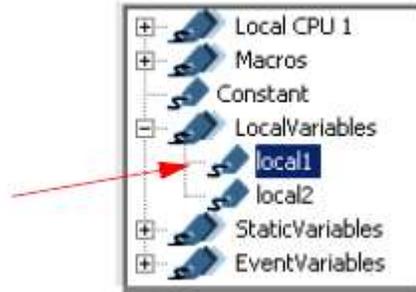
log10(X)	Log base 10
max(X1, X2...,Xn)	Maximum of a set of values
min(X1, X2...,Xn)	Minimum of a set of values
not(X)	Bitwise NOT "~"
or(X,Y)	Bitwise OR " "
sin(X)	Trigonometric sine
sinh(X)	Hyperbolic sine
sqrt(X)	Square root
sum(X1, X2...,Xn)	Sum of a set of values
tan(X)	Trigonometric tangent
tanh(X)	Hyperbolic tangent
xor(X,Y)	Bitwise XOR "^"

## Input tab

Name	Logical	Count	Value	Type
x	FLOAT8	1		
y	FLOAT8	1		

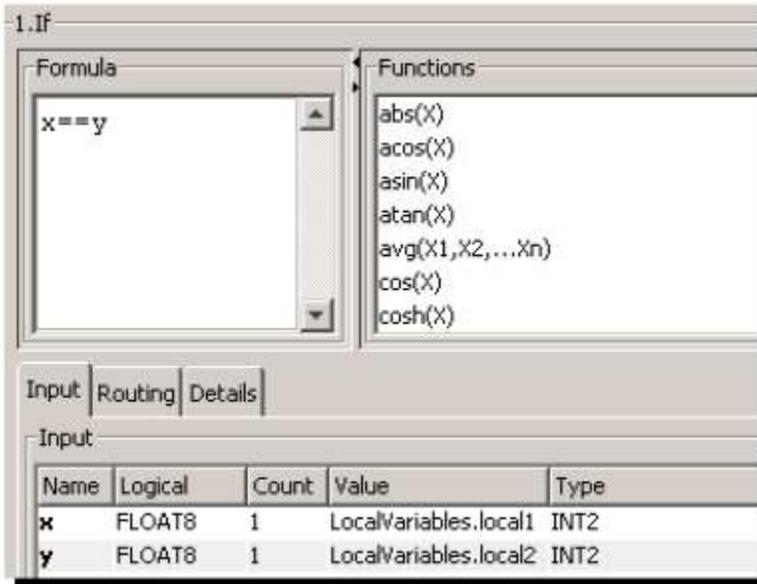
Parameter	Description
<b>Name</b>	<p>Each row is a logical variable automatically created when you type the expression. For this example, the logical variables <b>x</b> and <b>y</b> were entered in the expression</p> <p>All logical variables used in an expression have the data type <b>FLOAT8</b> which is an 8-byte floating point number. You might see <b>LREAL</b> as the default data type which is also an 8-byte floating point number.</p>

Select the **Value** column to select from a list of variables. For this example, two local variables were previously defined.

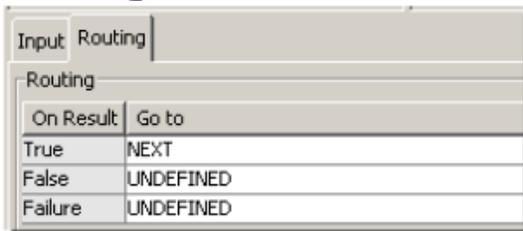


Although shown, do not specify a trigger macro. Also, you can only specify a numeric constant.

For this example, the completed **Input** tab will be similar to the following:



## Routing tab



Parameter	Description
<b>True</b>	The route to take when the expression evaluates to True.
<b>False</b>	The route to take when the expression evaluates to False.
<b>Failure</b>	The route to take if there is a failure.

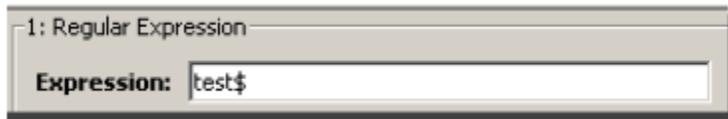
Related topics  
Expression

## IIoTA industrial IoT Platform: Regular Expression

The **Regular Expression** action tests an input string for a match to a regular expression.

### Parameter description

Use the **Expression** parameter to enter the expression to compare.



For a list of specific syntax rules for regular expressions, refer to:  
<http://www.w3.org/TR/xmlschema-2/#regexs>

**Characters not supported** - Do not use the ^ (caret) and \$ characters in a regular expression.

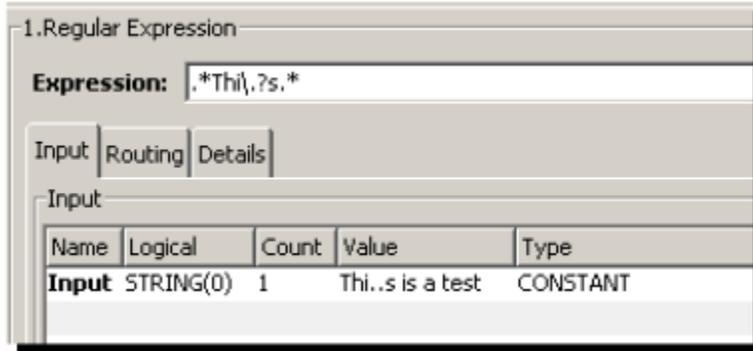
### Input tab

Parameter	Description
<b>Column</b>	<b>Description</b>
<b>Name</b>	This column provides the Input variable used for testing.
<b>Logical</b>	By default, the input variable for a regular expression has the data type <b>STRING (0)</b> .
<b>Count</b>	Indicates the quantity of the value is always a single integer. Thus, <b>Count</b> will always be one.
<b>Value</b>	The trigger variable whose value to use to compare against the expression. Click the <b>Value</b> column to select from a list of device variables, or a user defined local or static variable. You can also specify a macro or constant.

<b>Type</b>	When you specify <b>Value</b> , the data type of variable is automatically added to the <b>Type</b> column. If you select String, you must specify the length of the string.
-------------	--

## Example Regular Expression

The following shows an example Regular Expression.

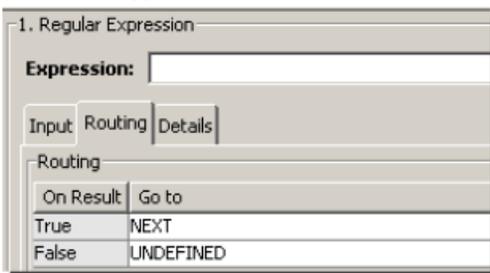


When the trigger executes, the expression is interpreted as:

.	Matches any character except a new line.
***	Causes the regular expression to match 0 or more repetitions of the preceding regular expression, or as many repetitions as are possible.
\	Escapes the special characters that follow.

The **Input** tab shows the value of the string to match as a constant. The string to look for is: This is a test

## Routing tab



Parameter	Description
-----------	-------------

<b>True</b>	The route to take when the expression evaluates to True.
<b>False</b>	The route to take when the expression evaluates to False.
<b>Failure</b>	The route to take if there is a failure.

## IIoTA industrial IoT Platform: Jump

The **Jump** action provides an execution route with no other action performed.

### Routing tab

The **On Result** column provides one route for the action:

Parameter	Description
<b>Destination</b>	The route to take from this action.

## IIoTA industrial IoT Platform: Connector

The **Connector** action primarily acts as a destination for a **Jump** action but may be routed to from any action. It performs no other action.

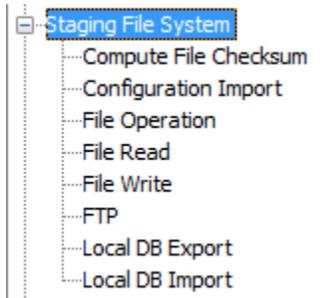
### Routing tab

The **On Result** column provides one route for the action:

Parameter	Description
<b>Normal Execution</b>	The route to take from this action.

## IIoTA industrial IoT Platform: Staging File System

The **Staging File System** category provides actions that work with files in the node's Staging Browser area.



A node's Staging Browser area supports the concepts of directories (or folders), subdirectories, and files.

In addition to the trigger actions that work with files in the Staging Browser are, the Workbench provides access by using the Administration **Staging Browser** tab.

The **Staging File System** category provides these actions:

- Computer File Checksum
- Configuration Import
- File Operation
- File Read
- File Write
- FTP
- Local DB Export
- Local DB Import
- Directory Operation

## IIoTA industrial IoT Platform: Compute File Checksum

The **Compute File Checksum** action computes the checksum value of a file by using a hash algorithm (such as CRC-32).

Resource intensive

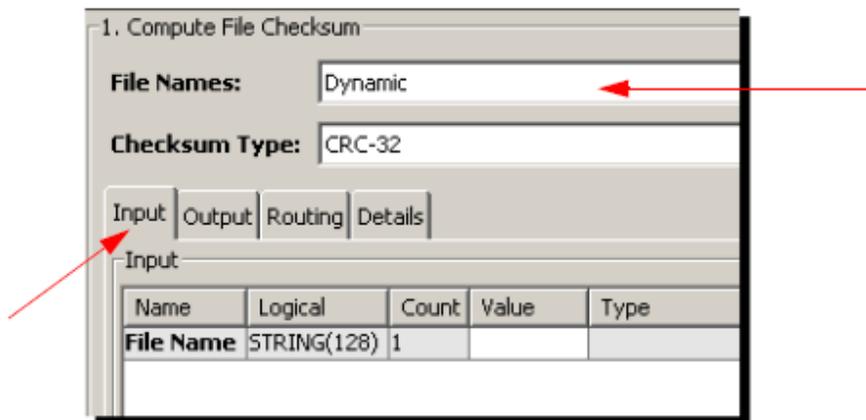
For platforms with limited memory resources or persistent storage access performance, this action can be resource intensive.

### Parameter description

Parameter	Description
<b>File Names</b>	<p>The options are:</p> <p><b>Static</b> - The <b>File Name</b> parameter becomes available and is selected from the existing files in the Staging Browser area.</p> <p><b>Dynamic</b> - The <b>File Name</b> parameter becomes available on the <b>Input</b> tab and can be specified from a variable.</p>
<b>File Name</b>	Available when the <b>File Names</b> option is <b>Static</b> . This is the complete path and file name. You can type the path and file name or use <b>Browse</b> . If you click <b>Browse</b> , the Staging Browser window appears.
<b>Checksum Type</b>	<b>CRC-32</b> is the supported Checksum algorithm.

### Input tab (Dynamic)

Available when the **File Names** option is **Dynamic**.



Parameter	Description
<b>File Name</b>	The path and file name of the file in the Staging Browser area whose checksum you want to compute.

### Output tab

The **Output** tab is the same for **Dynamic** and **Static** options.

Output				
Name	Logical	Count	Value	Type
<b>Value</b>	UJINT4	1		

Parameter	Description
<b>Value</b>	The computed checksum for the file.

## IIoTA industrial IoT Platform: Configuration Import

The **Configuration Import** action imports previously exported application definition and system configuration information, such as projects, triggers, devices, transport maps, transports, network configuration, security configuration, etc.

For information on a complete node Back Up and a node Restore, see Backing up and Restoring a node's configuration.

For information on the Import and Export function for items, see Exporting a project or trigger.

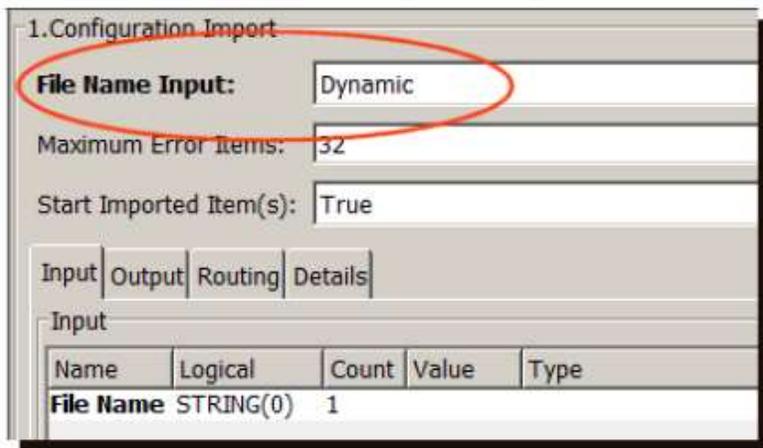
### Parameter description

Parameter	Description
<b>File Name Input</b>	<p>The options are:</p> <p><b>Static</b> – The <b>File Name</b> parameter becomes available and is selected from the existing files in the Staging Browser area.</p> <p><b>Dynamic</b> – The <b>File Name</b> parameter becomes available on the <b>Input</b> tab and can be specified from a variable.</p>
<b>File Name</b>	<p>Available when the <b>File Names</b> option is <b>Static</b>. This is the complete path and file name of a previously exported system Export file (<b>.DWX</b>). <b>You can type the path and file name, or use *Browse</b>. If you click <b>Browse</b>, the Staging Browser window appears.</p>

<b>Maximum Error Items</b>	Sets the maximum number of error messages stored for a failed Import. Works in conjunction with the <b>Output</b> tab <b>Failed Items</b> parameter. Setting the number too high might cause performance problems.
<b>Start Imported Item(s)</b>	The options are:  <b>True</b> – Any imported item (such as a project, trigger, device) that was in a started state when it was exported, will be imported in a started state. <b>False</b> – All imported items put in a stopped state.

## Input tab (Dynamic)

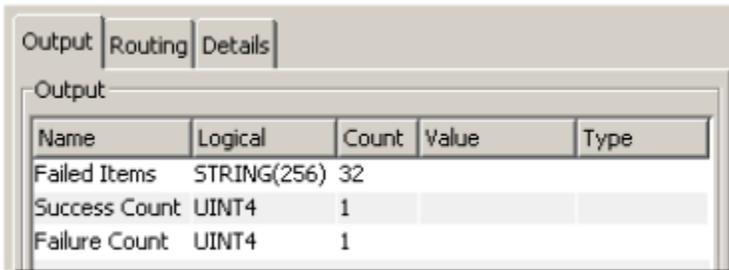
Available when the File Names option is **Dynamic**.



Parameter	Description
<b>File Name</b>	The path and file name in the Staging Browser area of a previously exported system Export file (*.DWX).

## Output tab

The Output tab is the same for Dynamic and Static options.



Parameter	Description
<b>Failed Items</b>	An array of strings to store the error messages that describe an item that failed during import. The maximum number of error messages to store is determined by the <b>Maximum Error Items</b> parameter.
<b>Success Count</b>	The number of successfully imported items.
<b>Failure Count</b>	The number items that failed during import.

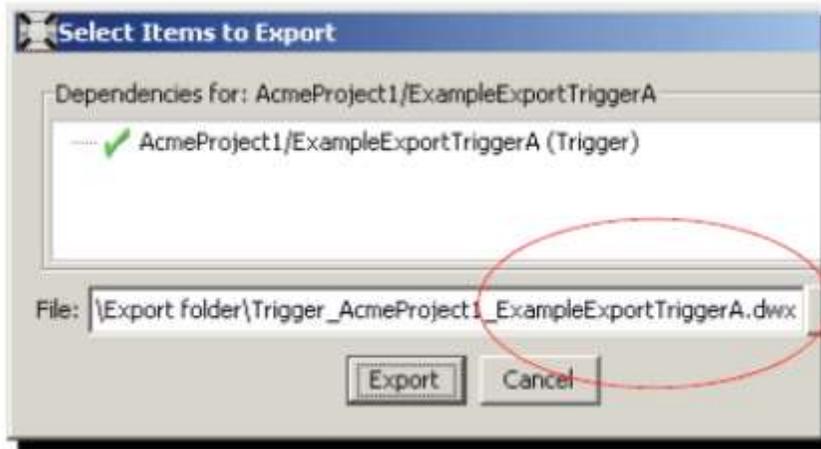
### Example Configuration Import action

The **Configuration Import** action imports previously exported application definition and system configuration information into a node. To understand the process, the following scenario is used:

- There are four nodes: A, B, C, and D.
- Node A has a trigger that needs to be distributed to nodes B, C, and D.

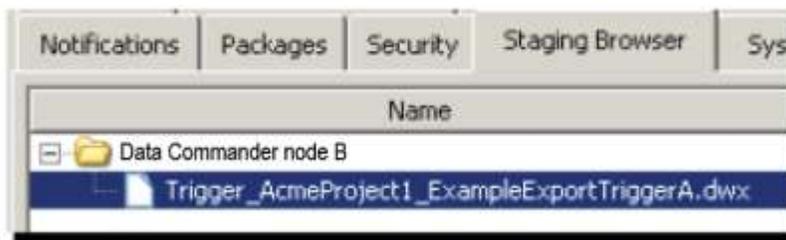
The following describes the export and import process for this scenario.

<b>From node A:</b>	<p>Create a trigger (ExampleExportTriggerA).</p>  <p>This is the trigger to be distributed to appropriate nodes: B, C, and D.</p> <p>Export the trigger.</p>
---------------------	--



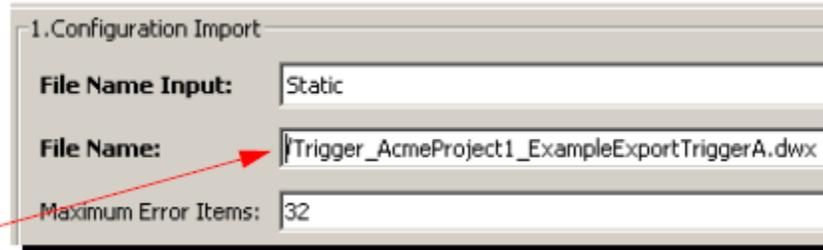
The name of the export file is ExampleExportTriggerA.dwx.

Using FTP, put the exported ExampleExportTriggerA.dwx file into the Staging Browser area nodes B, C, and D.



**From nodes B, C, and D:**

Create a trigger with a **Configuration Import** action.



The **Configuration Import** action will include the name of the file to import. For this example: ExampleExportTriggerA.dwx.

Save this trigger, and then start the trigger.

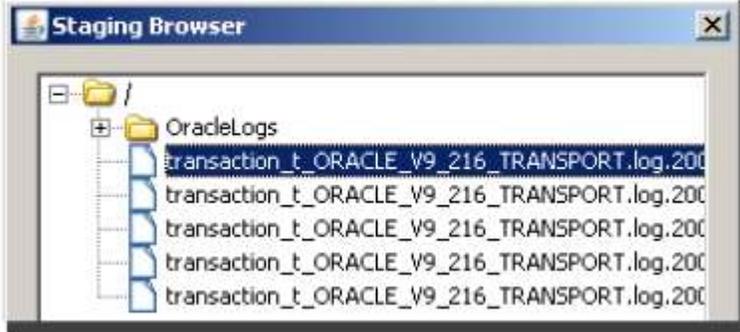
When the action executes, it will check the Staging Browser area for the file specified in the **File Name** parameter (in this case ExampleExportTriggerA.dwx).

The data specified in ExampleExportTriggerA.dwx is imported into the node and the state of the state of trigger is set as appropriate.

## IIoTA industrial IoT Platform: File Operation

The **File Operation** action provides Copy, Move, and Delete operations for files in the Staging Browser area of the node.

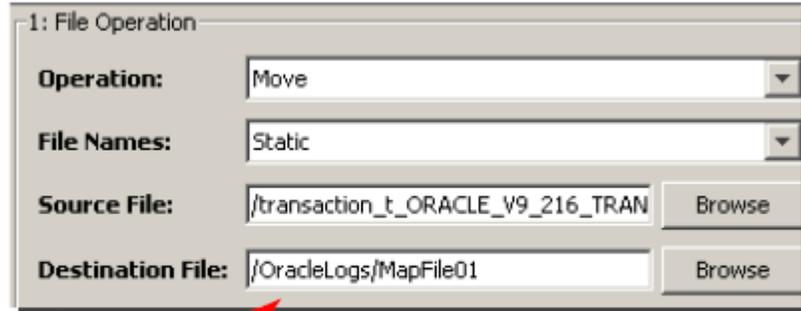
### Parameter description

Parameter	Description
<b>Operation</b>	<p>The options are:</p> <p><b>Copy</b> - copies a file to another directory.  <b>Move</b> - moves a file to another directory.  <b>Delete</b> - deletes a file.</p>
<b>File Names</b>	<p>The options are:</p> <p><b>Static</b> - When you select the <b>Copy</b> or <b>Move</b> operation, the <b>Source File</b> and <b>Destination File</b> parameters become available. When you select the <b>Delete</b> operation the <b>File</b> parameter becomes available.  <b>Dynamic - Input</b> tab parameters become available to specify variables for the file name parameters.</p>
<b>Source File</b>	<p>Available when <b>Operation</b> is <b>Copy</b> or <b>Move</b> and <b>File Names</b> is <b>Static</b>. You can enter the file name or select <b>Browse</b>. You must enter the entire file path and file name. If you select <b>Browse</b>, the Staging Browser window appears.</p>  <p>The window lists files on node in the Staging Browser area. For this example,</p>

the files shown are transport map logging files. They are in the **OracleLogs** subdirectory.

**Destination File**

Available when **Operation** is **Copy** or **Move** and **File Names** is **Static**. You can enter the file name or select **Browse**. You must enter the entire file path and file name. If you select **Browse**, the Staging Browser window appears.



The parameter cannot be left empty.

Consider the following:

You can type a destination directory and file name as follows:

XYZ/XYZ/filename.ext

/XYZ/XYZ/filename.ext

If you do not specify a file name, the trigger will fail.

For a **Copy** operation, if the file name already exists in the destination directory, the runtime will overwrite the existing file.

For a **Move** operation, if the file name already exists in the destination directory, the trigger might fail depending on the operating environment.

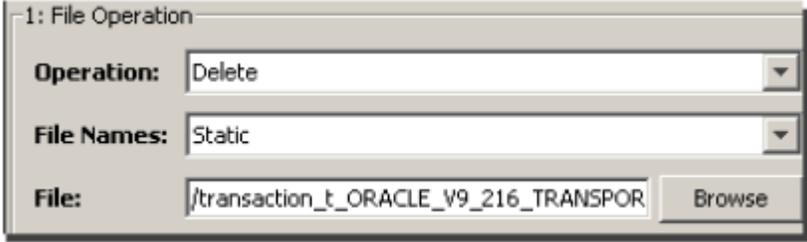
You can rename the source file that you are copying in the destination directory.

For example, if you select the FAN file from a directory called XYZ, you can copy and rename the FAN file to a destination directory named XYZ2 as FOO.txt by typing /XYZ2/FOO.txt.

When renaming a file, if the file name already exists in the destination directory, the runtime will overwrite the existing file.

You can specify a file but not a directory. The runtime copies (or moves) the file to the root of the Staging Browser area.

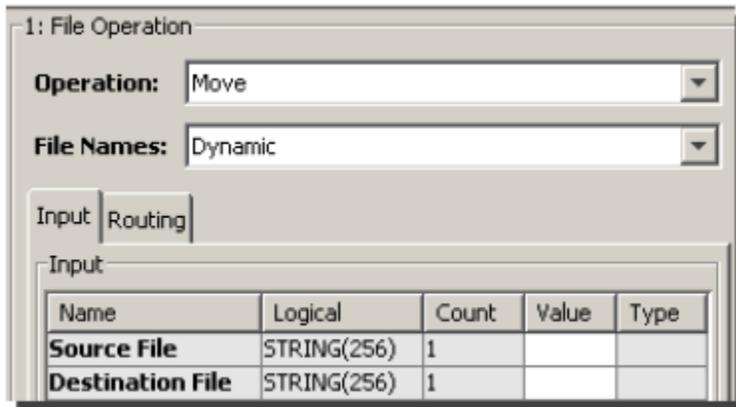
**File** Available when **Operation** is **Delete** and **File Names** is **Static**.



You can type the destination directory and file name or select **Browse**. If you do not supply the complete path, the trigger will fail.

## Input tab

When you select **Dynamic** for the **File Names** option, an **Input** tab appears.



Parameter	Description
Source File	The source file name when using the <b>Copy</b> or <b>Move</b> operations.
Destination File	The source file name when using the <b>Copy</b> or <b>Move</b> operations.
File	The target file name when using the <b>Delete</b> operation.

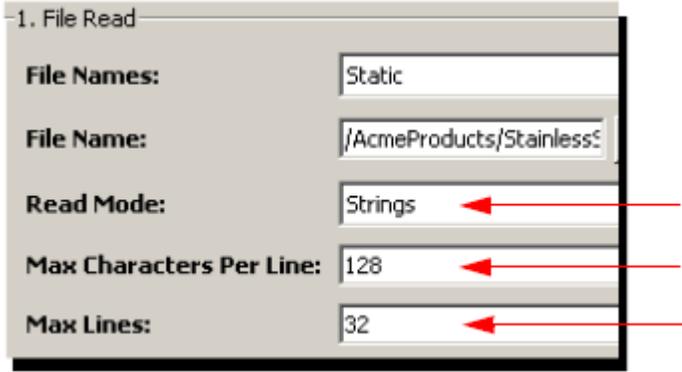
## IIOA industrial IoT Platform: File Read

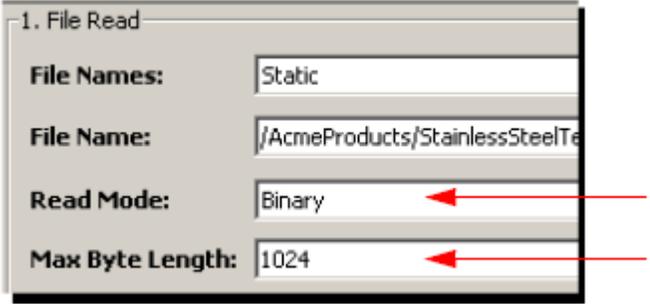
The **File Read** action reads a specified number of strings (lines) or bytes from a file in the Staging Browser area. The data is stored as strings (lines) or bytes in a destination variable.

In **Strings** mode, the file is read into strings as indicated by the line terminator.

In **Binary** mode, the file is read as one big binary variable.

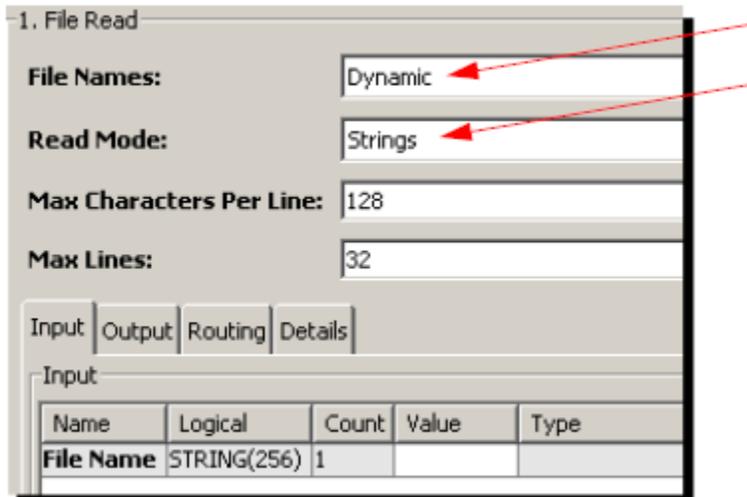
## Parameter description

Parameter	Description
<b>File Names</b>	<p>The options are:</p> <p><b>Static</b> – The <b>File Name</b> parameter becomes available to specify the file name in the Staging Browser area to read.</p> <p><b>Dynamic</b> – The <b>Input</b> tab <b>File Name</b> parameter becomes available to specify the file name.</p>
<b>File Name</b>	<p>Available when <b>File Names</b> is <b>Static</b>.</p> <p>You can enter the file name or select <b>Browse</b>. You must enter the entire file path and file name. If you select <b>Browse</b>, the Staging Browser window appears.</p>
<b>Read Mode</b>	<p>The options are:</p> <p><b>Strings</b> – When selected, <b>Max Characters Per Line</b> and <b>Max Lines</b> become available.</p>  <p>In <b>Strings</b> mode, the runtime treats <b>Line Feed</b> (0x0A hexadecimal, 10 in decimal) and <b>Carriage Return</b> (0x0D hexadecimal, 13 in decimal) as the line terminator depending on the runtime platform.</p>

	<p><b>Binary</b> – When selected, <b>Max Byte Length</b> becomes available.</p> 
<b>Max Characters Per Line</b>	The maximum number of characters in a line to read. The maximum value is 2147483647.
<b>Max Lines</b>	The maximum number of lines in the file to read. The maximum value is 2147483647.
<b>Max Byte Length</b>	The maximum number of bytes in the file to read. The maximum value is 2147483647.

## Input tab (Dynamic and Strings)

When the **Dynamic** and **Strings** options are used:



Parameter	Description
<b>File Name</b>	The file in the Staging Browser area to read. This can be a constant or a variable. If specified in a variable, the file name can change during run time.

## Input tab (Dynamic and Binary)

When the **Dynamic** and **Binary** options are used:

Parameter	Description
<b>File Name</b>	The file in the Staging Browser area to read. This can be a constant or a variable. If specified in a variable, the file name can change during run time.

## Output tab (Strings)

When the **Strings** option is used (for both **Static** and **Dynamic**):

Parameter	Description
<b>String Array</b>	The string array where the read lines are returned.
<b>Number of Lines</b>	The number of lines that were read. This value will not be larger than the <b>Max Lines</b> parameter.

## Output tab (Binary)

When the **Binary** option is used (for both **Static** and **Dynamic**):

The screenshot shows the '1. File Read' configuration window. The 'File Names' field is set to 'Static' and the 'Read Mode' is set to 'Binary'. Red arrows point to these two fields. Below the configuration is an 'Output' tab with a table showing the results of the read operation.

Name	Logical	Count	Value	Type
<b>Binary Buffer</b>	BINARY(1024)	1		
<b>Binary Length</b>	INT4	1		

Parameter	Description
<b>Binary Buffer</b>	The variable where the read bytes are returned.
<b>Binary Length</b>	The number of bytes that were read. This value will not be larger than the <b>Max Byte Length</b> parameter.

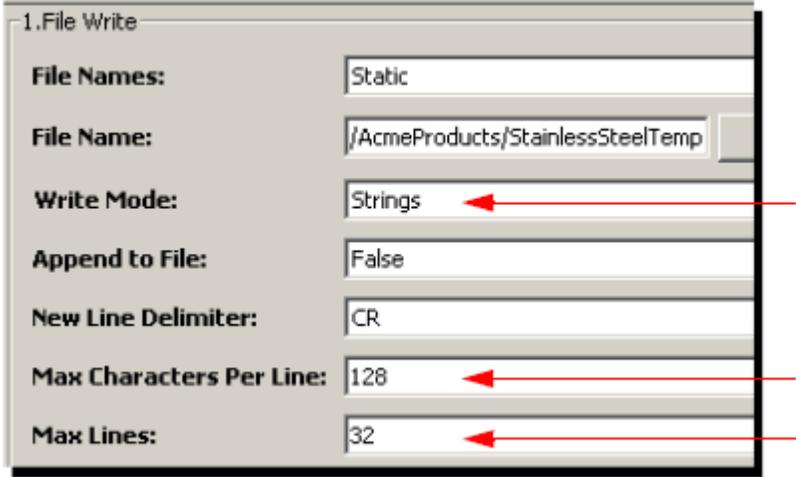
## IIoTA industrial IoT Platform: File Write

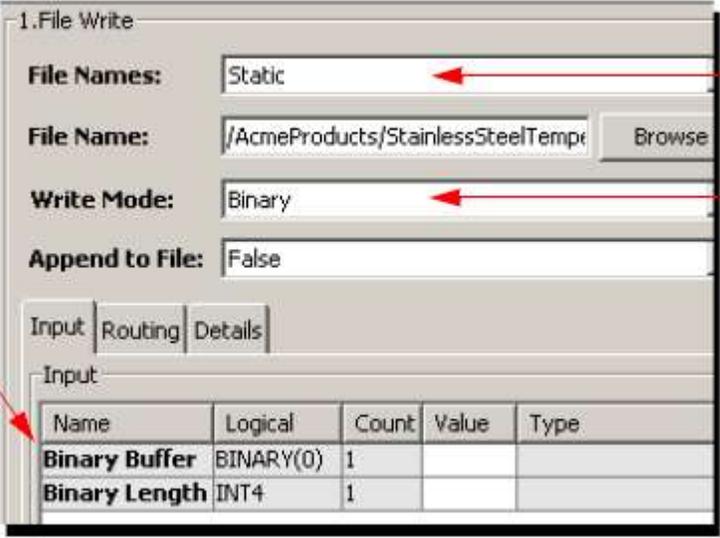
The **File Write** action appends data to the end of a file and returns the number of bytes written.

In **Strings** mode, the data is written and then a line terminator is added.

In **Binary** mode, the file is written read as one big binary variable.

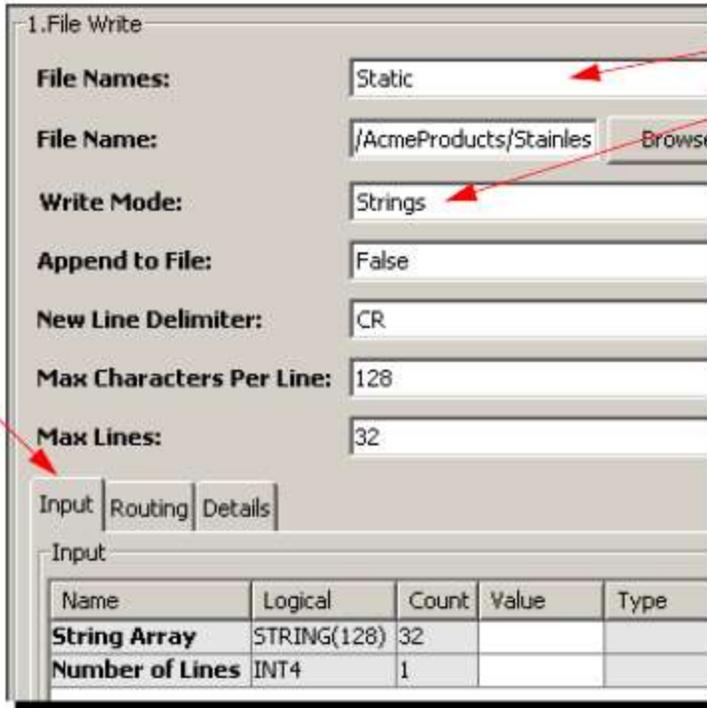
## Parameter description

Parameter	Description
<b>File Names</b>	<p>The options are:</p> <p><b>Static</b> - The <b>File Name</b> parameter becomes available to specify the file name in the Staging Browser area to write.</p> <p><b>Dynamic</b> - The <b>Input</b> tab <b>File Name</b> parameter becomes available to specify the file name.</p>
<b>File Name</b>	<p>Available when <b>File Names</b> is <b>Static</b>.</p> <p>You can enter the file name or select <b>Browse</b>. You must enter the entire file path and file name. If you select <b>Browse</b>, the Staging Browser window appears.</p>
<b>Write Mode</b>	<p>The options are:</p> <p><b>Strings</b> — When selected, <b>New Line Delimiter</b>, <b>Max Characters Per Line</b> and <b>Max Lines</b> parameters become available.</p>  <p>In <b>Strings</b> mode, the runtime writes the data and then adds a <b>CRLF</b> (Line Feed 0x0A hexadecimal, 10 in decimal) or <b>CR</b> (Carriage Return 0x0D hexadecimal, 13 in decimal) as the line terminator depending on the runtime platform.</p> <p><b>Binary</b> — When selected, the runtime writes the data as bytes with no line or record terminator.</p>

	
<p><b>Append to File</b></p>	<p>The options are:</p> <p><b>True</b> — Append to the file if it exists. If the file does not exist, create a new file.</p> <p><b>False</b> — Overwrite the original file if it exists. If the file does not exist, create a new file.</p>
<p><b>New Line Delimiter</b></p>	<p>Available when <b>Write Mode</b> is <b>Strings</b>.</p> <p>The options are:</p> <p><b>CR</b> — Carriage return. The carriage return code (0x0D) is inserted as the new line delimiter after each string. This is normally used for a UNIX-based platform.</p> <p><b>CRLF</b> — Carriage return line feed. The line feed code (0x0A) is inserted as the new line delimiter after each string.</p>
<p><b>Max Characters Per Line</b></p>	<p>Available when <b>Write Mode</b> is <b>Strings</b>.</p> <p>The maximum number of characters to write. The maximum value is 2147483647.</p>
<p><b>Max Lines</b></p>	<p>Available when <b>Write Mode</b> is <b>Strings</b>.</p> <p>The maximum number of lines to write. The maximum value is 2147483647.</p>

## Input tab (Static and Strings)

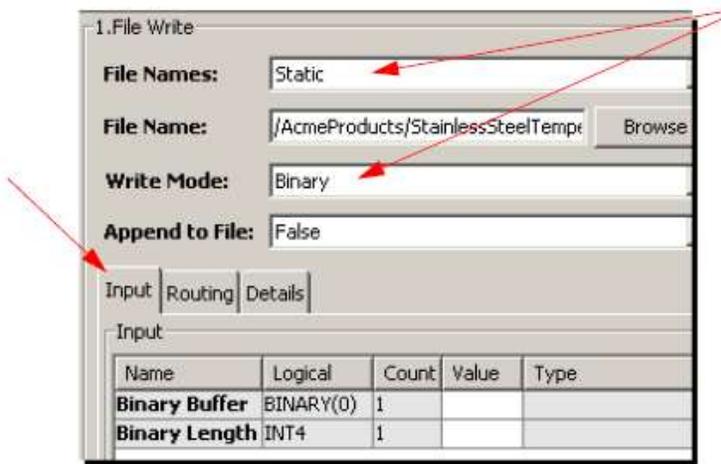
When the **Static** and **Strings** options are used:



Parameter	Description
String Array	The string array with the lines to write.
Number of Lines	The number of lines (number of array elements) to write.

## Input tab (Static and Binary)

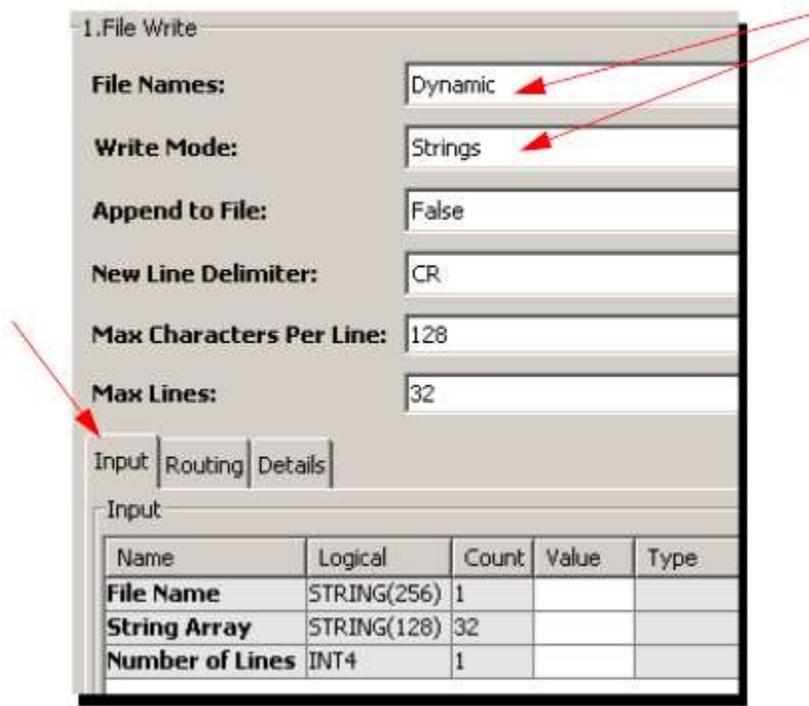
When the **Static** and **Binary** options are used:



Parameter	Description
<b>Binary Buffer</b>	The BINARY variable with the bytes to write
<b>Binary Length</b>	The number of bytes to write.

## Input tab (Dynamic and Strings)

When the **Dynamic** and **Strings** options are used:



Parameter	Description
<b>File Name</b>	The file in the Staging Browser area to write. This can be a constant or a variable. If specified in a variable, the file name can change during run time.
<b>String Array</b>	The string array with the lines to write.
<b>Number of Lines</b>	The number of lines (number of array elements) to write.

## Input tab (Dynamic and Binary)

When the **Dynamic** and **Binary** options are used:

1. File Write

**File Names:** Dynamic

**Write Mode:** Binary

**Append to File:** False

Input | Routing | Details

Input:

Name	Logical	Count	Value	Type
<b>File Name</b>	STRING(256)	1		
<b>Binary Buffer</b>	BINARY(0)	1		
<b>Binary Length</b>	INT4	1		

Parameter	Description
<b>File Name</b>	The file in the Staging Browser area to read. This can be a constant or a variable. If specified in a variable, the file name can change during run time.
<b>Binary Buffer</b>	The BINARY variable with the bytes to write
<b>Binary Length</b>	The number of bytes to write.

## IIoTA industrial IoT Platform: FTP

The **FTP** action provides Get, Put and Delete operations to use with a remote FTP server.

The **FTP** action provides the FTP client function. The remote FTP server can reside on any computer on the network, as long as the node can connect to and access the FTP server.

Files are transferred from and to the Staging Browser area of the node.

The Workbench Administration feature, Staging Browser tab provides file and directory operations for a node's Staging Browser area.

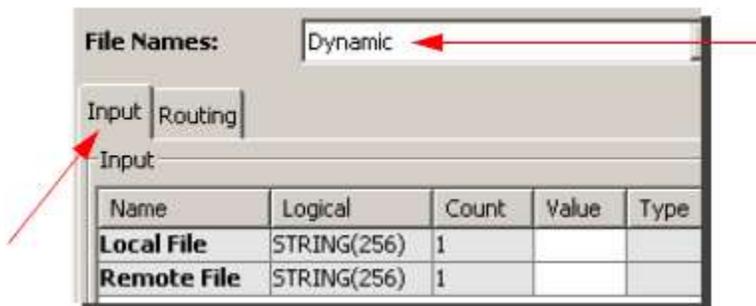
### Parameter description

Parameter	Description
<b>Operation Name</b>	<p>The options are:</p> <p><b>Put</b> - copies one file from the staging file system to a remote directory. Using <b>Put</b>, you can transfer a single file to the root directory, or any directory already created in the remote computer.</p> <p><b>Get</b> - copies one file from the remote directory to the staging file system. Using <b>Get</b>, you can transfer a single file from the remote computer to the root directory or any directory already created in the Staging Browser.</p> <p><b>Delete</b> - Removes a file in the current remote directory. Using <b>Delete</b>, you can delete a single file from the remote computer.</p>
<b>Server Address</b>	The IP address or hostname of the computer with the remote FTP server. If a hostname is used, a DNS must be able to resolve it to an IP address.
<b>Server Port</b>	The port number the FTP server listens to. The default is 21.
<b>Transfer Mode</b>	<p>Determines how the data connection is established. The options are:</p> <p><b>Active Mode</b> - Active mode places the responsibility of opening fire wall ports on the client computer and allows the server computer to make a connection from the outside.</p> <p><b>Passive Mode</b> - Passive mode puts the responsibility of opening up ports on the server computer side, allowing client computers with restrictive fire walls to connect.</p> <p><b>References:</b>            If you want to learn more about the FTP protocol, see <a href="http://www.ietf.org/rfc/rfc959.txt">http://www.ietf.org/rfc/rfc959.txt</a>.            For a discussion on active mode and passive mode, see <a href="http://slacksite.com/other/ftp.html">http://slacksite.com/other/ftp.html</a></p>
<b>Username</b>	The user name to use to access the FTP server.
<b>Password</b>	The password to use to access the FTP server.
<b>File Names</b>	<p>The options are:</p> <p><b>Static</b> — The <b>Local File</b> and <b>Remote File</b> parameters become available.</p> <p><b>Dynamic</b> — When you select <b>Dynamic</b>, an <b>Input</b> tab and parameters become available to specify the file names.</p>

<b>Local File</b>	Available when <b>File Names</b> is <b>Static</b> . The file that resides in the Staging Browser on this node. This is the source file for a <b>Put</b> , or the destination file for a <b>Get</b> . You can type the file name or select <b>Browse</b> . You must enter the entire path and file name. Wildcard characters such as an asterisk (*) are not allowed in the file name.
<b>Remote File</b>	Available when <b>File Names</b> is <b>Static</b> . The file that resides on the remote computer with the FTP server. This is the source file for a <b>Get</b> , or the destination file for a <b>Put</b> or a <b>Delete</b> .
<b>Append Timestamp to File</b>	Available when <b>Operation Name</b> is <b>Put</b> and <b>File Names</b> is <b>Static</b> . This option lets you add a timestamp to the end of a file name as follows:  <b>None</b> — Do not append a timestamp. <b>YYYYMMDD</b> — The output format would be year, month, and day. For example: 20080304 <b>YYYYMMDD-HHMMSS</b> — The output format would be year, month, day - hour minute, second. For example: 20080304-10:50:51

## Input tab

When you select the **Dynamic** option for the **File Names** parameter, an **Input** tab appears.



Parameter	Description
<b>Local File</b>	The file that resides in the Staging Browser on this node. This is the source file for a <b>Put</b> , or the destination file for a <b>Get</b> .
<b>Remote File</b>	The file that resides on the remote computer with the FTP server. This is the source file for a <b>Get</b> , or the destination file for a <b>Put</b> or a <b>Delete</b> .

Related topics  
Staging Browser

# IIoTA industrial IoT Platform: Local DB Export

The **Local DB Export** action exports the contents of a Local Database table to a comma-separated values (CSV) file in the Staging Browser area.

For information on exporting and importing the table definition, see Exporting and Importing Local Database tables.

1. Local DB Export

**Filename:**  ...

**Table:** AF2\_LET\_IPU\_EMAIL\_RESULTS ↻

**WHERE:**  ...

**Maximum Rows:**

**Delimiter:**

**Qualifier:**

**First Row Is Header:**

**Delete Exported Rows:**

Input **Output** Routing Details

Input

Name	Logical	Count	Value	Type
<b>Filename</b>	ANY	1		

?

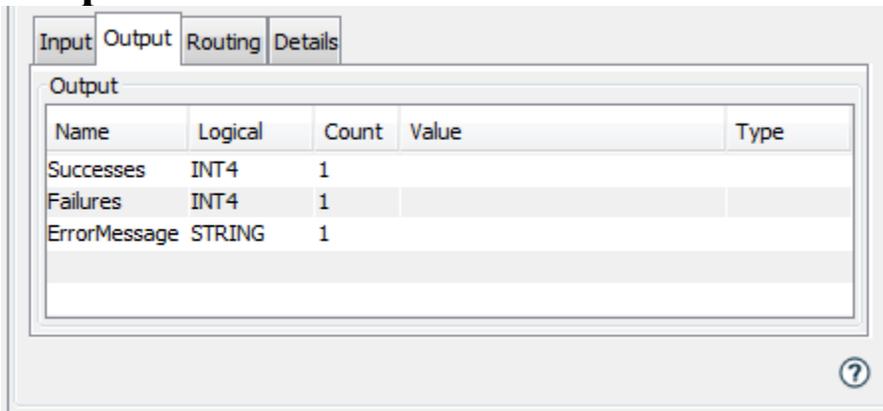
## Parameter description

Parameter	Description
<b>Filename</b>	<p>The name of the CSV file to write the exported data. If the file does not exist, it will be created.</p> <p>If the file already exists, it will be overwritten.</p> <p>The compound string feature and a substitution variable can be used in the <b>Filename</b> parameter to reference a variable that can be dynamically changed during runtime. For more information, see Using compound strings.</p>

	<p>For example, entering \$(filename) will result in a <b>filename</b> parameter in the Input tab that can reference any variable, whose contents can be changed at runtime.</p>
<b>Table</b>	The name of the Local Database table to export.
<b>Where</b>	<p>A Where clause can be used to restrict the rows included in the export.</p> <p>Both constants and substitution variables can be used in the Where clause.</p> <p>To construct a Where clause, use an operator (=, !=, &gt;, &gt;=, &lt;, &lt;=, like, is null, is not null ) to relate the column to either a constant or a substitution variable. Each of these operators can be combined with other operators using an <b>And</b> or <b>Or</b> statement.</p> <p>To use a substitution variable, insert \$(x) where x is a variable that will be associated to a variable on the <b>Input</b> tab (see the <b>Input</b> tab below).</p> <p><b>Note:</b> For columns of type TEXT, the constant or substitution variable must be enclosed in double quotes. For example:</p> <p>C01 = "JohnDoe"                  The constant is enclosed in double quotes.                  C01 = "\$(test)"                  The substitution variable is enclosed in double quotes. The variable (test in this example) will be added to the Input tab and can then be mapped to a STRING variable (or a constant).</p> <p>The where clause builder, accessed by selecting the icon , can be used to assist in building the where clause.</p>
<b>Maximum Rows</b>	The maximum number of rows to export.
<b>Delimiter</b>	The character to use to separate the column values in the CSV file.
<b>Qualifier</b>	<p>The character to use around column values to escape reserved characters such as commas or line breaks.</p> <p>The following shows example column data from a CSV file whose columns are identified by the qualifier " and each column is delimited by a comma.</p>

	<div style="text-align: center;"> <p><b>Qualifier</b></p> </div> <pre> 2008,Ford,750E,"ac, abs, moon",30000.00 (CRLF) 2007,Jeep,Grand Cherokee,"MUST SELL! air, moon roof, loaded",4799.00                     </pre> <p style="text-align: center;"><b>Delimiter</b></p> <p>Note that CRLF refers to carriage return line feed. Also note that when specifying a qualifier all values in the table column must be surrounded by quotes, not just the reserved character.</p>
<p><b>First Row is Header</b></p>	<p>An option to insert a header row in the CSV file. The table's column names will be used as the column names in the CSV file. The parameter can be set to <b>True</b> or <b>False</b>.</p>
<p><b>Delete Exported Rows</b></p>	<p>An option to delete the rows from the Local Database table once they are exported. The parameter can be set to <b>True</b>, <b>False</b>, or <b>On Complete Success</b>.</p>

## Output tab



Parameter	Description
<b>Successes</b>	The number of rows that were successfully exported.
<b>Failures</b>	The number of rows that could not be exported due to a Local Database error. This Output parameter does not return the number of rows that were not exported due to the <b>Maximum Rows</b> parameter being reached.

<b>ErrorMessage</b>	An error message string if a Local Database error is encountered and the Failure route is taken.
---------------------	--

Related topics

Exporting and Importing Local Database tables

Local DB Select

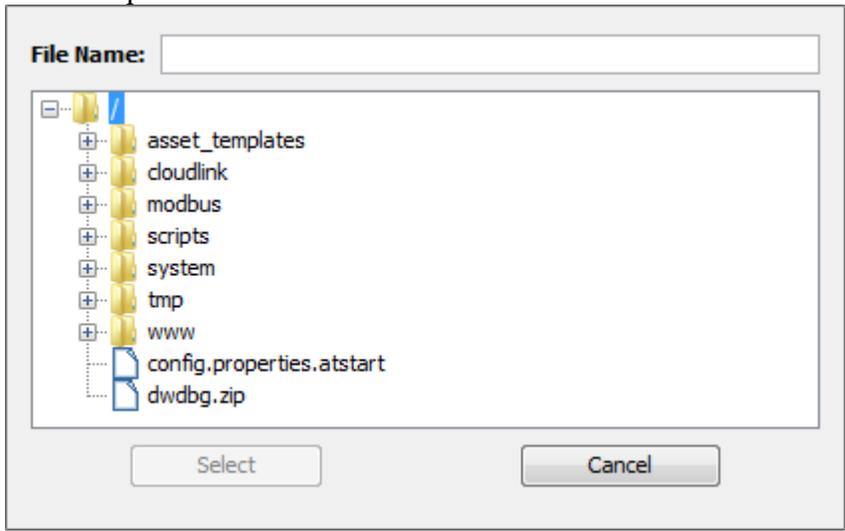
## IIoTA industrial IoT Platform: Local DB Import

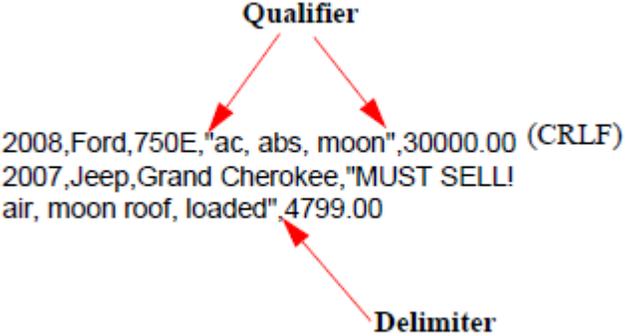
The **Local DB Import** action imports data from a CSV file in the Staging Browser area to a Local Database table.

The data imported from the CSV file is added to the Local Database table. Any existing data already in the Local Database table is not replaced or overwritten or removed.

For information on exporting and importing the table definition, see Exporting and Importing Local Database tables.

### Parameter description

Parameter	Description
<b>Filename</b>	<p>The name of the CSV file that has the data to import. You can enter the file name or select <b>Browse</b>. You must enter the entire file path and file name. If you select <b>Browse</b>, the Staging Browser window appears.</p> <p>For example:</p> 

	<p>The compound string feature and a substitution variable can be used in the <b>Filename</b> parameter to reference a variable that can be dynamically changed during runtime. For more information, see Using compound strings. For example, entering \$(filename) will result in a <b>filename</b> parameter in the Input tab that can reference any variable, whose contents can be changed at runtime.</p>
<p><b>Table</b></p>	<p>Select the Local Database table drop-down list. The table must have been previously been defined. The number of columns and type of data in the CVS file must match the number of columns and type of data in the Local Database table. For example:</p> 
<p><b>Delimiter</b></p>	<p>The character used to separate the column values in the CSV file. The default value is the comma character. The character to use around text that contains reserved characters such as commas or line breaks. The default value is the double quotation mark (").</p>
<p><b>Qualifier</b></p>	<p>The character to use around text that contains reserved characters such as commas or line breaks. The default value is the double quotation mark ("). The following shows example column data from a CSV file whose columns are identified by the qualifier " and each column is delimited by a comma.</p> <div style="text-align: center;"> <p><b>Qualifier</b></p>  </div> <p>2008,Ford,750E,"ac, abs, moon",30000.00 (CRLF)  2007,Jeep,Grand Cherokee,"MUST SELL!  air, moon roof, loaded",4799.00</p> <p>Note that CRLF refers to carriage return line feed.</p>
<p><b>First Row is Header</b></p>	<p>The parameter can be set to <b>True</b> or <b>False</b>. Indicates if the first row in the CSV file is a header row. If <b>True</b>, the first row in the CSV file will be treated as a header row. In this case, the data value in each column in the header row must match a column name in the Local Database table. If this requirement is not met, the rows in the CSV file will not be imported into the Local Database table. If <b>False</b>, the CSV file is considered not to have a header row. The data values from each row in the CSV file are imported into the Local Database table in the same order as the column definition order.</p>

## Output tab

Name	Logical	Count	Value	Type
Successes	INT4	1	Local CPU 1.D[10]	INT4
Failures	INT4	1	Local CPU 1.D[12]	INT4

Parameter	Description
<b>Successes</b>	The number of rows that were successfully imported.
<b>Failures</b>	The number of rows that could not be imported.

## Local DB Import CSV file considerations

Because legacy systems and enterprise applications that generate CSV files can use varying formats that may or may not conform to any specific standard, the CSV files that are imported should be reviewed for any formatting or discrepancies that might cause import problems.

The Local DB Import action attempts to handle varying formats of CSV files, including support of the following:

- Rows ending with a "/r/n"
- Rows ending without a "/r/n"
- Rows ending with a space
- Rows that contain a percent sign (%)
- Rows that contain a single quote ( ' )
- Rows that begin with an empty value
- Some rows that are separated by carriage returns, some that are not
- Rows that starts with the qualifier ( " )
- Rows that fail to import are handled, and then the importing attempts to continue with the remaining rows
- Row that contains data that represents a TIMESTAMP data type
- Import into a table with an auto-increment column from a CSV file that does not contain the data for the column.

Related topics

Exporting and Importing Local Database tables

# IIoTA industrial IoT Platform: Directory Operation

The **Directory Operation** action provides Create and Delete operations for directories in the Staging Browser area of the node.

## Parameter description

1.Directory Operation

**Operation:** Create

**Directory Definition:** Static

**Directory Name:**

Parameter	Description
<b>Operation</b>	<p>The options are:</p> <p><b>Create</b> - Create a directory in the Staging Browser area of the node.  <b>Delete</b> - Delete a directory in the Staging Browser area of the node.</p>
<b>Directory Definition</b>	<p>The options are:</p> <p><b>Static</b> - Selecting this option will cause the <b>Directory Name</b> parameter to become available.  <b>Dynamic</b> - The <b>Input</b> tab becomes available with a parameter to specify the directory name.</p>
<b>Directory Name</b>	<p>Available when the <b>Static</b> option is selected in the <b>Directory Definition</b> parameter. Enter the name of the directory that will either be created or deleted.</p>

## Input tab

The Input tab appears when you select the **Dynamic** option in the **Directory Definition** parameter.

1.Directory Operation

**Operation:** Create

**Directory Definition:** Dynamic

Input Output Routing Details

Input

Name	Logical	Count	Value	Type
Name	STRING(256)	1		

Parameter	Description
Name	The name of the directory.

## Output tab

1.Directory Operation

**Operation:** Create

**Directory Definition:** Static

**Directory Name:**

Output Routing Details

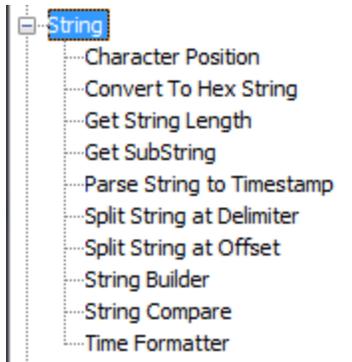
Output

Name	Logical	Count	Value	Type
resultStatus	INT4	1		

Parameter	Description
resultStatus	The result of the create or delete directory operation.

## IIoTA industrial IoT Platform: String

The **String** category provides actions that act on STRING data type variables.

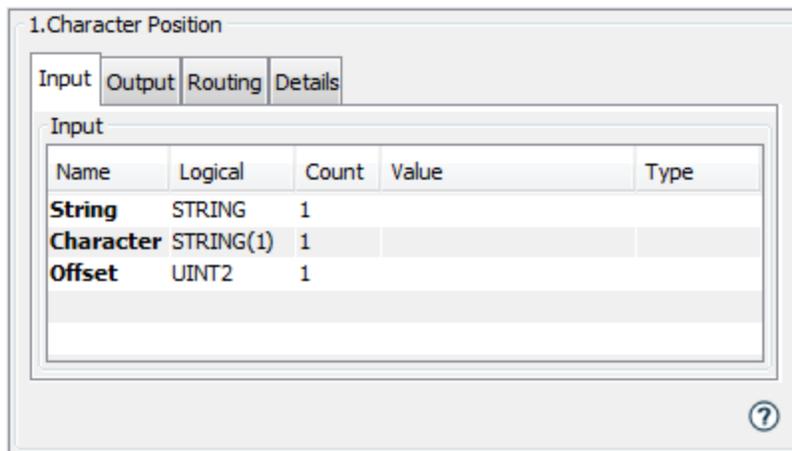


The **String** category provides these actions:

- Character Position
- Convert to Hex String
- Get String Length
- Get SubString
- Parse String to Timestamp
- Split String at Delimiter
- Split String at Offset
- String Builder
- String Compare
- Time Formatter

## IIoTA industrial IoT Platform: Character Position

The **Character Position** action returns the position of a character within a string.



The Character Position action is part of the Technology Preview Extension

This action is part of the Technology Preview Extension, which is also referred to as the sandbox package. The action will be found in the **String** trigger action category upon successful deployment of the sandbox package

See Technology Preview Extension for information on obtaining and installing the extension.

## Input tab

Parameter	Description
<b>String</b>	The string value that will be examined for the specific character.
<b>Character</b>	The character to try to find in the string value.
<b>Offset</b>	The offset within the string to start the search for the character.

## Output tab

Parameter	Description
<b>Location</b>	The offset within the string where the character was found. A -1 indicates that the character was not found.

For example:

- Given the **String** "Hello World", the **Character** "o", and an **Offset** of 0 (zero), the action will return the a value of 4 in the Output parameter **Location**.
- Using the action again with the same **String**, **Character**, and an **Offset** of 5 will return the a value of 7 in the Output parameter **Location**.
- Using the action a third time with the same **String**, **Character**, and an **Offset** of 8 will return a value of -1 in the Output parameter **Location**, indicating that the character doesn't exist in the string.

# IIoTA industrial IoT Platform: Convert to Hex String

The **Convert To Hex String** action converts the decimal value of an integer into its hexadecimal equivalent. That value will then be converted into a string.

1. Convert To Hex String

Input Data Type:

Input Output Routing Details

Input

Name	Logical	Count	Value	Type
Value	UINT2	1		

The Convert to Hex String action is part of the Technology Preview Extension

This action is part of the Technology Preview Extension, which is also referred to as the sandbox package. The action will be found in the **String** trigger action category upon successful deployment of the sandbox package

See Technology Preview Extension for information on obtaining and installing the extension.

## Parameter description

Parameter	Description
<b>Input Data Type</b>	The data type of the data to be converted. The options are: UINT1 – 1 byte unsigned integer UINT2 – 2 byte unsigned integer UINT4 – 4 byte unsigned integer

## Input tab

Parameter	Description
<b>Value</b>	The source that contains a decimal numeric value. This can be a variable or be a constant string.

## Output tab

Parameter	Description
<b>Result</b>	The hexadecimal value of the source decimal value. This is returned as a string.

# IIoTA industrial IoT Platform: Get String Length

The **Get String Length** action returns the length of the data in a STRING data type variable.

1. Get String Length

Failure on Empty String:

Input Output Routing Details

Input

Name	Logical	Count	Value	Type
String	STRING	1		

## Parameter description

Parameter	Description
<b>Failure on Empty String</b>	<p>An option to control the Route when an empty string is provided as the String parameter in the Input tab.</p> <p>False – The default. The Success Route will be taken when an empty string is provided as the String parameter in the Input tab.</p> <p>True - When an empty string is provided as the String parameter in the Input tab, the action's Failure Route will be taken. This allows your routing logic to check for an empty string without having to compare the Output tab Length parameter for a value of zero.</p>

## Input tab

Parameter	Description
<b>String</b>	The variable whose length you want to retrieve.

## Output tab

Parameter	Description
<b>Length</b>	The variable where the length of the string variable is returned.

## IIoTA industrial IoT Platform: Get SubString

The **Get SubString** action returns a portion of a string. The starting and ending offsets within the string determine the portion returned.

1.Get SubString

Input Output Routing Details

Input

Name	Logical	Count	Value	Type
<b>String</b>	STRING	1		
<b>Start Offset</b>	UINT4	1		
<b>End Offset</b>	UINT4	1		

The Get SubString action is part of the Technology Preview Extension

This action is part of the Technology Preview Extension, which is also referred to as the sandbox package. The action will be found in the **String** trigger action category upon successful deployment of the sandbox package,

See Technology Preview Extension for information on obtaining and installing the extension.

### Input tab

Parameter	Description
<b>String</b>	The string value that will be the source of the substring. This can be a variable or be a constant string.
<b>Start Offset</b>	The offset into the string value to begin the substring. To begin with the first character in the string this value should be 1. This value can be a variable or be a constant string.
<b>End Offset</b>	The offset into the string that will contain the final character in the substring. This value can be a variable or be a constant string.

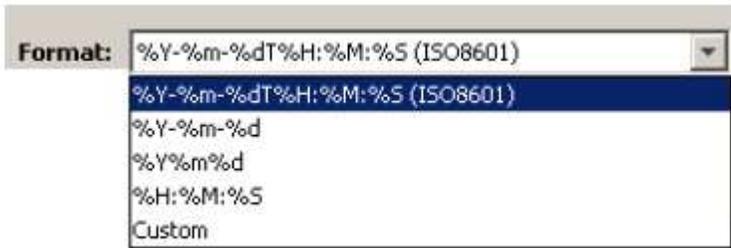
## Output tab

Parameter	Description
<b>Result String</b>	The variable that will contain the substring created from the source string.

## IIoTA industrial IoT Platform: Parse String to Timestamp

The **Parse String to Timestamp** action formats a STRING variable into a TIMESTAMP variable in several standard formats, including custom formats.

### Parameter description

Parameter	Description
<b>Format</b>	<p>This is the format in which you want the date and time information to appear. Select a format from the list.</p>  <p>The options are:</p> <ul style="list-style-type: none"> <li><b>Y-m-dTH:M:S</b> - Complete date plus hours, minutes and seconds. For example: 2008-04-22T19:20:30 As specified in ISO 8601.</li> <li><b>Y-m-d</b> - Complete date. For example: 2008-04-22</li> <li><b>Ymd</b> - Complete date. For example: 20080422</li> <li><b>H:M:S</b> - Hour, minute, second. For example: 11:36:06</li> <li><b>Custom</b> - Note to programmers: The custom format uses the C-runtime strftime() function under the covers, so the format string must be a valid</li> </ul>

	format string for strftime(). For more information, see the section below: <b>strftime function formats.</b>
<b>Use Advanced Properties</b>	Select this check box to turn on the <b>Array Size</b> parameters
<b>Array Size</b>	Used when there is an array of data type STRING that must be converted to an array of TIMESTAMP. If the input variable is a single STRING variable, enter the size of 1.

**Input tab**

Parameter	Description
<b>String</b>	The input STRING with the date and time formatting specified by the <b>Format</b> parameter to convert to a TIMESTAMP.
<b>Count to Parse</b>	The number of items in the array to format (this can be less than the maximum array size).

**Output tab**

Parameter	Description
<b>Output</b>	The output TIMESTAMP variable.

**Adding a Custom format**

The **Time Formatter** action also lets you define a **Custom** format for the input string. For information on the Custom format option, see Time Formatter.

**IIoTA industrial IoT Platform: Split String at Delimiter**

The **Split String at Delimiter** action is used to break up a string into an array of strings, or tokens, based on the existence of a delimiter within the string. An example use of this action would be to parse a comma delimited string into an array of strings.

5.Split String at Delimiter

**Maximum Tokens:**

**Token Length:**

Input Output Routing Details

Input

Name	Logical	Count	Value	Type
<b>String</b>	STRING	1		
<b>Delimiters</b>	STRING	1		

?

The Split String at Delimiter action is part of the Technology Preview Extension

This action is part of the **Technology Preview Extension**, which is also referred to as the sandbox package. The action will be found in the **String** trigger action category upon successful deployment of the sandbox package,

See Technology Preview Extension for information on obtaining and installing the extension.

## Parameters

Parameter	Description
<b>Maximum Tokens</b>	The maximum number of smaller strings, or tokens, to create.
<b>Token Length</b>	The maximum length that can be allocated to each token.

## Input tab

Parameter	Description
<b>String</b>	The string value to be parsed into smaller strings.
<b>Delimiters</b>	The string that contains the delimiter value.

## Output tab

Parameter	Description
<b>Tokens</b>	A string array that will hold the smaller strings parsed from the input string. The length of each string is defined by the <b>Token Length</b> parameter. The size of the array is determined by the <b>Maximum Tokens</b> parameter.

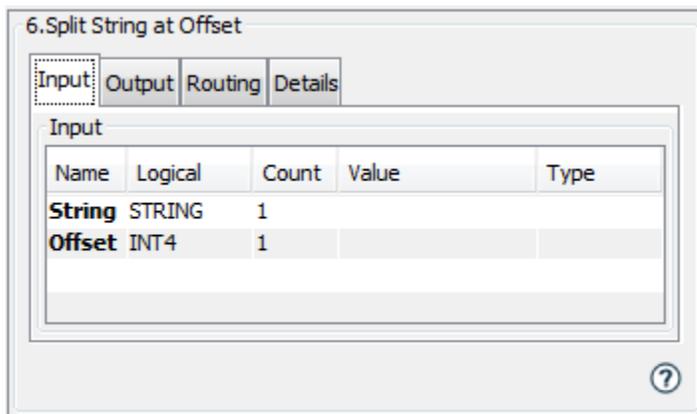
Tokens Count	The actual number of smaller strings that were created upon completion of the action. This value can be less than or equal to the value defined in the <b>Maximum Tokens</b> parameter.
--------------	---

For example:

- Given the **String** "Red,Green,Blue,Yellow,Purple,Orange", the **Delimiters** "," (comma), a **Maximum Tokens** value of 4, and an **Token Length** of 3, the action will return the values: "Red", "Gre", "Blu", and "Yel" in the Output parameter **Tokens** and a value of 4 in the **Tokens Count** parameter.
- Using the action again with the same parameters, except with a **Delimiters** value of "." (period), the action will return the value of "Red" and a value of 1 in the **Tokens Count** parameter. Only 1 token was created because the "." **Delimiters** value was not found in the string. The first three characters of the **String** value were used because the **Token Length** value was set to 3.
- Using the action a third time with the same parameters, except with a **Delimiters** value of "e" returns 4 values in the **Tokens** variable: "R", "d,Gr", "n,Blu", ",Y". The **Tokens Count** variable will contain a value of 4.

## IIoTA industrial IoT Platform: Split String at Offset

The **Split String at Offset** action is used to return two strings from a source string at a specific offset within the source string.



The Split String at Offset action is part of the Technology Preview Extension

This action is part of the Technology Preview Extension, which is also referred to as the sandbox package. The action will be found in the **String** trigger action category upon successful deployment of the sandbox package,

See Technology Preview Extension for information on obtaining and installing the extension.

## Input tab

Parameter	Description
<b>String</b>	The source string.
<b>Offset</b>	The location in the string that marks the end point of the <b>LeftString</b> and the begin point of <b>RightString</b> .

## Output tab

Parameter	Description
<b>LeftString</b>	The string created from the beginning of the <b>String</b> value to the <b>Offset</b> value.
<b>RightString</b>	The string created from the character after the <b>Offset</b> value to the end of the <b>String</b> value.

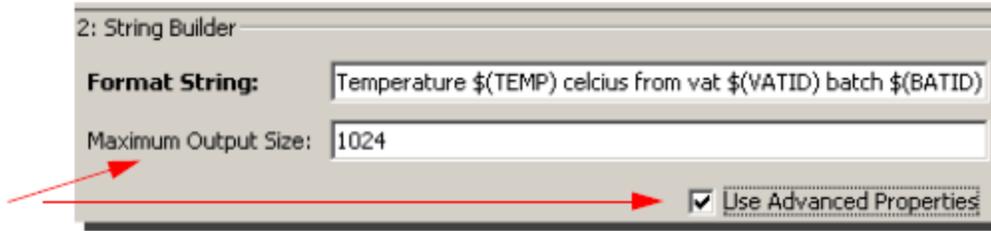
For example:

- Given the **String** "the quick brown fox jumped over the lazy dog", the **Offset** 10, the action will return the a value of "the quick " in the Output parameter **LeftString** and a value of "brown fox jumped over the lazy dog" in **RightString**.
- Using the action again with the same **String** and an **Offset** of 100 will result in an error because the **Offset** value is exceeds the size of the **String**.
- Using the action a third time with the same **String** and an **Offset** of 0 will return the an empty string value in the Output parameter **LeftString** and a value of "the quick brown fox jumped over the lazy dog" in **RightString**.

# IIoTA industrial IoT Platform: String Builder

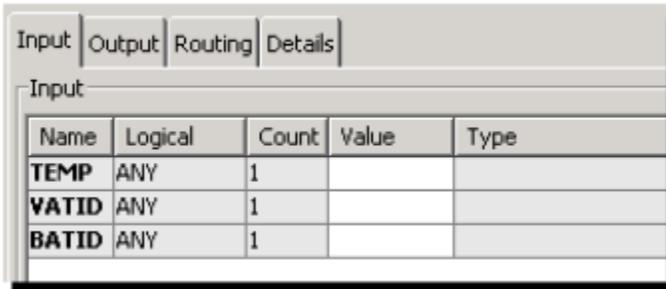
The **String Builder** action builds a string from multiple constants and variables. This can be used to build (or concatenate) a single output string from multiple strings variables and constants.

## Parameter description



Parameter	Description
<b>Format String</b>	Enter the set of text and substitution variables that are used to build the single output string. The <b>Input</b> tab appears when you type the first substitution variable in the form of \$(). For example: \$(TEMP). A row is added to the <b>Input</b> tab for each substitution variable, in this example TEMP. If you want to include a \$ character as part of the text, you must use double \$\$ characters. Otherwise, the \$ character will not appear in the output. For example: \$\$5.75.
<b>Maximum Output Size</b>	Available when the <b>Use Advanced Properties</b> checkbox is selected. Defaults to 1024. This parameter controls the maximum size of the <b>Output</b> parameter on the <b>Output</b> tab.

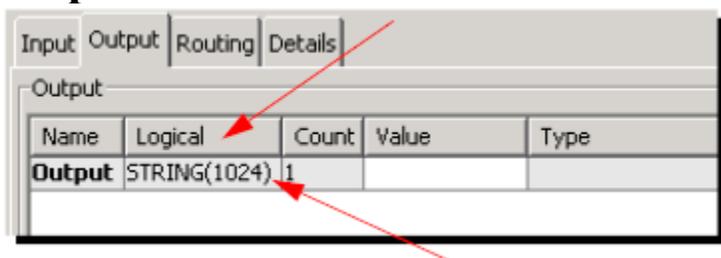
## Input tab



The **Input** tab appears when you type the first substitution variable in the form of \$() in the **Format String** parameter. The **Input** tab lets you map substitution variables to variables or constants.

Parameter	Description
<b>Name</b>	<p>A row is added to the <b>Input</b> tab for each substitution variable, in the example TEMP, VATID and BATID would be the added rows.</p> <p>The substitution variable is mapped to any trigger macro, constant, started device variable, trigger local variable, trigger static variable or trigger event variable by selecting the <b>Value</b> cell for the row. Once the cell is selected, a drop-down list of the available items is displayed.</p>

## Output tab



Parameter	Description
<b>Output</b>	<p>The output variable that receives the result of the multiple input parameters and text concatenated into a single string.</p> <p>For the example Format String: Temperature \$(TEMP) Celsius from vat \$(VATID) batch \$(BATID)</p> <p>Where:</p> <ul style="list-style-type: none"> <li>TEMP is mapped to a device variable with the value of 20</li> <li>VATID is mapped to a device variable with the value of 100</li> <li>BATID is mapped to a device variable with the value of 2</li> </ul> <p>Then the output string would be: Temperature 20 Celsius from vat 100 batch 2.</p>

## Supported control characters

The String Builder action supports control characters for the output of the action. The following control character patterns will be accepted:

Pattern	Result
\t	The tab character
\r	The line feed character

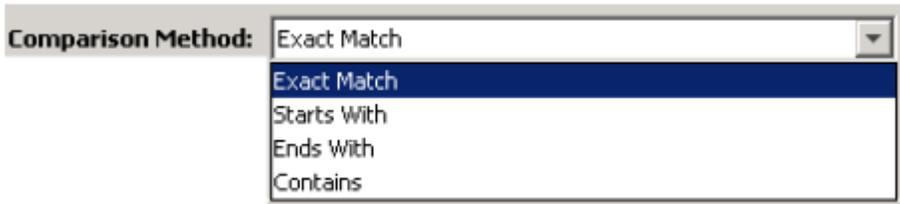
\n	The new line character (0x0A)
\xNN	Any character specified in hexadecimal notation, for example \x35 will print the number 5. Warning: use of \x00 to specify an embedded null should not be used.
\0	Null - not supported for embedded nulls. String functions that use the length of the string when processing will not give the expected results.
\\	Places a backslash \ character

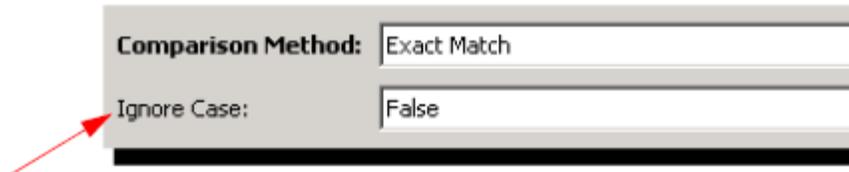
## IIoTA industrial IoT Platform: String Compare

A **String Compare** action compares strings based on a specific criterion such as exact match.

### Parameter description

This action compares all or part of a string with another string. You must specify whether to use case sensitivity and the number of characters in the string to compare.

Parameter	Description
<b>Comparison Method</b>	<p>The comparison criteria.</p>  <p>The options are:</p> <p><b>Exact Match</b> — The input string must exactly match the search string.  <b>Starts With</b> — The input string must start with the search string.  <b>Ends With</b> — The input string must end with the search string.  <b>Contains</b> — The input string must contain the search string.</p>
<b>Ignore Case</b>	Indicates if the comparison is case sensitive.



The options are:

**True** — Do not apply case-sensitivity. The string comparison is insensitive to uppercase and lowercase letters.

**False** — Apply case sensitivity. The string must exactly match uppercase and lowercase letters.

## Input tab

Input				
Name	Logical	Count	Value	Type
<b>String</b>	STRING(0)	1		
<b>Search String</b>	STRING(0)	1		

Parameter	Description
<b>String</b>	The input string, which will be compared to based upon the <b>Comparison Method</b> and the <b>Search String</b> parameter.
<b>Search String</b>	The pattern to match or find in the <b>String</b> parameter.

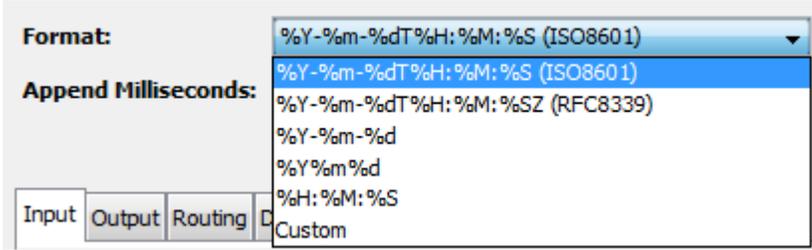
## Routing tab

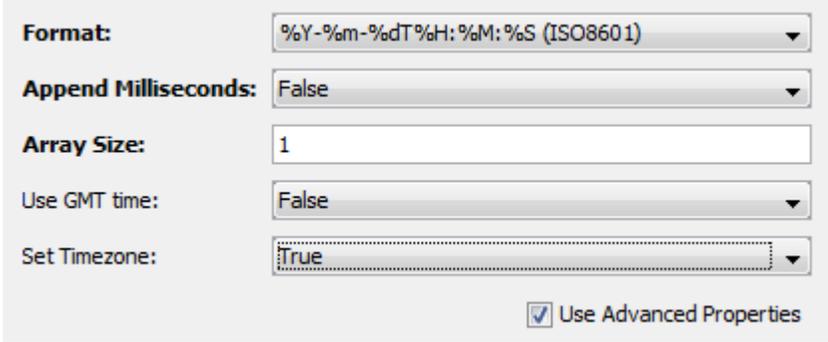
Parameter	Description
<b>True</b>	Route to take when the comparison is successful (pattern found based on the <b>Comparison Method</b> ) in the <b>String</b> parameter.
<b>False</b>	Route to take when the comparison is not successful (pattern is not found based on the <b>Comparison Method</b> ) in the <b>String</b> parameter.
<b>Failure</b>	Route when there is a failure.

# IIoTA industrial IoT Platform: Time Formatter

The **Time Formatter** action formats a **TIMESTAMP** variable into a **STRING** variable in several standard formats, including custom formats.

## Parameter description

Parameter	Description
<b>Format</b>	<p>This is the output format in which you want the date and time information to appear. Select a format from the list.</p>  <p>The options are:</p> <p><b>Y-m-dTH:M:S</b> – Complete date plus hours, minutes and seconds. For example: 2008-04-22T19:20:30 As specified in ISO 8601.</p> <p><b>Y-m-dTH:M:SZ</b> – Complete date plus hours, minutes, seconds and time zone. For example: 2008-04-22T19:20:30Z or 2008-04-22T19:20:30-04:00 As specified in RFC 8339.</p> <p><b>Y-m-d</b> – Complete date. For example: 2008-04-22</p> <p><b>Ymd</b> – Complete date. For example: 20080422</p> <p><b>H:M:S</b> – Hour, minute, second. For example: 11:36:06</p> <p><b>Custom</b> – Note to programmers: The custom format uses the C-runtime <code>strftime()</code> function under the covers, so the format string must be a valid format string for <code>strftime()</code>. For more information, see the section below: <b>strftime function formats</b>.</p>
<b>Append Milliseconds</b>	<p>The options are:</p> <p><b>True</b> – A period following by the three-digit millisecond value (for example .334) will be appended to the end of the formatted output string.</p>

	<p><b>False</b> – Milliseconds will not be appended to the end of the formatted output string. This is the default.</p>
<p><b>Use Advanced Properties</b></p>	<p>Select the <b>Use Advanced Properties</b> check box to display additional parameters: <b>Array Size</b>, <b>Use GMT time</b> and <b>Set Timezone</b>.</p> <p>The <b>Set Timezone</b> parameter will appear when the <b>Use GMT time</b> is set to <b>False</b>, and when the Staging Browser folder <code>/system/zoneinfo</code> exists.</p> 
<p><b>Array Size</b></p>	<p>Used when there is an array of data type <b>TIMESTAMP</b> that must be converted to an array of <b>STRING</b>. If the input variable is a single <b>TIMESTAMP</b> variable, enter the size of 1.</p>
<p><b>Use GMT time</b></p>	<p>Used to specify that the output variable will be in Greenwich Mean Time (GMT). The options are:</p> <p><b>True</b> – Format the output variable <b>STRING</b> in GMT. <b>False</b> – Do not format the output variable <b>STRING</b> in GMT. The output variable <b>STRING</b> will be in the timezone configured for the gateway. This is the default.</p>
<p><b>Set Timezone</b></p>	<p>When the <b>Use GMT time</b> is set to <b>False</b>, and when the Staging Browser folder <code>/system/zoneinfo</code> exists, the <b>Set Timezone</b> parameter will appear. The options are:</p> <p><b>True</b> – Format the output variable <b>STRING</b> using the alternate timezone specified in the <b>Input</b> tab, <b>Timezone</b> parameter. When this option is set to <b>True</b>, the <b>Input</b> tab, <b>Timezone</b> parameter becomes visible. <b>False</b> – Do not format the output variable <b>STRING</b> in an alternate timezone. The output variable <b>STRING</b> will be in the timezone configured for the gateway. This is the default.</p>

## Time zone files

When using an alternate time zone, meaning not the time zone configured for the gateway and not GMT; time zone files must be present in the Staging Browser /system/zoneinfo/ folder.

These time zone files can be an entire set of files for many time zones, or a subset of files for only the few time zones that apply to your gateways and your solution.

## Input tab

The Input tab when the **Set Timezone** parameter is **False**, or otherwise not available:

Name	Logical	Count	Value	Type
<b>Timestamp</b>	TIMESTAMP	1		
Count To Parse	UINT2	1		

The Input tab when the **Set Timezone** parameter is **True**:

Name	Logical	Count	Value	Type
<b>Timestamp</b>	TIMESTAMP	1		
Count To Parse	UINT2	1		
<b>Timezone</b>	STRING	1		

Parameter	Description
<b>Timestamp</b>	The input TIMESTAMP to convert to a String
<b>Count to Parse</b>	The number of items in the array to format (this can be less than the maximum array size).
<b>Timezone</b>	Used to specify the Time zone in which you want the date and time information to appear. This field is only enabled when the <b>Set Timezone</b> field is set to <b>True</b> . The filename or path to the time zone file relative to Staging Browser /system/zoneinfo/ folder. The input must be a String format. For example:

	<p>America/New_York  America/Indiana/Indianapolis  America/Sao_Paulo  America/Toronto</p> <p>Europe/London</p> <p><b>Note:</b> When specifying this input field and selecting a format that contains the %Z (custom or RFC8339), the output string will present the time zone as “Z or GMT” even though the time will be in the specified time zone.</p>
--	--

## Output tab

Name	Logical	Count	Value	Type
Output	STRING(64)	1		

Parameter	Description
<b>Output</b>	The output STRING variable with the date and time formatting specified by the <b>Format</b> parameter. Specifying a string that is greater than 64 characters long guarantees that all string formats can be accommodated.

## Exporting a trigger with reference to a time zone file

When you export a trigger or do a Back Up using the Workbench -> Back Up function, the time zone file referenced by the trigger Time Formatter action is not included in the export (or Back Up).

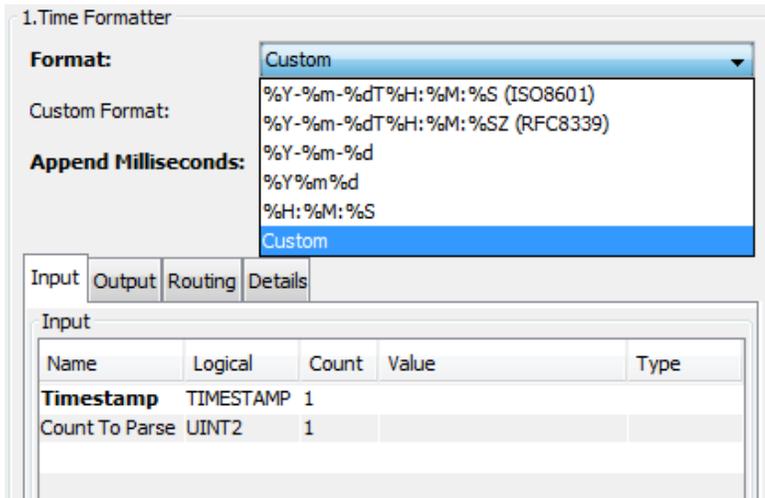
The time zone files must be copied onto any new node, in the Staging Browser /system/timezone folder, that gets the import of the exported trigger (or a Restore from the Back Up).

## Adding a Custom format

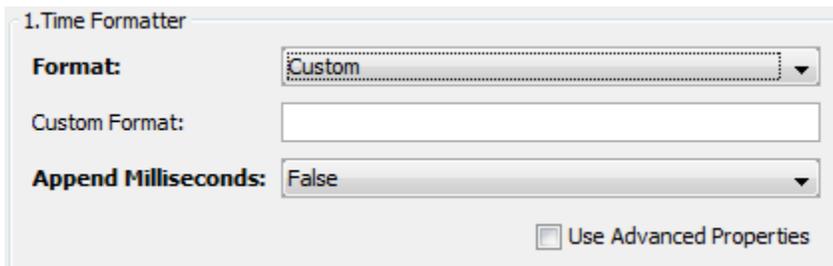
The **Time Formatter** action also lets you define a **Custom** format for the output string. A maximum of 9 format codes can be defined in the custom format.

You can add a **Custom** format as follows:

1. From the bottom of the **Actions** tab, select **Add**.  
The New Action window appears.
2. Expand the **String** category, select **Time Formatter**, and then select **Add**.  
The right pane changes to accommodate a **Time Formatter** action.



3. Next to **Format**, select the down-arrow, and then select **Custom**.  
The **Custom Format** box becomes available.



The **Custom** format will be a string that contains format codes for the strftime() function to replace with runtime values.

4. In the **Custom Format** box, type the format to use for the timestamp output variable.

## strftime function formats

The strftime() function converts the date and time information to a format specified by each format code and then stores the result in a string (up to the maximum size of characters).

**Note:** You must supply valid format codes for the C runtime library `strftime()` function and operating system environment. Incorrect format codes may cause runtime problems.

The following format codes can be used:

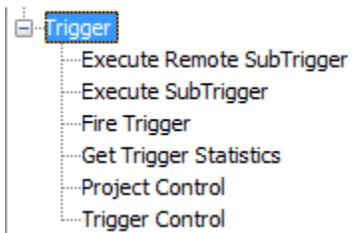
Code	Replaced by	Example
%a	The abbreviated weekday name. *	Fri
%A	The full weekday name. *	Friday
%b	The abbreviated month name. *	Oct
%B	The full month name. *	October
%c	The standard date and time string.	Thu Aug 23 14:55:02 2007
%d	The day of the month as a number (01-31).	23
%H	The hour in 24-hour format (00-23).	14
%I	The hour in 12-hour format (01-12).	02
%j	The day of the year as a number (001-366).	235
%m	The month as a number (01-11).	08
%M	The minute as a number [00-59].	55
%p	The locale's equivalent of a.m. or p.m.	PM
%S	The second as a number (00-59).	02
%U	The week of the year (Sunday as the first day of the week) as a number (00-53).	33
%w	The weekday as a number (0-6), with 0 representing Sunday.	4
%W	The week number of the year (Monday as the first day of the week) as a number (00-53). All days in a new year preceding the first Monday are considered to be in week 0.	34
%x	The standard date representation. *	08/23/07
%X	The standard time representation. *	14:55:02
%y	The year in decimal without the century (00-99).	01
%Y	The year in decimal with the century.	2008

%Z	The time zone name or abbreviation.	CDT
%%	A percent (%) sign.	%

\* Indicates Locale dependent.

## IIoTA industrial IoT Platform: Trigger

The **Trigger** category provides actions to control projects and triggers.



The **Trigger** category provides these actions:

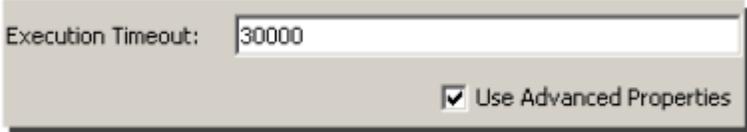
- Execute Remote SubTrigger
- Execute SubTrigger
- Fire Trigger
- Get Trigger Statistics
- Project Control
- Trigger Control

## IIoTA industrial IoT Platform: Execute Remote SubTrigger

The **Execute Remote SubTrigger** action executes a subtrigger defined on another node. This action operates much the same as **Execute SubTrigger** except the subtrigger executes on a remote node.

A PeerLINK device contains the connection information to the remote node where the subtrigger is defined.

### Parameter description

Parameter	Description
<b>Device Name</b>	The PeerLINK device that provides the connection information to a remote node.
<b>Project Name</b>	The project on the remote node that contains the subtrigger to execute.
<b>SubTrigger Name</b>	The subtrigger on the remote node to execute.
<b>Use Advanced Properties</b>	<p>When selected, this check box provides an execution timeout parameter.</p>  <p>The <b>Execution Timeout</b> parameter provides a time period in milliseconds to wait for the remote subtrigger to complete its execution.</p>

## IIoTA industrial IoT Platform: Execute SubTrigger

The **Execute SubTrigger** executes a subtrigger that is defined on the local node. The trigger that is the target of this action must be a **SubTrigger** event type trigger.

The **Fire Trigger** action is used to execute an **On-Demand** event type trigger.

Both **On-Demand** and **SubTrigger** event type triggers can use all of the available actions on the node.

- Subtriggers can define input and output variables pass data between the calling **Execute SubTrigger** action and the subtrigger.

For information on defining SubTrigger event type triggers, see SubTrigger

- On-Demand triggers can be executed by the calling Fire Trigger action and by the Workbench from the triggers project window.

For information on defining On-Demand event type triggers, see On-Demand

- Both can be executed with an option to wait for its completion.

Because subtriggers can define input and output parameters, they are very useful for defining common application logic that needs to be executed from multiple triggers. The data passed between the calling trigger and the subtrigger is used to provide the dynamic runtime information from the context of the calling trigger to the common subtrigger.

This is a common application pattern of consolidating common application logic into a sub routine or common function.

## Parameter description

Parameter	Description
<b>Project Name</b>	The project on the local node where the subtrigger resides. The selection ==Current Project== can be used to indicate the project that this calling trigger resides in.
<b>SubTrigger Name</b>	The subtrigger to execute.
<b>Wait for Completion</b>	The options are:  <b>True</b> - Initiates the execution of the subtrigger and waits for the subtrigger to complete its execution. This is referred to as synchronous processing. <b>False</b> - Initiates the execution of the subTrigger but does not wait for the subtrigger to complete its execution. This is referred to as asynchronous processing. There is no guarantee as to the concurrent execution of the calling trigger and its actions and the called subtrigger and its actions. Either one may be executed first or complete first. Any access to common resources, such as device variables must be understood by the application developer.

## Input tab

Name	Logical	Count	Value	Type
stringParameter	STRING(16)	1		
integerParameter	INT2	1		

The **Input** tab will display the input variables as defined by the subtrigger. These parameters are passed from the calling **Execute SubTrigger** action to the subtrigger.

Each input variable has the data type that it was defined with by the subtrigger. If the data types of the input variables do not match the data types as defined by the subtrigger, the subtrigger will fail.

## Output tab

1: Execute SubTrigger

**Project Name:** Publisher

**SubTrigger Name:** MySubTrigger

**Wait for Completion:** True

Input Output Routing

Output

Name	Logical	Count	Value	Type
outInteger	INT2	1		

The **Output** tab will display the output variables as defined by the subtrigger. These parameters are passed from the subtrigger back to the calling **Execute SubTrigger** action.

Output parameters can only be passed back from the subtrigger when the **Wait for Completion** parameter is **True**.

Each out variable has the data type that it was defined with by the subtrigger

## Example usage of the Execute SubTrigger action

In this scenario there is a need to select rows from a database at various phases of the production cycle. The selected data is used as input in various calculations and the results are fed down to a series of device variables, which vary depending upon the point in the production cycle the product currently resides. Many different triggers are defined that are launched when various device variables are activated. Each of these triggers could contain an action that would select the data from the database table and perform the calculations against the data. This presents a maintenance problem in that typos could be introduced into the definition of the calculations from one trigger to the next. Likewise, error or exception handling may be needed that would be replicated through each of the triggers.

A more desirable solution is to move these routines that are common to all the triggers into a subtrigger. Now each trigger will reference the subtrigger in the **Execute SubTrigger** action and the caller of the subtrigger can be assured that the data will be processed in a like manner regardless of where it is called.

Related topics

SubTrigger

## IIoTA industrial IoT Platform: Fire Trigger

The **Fire Trigger** action executes an On-Demand trigger. The trigger that is the target of this action must be a **On-Demand** event type trigger.

The **Execute SubTrigger** action is used to execute a **SubTrigger** event type trigger. Both **On-Demand** and **SubTrigger** event type triggers can use all the available actions on the node.

- Subtriggers can define input and output variables pass data between the calling **Execute SubTrigger** action and the subtrigger.

For information on defining SubTrigger event type triggers, see SubTrigger

- On-Demand triggers can be executed by the calling Fire Trigger action and by the Workbench from the triggers project window.

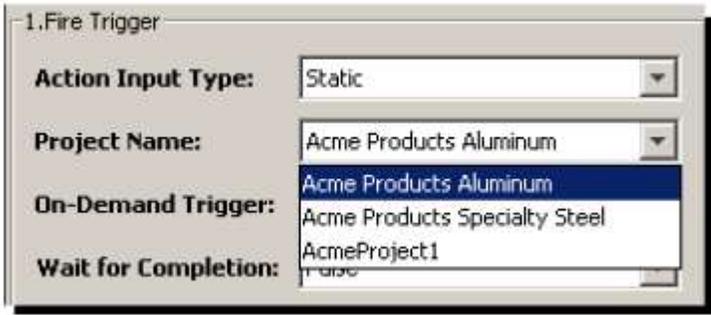
For information on defining On-Demand event type triggers, see On-Demand

- Both can be executed with an option to wait for its completion.

### Parameter description

On Result	Go to
Success	NEXT
Failure	UNDEFINED

Parameter	Description
<b>Action Input Type</b>	The options are:

	<p><b>Static</b> - The <b>Project Name</b> and <b>On-Demand Trigger</b> parameters becomes available.</p> <p><b>Dynamic</b> - The <b>Input</b> tab <b>Project Name</b> and <b>On-Demand Trigger</b> parameters becomes available.</p>
<b>Project Name</b>	<p>Available when <b>Action Input Type</b> is <b>Static</b>.</p> <p>Select a project from the list of all projects on the current node. The project where the On-Demand trigger resides.</p> <p>The selection <b>==Current Project==</b> can be used to indicate the project that this calling trigger resides in.</p> 
<b>On-Demand Trigger</b>	<p>Available when <b>Action Input Type</b> is <b>Static</b>.</p> <p>The On-Demand trigger to execute.</p>
<b>Wait for Completion</b>	<p>The options are:</p> <p><b>True</b> - Initiates the execution of the On-Demand trigger and waits for the On-Demand trigger to complete its execution. This is referred to as synchronous processing.</p> <p><b>False</b> - Initiates the execution of the On-Demand trigger but does not wait for the On-Demand trigger to complete its execution. This is referred to as asynchronous processing. There is no guarantee as to the concurrent execution of the calling trigger and its actions and the called On-Demand trigger and its actions. Either one may be executed first or complete first. Any access to common resources, such as device variables must be understood by the application developer.</p>

## Input tab

When you select **Dynamic** for the **Action Input Type** option, an **Input** tab appears:

Parameter	Description
<b>Project Name</b>	The project where the On-Demand trigger resides.
<b>Trigger Name</b>	The On-Demand trigger to execute.

Related topics

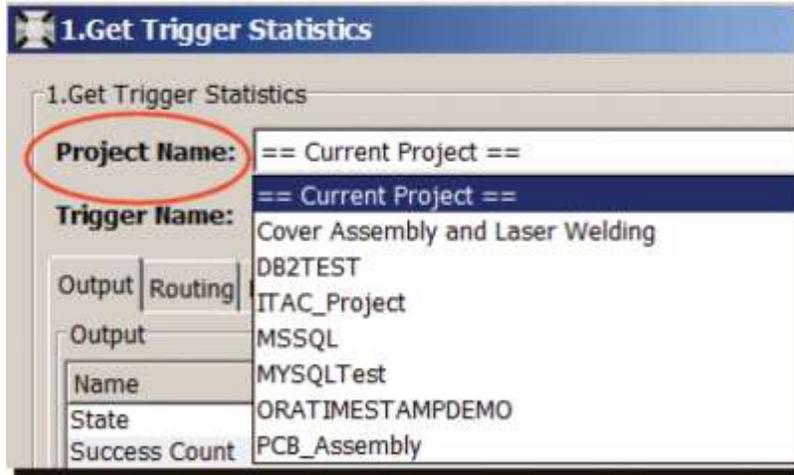
On-Demand

## IIoTA industrial IoT Platform: Get Trigger Statistics

The **Get Trigger Statistics** action returns statistics related to the execution of a trigger.

### Parameter descriptions

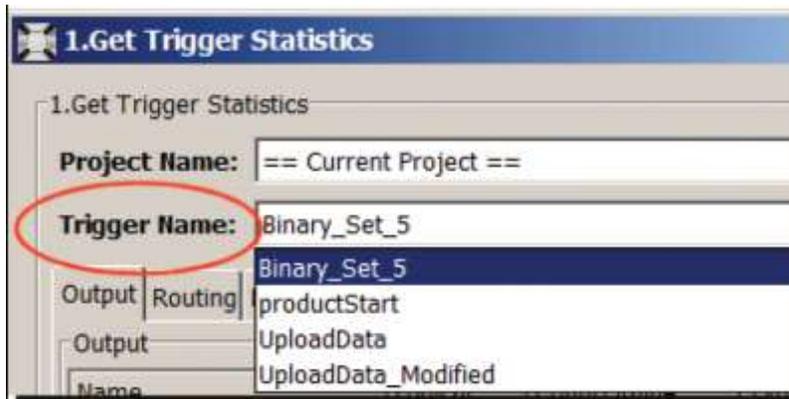
Parameter	Description
<b>Project Name</b>	Select a project from the list of all projects on the current node. The project where the trigger resides.



The selection **==Current Project==** can be used to specify the current project.

**Trigger Name**

Select the trigger defined in the project that you specified in the **Project Name** parameter.



## Output tab

You can specify an output variable to hold the statistic for the item. If you do not specify an output variable, no statistic will be generated. The statistics returned are like the parameters displayed on the bottom of the Project tab when a specific trigger is selected.

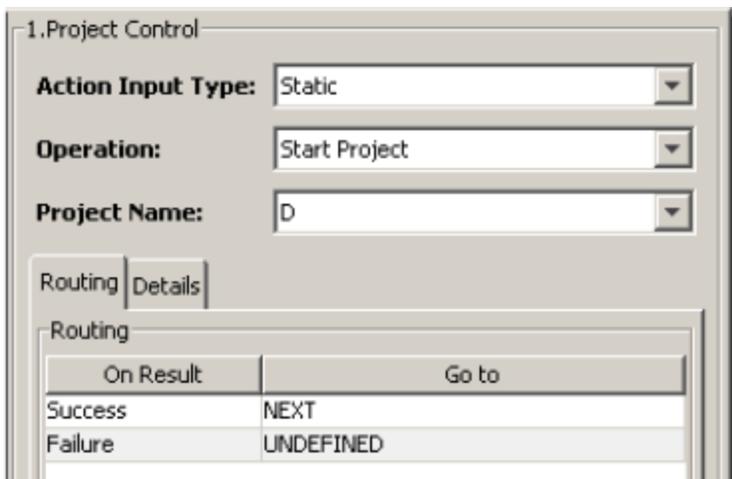
Parameter	Description
State	The current state of the trigger, which can be Started, Stopped or Disabled.
Success Count	The successful execution count.
Failure Count	The failed execution count.

Overflow Count	The number of triggers instances that were not executed due to the trigger's <b>Pending Count</b> value equaling the trigger's Settings tab <b>Max in Progress</b> value.
Pending Count	The number of instances of the trigger that are currently executing. This is the value displayed in the In Progress column of the Project tab.
In-Queue Count	The number of trigger executions that are waiting to execute.
Last Execution Time	The last execution time of the trigger in milliseconds.
Maximum Execution Time	The maximum execution time of the trigger in milliseconds.
Minimum Execution Time	The minimum execution time of the trigger in milliseconds.
Average Execution Time	The average execution time of the trigger in milliseconds.
Last Execution	The date and time the last execution of the trigger occurred.
Last Error	The date and time the last execution failure of the trigger occurred.
Queue Watermark Count	The maximum number of items at one time in the In-Queue queue (highest In-Queue value).
Queue Watermark Timestamp	The date and time the highest Queue Watermark Count occurred.

## IIoTA industrial IoT Platform: Project Control

The **Project Control** action starts, or stops, or deletes a project.

### Parameter description



Parameter	Description
<b>Action Input Type</b>	<p>The options are:</p> <p><b>Static</b> – The <b>Project Name</b> parameter becomes available.  <b>Dynamic</b> – The <b>Input</b> tab <b>Project Name</b> parameter becomes available.</p>
<b>Operation</b>	<p>The options are:</p> <p><b>Start Project</b> – The target project must be in a Stopped state. The project is started and all of its triggers are Loaded. In order for a trigger to execute, its state must be Started and its status must be Loaded.  <b>Stop Project</b> – The target project must be in a Started state. The project is stopped and all of its triggers are Unloaded. In order for a trigger to execute, its state must be Started and its status must be Loaded.  <b>Delete Project</b> – The target project must be in a Stopped state. The project and all its triggers are deleted from the node.</p>
<b>Project Name</b>	<p>Available when <b>Action Input Type</b> is <b>Static</b>.            Select a project from the list of all projects on the current node to control.</p>

## Input tab

When you select **Dynamic** for the **Action Input Type** option, an **Input** tab appears:

Parameter	Description
<b>Project Name</b>	The project on the current node to control.

# IIoTA industrial IoT Platform: Trigger Control

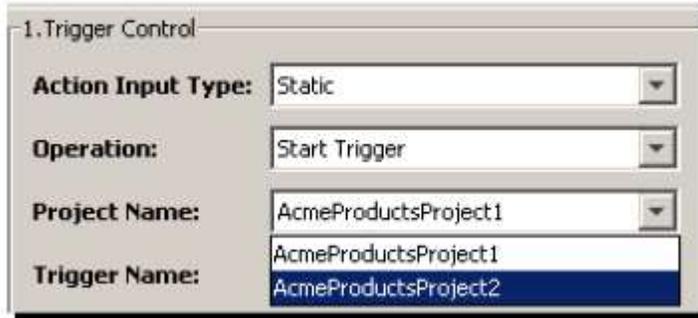
The **Trigger Control** action starts, stops, or deletes a trigger.

## Parameter description

On Result	Go to
Success	NEXT
Failure	UNDEFINED

Parameter	Description
<b>Action Input Type</b>	<p>The options are:</p> <p><b>Static</b> – The <b>Project Name</b> and <b>Trigger Name</b> parameters become available.  <b>Dynamic</b> – The <b>Input</b> tab <b>Project Name</b> and <b>Trigger Name</b> parameters become available.</p>
<b>Operation</b>	<p>The options are:</p> <p><b>Start Trigger</b> – The target trigger must be in a Stopped state. The trigger is started. In order for a trigger to execute, its state must be Started and its project state must be Started. This puts the trigger in a Loaded status.  <b>Stop Trigger</b> – The target trigger must be in a Started state. The trigger is stopped. The trigger will not be able to be executed.  <b>Delete Trigger</b> – The target trigger must be in a Stopped state. The triggers is deleted from the node.</p>
<b>Project Name</b>	<p>Available when <b>Action Input Type</b> is <b>Static</b>.  Select a project from the list of all projects on the current node. The project where the target trigger resides.</p>

The selection **==Current Project==** can be used to indicate the project that this calling trigger resides in.

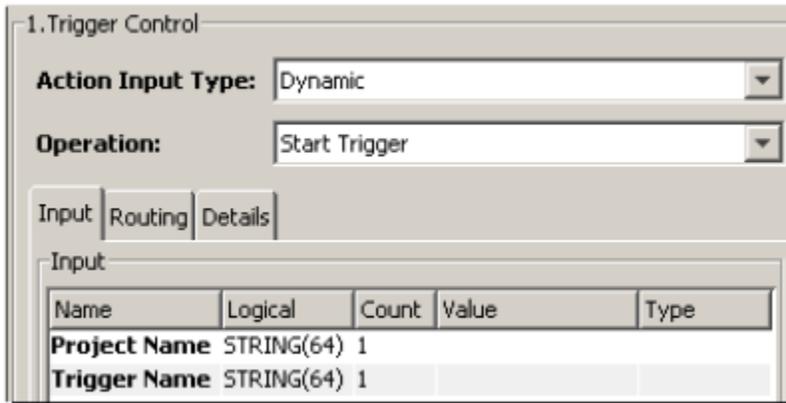


**Trigger name**

Available when **Action Input Type** is **Static**.  
Select the trigger to control.

## Input tab

When you select **Dynamic** for the **Action Input Type** option, an **Input** tab appears:



Parameter	Description
<b>Project Name</b>	The project where the trigger to control resides.
<b>Trigger Name</b>	The trigger to control

# IIoTA industrial IoT Platform: Using compound strings

## Overview

Many trigger actions use compound strings that let you build complex strings using the values from variables in the system. If you are familiar with programming, compound strings are similar to functions such as "printf()".

The following example shows a **Log Message** action using a compound string in its **Message** parameter. When the action executes, the message is dynamically built using the text in the parameter and the substitution variables \$(trig) and \$(devicevar). The variable \$(trig) is defined to use a trigger macro, Macros.\$TRIGGER, that contains the name of the trigger. The variable \$(devicevar) is defined to use the value of a device variable, globVar.TriggerStatusChange1.

Notice the substitution variables \$(trig) and \$(devicevar) are added as parameters to the **Input** tab when they are typed into the **Message** parameter. You can define them to reference any started device, constant, trigger macro, trigger local variable, trigger static variable, or event variable available from the pull-down list displayed for the **Value** cell for that row.

1. Log Message

**Message Type:** EXCEPTION

**Message Level:** INFO

**Component:** User Trigger

**Message:** From trigger \$(trig), status value = \$(devicevar)

Input Routing Details

Input

Name	Logical	Count	Value	Type
<b>trig</b>	ANY	1	Macros.\$TRIGGER	STRING(64)
<b>devicevar</b>	ANY	1	globVar.TriggerStatusChange1	INT2

When the trigger and this example **Log Message** action executes, the message that is dynamically built and written to the Exceptions Log:

- The substitution variable \$(trig) has been replaced by the value of Macros.\$TRIGGER. In this case, the name of the example trigger is Compound\_string\_example.
- The substitution variable \$(devicevar) has been replaced by the value of the device variable globVar.TriggerStatusChange1. In this case that device variable had a value of 1.

Filter (1 entries found)

**From Date:**  **To Date:**

**Message Type:**  **Component:**

**Message Contains:**

Date	Time	Type	Component	Message
2013-07-08	19:53:54,437	INFO	[User Trigger]	From trigger Compound_string_example, status value = 1

### Floating point limitation

Currently there is no way to limit the precision of floating-point values when printed, which may result in many digits after the decimal point when printing floating point values. To avoid this problem, you may wish to first convert the floating-point value to an integer value, which will prevent any decimal points from being printed.

## Supported escape characters

When building your string, you may need to insert special characters such as new lines or null terminators into your string. The compound string handler uses a backslash as an escape character to note the start of an escape sequence. The supported escape sequences are described in the table below. If an invalid escape character is specified, the sequence will be ignored, and the compound string will be built without the escape sequence as well as without the '\' in the string.

The following control characters are supported when using the compound string feature to build your string.

Pattern	Result
\t	The tab character
\r	The line feed character
\n	The new line character (0x0A)
\xNN	Any character specified in hexadecimal notation, for example \x35 will print the number 5. Warning: use of \x00 to specify an embedded null should not be used.

\0	Null - not supported for embedded nulls. String functions that use the length of the string when processing will not give the expected results.
\\	Places a backslash \ character
\$\$	Places a dollar sign \$ character

## IIoTA industrial IoT Platform: Enterprise connectivity

### Overview

The **Enterprise** feature provides the connectivity to your end point enterprise application programs. This includes any-to-any data transformation where the source data type and target data can be in different formats.

The Enterprise feature is used to define connections to your enterprise applications, such as database servers, and to define the operations, such as a database **Insert** operation, to send data to and receive data from the enterprise applications.

The IIoTA application logic, defined in triggers, then make use of the Enterprise feature definitions.

This allows you to create your M2M solution using your enterprise applications in a vendor neutral methodology. The triggers (your application logic) can be shielded from the majority of the enterprise connectivity details, while still providing the two-way enterprise application access functions required by your M2M solution.

### Enterprise connectivity feature summary

The details of the Enterprise connectivity feature are provided on the pages in this guide. A summary is:

- The Enterprise connectivity features are provided by the **Transaction Server** component of the runtime.
- A **Transport** defines the connectivity details and features, such as store and forward and transport map logging, when initiating a transaction (or request) from a trigger that is destined for an enterprise application.

Transports are defined to represent databases, message and queuing systems, TCP applications, ERP applications, and web services.

- A **Transport Map** defines the data format of the data sent to the enterprise application and, when there is response data, the format of the response data returned by the enterprise application.
- A **Listener** defines the connectivity details and features, such as mapping logs, when a message (or request) is received from an enterprise application. The received message is processed by a trigger.

Listeners are defined to represent message and queuing applications (such as JMS, MSMQ, and WebSphereMQ) and TCP applications.

- A **Listener Map** defines the data format of the data received from the enterprise application and, when there is response data, the format of the response data sent from the trigger back to the enterprise application.

When the Enterprise icon is selected for a node, each of the Enterprise connectivity features is available as a sub icon. They also are available as a separate tab on the Enterprise window.

The screenshot shows the 'Acme Products Enterprise Gateway / Enterprise' window. The left-hand navigation pane has 'Enterprise' selected, with its sub-items 'Transport Maps', 'Transports', 'Listener Maps', and 'Listeners' also visible. The main area shows a table of database connections:

Name	Type	State	Successes	Failures	S/F Queued	S/F Deleted	Pending	Pending Queue Overflow
DB2v9_0_59	DB	Up	1019208	0	0	0	0	0
MySQLv5_0_59	DB	Up	1016645	2	0	0	0	0
LOCALDB	DB	Down	0	0	0	0	0	0
LOCALDB1	DB	Down	0	0	0	0	0	0
LOCALDB2	DB	Down	0	0	0	0	0	0
LOCALDB3	DB	Down	0	0	0	0	0	0
LOCALDB4	DB	Down	0	0	0	0	0	0
LOCALDB5	DB	Down	0	0	0	0	0	0

Below the table is a 'Filter: None' section with 'Clear' and 'Modify' buttons. At the bottom of the window are buttons for 'New', 'Edit', 'Delete', 'Suspend', 'Resume', and 'Refresh'. A yellow banner at the very bottom reads 'Not for resale license in use'.

## Assumptions

The information in this section assumes that you have:

- Activated the appropriate licenses.
- The connectivity information for your enterprise applications (IP address, port, user ID, password, etc.).

## Highlights

This guide contains the following:

- Transports
- Transport maps
- Listeners
- Creating a TCP listener
- XML listener commands
- Creating a listener trigger
- Using the Listeners tab
- Editing a listener
- Export and importing listeners
- Using the Listener Maps tab
- Configuring a default listener map
- Tabs on the Listener window
- Listener maps
- Understanding data types

## IIoTA industrial IoT Platform: Transports

A **Transport** defines the connectivity details and features, such as store and forward and transport map logging, when initiating a transaction (or request) from a trigger that is destined for an enterprise application.

Transports are defined to represent databases, message and queuing systems, TCP applications, ERP applications, and web services.

The first several sections of this guide describe the general functions and tasks for all transport types.

This guide contains the following:

- Learning the basics
- Using tabs on the Transport window
- Using the Transports tab
- Transport types

## IIoTA industrial IoT Platform: Learning the basics

This page will walk you through the preliminary steps to creating a transport. You will learn how to create specific transports including database and WebSphere MQ transports.

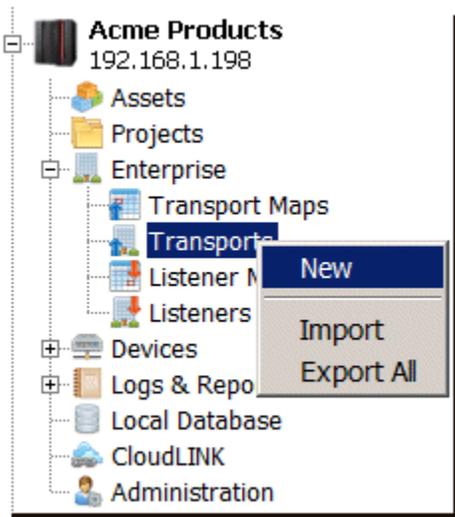
### Required software and protocol support

Before you can begin to create any of the supported transports, the appropriate third-party software must be installed and configured.

### Creating a transport

It is assumed that the Workbench is started, and you have logged on.

1. From the Workbench left pane, expand the that you want to associate the new transport with, and then expand **Enterprise**.
2. On the **Transports** icon, right click to display its pop-up menu.



3. Click **New**.

The Transport window appears with a set of tabs. The tabs that appear on the Transport window depend on the type of transport selected.

**Transport**

Name:

Type:

Parameters | Timeout | Custom Payloads

Host:  Port:

Mode:

Load transport at initialization  Include header in payload

Save Validate Cancel

This is the default window. Every Transport window has a **Parameters** tab. The appearance of the **Parameters** tab changes according to the type of transport selected.

### Naming the transport

Now that you have displayed the Transport window, the first step is to name the transport. A transport name can be up to 64 characters in length and can include letters, numbers, and the underscore character. Spaces are not allowed.

### Selecting the transport type

The next step is to select the type of transport you want to create. Click the down-arrow next to **Type**.

**Transport**

Name:

Type:

Parameters | Timeout | Custom Payloads

Host:  Port:

Mode:

Load transport at initialization  Include header in payload

Save Validate Cancel

The choices depend on the specific product on the node and what transport protocol licenses have been activated.

For information on the different transports, see [Transport types](#).

### Setting the transport to load at initialization

Every **Parameters** tab has a **Load transport at initialization** check box.

The **Load transport at initialization** feature lets the transport connect to the enterprise application as soon as the node starts up (or immediately after leaving store and forward). Then, when the first transaction for the transport is ready to be processed, processing is faster because the transaction does not have to initialize the transport (the connection is already established). By default, a transport is in a Down state until it processes a transaction.

Once you have selected a transport type, you can specify other parameters using specific tabs available for that transport.

### Validating and saving the transport

All transports can be validated. The Transport window provides **Save**, **Validate**, and **Cancel** buttons.

- After you have filled in each parameter, click **Validate**.

When you click **Validate**, the connection to the transport target application is tested. For example, if you had selected **DB** as the transport type, the Workbench tests the connection to the database. A message will tell you whether the validation was successful or not.

- If no errors are received, click **Save**.

The new transport is saved to the node and added to the **Transports** tab. The **Transports** tab provides information about transport processing such as whether a transport is operational or in a store and forward mode. From the **Transports** tab, you can also create a new transport, edit an existing transport, delete an unwanted transport, and more. For more information, click [Using the Transports tab](#).

Related topics

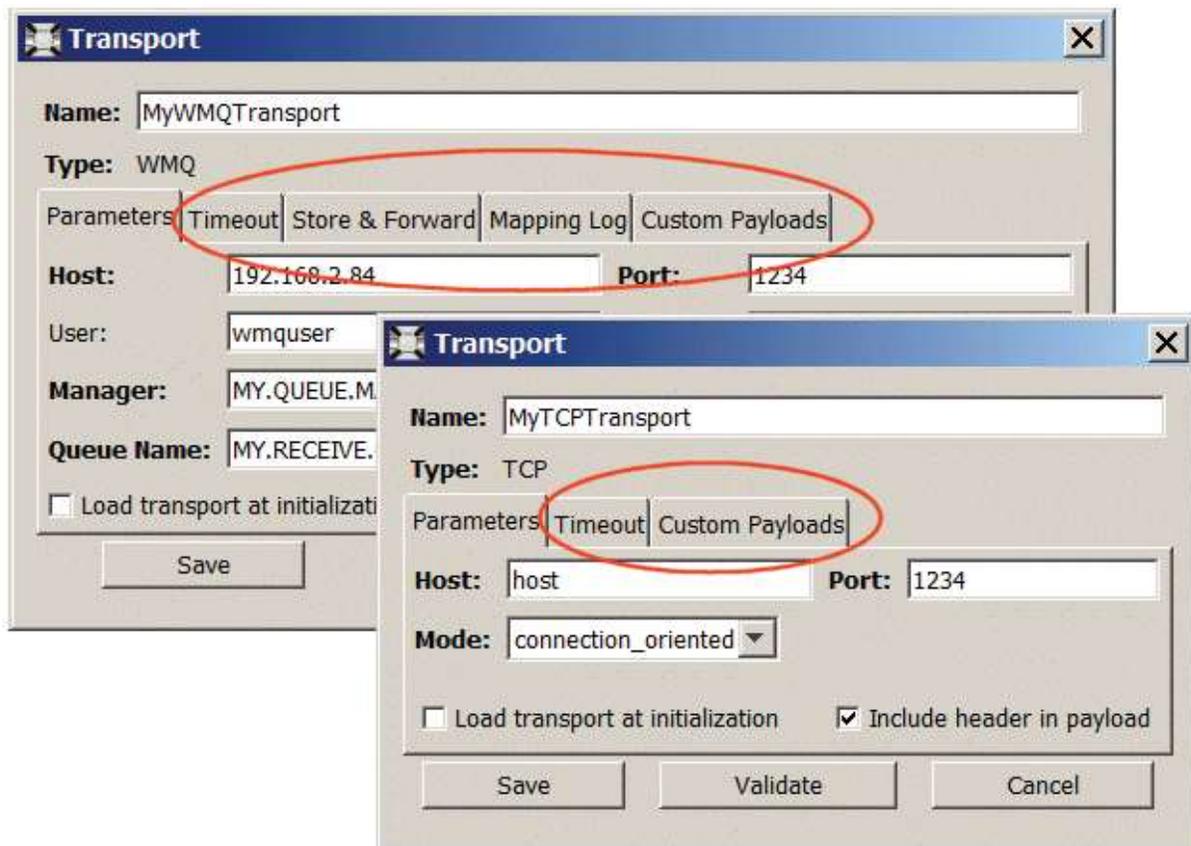
[Store & Forward tab](#)

[Mapping Log tab](#)

[Adding timeout values](#)

## IIoTA industrial IoT Platform: Using tabs on the Transport window

When creating a transport and specifying a transport type, the Transport window provides several tabs that accommodate the definition of the transport. For example, a transport window for a **TCP** transport has a **Timeout** and **Custom Payloads** tab; while a **WMQ** (WebSphere MQ) transport window has **Timeout**, **Store & Forward**, **Mapping Log**, and **Custom Payloads** tabs. These tabs and their parameters are common across the different types of transport windows.



Notice that the **WMQ** transport and the **TCP** transport windows have a **Timeout** tab and **Custom Payloads** tab. Information for these common tabs is located under **Related Topics**.

Related Topics

Timeout tab

Store & Forward tab

Mapping Log tab

Custom Payloads tab

## IIoTA industrial IoT Platform: Timeout tab

### Overview

Timeouts are used to fine tune the transport behavior and need to be tailored to application specific needs. They determine when a transaction will fail or be sent to the store and forward queue, when the host is unreachable or cannot be reached within a reasonable amount of time.

If a transaction consistently takes 10 seconds to execute, but the transport **Execution** timeout parameter is set to 5 seconds, then the transaction will cause the transport to switch to store and forward mode or be counted as failure if Store and Forward is not enabled.

Similarly, if it takes more than the **Connection** time to establish a connection to a host, the transport will switch to store and forward mode.

### Parameter values for a Timeout tab

The following shows the example SAP transport and its **Timeout** tab:

The screenshot shows the 'Transport' configuration window. The 'Name' field contains 'My\_SAP\_Transport' and the 'Type' dropdown is set to 'SAP'. Below these are four tabs: 'Parameters', 'Timeout', 'Store & Forward', and 'Mapping Log'. The 'Timeout' tab is selected and highlighted with a red circle. Under the 'Timeout' tab, there are three input fields: 'Connection (sec):' with the value '10', 'Execution (sec):' with the value '5', and 'Inactivity (sec):' with the value '14400'.

Option	Description
<b>Connection (sec)</b>	<p>Required. Defaults to 10 seconds.</p> <p>Specifies the length of time the system will try to connect to a target computer (where an associated database program or TCP application are running). If the connection is not made in the specified time period, an error message is sent to the exception log. Also, the <b>State</b> column on the <b>Transports</b> tab will indicate <b>Down</b>. If the transport has store and forward enabled, the <b>State</b> column will indicate <b>Store and Forward</b>.</p>
<b>Execution (sec)</b>	<p>Required. Defaults to 5 seconds.</p> <p>Specifies how long the system should wait (once the connection is made) for a transaction to complete. The time value that you specify should be the outer limit for how long you expect a typical transaction to take using a transport.</p> <p>If store and forward is turned on, the <b>Execution</b> timeout value will also specify how long the Transaction Server will wait before attempting to either switch to store and forward mode (for a database <b>Insert</b> or <b>Update</b> operation and WebSphere MQ transports) or fail the transaction (for a <b>Select</b> operation or stored procedure with <b>OUTPUT</b> or <b>INOUT</b> parameters).</p> <p>In the case of a transport map with a <b>Select</b> operation or stored procedure with <b>OUTPUT</b> or <b>INOUT</b> parameters, if the execution fails due to a communication problem, the transaction is not put into store and forward mode. The reason is so that the value of a device variable on the controller is not changed at some future point in time when the connection to the database is restored.</p> <p>There can be any number of reasons for transactions to take a long time: The server is extremely busy at certain times of the day; the network has a disruption due to a pulled cable or a switch shutoff that may be temporary or permanent.</p> <p><b>About long execution timeouts:</b></p> <p>Keep in mind that specifying a long timeout can cause transactions to backup when triggers are firing at a very fast rate.</p> <p>In addition, some transports take longer to interact with the business enterprise target (such as a database <b>Update</b> operation or stored procedure calls). For these types of transports, it is better to create a separate transport for each transport type. This would prevent a slower transaction from blocking other transactions that are queued behind it.</p> <p>If a re-connection cannot be established, an error message is written to the exception log. The <b>Failures</b> column in the project tab for the specified trigger is incremented by the number of times the trigger failed. If store and</p>

	<p>forward is enabled (for database and WebSphere transports), the <b>State</b> column will indicate <b>Store and Forward</b>.</p> <p><b>MSSQL Transport</b></p> <p>The lock timeout on transactions to a MSSQL database is set to 90% of the value specified as the execution timeout. This means that if the time taken by a MSSQL transaction exceeds 90% of the execution timeout it will fail, and the transport will process the transaction as if it had timed out.</p>
<b>Inactivity (sec)</b>	<p>Required. Defaults to 14,400 seconds.</p> <p>Specifies the maximum number of seconds of inactivity that the Transaction Server will wait before disconnecting from the host.</p>

The following **Related Topics** links pertain to tabs available from different transport type windows. For example, a transport window for a TCP transport has Timeout and Custom Payloads tabs; while a WebSphere MQ transport type has a Timeout, Store & Forward, Mapping Log, and Custom Payloads tabs.

Related topics  
 Store & Forward tab  
 Mapping Log tab  
 Custom Payloads tab

## IIoTA industrial IoT Platform: Store & Forward tab

### Overview

Except for the TCP transport, all other transport types support store and forward. In the event of an enterprise outage or network failure, the store and forward feature saves all transactions on the node, and then delivers them when the connection to the enterprise host can be established again.

When the connection to the enterprise host is re-established, messages from the store and forward queue are delivered to the enterprise application until the queue is cleared.

By default, store and forward is turned on with the **Process queue before leaving S/F** option. See parameter details below.

### Parameter values for the Store & Forward tab

The following shows an example SAP transport and its **Store & Forward** tab:

The screenshot shows the 'Transport' configuration window. The 'Store & Forward' tab is active and circled in red. Below the tabs, the 'Store & Forward' checkbox is checked. The 'TTL (sec)' field is set to 86400, and the 'Max Storage (MB)' field is set to 10. The 'On overflow' dropdown is set to 'Discard new message'. The 'Process queue before leaving' checkbox is also checked.

## Store & Forward

When the **Store & Forward** check box is selected, the transport will switch to store and forward mode in case the connection to the enterprise host is lost.

If unchecked, store and forward mode is not active, and transactions will be counted as failures in case the connection to the enterprise host is lost.

## TTL(sec)

You can use the **Time to Live (TTL)** parameters to control the use of the Store and Forward queue while the transport is suspended. The value in seconds determines how long to retain messages that are in the store and forward queue.

## Max Storage (MB)

The **Max Storage** parameter is the maximum size of the store and forward queue for the current transport. The default is 20 MB. The only limit is the amount of disk space on the node.

## On overflow

There are two options:

**Discard new message** — Do not accept any new messages once the store and forward queue reaches its maximum size. New messages are counted as failures.

**Delete oldest message** — Have the oldest messages discarded in a first in/first out (FIFO) manner when the store and forward queue becomes full.

## Process queue before leaving S/F

When creating a transport, the **Process queue before leaving S/F** check box can be used to modify the behavior of store and forward.

Parameters | Timeout | Store & Forward | Mapping Log

Store & Forward

TTL (sec): 86400 On overflow: Discard new message

Max Storage (MB): 10  Process queue before leaving S/F

By default, the **Process queue before leaving S/F** check box is selected and data in the store and forward queue behaves in a first in/first out (FIFO) manner. This means that once the connection is re-established to the end point application, data is delivered in the order in which it entered the store and forward queue.

When the **Process queue before leaving S/F** check box is not selected, as soon as the connection is re-established with the end point application, new data is not added to the store and forward queue but; instead, is sent directly to the end-point application, and the data residing in the store and forward queue is also being delivered to the end point application. The order of delivery is not important in this non-FIFO mode. It is possible to have the state of a transport shown as **Up** (on the **Transports** tab) while data is still in the store and forward queue.

Non-FIFO mode guarantees to empty the store and forward queue. However, in FIFO mode, clearing the queue depends on the rate data is added to the queue. For FIFO mode, when data is added to the queue at a rate faster than it is being delivered, the transport will always remain in store and forward.

To fine tune an application's ability to switch to store and forward mode, use the **Connection** and **Execution** timeouts parameters from the **Timeout** tab.

Transport

Name: My\_SAP\_Transport

Type: SAP

Parameters | Timeout | Store & Forward | Mapping Log

Timeout

Connection (sec): 10 Execution (sec): 5

Inactivity (sec): 14400

The following **Related Topics** links pertain to tabs available from different transport type windows. For example, a transport window for a TCP transport has Timeout and Custom Payloads tabs; while a WebSphere MQ transport type has a Timeout, Store & Forward, Mapping Log, and Custom Payloads tabs.

Related topics

Timeout tab

Mapping Log tab

Custom Payloads tab

Purging data from store and forward

## IIoTA industrial IoT Platform: Mapping Log tab

### Overview

Except for the TCP transport, all other transport types support transport map logging. Transport map logging enables the content of every outbound or inbound transaction for the transport to be recorded in a mapping log.

For the runtime to provide a map log file, you must turn the logging on for the transport or listener. Each transport or listener that has logging enabled will send transactions to a file specific to that transport or listener. For example, suppose you create a listener called MyWMQListener and two transports called MyWMQTransportOne and MyWMQTransportTwo. Three separate log files will be created and maintained.

### Parameter values for a Mapping Log tab

The following shows the **Mapping Log** tab that is common to both the Transport window (for a database, JMS, MSMQ, WMQ, SAP, SAP MII, iTAC, and web service transport) and Listener window (for a MSMQ, WMQ, JMS, and TCP listener).

The screenshot shows the 'Transport' configuration window for 'My\_SAP\_Transport'. The 'Type' is set to 'SAP'. The 'Mapping Log' tab is selected and highlighted with a red circle. The 'Mapping Log' checkbox is checked. The 'Log Size (MB)' is set to 1, and the 'Number of Log Files' is set to 1. The 'Message Size (bytes)' is set to 'Entire Message'. There is also an unchecked checkbox for 'Copy rolled log to staging'.

The following describes the parameters on the **Mapping Log** tab:

## Mapping Log

To turn on logging, select the **Mapping Log** check box. The following parameters become available.

### Log Size (MB)

The value is the maximum size of the mapping log file for the current transport or listener. You can select a maximum log size anywhere from 1 MB to 10 MB. Once the log file reaches that size, a new file is created and named as *Name\_1.log*. Subsequent transactions are recorded in the current log file. If the **Copy rolled log to staging** check box is selected, that file is also copied to the staging file system.

### Message Size (byte)

The value controls the length of the line recorded in the mapping log for the current transport or listener.

You can specify that the entire message be recorded or limit the line to 512 or 1024 bytes. Select **Entire Message** to have no limitations.

### Number of Log Files

The value is the number of mapping log files that can be maintained for the current transport or listener.

Log files are maintained as numbered logs (1 to 10). Once the log file reaches the size specified in **Log Size**, a new file is created and named as *TRANSPORTtransport\_n.log*. For example, *PLANTDBtransport\_1.LOG*, *PLANTDBtransport\_2.LOG*, *PLANTDBtransport\_3.LOG*, and so forth.

If you change the value of the **Number of Log Files** parameter to a lower number for a saved transport or listener (for example from 4 to 2), the two older log files will remain in the Mapping Logs tab. You cannot delete a specific log file. You might export all the log files, and then use **Delete Log** to clear all entries in the **Mapping Log** tab.

### Copy rolled log to staging

If selected, a copy of the mapping log file is sent to the root of the staging file system (the Staging Browser tab) when the file size reaches the value specified in the Log Size (MB) parameter. From the **Staging Browser** tab, you can perform file system operations such as putting the file on a remote computer.

The file will have the suffix appended with the following format:

yyyyMMddHHmmssS

where:

yyyy	is the current year.
MM	is the number of months in the year.
Dd	is the day of the month.
HH	is the hour (24 hour).
Mm	is the minutes past the hour.
Ss	is seconds.
S	is the first digit of milliseconds.

For example, the file name for a transport named PLANTDB that was copied to the staging file system will be transaction\_t\_PLANTDB.log.200804151020154.

**Turning mapping logging on or off at any time.** You can turn mapping logging on or off whenever you want. If you have already created the transport or listener, you can edit the respective Transport or Listener window to change the **Mapping Log** option.

The following **Related Topics** links pertain to tabs available from different transport type windows. For example, a transport window for a TCP transport has Timeout and Custom Payloads tabs; while a WebSphere MQ transport type has a Timeout, Store & Forward, Mapping Log, and Custom Payloads tabs.

Related Topics

Timeout tab

Store & Forward tab

Custom Payloads tab

## IIoTA industrial IoT Platform: Custom Payloads tab

JMS, TCP, SAP MII, Microsoft Message Queuing, and WebSphere MQ transport types facilitate custom formats for their payload data.

### Assumptions

These pages assume that you understand how to create a transport and are ready to develop a custom payload format.

## Parameter values for the Custom Payloads tab

The following shows a portion of the Transport window that lets you specify a custom payload.

Parameters | Timeout | Store & Forward | Mapping Log | Custom Payloads

Allow Custom Payloads

Jar Name:

Transform Class Name:

A custom payload format requires the use of a client created java application program that defines method(s) to populate values for each transport input map variable.

The following describes the parameters on the **Custom Payloads** tab:

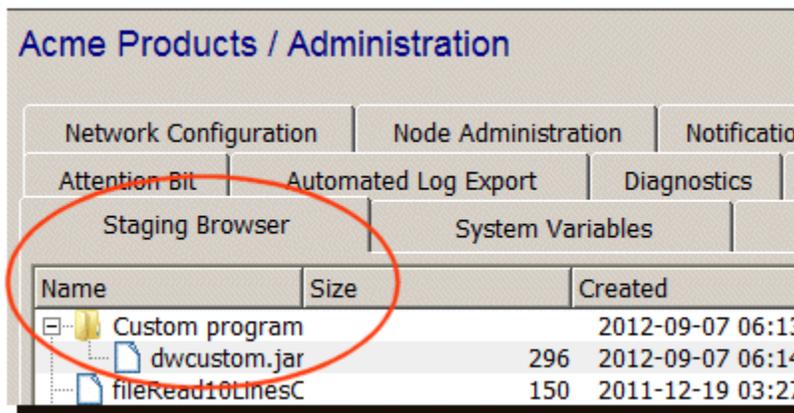
### Allow Custom Payloads

Select the **Allow Custom Payloads** check box to enable a transport to use a custom payload. You must specify the name of a jar file and class name.

### Jar Name

When creating a transport that will use a custom format payload, you must specify a jar file and class name.

Use the **Jar Name** box to specify the name of the jar file that contains the client java application program. The jar file must be available in staging file system (the **Staging Browser** tab) on the node that contains the transport.



For more information, see **Staging Browser**.

## Transform Class Name

Use the **Transform Class Name** box to specify the name of the class where the methods are defined.

The screenshot shows a configuration window with five tabs: Parameters, Timeout, Store & Forward, Mapping Log, and Custom Payloads. The Custom Payloads tab is active. It contains a checked checkbox labeled 'Allow Custom Payloads'. Below this, there are two text input fields: 'Jar Name' with the value 'dwcustom.jar' and 'Transform Class Name' with the value 'com.ils\_tech.custom.DWTransformRequest'.

For this example, the client java application program is packaged in **dwcustom.jar**. The class name is **DWTransformRequest**.

For information on how to create a transport map that uses the custom format, refer to Adding a custom payload.

The following **Related Topics** links pertain to tabs available from different transport type windows. For example, a transport window for a TCP transport has Timeout and Custom Payloads tabs; while a WebSphere MQ transport type has a Timeout, Store & Forward, Mapping Log, and Custom Payloads tabs.

### Related Topics

- Timeout tab
- Store & Forward tab
- Mapping Log tab
- Creating a JMS transport
- Creating the MSMQ transport
- Defining a TCP transport
- Creating the WebSphere MQ transport

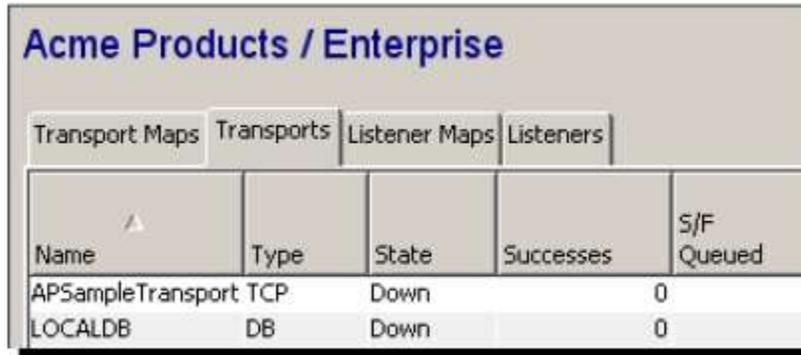
## IIoTA industrial IoT Platform: Using the Transports tab

The **Transports** tab is used to view different processing status for a specific transport such as whether or not the transport has been initialized and the number of times the transport successfully delivered data. The information is valuable during runtime especially when deciphering transport processing problems.

To use the **Transports** tab, follow these steps:

1. From the Workbench left pane, expand the node whose transports you want to review.
2. Expand **Enterprise**, and then click **Transports**.

The **Transports** tab appears in the right pane.



Acme Products / Enterprise				
Transport Maps   <b>Transports</b>   Listener Maps   Listeners				
Name	Type	State	Successes	S/F Queued
APSampleTransport	TCP	Down		0
LOCALDB	DB	Down		0

The following describes each column on the **Transports** tab:

### *Name*

This is the unique name of the transport. Click the Name heading to list the transports in ascending or descending alphabetical order.

### *Type*

The type of transport: For example, **TCP**, **WMQ** (for WebSphere MQ), **DB** (for relational database), and **SAP**.

### *State*

The state of the transport can be:

- **Up** — indicates that the transport software is operational, and that data can be sent. The transport could be either successfully sending transactions or having errors sending them.
- **Down** — indicates that the transport has not been initialized. The state will change when the first transaction is sent out through this transport. For more information, see Suspending a transport.

- **Store and Forward** — applies to all transports except for TCP. In this state, the transport has encountered an error sending transactions to a remote message queue or database table and is storing the transactions for a later delivery. The node will periodically attempt to retransmit the transactions. When all the transactions have been sent, the **State** column will change to **Up**.
- **Suspended** — indicates that the transport cannot deliver transactions to its final destination even if that destination is reachable and services are up and running on the destination host. A suspended transport will attempt to store the transactions in store and forward if store and forward is enabled for that transport. A transport that has been suspended can be returned to its normal state by using the **Resume** menu from the **Transports** tab. For more information, refer to Suspending a transport.
- **Disabled** — indicates an unrecoverable error occurred in the transport. Because of the seriousness of the problem, the system disabled the transport to prevent any further processing. Once the problem is corrected, a disabled transport can be enabled by using the **Enable** menu from the **Transports** tab.
- **S&F Suspended** — indicates that the transport cannot deliver transactions to its destination. However, the transport has store and forward enabled so that transactions will be saved into a store and forward queue.

## Successes

Tracks the number of times the transport successfully delivered payload. During normal operations, this column should have a non-zero value. A transport can be referenced by many transactions making the number a higher level of accumulation.

## Failures

The number of failures due to application errors incurred by the transport. For example, SQL exception, TCP communication error, and so forth. During normal operations, this column should have a zero value.

## S/F Queued

The number of transactions currently in the store and forward queue for the transport. As these transactions are processed, this number will be decremented by 1, and the Successes column or the Failures column will be incremented as appropriate. During normal operations, this column should have a zero value.

## S/F Deleted

The number of transactions dropped while in the store and forward mode (lost data). This is due to **TTL** (time to live) parameter violation or **Max Storage** parameter violation. The TTL parameter is available on database or WebSphere MQ transports. During normal operations, this column should have a zero value. If this column has a non-zero value, it is an indication that the store and forward configuration values (**TTL** and **Max Storage** parameters) should be analyzed and possibly increased.

## Pending

The number of transactions currently in the internal pending queue for the transport. During normal operation, this column should be zero (most of the time) or a very small number.

## Pending Queue Overflow

The number of transactions dropped due to the pending queue full condition (lost data). If this column is non-zero, it is an indication of a serious condition, such as a trigger executing too frequently, a network outage, and so forth, and should be addressed quickly. During normal operations, this column should have a zero value.

There are other features available from the **Transports** tab. You can display a pop-up menu that lets you create a new transport, delete an unwanted transport, and more. The pop-up menu provides these options:



For information about **Edit** and **Delete**, refer to Editing a transport and Deleting a transport.

For information about **Suspend** and **Resume**, see Suspending a transport.

For information about **Import** and **Export**, please refer to Exporting a single transport, Exporting multiple transports, and Importing one or more transports.

For information about the **Purge Store/Forward queue** menu, see Purging data from store and

forward.

The **Enable** menu is not available unless the transport encounters a serious problem. If that is the case, the **State** column on the **Transports** tab indicates **Disabled** (and the **Enable** menu becomes available). Once the problem is corrected, you can enable the flow of transactions by using the **Enable** menu.

The **Clear counters** menu allows you to clear the all columns on the **Transports** tab.

Related topics

Suspending a transport

Purging data from store and forward

Editing a transport

Deleting a transport

Exporting a single transport

Exporting multiple transports

Importing one or more transports

## IIoTA industrial IoT Platform: Editing a transport

Using the **Transports** tab, you can edit a transport. When editing a transport, keep the following in mind:

- If the transport is currently processing, the newly modified transport parameters will be used the next time a trigger that is associated with this transport is executed.
- If there are transactions in the internal pending queue, these transactions will be processed using the newly modified transport parameters after the currently executing transaction is processed.

Follow these steps:

1. From the **Transports** tab, select the transport you want to change.

The lower portion of the **Transports** tab displays the values of the transport.

<b>Name:</b>	testMYSQLtransport
<b>State:</b>	Up
<b>Database:</b>	dwdb
<b>Last Run:</b>	2011-02-04 12:54:22
<b>Connect Time - Maximum:</b>	10400
<b>Last Error Code:</b>	-4707

- From the bottom of the **Transports** tab, click **Edit**.

You can also display the pop-up menu from the top portion of the **Transports** tab. Select the transport, and then click **Edit**.

- If applicable, a message appears with a list of the transport maps that reference the transport that you want to edit. To continue, click **OK**.

Note that the changes to the transport will not disable a trigger. However, if incorrect information is introduced to the transport, when the transport is edited and saved, an error will be issued, or the data could be sent to a store and forward queue.

The Transport window appears.

The screenshot shows a window titled "Transport" with a blue header. Below the header, the "Name" field contains "MySQLv5\_0\_59". The "Type" is set to "DB". There are three tabs: "Parameters" (selected), "Timeout", "Store & Forward", and "Mapping Log". The "Parameters" tab contains the following fields:

<b>DB Name:</b>	dwdb	<b>DB Type:</b>	MySQL
<b>Host:</b>	192.168.0.59	<b>Port:</b>	3306
<b>User:</b>	acuser	<b>Password:</b>	*****

Note that the Transport window is unique depending upon the type of transport. For this example, a DB transport is used. Therefore, the Transport window reflects parameters required for a database transport. |

4. Make your changes, and then click **Validate**. A message will tell you if there are errors.
5. If no errors are received, click **Save**. The changes are saved to the node.

#### Related topics

Purging data from store and forward  
 Deleting a transport  
 Exporting a single transport  
 Exporting multiple transports  
 Importing one or more transports

## IIoTA industrial IoT Platform: Deleting a transport

Before you delete a transport make sure it is not in use by a transport map; otherwise, the Workbench will not let you delete it.

To delete a transport:

1. From the Workbench left pane, expand the appropriate node that contains the transport you want to delete.
2. Expand **Enterprise**, and then click the **Transports** icon.
3. From the **Transports** tab, select the transport you want to delete, display its pop-up menu, and then click **Delete**.
  - If you try to delete a transport that is being used by a transport map, an error message will prevent you from deleting the transport. Click **OK** to remove the message.
  - If the transport you want to delete is not in use by a transport map, a message will ask you if you are sure you want to delete the transport, click **Yes**.



The transport is immediately removed from the **Transports** tab. In addition, the transport definition is immediately deleted from the node.

### Deleting multiple transports at the same time

You can delete multiple transports at the same time. Follow these steps:

1. From the Workbench left pane, expand the appropriate node that contains the transports you want to delete.
2. Expand **Enterprise**, and then click the **Transports** icon.  
The **Transports** tab appears. The transport you want to delete must be in a **Down** state.
3. To select all transports, anywhere in the **Transports** tab, display the pop-up menu, and then click **Select All**.
  1. To select consecutive transports, click the first transport, press and hold down SHIFT, and then click the last transport.
  2. To select transports that are not consecutive, press and hold down CTRL, and then click each transport.
4. Display the pop-up menu, and then click **Delete**.
5. A message will ask you if you want to delete all the selected transports. Click **Yes**.

The transports are immediately removed from the **Transports** tab.

Related topics

Editing a transport

Exporting a single transport

Exporting multiple transports

Importing one or more transports

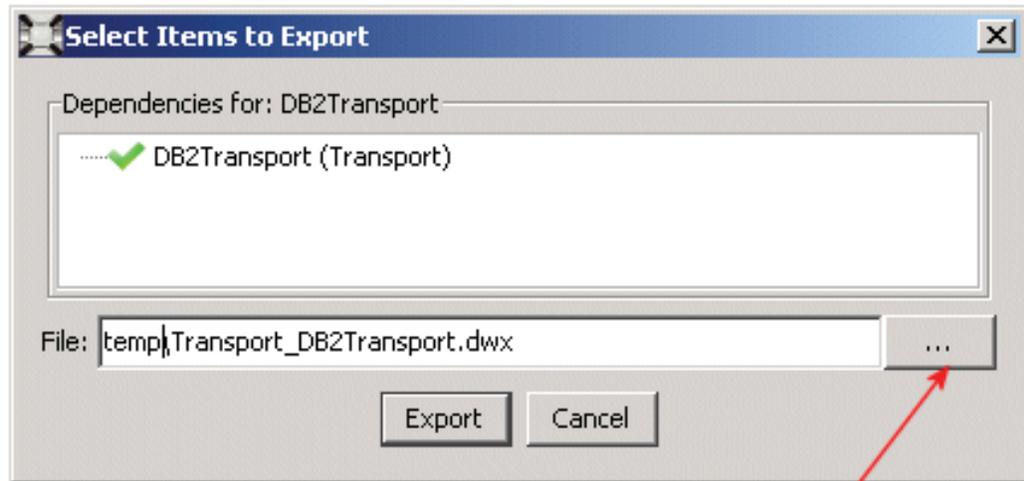
## IIoTA industrial IoT Platform: Exporting a single transport

The Workbench gives you the ability to export a transport and then import the transport into a different node. You can also export and import more than one transport at the same time. For more information, see Exporting multiple transports.

To export a single transport:

1. From Workbench left pane, expand the node whose transport you want to export.
2. Expand **Enterprise**, and then click the **Transports** icon.  
The **Transports** tab appears.
3. Select the transport you want to export, display its pop-up menu, and then click **Export**.

The Select Items to Export window appears.



Browse button

The window contains the name of selected transport.

4. Click the browse button to locate the folder to export the transport to.

The Export File Location window appears.

The transport you selected in Step 3 is automatically added to the **File name** box. The word **transport\_** is added to the front of the name (for example `transport_MyDB2transport`). The transport file name will be appended with a `.DWX` file extension. You can name the exported transport anything you want. When you type a name for the exported transport, do not type a file extension, the `DWX` file extension is automatically added when exportation takes place.

5. Navigate to the drive and folder that you want to save the transport to, and then double-click the folder. The folder name is added to the **Look in** box.
6. Click **Select**.
7. The Select Items to Export window reappears with the path and file name added to the **File** box. Click **Export**.
8. A message will tell you that the transport was successfully exported. Click **OK**.

#### Related topics

- Suspending a transport
- Purging data from store and forward
- Editing a transport
- Deleting a transport
- Exporting multiple transports
- Importing one or more transports

## IIoTA industrial IoT Platform: Exporting multiple transports

You can also export multiple transports at the same time.

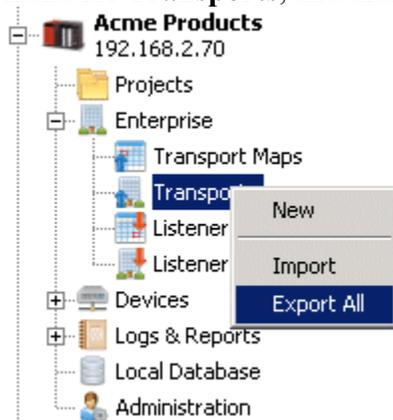
1. From anywhere on the **Transports** tab, displays its pop-up menu, and then click **Select All**.
2. Display the pop-up menu again, and then click **Export**.

The following describes three additional selection methods for exporting more than one transport from the Transports tab.

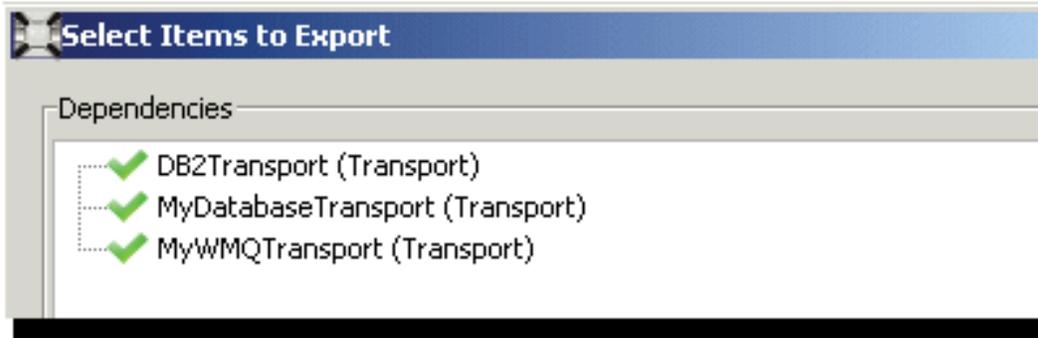
1. To select consecutive transports: Click the first transport, press and hold down SHIFT, and then click the last transport. Anywhere on the **Transports** tab, display its pop-up menu, and then click **Export**.
2. To select transports that are not consecutive: Press and hold down CTRL, and then click each transport. Anywhere on the **Transports** tab, display its pop-up menu, and then click **Export**.

You can also export all transports simultaneously using this method:

3. From the node that contains transport you want to export, expand **Enterprise**, display the pop-up menu for **Transports**, and then click **Export All**.

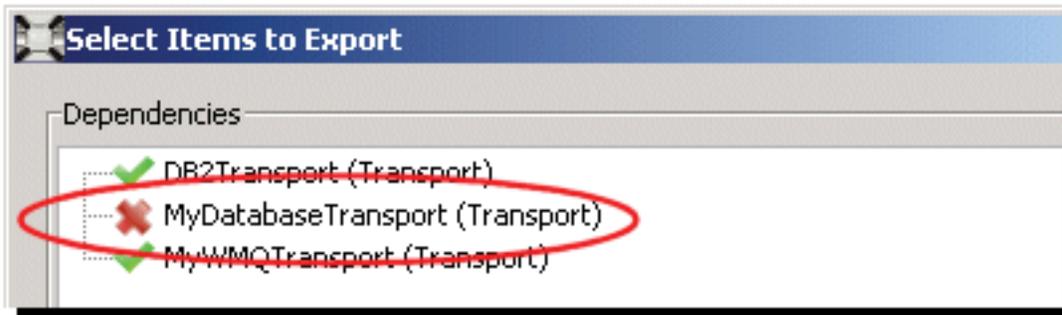


The Select Items to Export window appears.



All the transports are marked to be exported.

If you decide to not export one or more of the transports, simply select it, and it will not be exported.



4. Click the browse button to locate the folder to export the transports to.

The Export File Location window appears.

The word **transports** is automatically added to the **File name** box. The *transports* file name will be appended with a .DWX file extension. All exported transports will be bundled within this file name.

5. Navigate to the drive and folder that you want to save the transports to, and then double-click the folder. The folder name is added to the **Look in** box.
6. Click **Select**.
7. The Select Items to Export window reappears with the path and file name added to the **File** box.

- Click **Export**.

All the transports are exported into a single .DWX file.

A message will tell you that the transports were successfully exported.

- Click **OK**.

You are now ready to import one or more transports.

Related topics

Suspending a transport

Purging data from store and forward

Editing a transport

Deleting a transport

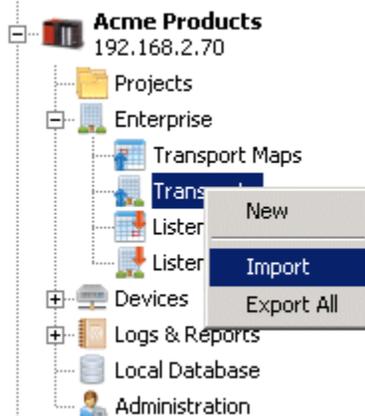
Exporting a single transport

Importing one or more transports

## IIoTA industrial IoT Platform: Importing one or more transports

It is assumed that you have previously exported a transport. You can import a single transport or multiple transports; the steps are the same.

- From the node that you want to import one or more transports into, expand **Enterprise**, display the **Transports** pop-up menu, and then click **Import**.

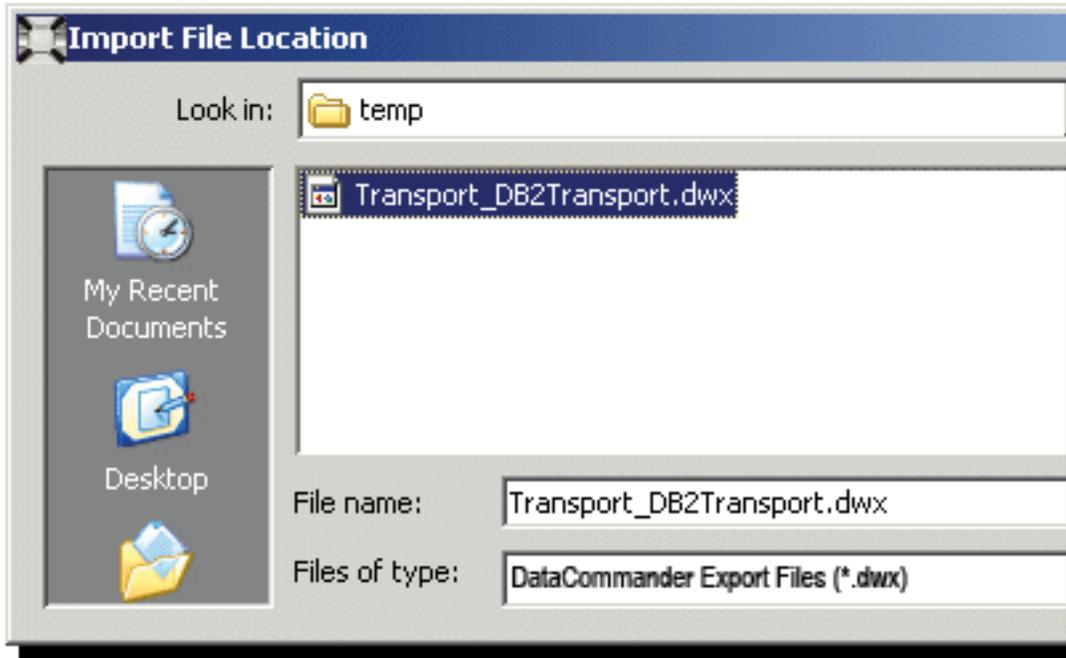


You can also import from the **Transports** tab. Anywhere on the **Transports** tab, display its pop-up menu, and then click **Import**.

The Import File Location window appears.

2. Navigate to the drive and folder that contains the exported file, and then double-click the folder. The folder name is added to the **Look in** box.
3. Select the transport file you want to import.

The file name is added to the **File name** box.



4. Click **Select**.  
The Import window appears.  
The **File** box shows the location and file of the exported file. You change the path specification by using the browse button.  
The **Items in File** table shows the transports that were exported from the specific node.
5. Click **Import**.
6. A message will tell you that the importation of the transports was successful. Click **OK**.

The transports are immediately added to the **Transports** tab.

Related topics

Suspending a transport

Purging data from store and forward

Editing a transport

Deleting a transport

Exporting a single transport

Exporting multiple transports

## IIoTA industrial IoT Platform: Suspending a transport

By default, a transport is down until it processes a transport map. If you checked the **Load transport at initialization** option when you created the transport, a connection to the host is attempted as soon as the node boots up (or immediately after leaving store and forward). Thus, when the first transaction comes in, processing is faster because the transaction does not have to initialize the transport (the connection is already established).

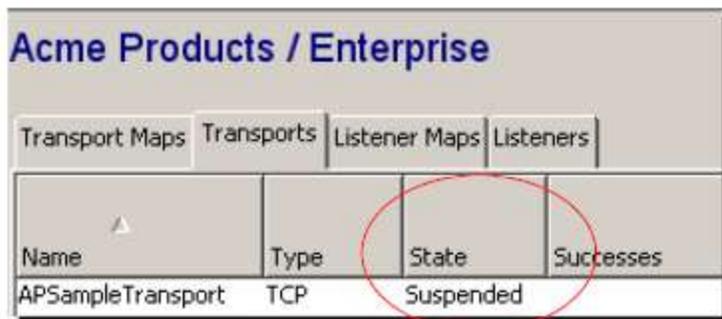
If the node is stopped and restarted, a started transport will be automatically initialized and connected to the host when the device comes back up.

A suspended transport will not deliver transactions to its destination, even if that destination is reachable and services are up and running on the destination host. A suspended transport will attempt to store the transactions in Stored and Forward, if enabled for that transport, until it is resumed. The **Time to Live** and **Max Storage** parameters will be used to control the use of the Store and Forward queue while the transport is suspended. However, if the Store and Forward capability has been turned off, or the transport does not support Store and Forward, then suspending it will cause it to fail all transactions coming in.

To suspend a transport:

1. From the Transports tab, select the transport whose transactions you want to suspend.
2. Display its pop-up menu, and then click **Suspend**.
3. A message will ask you if you want to suspend the transport, click **Yes**.

The **State** column on the **Transports** tab for the currently selected transport is changed to **Suspended**.



Acme Products / Enterprise			
Transport Maps	Transports	Listener Maps	Listeners
Name	Type	State	Successes
APSampleTransport	TCP	Suspended	

The following describes the processing for transports states whenever the status of transport is changed to suspend.

## Suspending a transport in normal mode

A transport with an **Up** state is in normal mode and transactions have been successfully delivered. The transport is connected to its host. When a transport with an **Up** state is suspended, the transport will disconnect from the host. Its status on the **Transports** tab is changed to **Suspended**. Once suspended, the first transaction that comes through the transport will be counted as a failure — if the transport does not have store and forward capability. On the other hand, if the transport has store and forward enabled, then the transaction will be saved into a store and forward queue and the transport state updated from **Suspended** to **S&F Suspended**.

The transactions will stay in the store and forward queue until their time to live expires, their maximum outstanding messages is reached, or until the transport is returned to normal mode.

If the node is rebooted, the transport state will be restored to **Suspended** or **S&F Suspended** depending on the state it was when the device was rebooted.

## Suspending a down transport

A transport with a **Down** status is not processing a transport map. When a transport with a **Down** state is suspended, the first transaction that comes through the transport will be counted as a failure — if the transport does not have store and forward capability. On the other hand, if the transport has store and forward enabled, then the transaction will be saved into a store and forward queue and the transport state updated from **Suspended** to **S&F Suspended**.

The transactions will stay in the store and forward queue until their time to live expires, their maximum outstanding messages is reached, or until the transport is returned to normal mode.

If the node is rebooted, the transport state will be restored to **Suspended** or **S&F Suspended** depending on the state it was when the device was rebooted.

## Suspending a transport in Store and Forward

A transport with a **Store and Forward** status has encountered a problem and transactions are being saved in a store and forward queue. When a transport with a **Store and Forward** state is suspended its status changes to **S&F Suspended**. The transport will no longer attempt to connect to the host. If the host becomes available, the transport will remain in a **S&F Suspended** state until it is resumed.

If the node is rebooted, the transport state will be restored to **S&F Suspended**.

## Resuming a suspended transport

A transport that has been suspended can be resumed, where it will return to its normal mode.

To resume a suspended transport:

1. From the **Transports** tab, select the transport whose transactions you want to resume.
1. Display its pop-up menu, and then click **Resume**.
1. A message will ask you if you want to resume the transport, click **Yes**.

The transport is returned to a **Down** state, unless you checked the **Load transport at Initialization** option when you created the transport and the host is available. In that case, the state of the transport will be changed to **Up**. If the state of the transport is **Down** after you resumed it, the first transaction that comes through the transport will switch the state from **Down** to **Up**. If the host is not available and store and forward is disabled, the transaction will be counted as a failure, and the state of transport will remain at **Down**.

Related topics

Purging data from store and forward

Editing a transport

Deleting a transport

Exporting a single transport

Exporting multiple transports

Importing one or more transports

## IIoTA industrial IoT Platform: Purging data from store and forward

The Transaction Server provides a store and forward queue that temporarily holds transactions that cannot be delivered successfully to the endpoint enterprise application. When you define a database or WebSphere MQ transport, you set the length of time (the **TTL** parameter) for the transaction to remain in the store and forward queue. However, there might be an occasion when you want to remove data from the store and forward queue. For example, suppose you are viewing the status window and the information in the **Store and Forward** column alerts you that the queue for a specific transport is less than 50 percent full.

<b>Node Name:</b>	AcmeProducts	<b>IP</b>	
Current Time:	2008-05-16 14:32:32	Total	
Description:			
Extended Transport Map Status			
Transport	Store & Forward	Data Loss	Pending Queue C
MySQLTransport	Under 50%	No	No

However, at that point, you decide to purge the queue of data. Follow these steps:

1. From the Workbench left pane, click the appropriate node.
2. Expand **Enterprise**, and then click **Transports**.

The **Transports** tab appears.

Acme Products / Enterprise				
Transport Maps   <b>Transports</b>   Listener Maps   Listeners				
Name	Type	State	S/F Queued	Su
testMYSQLtransport	DB	Store & Forward		5

3. Select the appropriate transport, display its pop-up menu, and then click **Purge Store/Forward queue.**

Although the store and forward queue is immediately purged of data, if the transaction failures continue, the data will also continue to be delivered to the queue. If the situation is not immediately corrected, you might want to consider stopping the trigger.

Related topics

Store & Forward tab

Editing a transport

Deleting a transport

Exporting a single transport

Exporting multiple transports

Importing one or more transports

## IIoTA industrial IoT Platform: Using the payload error log

The payload error log is an optional feature that can be used to view request and response payloads that correspond to transactions that have failed due to an application level error. The failures could be due to any of the following:

- Requests that fail because insufficient data has been supplied in the request
- The end point specified in the request is not configured to process the specific request
- The response received is not compatible with what is configured on the transport map

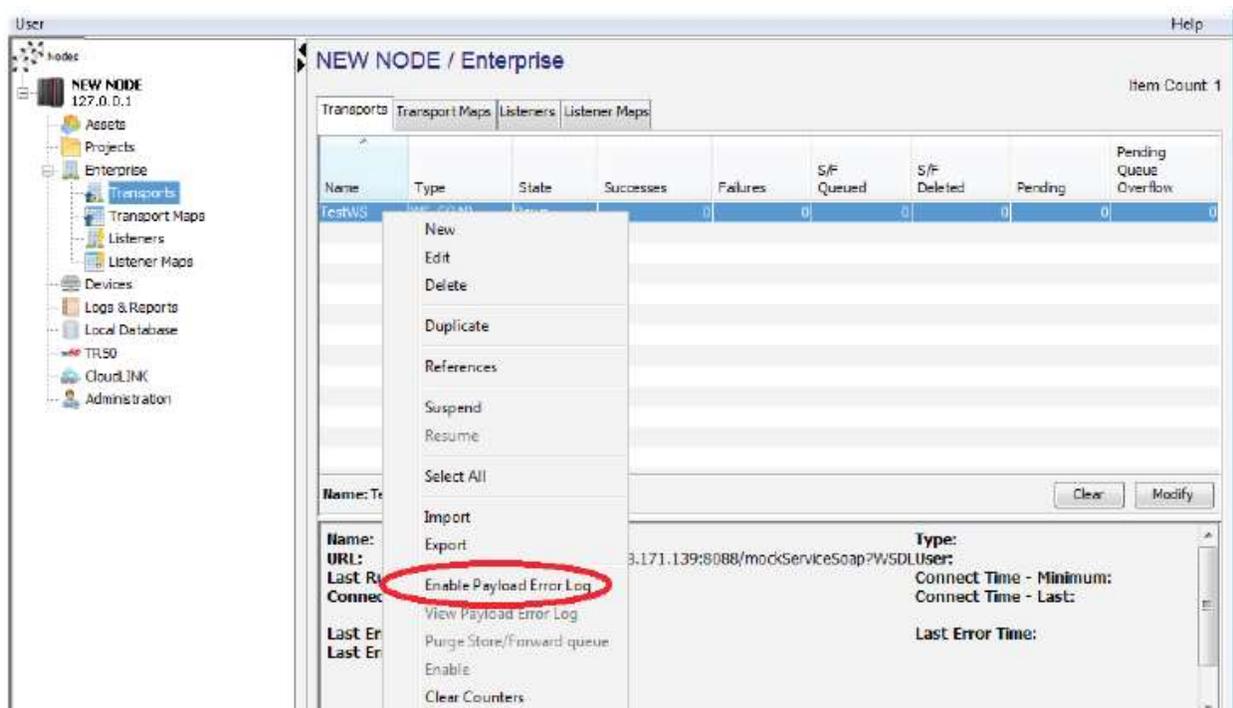
This feature is available on the following transport types.

- TCP
- HTTP
- SAP\_MII
- WS\_SOAP

## Enabling the payload error log

Navigate to the Transports panel and enable the payload error log feature by selecting the transport you want to work with. Bring up the context menu by selecting the right mouse button.

Select the **Enable Payload Error Log** option:



Once this feature is selected, the **Enable Payload Error Log** entry displays a check mark to indicate that this feature has been enabled. Once configured the setting is in place until the node restarts.

The capacity of the log, which determines how many active requests (and corresponding response) payloads are stored, is determined by the runtime platform as shown in the table below.

When a payload error log is at capacity, a new entry will replace the oldest entry in the log.

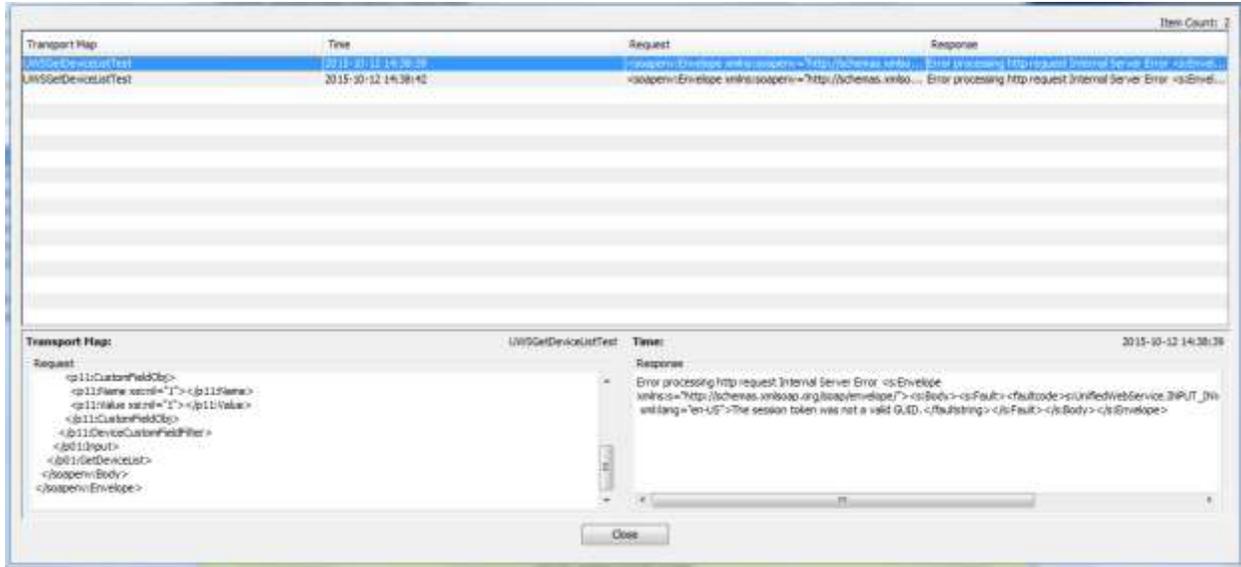
## Viewing the payload error log

If a request processed by this transport fails due to an application level error, the request and response payloads are stored in the payload error log.

Once there are view-able error log entries, the **View Payload Error Log** context menu option is enabled:

The screenshot shows the 'NEW NODE / Enterprise' interface. At the top, there are tabs for 'Transports', 'Transport Maps', 'Listeners', and 'Listener Maps'. Below these is a table with columns: Name, Type, State, Successes, Failures, S/F Queued, S/F Deleted, Pending, and Pending Queue Overflow. The 'TestWS' transport is selected, and a context menu is open over it. The menu items are: New, Edit, Delete, Duplicate, References, Suspend, Resume, Select All, Import, Export, **View Payload Error Log** (circled in red), Purge Store/Forward queue, Enable, and Clear Counters. Below the table, there are fields for Name, URL, Last Run, and Last Error, along with buttons for Clear and Modify. The 'View Payload Error Log' option is highlighted with a red circle.

Select the **View Payload Error Log** option to display the request and response payloads:



The payload error log is cleared under the following conditions:

- The Clear Counters context menu option is selected on the Transport.
- An interval of 4 hours has elapsed since the last failed request was added to the payload error log.

## IIoTA industrial IoT Platform: Transport types

### Overview

Each transport type is supported by the Transaction Server component, including the details of the following:

- Communication and interface protocol for the enterprise application
- Operations (or actions) if applicable. For example, the database transport supports Select, Insert, Update, Delete, etc. operations.
- Data type conversions between the data in the runtime system and the data in the enterprise application
- Request store and forward support if applicable
- Mapping log support if applicable.

This allows you to create your M2M solution using your enterprise applications in a vendor neutral methodology. The triggers (application logic) and your devices can be shielded from many of the enterprise application connectivity details, while still providing the two-way enterprise application access required by your M2M solution.

The information in these sections provides the details for the supported transport types.

## Assumptions

The general tasks that apply to all transport types are documented in the first several sections of Transports.

It is assumed that you are familiar with those general tasks.

The tasks that apply to building your M2M solution's application logic in triggers are documented in Projects and triggers.

Information related to using trigger actions to access your enterprise applications is documented in those sections.

## IIoTA industrial IoT Platform: Database transports

In a relational database, data is stored in tables. A table is a collection of rows and columns. Structured Query Language (SQL) is used to retrieve or update data by specifying columns, tables, and various relationships between them.

### Supported databases

You can setup a transport to take advantage of the following database servers:

- IBM DB2 Universal Database
- IBM DB2/400
- Microsoft SQL Server
- MySQL
- Oracle
- OSIP Data Archive
- Postgres
- Raima Database Manager (RDM)
- SAP HANA

#### Supported databases

Support for the database servers may vary for your specific product.

Support for some SQL operations is limited on the following database servers:

- RDM does not support Stored Procedures, Select with Update, and Select with Delete operations
- SAP HANA does not support Select with Update and Select with Delete operations.

### Assumptions

Before you begin, it is assumed:

- That your database administrator created a database and populated the database with tables. The administrator gave you the name of one or more databases.
- You have the IP address of the computer where the database resides.
- You have the user ID and password needed to log on to the database.
- You have some understanding of relational databases and SQL.

## IIoTA industrial IoT Platform: Default local database transports

You do not have to create a local database transport. When you installed the node, support for a local database transport automatically became available. The first time, you start the Workbench, and expand **Enterprise** and then click **Transports** from the right pane, the **Transports** tab shows the **LOCALDB** transports.

NEW NODE / Enterprise							
Transport Maps		Transports		Listener Maps		Listeners	
Name	Type	State	S/F Queued	Successes	F		
LOCALDB5	DB	Down		0	0		
LOCALDB4	DB	Down		0	0		
LOCALDB3	DB	Down		0	0		
LOCALDB2	DB	Down		0	0		
LOCALDB1	DB	Down		0	0		
LOCALDB	DB	Down		0	0		

This local database resides on the node. There are six predefined transports for the database:

- LOCALDB
- LOCALDB1
- LOCALDB2
- LOCALDB3
- LOCALDB4

- LOCALDB5

By having six predefined transports on the local database, you can have up to six concurrent operations on different database tables. This means that one database operation does not have to wait for an earlier operation to complete which improves throughput on the local database.

All six transports reference the same database on the node and the same set of tables. You can create, delete, and view tables in the local database using the **Local Database** window. For more information, see Local Database.

For information on how to specify a local database as a transport in a transport map, see Referencing a local database transport.

## IIoTA industrial IoT Platform: Creating the database transport

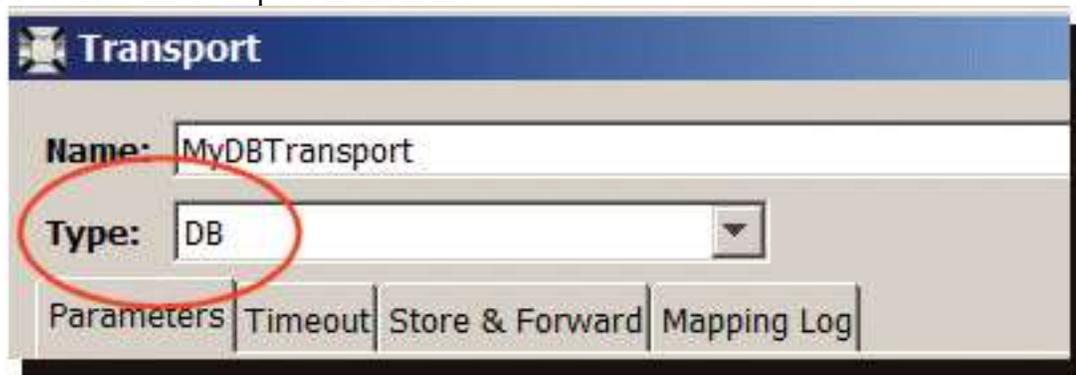
You can create a transport that will be able to generate a value for a newly inserted row or update the value of an existing row in a database table.

1. From the Workbench left pane, expand the node that you want to associate the new transport with.
2. Expand **Enterprise**, right-click the **Transports** icon to display its pop-up menu, and then click **New**.

The Transport window appears.

3. In the **Name** box, type a unique name for the transport.

This is the unique name of the transport. A database transport name can be up to 64 characters in length and can include letters, numbers, and the underscore character. You will not be able to type invalid characters. For example, spaces are not allowed. You will not be able to insert a space in the name.

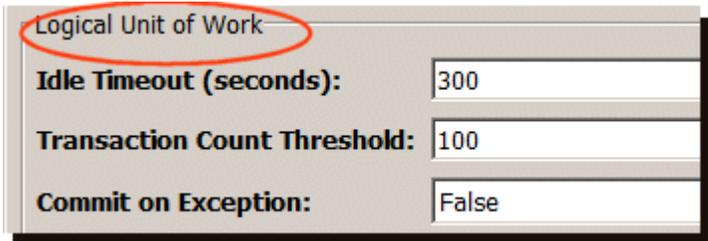


- Click the **Type** down arrow, and then select **DB**.

The Transport window changes to accommodate the definition of a database transport.

### Parameters tab

Parameter name	Description
DB Name	The name of the database. The database must exist and be populated with tables.
DB Type	The down-arrow provides a list of installed databases like the following: <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <p>DB2                          DB2                          DB2400                          MySQL                          Oracle                          MSSQL                          RDM</p> </div>

<b>Host</b>	The host name or IP address of the computer whose name you just typed in <b>DB Name</b> resides.
<b>Port</b>	Defaults to the port that is used by the database.
<b>User</b>	This is the user ID to use to access the database.
<b>Password</b>	This is the password to use to access the database.
<b>Extended Attributes</b>	Displays the Extended Attributes window. Each database has published a list of properties that can be specified to alter the connection settings when connecting to the database server. You can use the Extended Attributes window to specify these settings. Typically, you should only modify these values if required by your database administrator. For an explanation of the individual properties, refer to the web sites or documentation of the specific database vendor.
<b>Connection Pool Size</b>	A connection pool is a cache of database connections maintained so that the connections can be reused when future requests to the database are required. The <b>Connection Pool Size</b> value is the maximum number of new connections to create and store in the pool. The default value is 1. <b>Logical unit of work consideration:</b> If you are working with the <b>logical unit of work</b> feature, you must specify a connection pool size of at least 2. The Transaction Server will not allocate a connection to a Logical Unit of Work Begin action request if it is the last connection in the pool.
<b>Logical Unit of Work</b> (next three parameters)	
<b>Idle Timeout (seconds)</b>	This parameter is the number of seconds for which a logical unit of work will be kept open with no database transactions being executed. Every database transaction submitted within that logical unit of work will cause the idle timeout to reset. If the time period elapses, then the logical unit of work will be closed with a <b>Commit</b> or a <b>Rollback</b> operation based on the <b>Commit on Exception</b> parameter. The default value is 300 seconds. The minimum value is 5 seconds.
<b>Transaction Count Threshold</b>	This parameter is the maximum number of transactions that will be executed by a logical unit of work before it is closed with a <b>Commit</b> or a <b>Rollback</b> operation based on the <b>Commit on Exception</b> parameter. The default value is 100.

<b>Commit on Exception</b>	When either the <b>Idle Timeout</b> or <b>Transaction Count Threshold</b> limits are exceeded, the following will occur: <b>True</b> --- Specifies the logical unit of work to be committed. <b>False</b> — Specifies the logical unit of work to be rolled back.
<b>Load transport at initialization</b>	Select the <b>Load transport at initialization</b> check box to have the transport connect to the host as soon as the node starts up (or immediately after leaving store and forward).

## Testing and saving the database transport

To test the connection, click **Validate**. In the case of a relational database transport, validation consists of connecting to the database with the specified user ID and password credentials.

If no errors are received, click **Save**. The transport is persisted to the node.

The name of the new transport is added to the **Transports** tab and will be available for use by a transport map.

Related topics

The following **Related Topics** describes the set of tabs that let you specify values to create a database transport.

Timeout tab

Store & Forward tab

Mapping Log tab

## IIoTA industrial IoT Platform: TCP transport

A TCP transport provides a configurable TCP client to interact with a TCP server application.

The TCP transport supports the following:

- The IP address or host name, and the port number to use to connect to a TCP server application
- The type of TCP connection, `connection_oriented` or `connectionless`
- For some connection options, the ability to receive response data from the TCP server application using the same TCP connection used for sending data.

## Assumptions

You are familiar with the concepts of TCP application functions, and the specific TCP server application that will be accessed by the TCP transport.

# IIoTA industrial IoT Platform: Defining a TCP transport

The TCP transport definition panel allows you to specify the IP address or host name, and port number of the TCP server application. It also allows you to specify:

- Connection and Transaction timeouts
- The connection mode: `connection_oriented` or `connectionless`
- If the transport will receive and process response data sent from the TCP server application.

## Procedure

Follow these steps:

1. Using the Workbench access a node and navigate to the transport panel and select the New button to bring up a transport definition panel.
2. In the **Name** parameter, type a unique name for the transport.

A TCP transport name can be up to 64 characters in length and can include letters, numbers, and the underscore character. You will not be able to type invalid characters. For example, spaces are not allowed. You will not be able to insert a space in the name.

3. Use the **Type** down-arrow, and then select **TCP**.

The Transport window changes to accommodate the definition of a TCP transport.

## Parameters tab

The screenshot shows the 'Parameters' tab of the transport definition panel. The 'Name' field is empty. The 'Type' dropdown is set to 'TCP'. The 'Host' field contains '192.168.0..123' and the 'Port' field contains '4444'. The 'Mode' dropdown is set to 'connectionless'. There are three checkboxes: 'Load transport at initialization' (checked), 'Include header in payload' (unchecked), and 'Handle Response' (checked). The 'Timeout (sec)' field contains '30' and the 'Max Msg Size (KB)' field contains '2'. At the bottom are buttons for 'Save', 'Validate', and 'Cancel'.

Parameter	Description
<b>Host</b>	The IP address or host name of the computer where you want the messages sent. This is where the TCP server application program is running.
<b>Port</b>	The port number to use to connect to the TCP server application.
<b>Mode</b>	<p>The type of TCP connection management The options are:</p> <p><b>connection_oriented</b> - a TCP connection is established by the Transaction Server and maintained for subsequent sending of data to the TCP server application.</p> <p><b>connectionless</b> - a TCP connection is established by the Transaction Server; the data is sent to the TCP server application and then the connection is closed. When the next <b>Transaction</b> action supplied data is processed by the transport, a new connection is established.</p>
<b>Load transport at initialization</b>	An option to have the TCP transport connect to the TCP server application when the node starts.
<b>Include header in payload</b>	<p>The data, or payload, that is sent by the TCP transport will begin with an 8-byte header. The first 4-bytes will be the length of the data (not including the 8-byte header), and the next 4 bytes are set to zero and reserved for future use. Sending a header with the data makes it easier and more efficient for the TCP server application program to know how many bytes to receive and process.</p> <p>The byte ordering of the header is big-endian.</p>
<b>Handle Response</b>	<p>An option to indicate that the TCP Transport can receive response data sent by the TCP application.</p> <p>This feature is only allowed when the <b>Mode</b> is set to <b>connectionless</b> and the <b>Include header in payload</b> checkbox is unchecked.</p> <p>You specify how the response data is to be processed in a <b>Transport Map</b>.</p> <p>When this option is selected, additional fields become enabled as described below.</p>
<b>Timeout (sec)</b>	The number of seconds that the TCP transport will wait for response data from the TCP server application. This value should be less than the <b>Execution</b> timeout specified on the <b>Timeout</b> tab. If the Transaction action that is being processed has output data parameters, the TCP transport will wait until one of the following conditions occurs:

	<p>The TCP server application sends response data and closes the socket. The TCP transport will map response data to the output map parameters as defined in the transport map.</p> <p>The <b>Max Msg Size(KB)</b> amount of data has been received. This will result in a failure of the Transaction action.</p> <p>The <b>Timeout (sec)</b> number of seconds has elapsed. This will result in a failure of the Transaction action.</p>
<b>Max Msg Size(KB)</b>	<p>The maximum response message size in Kilo-Bytes that the TCP transport will receive in a response from the TCP server application. If there are more bytes available in the response, the transport will report this as an error.</p>

### Timeout tab

Parameter	Description
<b>Connection (seconds)</b>	The length of time the system will try to connect to a target TCP server application. For more information, see Connection timeout.
<b>Execution (seconds)</b>	How long the system should wait (once the connection is made) for a TCP transaction to complete. For more information, see Execution (sec). If you have defined the TCP transport to handled response data, make sure that this value is greater than the Timeout specified in the <b>Handle Response</b> section.

## Custom Payloads tab

For information on the Custom Payloads tab, see Custom Payloads tab.

## Testing and saving the TCP transport

1. To test the connection, select **Validate**. The validation consists of establishing a connection to the TCP server application.
2. If no errors are received, select **Save**. The transport definition is saved to the node. The name of the new transport is added to the Transports tab and will be available for use by a transport map.

Related topics

Timeout tab

Custom Payloads tab

# IIoTA industrial IoT Platform: Transport maps

A Transport Map defines the data format of the data sent to the enterprise application and, when there is response data, the format of the response data returned by the enterprise application.

A transport map gives you the ability to use map variables (that you create) instead of plant floor PLC device variable names as the source or destination of enterprise data. The Transaction Server automatically associates the map variable to a physical destination such as a specific column in a database table. The transport map feature provides a level of control between the factory floor production group and the IT department.

This section describes transport map elements and then provide step-by-step instructions for building different types of transport maps as follows:

- Overview of the transport map process
- Reviewing the Transport Map window
- Specifying transport map characteristics
- Creating map variables (Input or Output tab)
- Transport map macros
- About payloads
- Using the Transport Maps tab
- Transport map types

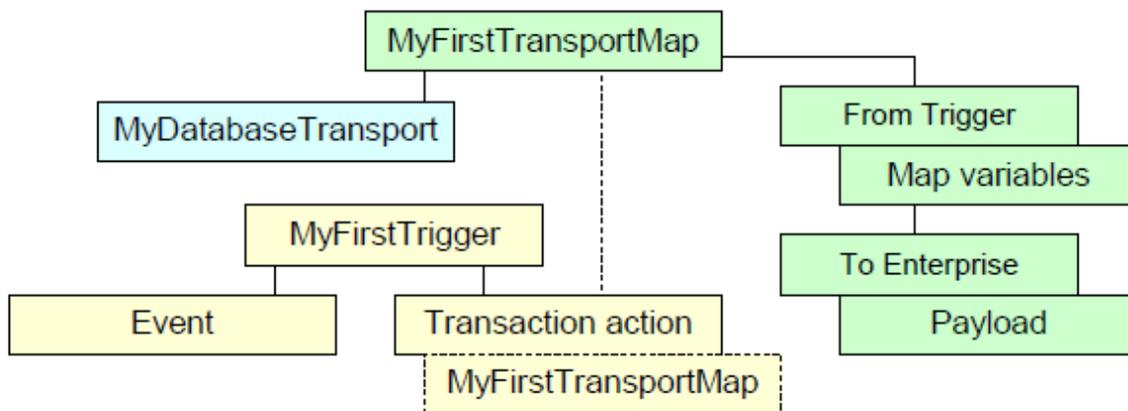
# IIoTA industrial IoT Platform: Overview of the transport map process

You can think of a transport map as the mapping (or organization) of plant floor data into a format that the enterprise application expects.

A transport map is comprised of its name, a previously defined transport, map variables (the runtime data from the trigger), and payload (the data that is delivered to the enterprise application).

The transport map is specified within a trigger as a **Transaction** action to be executed whenever an event occurs.

The following illustrates these components.



Combined, these components produce the communication between the production controller and the endpoint enterprise application program. The Workbench provides the means to enable the process to produce these components as follows:

- Create a transport. The transport is created before the transport map is created. The transport provides the protocol that is used to transfer the data. For more information, see Transports.
- Create a transport map and define the map variables and payload. All transport maps require a transport. The transport that is specified (for example a database transport) determines the parameters to set for the transport map.
- Create a trigger and then add the **Transaction** action (the transport map). The trigger is associated with the transport map and provides the event type and perhaps one or more conditions.

#### Transport map authorization

In order to use the information in this chapter, you must have been given full transport map authorization. You must be able to view, add, edit, and delete a transport map.

## IIoTA industrial IoT Platform: Reviewing the Transport Map window

A trigger contains one or many actions that are executed whenever a runtime event occurs.

At that time, the trigger that is associated with a **Transaction** action performs several operations including the building of a runtime payload per the map variables defined in the transport map.

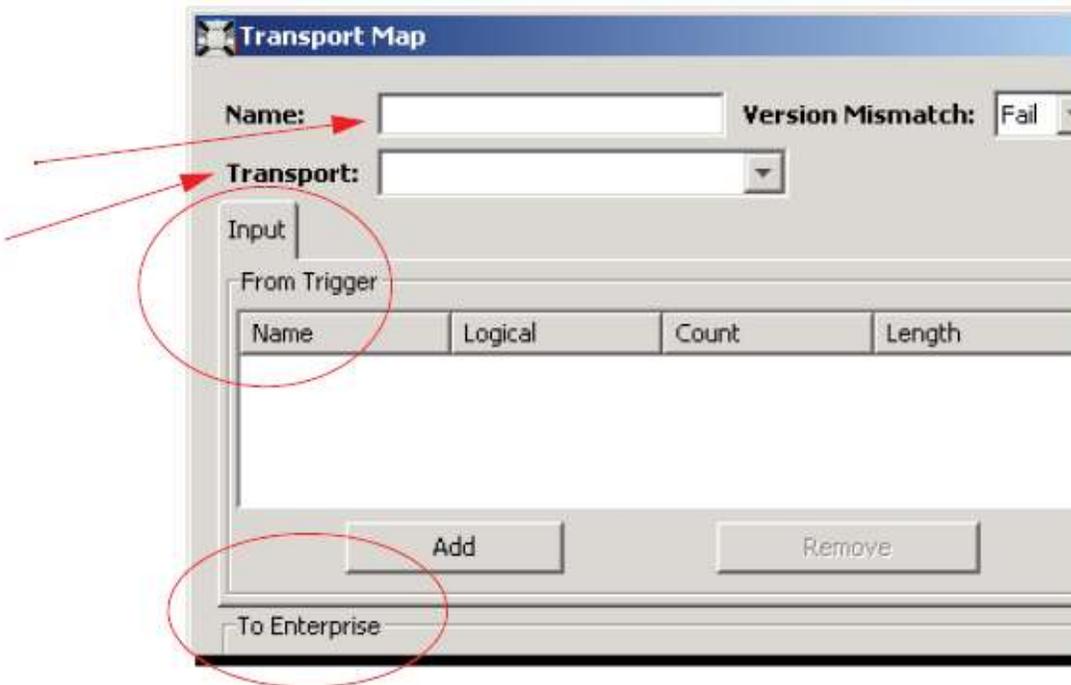
This payload is formatted for delivery to an enterprise application. In other words, the trigger will be input to the transport map.

If the transport map is bidirectional, then the system will expect data that is queried from an enterprise system to be sent to it in the output map variables.

For more information about the **Transaction** action, refer to Transaction.

The Transport Map window gives you the ability to assign a previously defined transport, create the input (or output) map variables, and build a runtime payload data definition.

The following shows the Transport Map window before a transport is assigned.



The Transport Map window provides these distinct sections:

- The very top of the window is where you specify a unique name for the transport map, the transport, and a version designation. For more information, see Specifying transport map

characteristics.

- **Input** and **Output** tabs. The **From Trigger** section displays a single **Input** tab or both an **Input** tab and **Output** tab depending on the transport type and the SQL action (for database transports). The **Input** tab is where you add map variables for the data elements that the enterprise system will store. An **Output** tab also appears when the transport map contains a stored procedure or Select, Select with Delete, and Select with Update operation. For more information, please reference Creating map variables (Input or Output tab).
- The **To Enterprise** section title indicates the direction of the data flow with regards to the transport map. The **To Enterprise** section provides the data that will be stored on the enterprise system (sometimes referred to as the payload). When you select a transport, the **To Enterprise** section of the Transport Map window changes to accommodate the transport type. The title changes to **From Enterprise** when the transport map uses a Select operation (the data will be an output of the transaction). The title changes to **From/To Enterprise** when the transport map uses a Select with Update, Select with Delete, or Stored Procedure.
- The very bottom of the window is where you validate (test) and save the transport map. For more information, refer to Testing the transport map.

## IIoTA industrial IoT Platform: Specifying transport map characteristics

The top of the Transport Map window is where you specify a unique name for the transport map, the transport to send the data on, and version information.

**Transport Map**

Name:  Version Mismatch:

Transport:

Input

From Trigger

Name	Logical	Count	Length

To Enterprise

When the transport map is saved, the name of the transport map, the name of the transport, and the transport type are shown in the Transport Maps window. For more information, see Using the Transport Maps tab.

The following describes the parameters that are available from the top of the Transport Map window:

### Name

A unique name for the transport map. A transport map name can be up to 64 characters and include letters, numbers, and the underscore character. Spaces are allowed.

### Transport

From the **Transport** drop-down list, select a previously defined transport. When you select a transport, the bottom section of the Transport Map window changes to accommodate the transport type. For more information, see

- Transport map for a database transport

### Version Mismatch

From the **Version Mismatch** drop-down list, select either **Fail** or **Pass**.

**Fail** — Immediately fail any trigger event that has a version mismatch between the current transport map and the transport map that was used in the trigger.

If the transport map was expecting data in a certain format, and you change that format, the results will be a trigger failure. If on the other hand, you change the transport map but maintain the same format of the transport map data, the fact that the versions differ will be of no consequence.

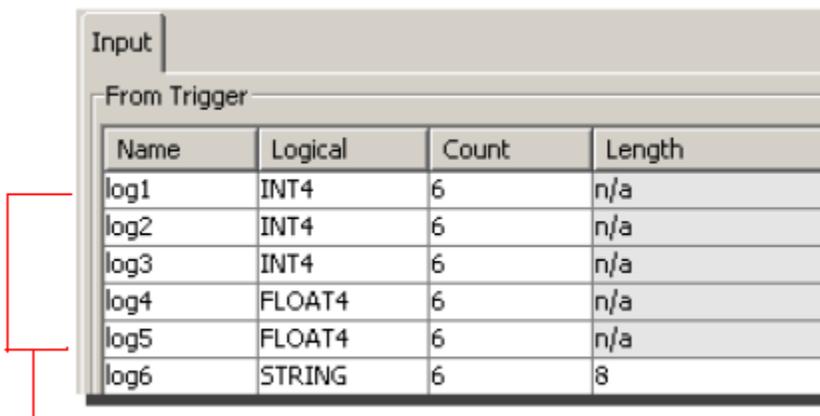
**Pass** — Allow a trigger event that has a version mismatch between the current transport map and the transport map that was used in the trigger.

The **Pass** option could be safely selected if the number of variables and the data type of the variables in the transport map did not change between versions. Changing the content of a transport map by altering either the number of variables or the data type of the variables will result in transaction data that is processed using an outdated transport map definition. This in turn could result in trigger failures as the transaction data is translated incorrectly or added to the wrong database column.

## IIoTA industrial IoT Platform: Creating map variables (Input or Output tab)

Once you establish the transport type (and the SQL action for database transports), you can create map variables using an **Input** or **Output** tab on the Transport Map window. During runtime, these variables are mapped to physical PLC device variables (when a plant floor event occurs).

The following shows an example transport map and the **Input** tab where the map variables are defined.



Name	Logical	Count	Length
log1	INT4	6	n/a
log2	INT4	6	n/a
log3	INT4	6	n/a
log4	FLOAT4	6	n/a
log5	FLOAT4	6	n/a
log6	STRING	6	8

map variables

The **Input** tab represents the definition of runtime data that is sent by a trigger execution to the transport map. This is the source data that is used to build a payload for an outbound message (such database or message system payloads).

The **Output** tab becomes available when a database transport is specified with either a Select, Select with Update, Select with Delete, or Stored Procedure. The **Output** tab also becomes available whenever an iTAC API has Out or In/Out parameters. The **Output** tab represents the definition of runtime data that is bidirectional.

Name	Logical	Count	Length
log1	STRING	5	4
log2	FLOAT4	5	n/a
log3	INT2	5	n/a
log4	FLOAT4	5	n/a
log5	INT2	5	n/a

When a transport map is bidirectional, the trigger will expect data that is queried from an enterprise system to be sent to it from the map variables on the **Output** tab.

Transport maps that use database transports with Insert, Batch Insert, and Update operations, or transport maps that use WebSphere MQ or TCP transports, will not have **Output** tabs since these transport maps do not support the ability to write data back to the PLC as Select operations, Stored Procedures, and iTAC API (with In/Out or Out parameters) do.

Both the **Input** and **Output** tab will contain these columns:

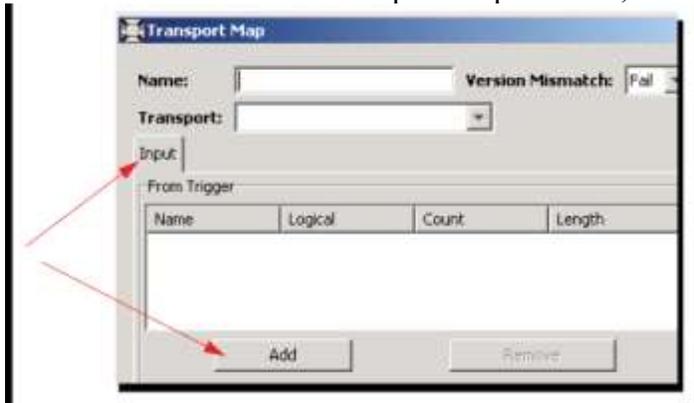
Column name	Description
<b>Name</b>	The logical name for the map variable. You can create a unique name and add each map variable separately. For more information, see <a href="#">Creating each map variable separately</a> . For database transports, you can automatically create map variables based on the column name in the database table. For more information, see <a href="#">Creating map variables automatically</a> .
<b>Logical</b>	The data type that you associate with the name. The data types are available from a list. For database transports, when the <b>Map Table</b> button is used, the data type is automatically added based on the vendor data type assigned when the database table was created. For more information, see <a href="#">vendor database data types</a> .
<b>Count</b>	A value that specifies the dimension of the map variable such as a scalar or an array. For an array, the value would be the number of elements in the array. You can double-click inside the <b>Count</b> column to change the value. <b>Output</b> tab. When you add a value for <b>Max Rows</b> , that value is also automatically added to the <b>Count</b> column on the <b>Output</b> tab. This feature applies only to Select, Select with Delete, and Select with Update operations.

<b>Length</b>	An integer value. If the data type in the <b>Logical</b> column is a string, then the value for the <b>Length</b> column is the length of the string. A string can be any length.
---------------	---

## IIoTA industrial IoT Platform: Creating each map variable separately

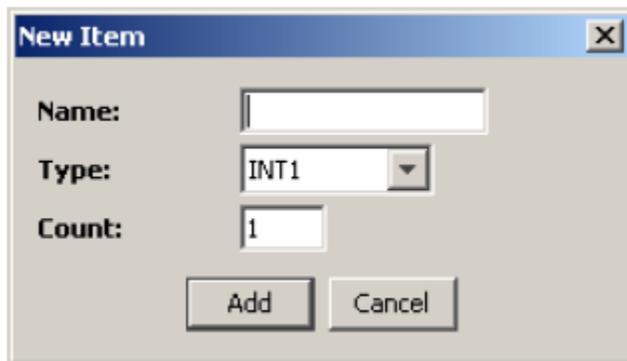
To create each map variable separately, follow these steps:

1. From the Transport Map window, under the **Input** tab, click **Add**



For transport maps that have an **Output** tab, you can also click **Add**. The process for adding a map variable to an **Output** tab is the same as adding a map variable to the **Input** tab.

The New Item window appears.



2. In the **Name** box, type the name for the map variable. For this example, **log1**.

The name can be up to 64 characters and include letters, numbers, underscore, dash characters, and spaces. Special characters such as <>' (single quotation) " (double quotation) are not allowed.

This name becomes available from the **Variable** column on the Transport Map window so that you can associate it with a payload format or database table.

- Click the **Type** down-arrow, and then select the data type that you want assigned to the name. For this example, INT2.

The screenshot shows a dialog box titled "New Item" with a close button (X) in the top right corner. It contains three input fields: "Name:" with the text "log1", "Type:" with a dropdown menu showing "INT2", and "Count:" with the text "1". At the bottom, there are two buttons: "Add" and "Cancel".

- In the **Count** box, accept the default 1. The value specifies the dimension of the map variable (for this example a scalar).

#### **Input** tab.

**Arrays:** If the data type were an array, you would change the value in the **Count** box to the number of elements in the array.

#### **Output** tab.

For transport maps that have an **Output** tab, the **Count** box value indicates the maximum number of rows that you want the Select, Select with Delete, and Select with Update operation to handle. If **Max Rows** already contains a value, the value automatically appears in the **Count** box. Note that the value for **Max Rows** can be overridden on the trigger definition that uses the transport map.

- Click **Add**.

A row appears on the **Input** tab with the information you added.

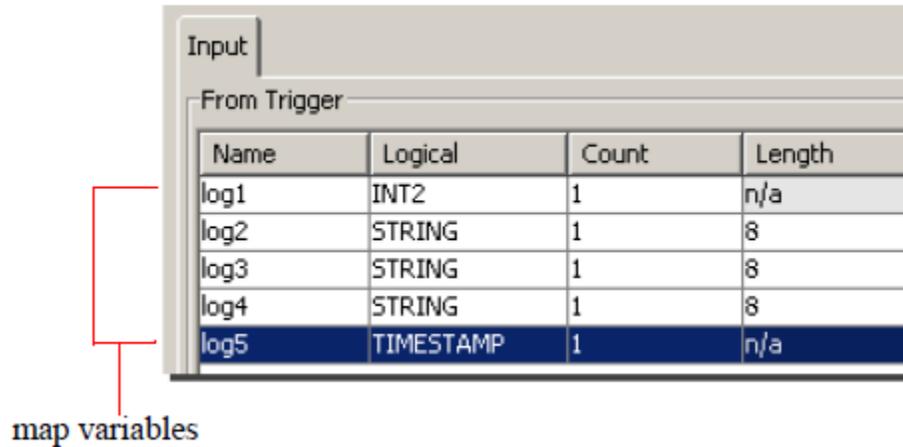
The screenshot shows the "Input" tab of a software interface. Below the tab name, there is a section labeled "From Trigger" containing a table. The table has four columns: "Name", "Logical", "Count", and "Length". A single row is highlighted in blue, containing the values "log1", "INT2", "1", and "n/a".

Name	Logical	Count	Length
log1	INT2	1	n/a

6. Repeat the steps to add all required rows naming the map variables appropriately.

7.

The final **Input** tab might look like this:



Name	Logical	Count	Length
log1	INT2	1	n/a
log2	STRING	1	8
log3	STRING	1	8
log4	STRING	1	8
log5	TIMESTAMP	1	n/a

map variables

Related topics

Creating map variables automatically

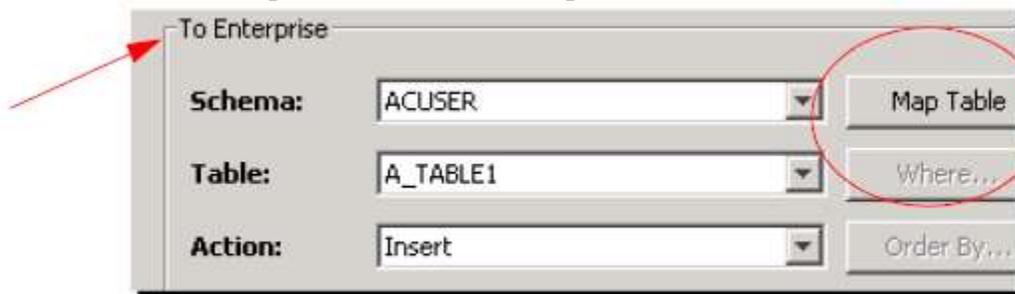
Keeping map variables when changing to another table

## IIoTA industrial IoT Platform: Creating map variables automatically

When you specify a database transport, you can automatically create the map variables. To automatically create the map variables, use the **Map Table** button that became available when you selected the database transport.

**Map Table** automatically creates a set of map variables that match all the columns and vendor database specific data types in the selected table. All the columns in the table will be populated with a name and appropriate data type.

- From the **To Enterprise** section, click **Map Table**.



The map variables are automatically added to the **Input** tab.

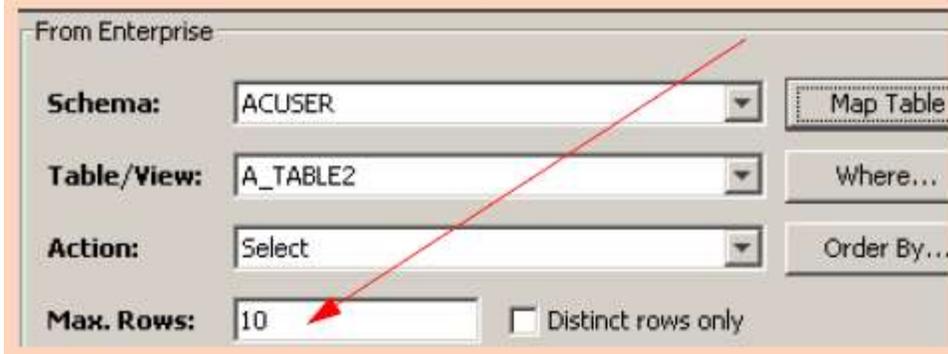
Name	Logical	Count	Length
col_C1	INT2	1	n/a
col_C2	INT4	1	n/a
col_C3	INT8	1	n/a
col_C4	FLOAT4	1	n/a
col_C5	FLOAT8	1	n/a
col_C6	STRING	1	16

You can use **Map Table** to populate the **Input** tab for Insert, Batch Insert, Update, operations and Stored Procedures.

**Output** tab. **Map Table** will also populate the **Output** tab for Select, Select with Delete, and Select with Update operations.

Name	Logical	Count	Length
col_out_C1	INT2	10	n/a
col_out_C2	INT4	10	n/a
col_out_C3	INT8	10	n/a
col_out_C4	FLOAT4	10	n/a
col_out_C5	FLOAT8	10	n/a
col_out_C6	STRING	10	16

If the **Max Rows** box contains a value, that value automatically appears in the **Count** column.



Note that the value for **Max Rows** can be overridden on the trigger definition that uses the transport map.

You cannot use the **Map Table** in conjunction with the **Output** tab for stored procedures.

The variable names are also added to the **Variable** column at the bottom of the Transport Map window (for this example the **Insert** tab).

Column	DB Type	Required	Variable
C1	SMALLINT	No	col_C1
C2	INTEGER	No	col_C2
C3	BIGINT	No	col_C3
C4	REAL	No	col_C4
C5	DOUBLE	No	col_C5
C6	VARCHAR	No	col_C6

Once you have created the map variables, you can edit the columns on the **Input** tab and **Output** tab as follows:

Name	Logical	Count	Length
col_C1	INT2	1	n/a
col_C2	INT4	1	n/a

- To change the name under the **Name** column, double-click the name, and then retype it.
- To change a data type, click the row under the **Logical** column, and then select a data type from the list.

- To change a value that specifies the dimension of the map variable, double-click the row under the **Count** column, and then type a different value.
- If the data type is a string, you will be able to type into the **Length** column to change the value of the string.

Related topics

Creating each map variable separately

Keeping map variables when changing to another table

## IIoTA industrial IoT Platform: Keeping map variables when changing to another table

Suppose you have created a set of map variables for an Insert operation but decide to change from one database table to another. You can keep the previously defined mapped variables intact.

For this example, the Insert operation was selected from the **Action** drop-down list. It is assumed that you have created a set of map variables and they appear on the **Input** tab.

The screenshot displays the configuration interface for an Insert operation. On the left, the 'Input' tab shows a table of columns from a trigger:

Name	Lo
col_C1	INT
col_C2	INT
col_C3	INT
col_C4	FLC
col_C5	FLC
col_C6	STR

Red arrows point from the text 'map variables' to the 'col\_C1' row and from 'table column names' to the 'col\_C6' row.

The main configuration panel, titled 'To Enterprise', shows the following settings:

- Schema:** ACUSER
- Table:** A\_TABLE1
- Action:** Insert

Below these settings, the 'Insert' tab shows a table mapping columns to variables:

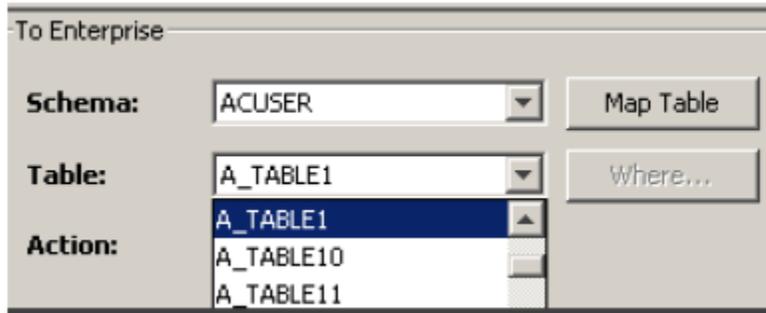
Column	DB Type	Required	Variable
C1	SMALLINT	No	col_C1
C2	INTEGER	No	col_C2
C3	BIGINT	No	col_C3
C4	REAL	No	col_C4
C5	DOUBLE	No	col_C5
C6	VARCHAR	No	col_C6

The **Insert** tab shows the current table column names and other meta data.

To retain the map variables when changing to another database table, follow these steps:

1. From **To Enterprise** section, click the **Table** down-arrow.

A list of tables appears.



2. Select the table that you want to change to.

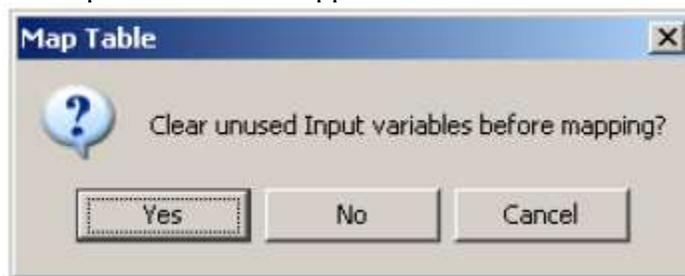
The Insert tab changes to accommodate the new table.

Column	DB Type	Required	Variable	Value
ENGINE_S...	CHAR	No		
STATION_ID	CHAR	No		
TORQUE1	REAL	No		
ANGLE1	SMALLINT	No		
TORQUE2	REAL	No		
ANGLE2	SMALLINT	No		

At this point, the **Variable** column is empty.

3. Click **Map Table**.

The Map Table window appears.



4. To keep the existing map variables, click **No**.

If you click **Yes**, the current map variables on the **Input** tab are removed. The new variables are created and added to the **Variable** column on the **Insert** tab.

Column	DB Type	Required	Variable	Value
ENGINE_SE...	CHAR	No	col_ENGINE_SE...	
STATION_ID	CHAR	No	col_STATION_ID	
TORQUE1	REAL	No	col_TORQUE1	
ANGLE1	SMALLINT	No	col_ANGLE1	
TORQUE2	REAL	No	col_TORQUE2	
ANGLE2	SMALLINT	No	col_ANGLE2	

A new set of map variables appear on the **Input** tab along with the existing map variables.

From the **Insert** tab, you can associate whatever map variables you want with the map variables on the **Input** tab.

You can also remove the map variables that do not want to keep.

1. From the **Input** tab, select the map variable that you want to remove, display its pop-up menu, and then click **Clear**.

Name	Logical	Count	Length
col_C1	WORD	1	n/a
col_C2	WORD	1	n/a
col_C3	RD	1	n/a
col_C4		1	n/a
col_C5		1	n/a

2. If you click **Clear All**, all the map variables are removed from the **Input** tab. You can click **Map Table** to create another set of map variables.

The map variable is removed from the **Input** tab.

Related topics

Creating each map variable separately

Creating map variables automatically

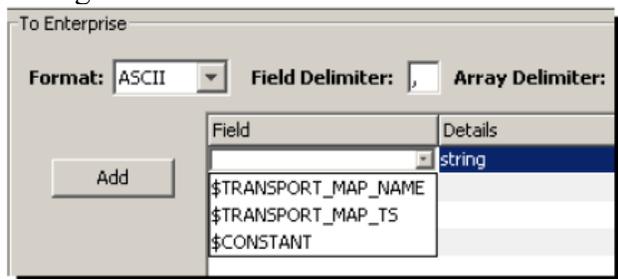
## IIoTA industrial IoT Platform: Transport map macros

A transport map can also contain macros. A macro is a pre-defined variable that does not exist in a device; however, the runtime environment has knowledge of its existence.

The following macros are available when defining a payload from the Transport Map window:

### **\$TRANSPORT\_MAP\_NAME**

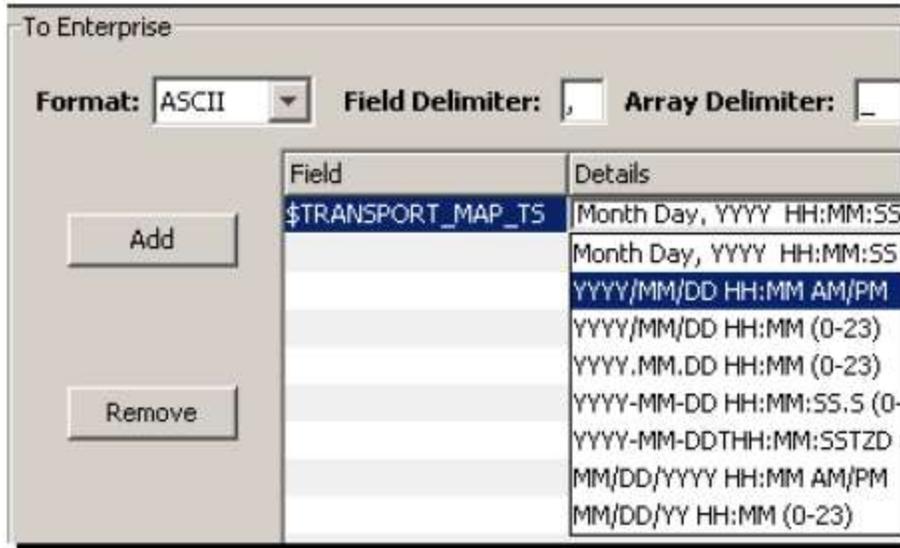
The \$TRANSPORT\_MAP\_NAME macro is used to place the name of the transaction in the runtime payload. For ASCII payloads, the name of the transport map is added to the outbound message.



For XML payloads the name of the transport map is placed within an <Item> tag. For database Insert, Batch Insert, Update, Select, Select with Update, and Stored Procedure operations, the transport map name is provided as input to the database operation when the transaction is executed.

### **\$TRANSPORT\_MAP\_TS**

The \$TRANSPORT\_MAP\_TS macro is used to place the current timestamp when the transaction payload was written to the enterprise destination as part of processing a transaction. \$TRANSPORT\_MAP\_TS is a valuable macro for automatically logging events in order to provide an audit trail that can be used to diagnose problems. You can add the \$TRANSPORT\_MAP\_TS macro to a transport map so that current time and date are recorded as part of the output format. For example, for ASCII payloads the timestamp is added to the outbound message. You can select the timestamp format from a list.



An example output for the Month Day, YYYY HH:MM:SS AM/PM TZ format is June 3, 2007 10:50:51 AM EDT.

When the **Month Day, YYYY HH:MM:SS AM/PM TZ** format is used in combination with a transport, ASCII as the output format, and a comma delimiter, the Transaction Server encloses the output timestamp with double-quotations, for example: "June 8, 2008 10:50:51 AM EDT"

## \$GENERATED

SQL databases can automatically generate values for some columns. This is useful in situations that require the database engine to:

- Generate a unique identifier for a row whose value will be stored in this column (for example, an SQLServer **TIMESTAMP** column).
- Generate a numerically increasing or decreasing value (for example, the DB2 **IDENTITY** column usually specified in the **CREATE TABLE** clause as **GENERATED ALWAYS AS IDENTITY**).
- Optionally allow the database to use the next **SEQUENCE** value as the column value (for example, an Oracle **SEQUENCE** and corresponding database **Insert** operation).

For these situations, you must not map a value to the database column; instead, map the **\$GENERATED** macro to the table column. The **\$GENERATED** macro is used in conjunction with **Insert** and **Batch Insert** operations.

You can access the value of the column that is automatically generated by the database when the row is inserted. It is available as an output parameter called **resultKey** in the output tab of

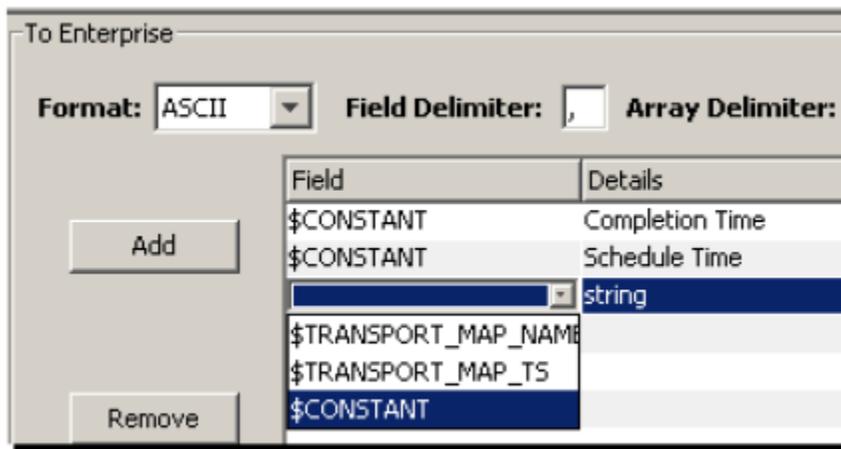
the **Insert** Enterprise Transaction action in the trigger. The value is returned only when accessing DB2, SQLServer, Oracle and mySQL.

## \$CONSTANT

The \$CONSTANT macro adds a value that does not change. Constants can be numbers or non-numeric characters such as:

- A string (Hello)
- An integer (12345)
- A real: 2.34567 (the value always has a decimal point)

You can add a constant value to all transport maps. The following shows a \$CONSTANT macro that was added to a transport map that is using a TCP transport and ASCII format as the payload output.



When the \$CONSTANT macro is selected from the list, the **Field** column becomes available as a box you can type in.

## \$NULL

This macro is available on transport maps that are defined for a database transport. It is used to specify that a NULL value gets assigned to a column or stored procedure parameter when the map executes.

The \$NULL macro can be used for the **Insert**, **Batch Insert**, **Update**, **Select with Update** and **Stored Procedure** SQL operations.

If you want to specify a NULL value from a trigger refer to the \$NULL macro section of the Trigger macros page.

Related Topics

Creating map variables (Input or Output tab)

## IIoTA industrial IoT Platform: About payloads

A transport is the mechanism that transfers the data that is generated when a trigger executes.

A payload is the useful data that you are sending or receiving.

The data can be any type including an engine serial number, machine torque reading, part count, and so forth.

The transport type you assign to the transport map determines the parameters to set for the payload.

A payload can be delivered to a message queue, relational database, TCP application program, and other enterprise application programs.

The following lists each payload that can be specified and the transport that facilitates the payload.

Payload type	Transport support
ASCII message	JMS-based, WebSphere MQ, MSMQ, HTTP, and TCP.
Freeform text	JMS-based, WebSphere MQ, MSMQ, HTTP, and TCP.
XML	JMS-based, WebSphere MQ, MSMQ, HTTP, and TCP.
Custom	JMS-based, WebSphere MQ, MSMQ, HTTP, and TCP.
Map message	JMS-based
XSD	JMS-based, WebSphere MQ, MSMQ, HTTP, and TCP.
JSON	JMS-based, WebSphere MQ, MSMQ, HTTP, and TCP
Not applicable	Relational database Supported databases: DB2, DB2400, MySQL, Oracle, MSSQL, RDM, PostgreSQL, SAP HANA.

## IIoTA industrial IoT Platform: Using the Transport Maps tab

The **Transports Maps** tab provides information about all transport maps for the current node.

To use the **Transport Maps** tab, follow these steps:

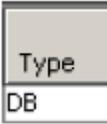
1. From the Workbench left pane, expand the node whose transport map you want to review.

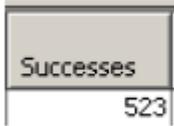
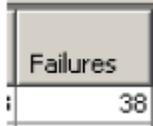
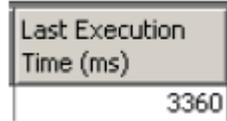
2. Expand **Enterprise**, and then click **Transport Maps**.

The **Transport Maps** tab appears in the right pane.

Acme Products / Enterprise			
Transport Maps   Transports   Listener Maps   Listeners			
Name	Transport	Type	Success
ExampleITAC_TransportMap	iTACTransport	iTAC	
ExampleTCP_TransportMap	EngineSerialNumbers	TCP	
ExampleWMQ_TransportMap	WMQTransport	WMQ	

The Transport Maps tab provides a table format with these columns:

Column name	Description
<b>Name</b>	This is the name of the transport map. Transport map names are listed in alphabetic order. The column provides standard ascending and descending alphabetical sorting. When you click the header of this column, the list of transport maps is sorted back and forth between ascending and descending order based on the alphabetic ordering of the contents in that column. When you select the row for this transport map, details are provided at the bottom of the Transport Maps tab.
<b>Transport</b>	The name of the transport assigned to the transport map.
<b>Type</b>	This is the type of transport that is associated with the transport map.  For this example, DB for database.

<p><b>Successes</b></p>	<p>Tracks the number of times the transaction successfully completed.</p>  <p>Tracks the number of times the transactions were successfully sent from the node to the enterprise system. Multiple triggers can reference a single transport map, so the successes value gives a cumulative total of all the transactions that were written to an enterprise system. A transport can be referenced by many transport maps so that counter is an even higher level of accumulation.</p>
<p><b>Failures</b></p>	<p>The number of failures due to an abnormal system event such as a database disconnect, hardware failure, device variable not found, and so forth.</p>  <p>When the failure occurs, you can view all system event messages for the current node in the <b>Exceptions Log</b> available from the <b>System Log</b>. For more information, see Exceptions Log</p>
<p><b>Last Execution Time (ms)</b></p>	<p>The time in milliseconds of the duration of the last transaction.</p> 
<p><b>Average Execution Time (ms)</b></p>	<p>The time in milliseconds computed as the average duration of the last ten transactions.</p> 

### Viewing details of the transport map

The bottom of the Transport Maps tab provides an area to view the details of the transport map definition.

<b>Name:</b>	AS400countWMQ	
<b>Transport Type:</b>	WMQ	
<b>Input Map List:</b>	v1 (STRING) v2 (INT2) v3 (STRING) v4 (INT2)	
<b>Last Error Code:</b>	<b>Transport Name:</b>	AS400WMQcount
<b>Last Error Message:</b>	<b>Last Run:</b>	n/a
	<b>Output Map List:</b>	n/a
	<b>Last Error Time:</b>	n/a

The information presented depends on the transport type and includes the list of input or output variables that might be associated with the transport map. This section is useful because it also provides error codes and error messages.

## IIOA industrial IoT Platform: Understanding transport map execution times

There is **Execution Time** section at the very bottom of the Transport Maps tab that contains current data that pertains to a transport map execution times. Transport map execution times reflect the performance of the system.

Execution Time(ms)					
Time	Minimum	Maximum	Last	Average	
Queue	0	0	0	0	
Process	0	47	47	18	
Transmit	15	47	31	37	
Post-Transmit	0	0	0	0	
Store & Forward	0	0	0	0	

The value in the **Average** column is the execution time average computed as the average of the last ten transactions. The value in the **Minimum** and **Maximum** columns are also obtained from execution times of the last ten transactions. The value in the **Last** column is the execution time of the most recent transaction.

The following describes the rows under the **Time** column\*:\*

<b>Row</b>	<b>Purpose</b>
<b>Queue</b>	The time the transaction spent in the system internal queue. If the queue time in is seconds, then the system becomes overburdened and cannot perform efficiently.
<b>Process</b>	The time taken to process variables and their values from the trigger and then convert them to the format of the endpoint enterprise application.
<b>Transmit</b>	The time spent sending the data to the endpoint enterprise application. The value of <b>Transmit</b> is perhaps the most important to monitor. For example, if the value of transmit time is too close to the transaction timeout value of the transport, the transport might switch to store and forward (if enabled) or the transaction could be dropped.
<b>Post-Transmit</b>	The time spent processing the acknowledgment or reply or data (in the case of a database Select operation) from the endpoint enterprise application. Monitoring the values of <b>Process</b> and <b>Post-Transmit</b> educates you about the complexity of the transport map being processed.
<b>Store &amp; Forward</b>	The time taken to store a transport map in the store and forward queue. The store and forward time indicate how fast transactions can be stored in the store and forward queue. If that time is too close to the frequency at which the trigger is executing, then transactions are going to be queued and could potentially exhaust the memory on the system. When coming out of store and forward, if the post-transmit time is greater than the trigger frequency, then the transport will never get out of store and forward.

**Transport map execution time notes:**

When a transport map goes into store and forward, and the value in the **Transmit** and **Post-Transmit** columns are zero, the column values will not be updated for those two execution times while the transport map is in store and forward.

When a transport map that was in store and forward is being delivered to the endpoint enterprise application (and assuming no other transport maps are being added to store and forward) the value in the **Queue** and **Process** columns are zero.

The following section describe other features available from the Transport Maps tab.

## IIoTA industrial IoT Platform: Using the pop-up menu of the Transport Maps tab

From the **Transport Maps** tab, you can also select a row, and then display its pop-up menu.



Using the pop-up menu, you can add a new transport map, change and duplicate an existing transport map, and delete a transport map. When deleting a transport map, a message will ask you if you are sure.

You can also perform global operations using **Select All** for **Import** and **Export**. For more information, see:

- Exporting a transport map
- Exporting multiple transport maps
- Importing a transport map

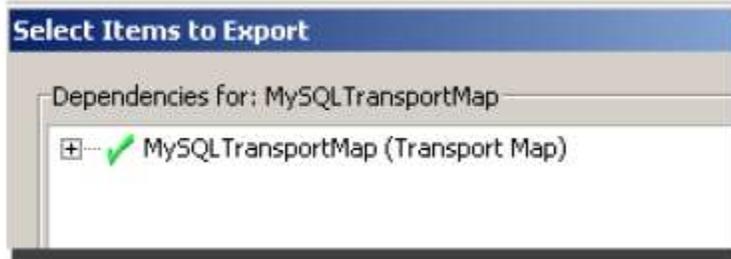
You can also perform global operations using **Select All** for **Delete** and **Clear Counters**. **Clear Counters** removes all number counts for the selected transport maps.

## IIoTA industrial IoT Platform: Exporting a transport map

The IIoTA Workbench gives you the ability to export a transport map and then import the transport map into a different node. You can also export more than one transport map at the same time.

1. From the **Transport Maps** tab, select the transport map you want to export, display the pop-up menu, and then click **Export**.

The Select Items to Export window appears.



2. Click the plus sign in front of the transport map name, to see the transport that is assigned to the transport map.



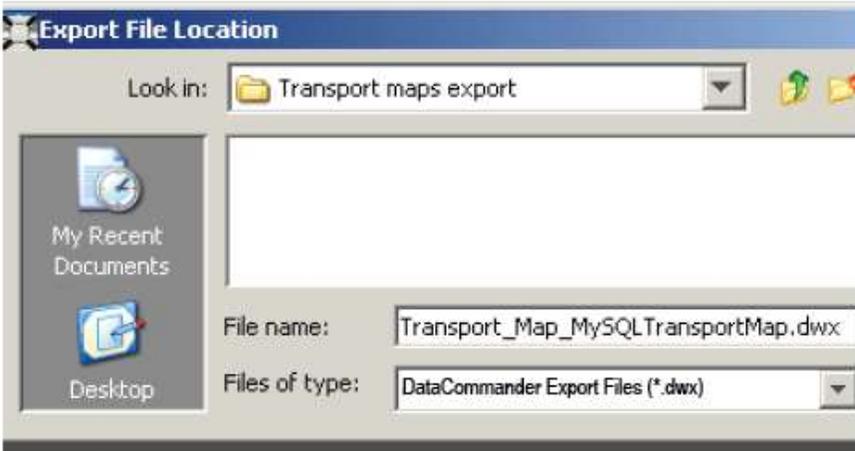
By default, both the transport map and the transport are selected for exportation.

If you do not want to export one of the transport map components simple select it and the check mark will turn to an X and the component will not be exported.



3. From the bottom of the Select Items to Export window, click the browse button.

The Export File Location window similar to the following appears:



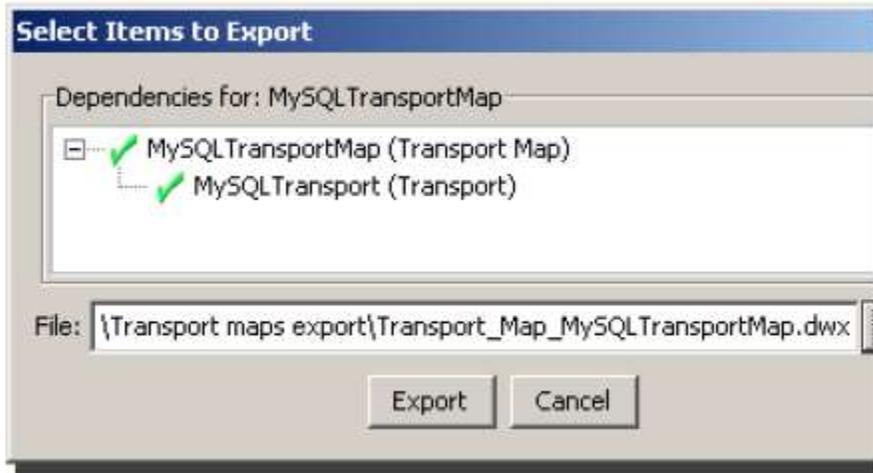
The transport map name that you selected in Step 1 is automatically added to the **File name** box. The word **Transport\_Map\_** is added to the front of the file name (for example Transport\_Map\_MySQLTransportMap). The transport map file name will be appended with a DWX file extension.

You can name the exported transport map anything you want. When you type a name for the exported transport map, do not type a file extension, the DWX file extension is automatically added when exportation takes place.

4. Navigate to the drive and folder that you want to save the transport map to, and then double-click the folder. The folder name is added to the **Look in** box.
5. Click **Select**.

The Select Items to Export window reappears with the path and file name added to the

**File box.**



6. Click **Export**.
7. A message will tell you the transport map was successfully exported. Click **OK**.

## IIoTA industrial IoT Platform: Exporting multiple transport maps

You can export more than one transport map at the same time.

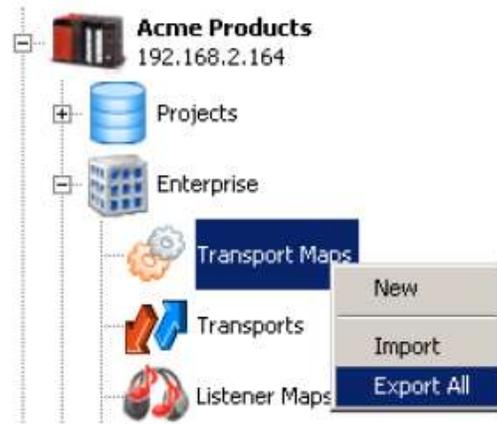
1. From anywhere in the Transport Maps tab, display its pop-up menu, and then click **Select All**.
2. Display the pop-up menu again, and then click **Export**.

The following describes three additional selection methods for exporting more than one transport map from the Transport Maps window.

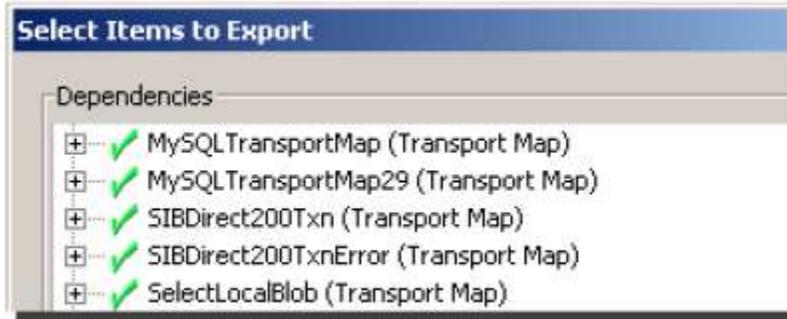
1. To select consecutive transport maps: Click the first transport map, press and hold down SHIFT, and then click the last transport map. Anywhere on the Transport Maps window, display its pop-up menu, and then click **Export**.
2. To select transport maps that are not consecutive: Press and hold down CTRL, and then click each transport map. Anywhere on the Transport Maps window, display its pop-up menu, and then click **Export**.

You can also export all transport maps simultaneously using this method:

- From the Workbench left pane, expand the node that contains the transport maps you want to export.
- Expand **Enterprise**, display the **Transport Maps** pop-up menu, and then click **Export All**.



The Select Items to Export window appears with the appropriate listing of transport maps.



All the transport maps are marked to be exported.

If you decide to not export one or more of the transport maps, simply select it, and it will not be exported.

- From the bottom of the Select Items to Export window, click the browse button.

The Export File Location window appears.

The word `Transport_Map` is automatically added to the **File name** box. The transport map file name will be appended with a DWX file extension. All the exported transport

maps will be bundled within this file name.

4. Navigate to the drive and folder that you want to save the transport maps to, and then double-click the folder. The folder name is added to the **Look in** box.
5. Click **Select**.

The Select Items to Export window reappears with the path and file name added to the **File** box.

6. Click **Export**.
7. A message will tell you the transport maps were successfully exported. Click **OK**.

You are now ready to import one or more transport maps.

## IIoTA industrial IoT Platform: Importing a transport map

It is assumed that you have previously exported a transport map. You can import a single transport map or all transport maps that were exported as a single exported file, the steps are the same.

1. From the Workbench left pane, expand the node that you want to import one or more transport maps into.

- Expand **Enterprise**, display the **Transport Maps** pop-up menu, and then click **Import**.

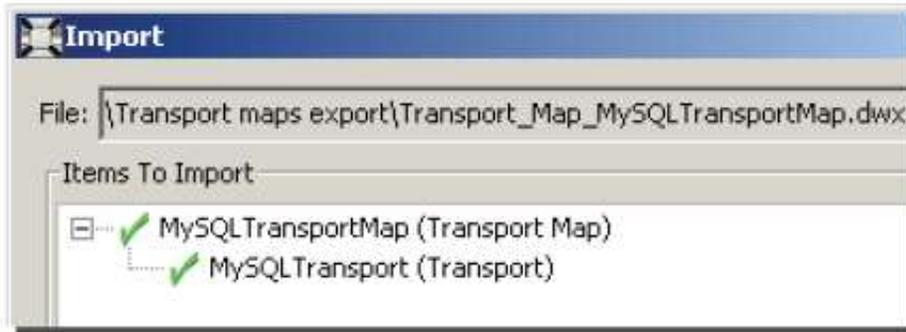


The Import File Location window appears.

- Navigate to the drive and folder that contains the previously exported transport map, and then double-click the folder. The folder name is added to the **Look in** box.
- Select the transport map file you want to import.
- When the file name is added to the **File name** box, click **Select**.



The Import window appears.



The **File** box shows the location and file name of the exported file. You change the path specification by using the browse button.

The **Items in File** table shows the components that are within the exported file. Both components will be imported.

6. Click **Import**.
7. A message will tell you that the import completed. Click **OK**.

The transport map appears in the Transport Maps tab. If the transport was also imported, it appears in the Transports tab.

#### Error

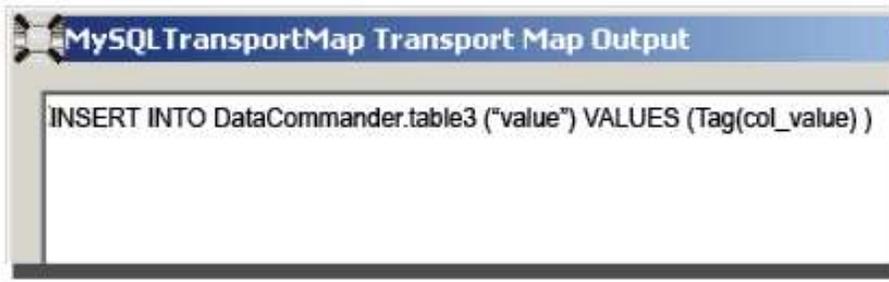
If you import a transport map that does not include a transport, when you open the transport map you will receive an error message indicating that the referenced transport was not defined on the node. You must either import the corresponding transport or assign another transport to the transport map.

## IIoTA industrial IoT Platform: Testing the transport map

Once you have saved the transport map, you can verify the content of the data that will be sent out when the trigger is actuated at runtime.

1. From the bottom of the Transport Map window, click **Validate**.

If the test is successful, a window like the following appears.



2. Click **OK** to close the window.

When the test request is submitted to the node, the node exercises the code path as if a real trigger condition had occurred and data was going to be sent out on the transport.

3. Click **Save**.

The name of the transport map, the name of the transport, and the transport type are immediately added to the **Transport Maps** tab.

## IIoTA industrial IoT Platform: Transport map types

### Overview

Each transport type supports the specifics of the enterprise application function in a transport map, including the following:

- Operations (or actions) if applicable. For example, the database transport supports Select, Insert, Update, Delete, etc. operations. The specific operation is specified in a transport map.
- Data format and mapping from runtime and device variables to enterprise application variables
- Data type conversions between the data in the runtime system and the data in the enterprise application.

The information in these sections provides the details for the supported transport map types.

### Assumptions

The general tasks that apply to all transport map types are documented in the first several sections of Transport maps.

It is assumed that you are familiar with those general tasks.

The tasks that apply to building your M2M solution's application logic in triggers are documented in Projects and triggers.

Information related to using trigger actions to access your enterprise applications is documented in those sections.

## IIoTA industrial IoT Platform: Transport map for a database transport

When you create a transport map and then associate the transport map with a database transport, you must specify a schema and then a table that you want to route data to.

Selection of the table specifies where you want to store values from map variables you set using the **Input** tab.

One of your tasks will be to associate the map variables with database table columns.

This is the payload that will be stored on the enterprise system.

### Assumptions

The following is assumed:

- You are familiar with Structured Query Language (SQL) and relational databases.
- A database transport has been created and is available on the node. If you want to learn more about transports and how the node uses them to deliver data, see Transports.
- The Workbench is started, and you have logged on.

## IIoTA industrial IoT Platform: Database destined payloads

A payload can be delivered to a relational database including IBM's DB2, Oracle, or Microsoft's SQL Server.

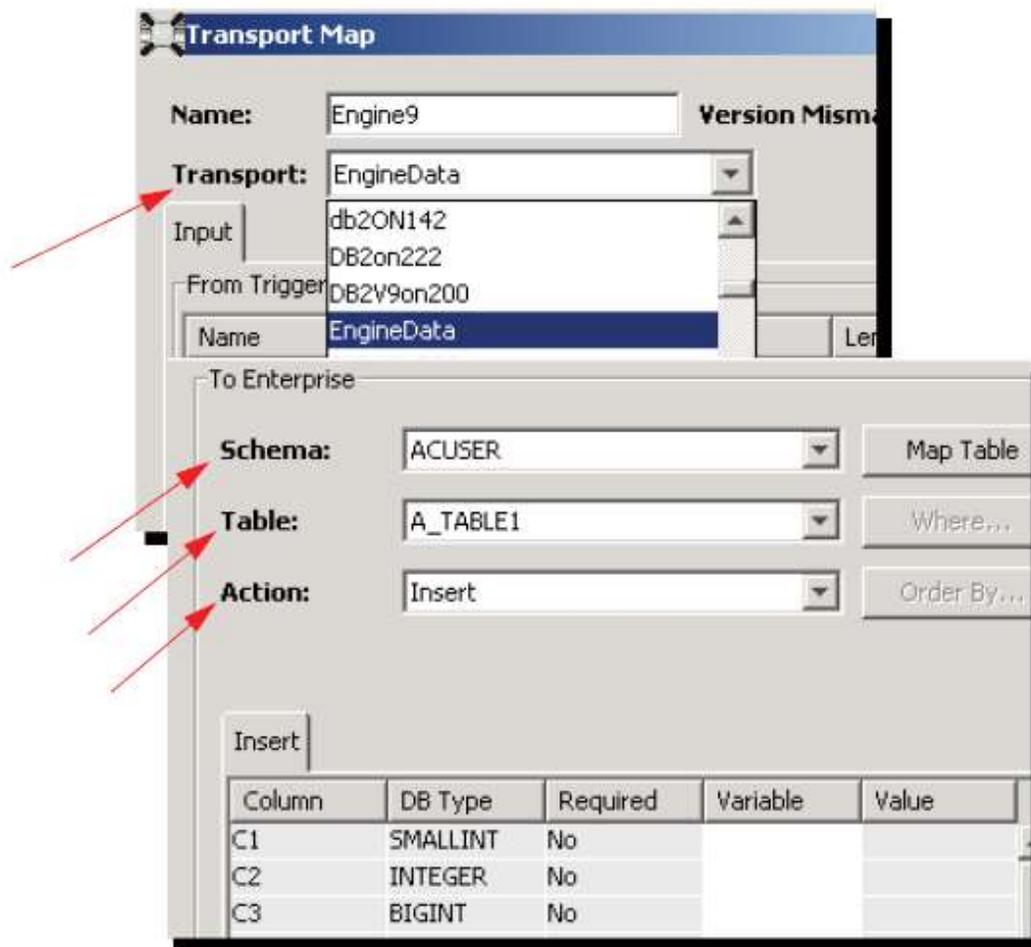
The database can reside on the local node or on an external enterprise system.

The transaction can be one way — data is transferred only to the database table.

A transaction can also transfer data in two directions — from the PLC to a database table and from a database table to the PLC.

When the transaction transfers data in two directions, it is referred to as a bidirectional transaction.

Selection of the transport will display the schemas associated with that database.



Selection of a schema will make available its associated tables. Selection of a table will display the columns defined in the table. Specifying an action determines the SQL operation or stored procedure to perform.

The following describes the default parameters that typically appear on the Transport Map window when a database transport is selected.

### Schema

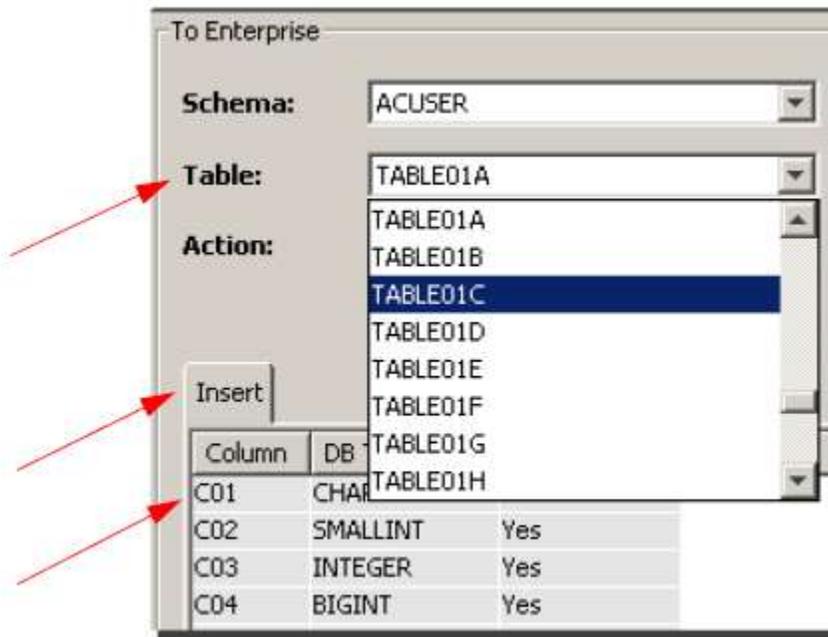
These are the schemas that are available for the database transport that you selected. The schema will contain tables whose columns and rows you want to store values from the map variable.

**Schema** defaults to the user ID specified in the transport (the user ID associated with the database). The schema is used to differentiate tables within the database.

When you select **Schema**, all tables in that schema are automatically available from the **Table** drop-down list.

### Table

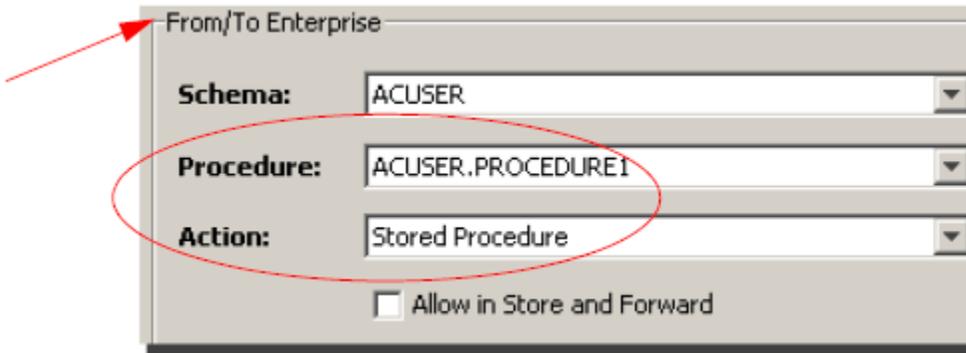
When you select **Insert**, **Batch Insert**, **Select**, **Select with Update**, **Select with Delete**, **Update**, or **Delete** from the **Action** list, you can use the **Table** drop-down list. These are the available database tables.



Select the table that you want to update or add data to. The associated SQL tab will be populated with column information for that table.

### Procedure

When you select **Stored Procedure** from the **Action** list, the **To Enterprise** title changes to **From/To Enterprise** and you will also see **Procedure**.

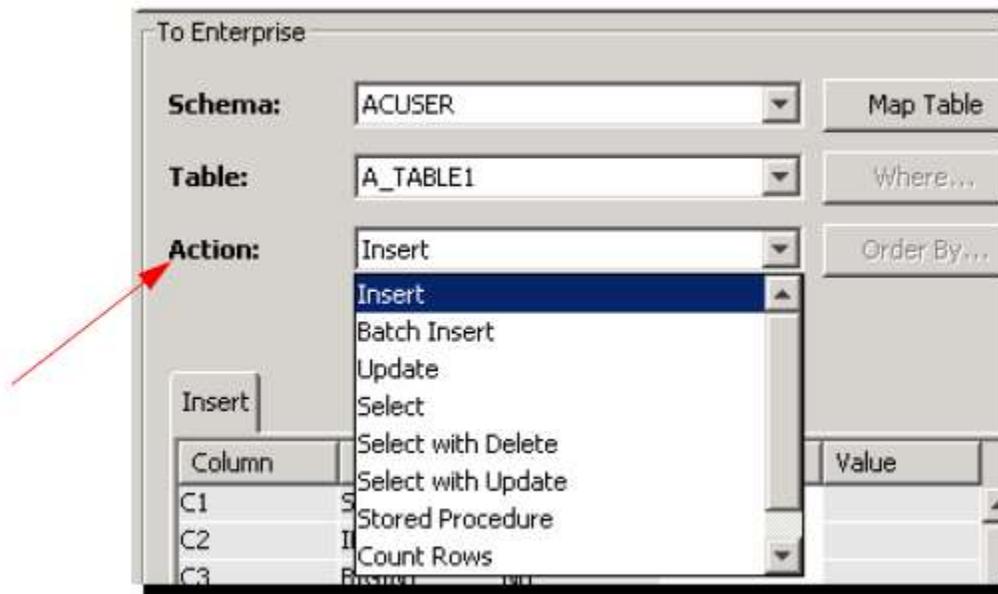


The **Procedure** drop-down list provides the available subroutines for the selected stored procedure. For information about the **Allow in Store and Forward** check box, see Using a stored procedure.

### Action

Once the table is added to the Transport Map window, you can select an action. Specifying an action determines the SQL operation or stored procedure that handle the values returned from the PLC. The direction of the flow of data provided by the specific SQL operation or stored procedure is reflected as a **To Enterprise**, **From Enterprise**, and **From/To Enterprise** section on Transport Map window.

The following shows the options available from the **Action** drop-down list.



A database transport facilitates SQL Insert, Batch Insert, Update, Select, Select with Delete, Select with Update, Stored Procedure, and Delete operations as follows:

- **Insert** — Adds a single row of data into a database table. For more information, see Inserting data (Insert).
- **Batch Insert** — Adds multiple rows of data into a database table. For more information, see Inserting multiple rows of data (Batch Insert).
- **Update** — Changes the value of a specific column in a database table. For more information, see Changing data (Update).
- **Stored Procedure** — Invokes a procedure that might update a single table or multiple tables in the same database. For more information, see Using a stored procedure.
- **Select** — Uses specific columns from a table and then updates PLC device variables with data from those columns. For more information, see Selecting columns (Select).
- **Select with Update** — Updates a set of columns on a record after the row has been fetched from the table. For more information, see Updating a set of columns (Select with Update).
- **Count Rows** — Returns the number of rows in a database table. The data is fetched from the database using criteria based on a Where clause. For more information, see Determining the number of rows on a table (Count Rows).
- **Select with Delete** — Deletes a record after the row was fetched from the table. For more information, see Deleting a set of columns on a record (Select with Delete).
- **Delete** — Deletes rows from a table. For more information, see Deleting rows in a table (Delete).

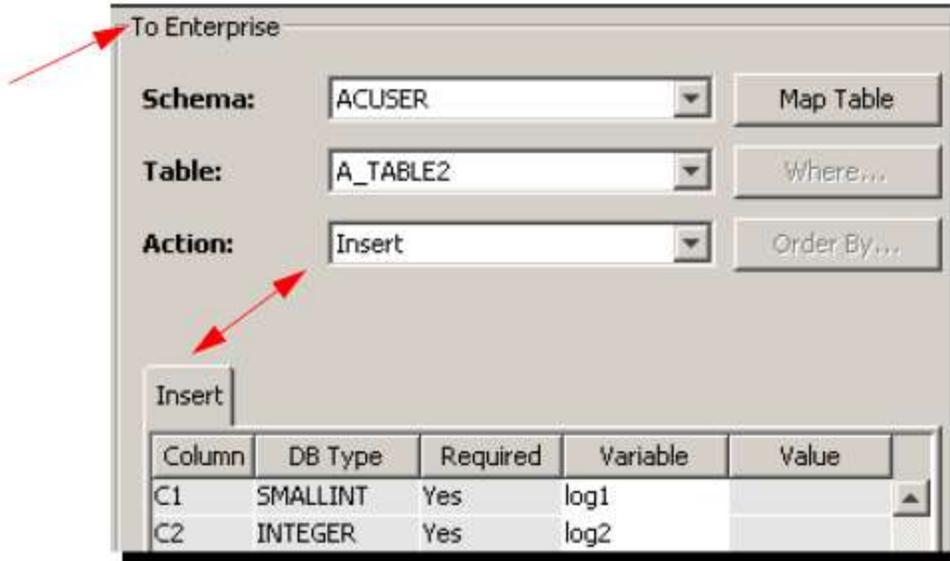
Once the action is selected, the next step is to create the map variables. You can automatically create the map variables using the **Map Table** button or you can add each map variable separately. For more information, see Creating map variables automatically.

The direction of the flow of data provided by the specific SQL operation or stored procedure appears as **To Enterprise**, **From Enterprise**, and **From/To Enterprise** on the bottom of the Transport Map window.

### **To Enterprise**

The data flows from the PLC to the enterprise database system.

The following shows an example of the **To Enterprise** section for a database destined payload.



A tab appears that matches the specified SQL operation.

For transport maps with Insert, Batch Insert, Update, and Delete operations, the **To Enterprise** section will contain these columns:

Column name	Description
<b>Column</b>	The names in this column correspond to the column names from the selected database table.
<b>DB Type</b>	These are the custom data types that are associated with the database transport. For more information, see Understanding data types.
<b>Required</b>	This column reflects the meta data for the <b>Required</b> column as supplied by the database table specification. When <b>Yes</b> , the column must have data. For example, if you clear a value from a row of the <b>Variable</b> column that says <b>Yes</b> , and then try to save the transport map, a message will tell you that required column must have data. If the <b>Required</b> column shows <b>No</b> , the column will not become populated unless you set a value using a map variable or macro. Note that Update operations do not have a <b>Required</b> column.
<b>Variable</b>	This column contains the same map variables as the <b>Input</b> tab. Using the <b>Map Table</b> button, you can automatically create map variables and they will appear on the <b>Variable</b> column. For more information, see Creating map variables automatically.
<b>Value</b>	This column contains the value of a constant. You add the constant from the <b>Variable</b> column. Note that Select operations do not have a <b>Value</b> column.

For more information on how to define the **To Enterprise** section for a transport map that uses an Insert operation, see Inserting data (Insert).

## From Enterprise

The data flows from the database table to the PLC.

The following shows an example of the **From Enterprise** section for a Select operation. A Select is considered a read-only operation because no data is modified on the local or enterprise database system.

Column	DB Type	Nullable	Variable	Default Value
C1	TIMESTAMP	Yes	col_out_C1	03/03/13 12:13...
C2	SMALLINT	Yes	col_out_C2	
C3	INTEGER	Yes	col_out_C3	
C4	SMALLINT	Yes	col_out_C4	
C5	SMALLINT	Yes	col_out_C5	
C6	INTEGER	Yes	col_out_C6	

For transport maps with Select operations, the **From Enterprise** section will contain these columns:

Column name	Description
<b>Column</b>	The names in this column correspond to the column names from the selected database table.
<b>DB Type</b>	These are the custom data types that are associated with the database transport. For more information, see Understanding data types.
<b>Nullable</b>	This column reflects the meta data for the <b>Required</b> column as supplied by the database table specification. When <b>Yes</b> , the column must have data. For

	example, if you clear a value from a row of the <b>Variable</b> column that says <b>Yes</b> , and then try to save the transport map, a message will tell you that required column must have data. If the <b>Required</b> column shows <b>No</b> , the column will not become populated unless you set a value using a map variable or macro.
<b>Variable</b>	This column contains the same map variables as the <b>Input</b> tab or <b>Output</b> tab. Using the <b>Map Table</b> button, you can automatically create map variables and they will appear on the <b>Variable</b> column. For more information, see Creating map variables automatically.
Default Value	Optional. This column is used in <b>Select</b> operations to handle null values from the database. When specifying a value in the <b>Default Value</b> column, the null returned from the database is substituted with the default value. If the database returns a value, the default value is not used.

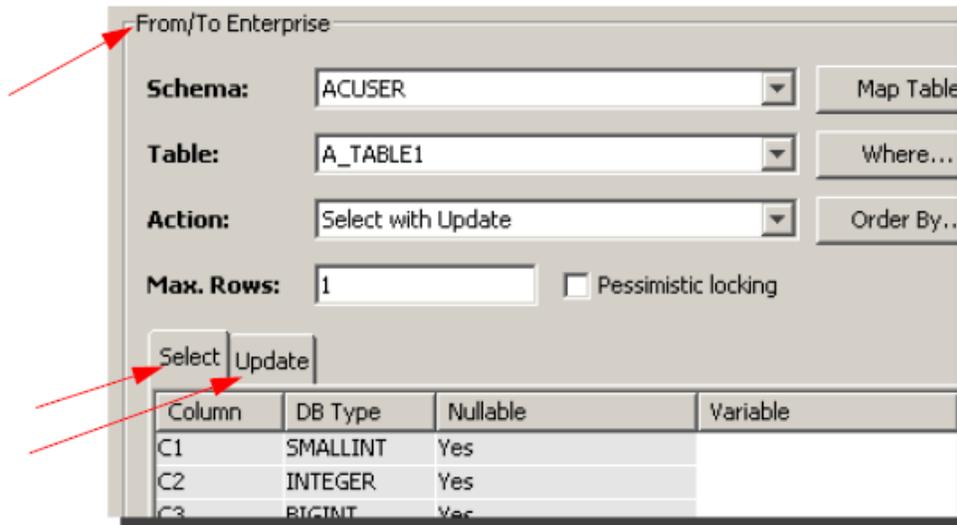
For more information, refer to Selecting columns (Select) .

### From/To Enterprise

The data flows in two directions — from the database table to the PLC and from the PLC to the database table.

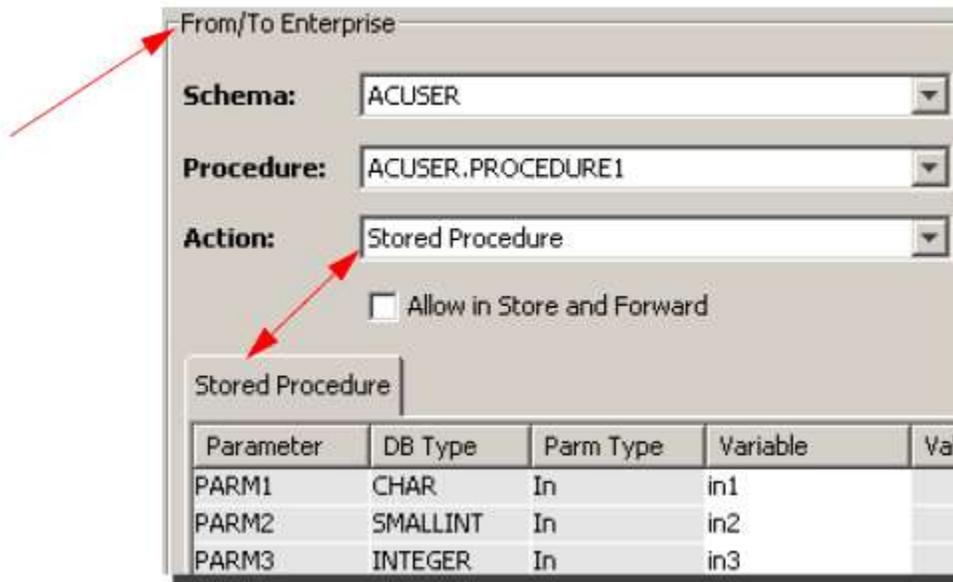
The **From/To Enterprise** section title indicates the direction of the data flows both ways and applies to read and write operations. The **From/To Enterprise** section applies to a Select with Update, Select with Delete, and Stored Procedures.

The following shows an example of the **From/To Enterprise** section for a Select with Update operation



Notice the **Select** and **Update** tabs. The Select with Update operation is used to update a set of columns on a record after the row has been fetched from the table. For more information, see *Updating a set of columns (Select with Update)*. For information about the **Pessimistic locking** check box, see Pessimistic locking.

For stored procedures, the tab is slightly different.



The **From/To Enterprise** section provides a **Stored Procedure** tab with these columns:

Column name	Description
<b>Parameter</b>	The names in this column correspond to the parameter names from the selected stored procedure.
<b>DB Type</b>	These are the custom data types that are associated with the database transport. For more information, see Understanding data types.
<b>Parm Type</b>	<p>The parameter was declared as IN, OUT, or INOUT.</p> <p><b>IN</b> — the data that is being sent to the stored procedure.</p> <p><b>OUT</b> — the data that is being returned from the stored procedure (bidirectional).</p> <p><b>INOUT</b> — the data that is being sent to the stored procedure and then returned from the stored procedure (bidirectional).</p> <p>Note if you are running Microsoft SQL Server, you might see a <b>Return</b> parameter. This parameter type is automatically added by the SQL Server program. For more information, refer to the stored procedure documentation that came with Microsoft SQL Server.</p>

<b>Variable</b>	This column contains the map variables from the <b>Input</b> tab or <b>Output</b> tab. Using the <b>Map Table</b> button, you can automatically create map variables and they will appear on the <b>Variable</b> column. For more information, click <a href="#">Creating map variables automatically</a> .
<b>Value</b>	This column contains the value of a constant. You add the constant from the <b>Variable</b> column.

**Stored procedure:** Note that a stored procedure with only an IN parameter is considered a one way transaction to the enterprise database system.

### Related topics

[Using a stored procedure](#)

## IIoTA industrial IoT Platform: Creating the transport map for a database transport

This section describes how to add a database transport to a transport map and configure the map variables and payload.

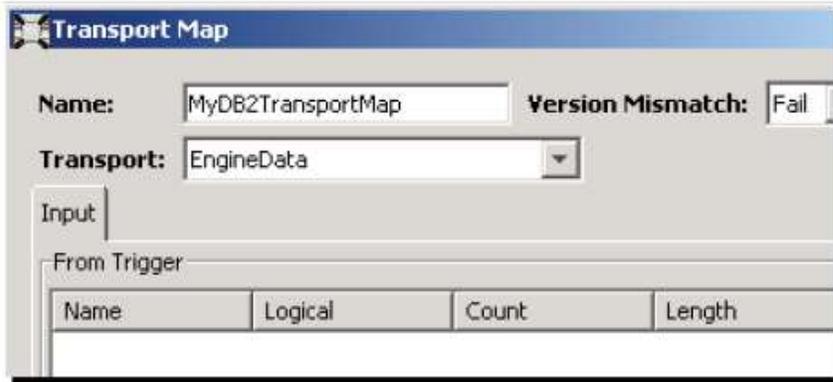
1. From the Workbench left pane, expand the node that you want to add the transport map to.
2. Expand **Enterprise**, right-click the **Transport Maps** icon to display its pop-up menu, and then click **New**.

The Transport Map window appears.

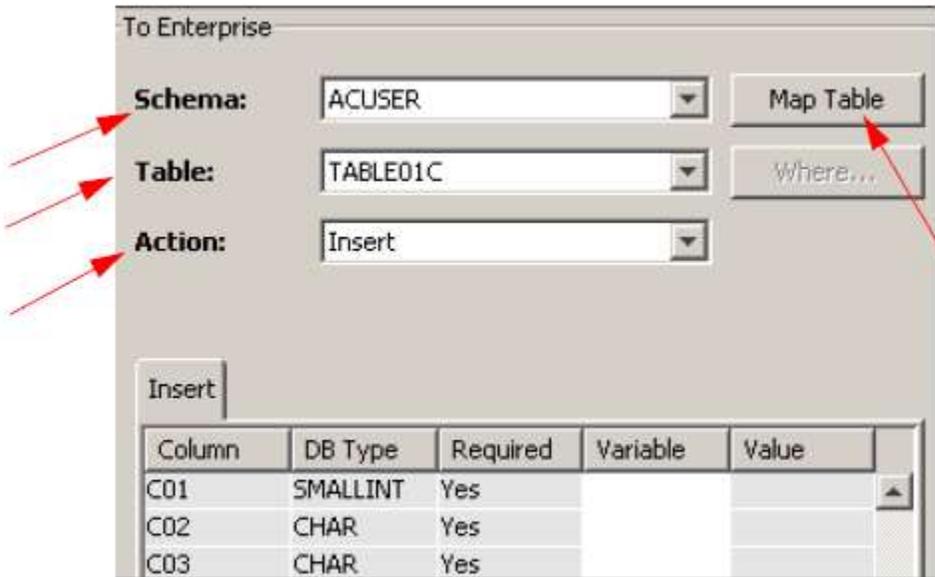
3. Name the transport map as appropriate, and then set the **Version Mismatch** option to either **Pass** or **Fail**.
4. From the Transport Map window, next to **Transport**, click the down arrow.

A list of pre-defined transports appears.

5. Select the transport whose database you want to connect in.



The following shows a partial view of the Transport Map window when a database transport is selected.



For this example, the **To Enterprise** title appears because the window defaults to an Insert operation.

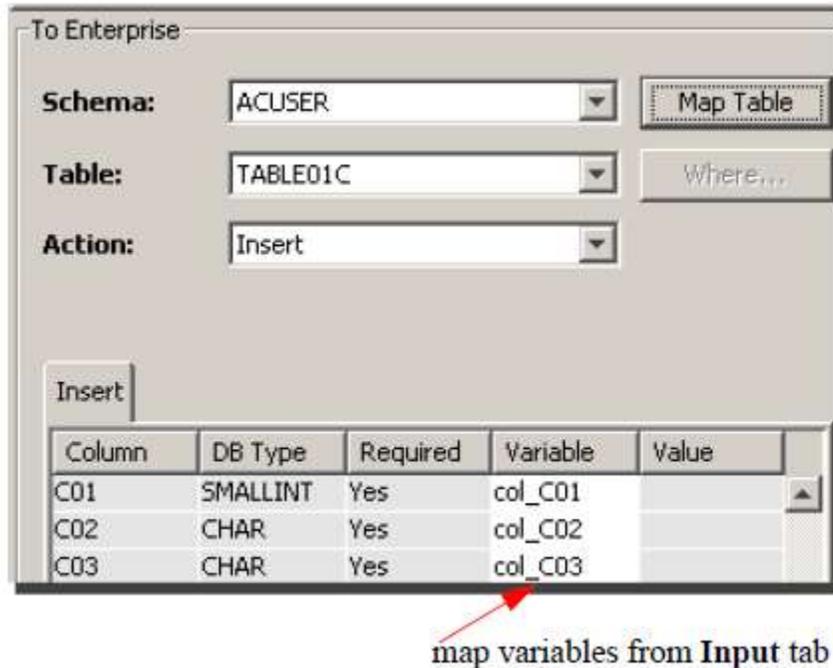
## IIoTA industrial IoT Platform: Inserting data (Insert)

The **Insert** option allows you to add a new row of data into a database table whenever a triggered event occurs.

You select the schema and table that is used as the target for this operation.

The **Input** tab will consist of a collection of map variables whose values will be mapped to columns of that table.

You can add the map variables automatically using the **Map Table** button (see Creating map variables automatically) or create each map variable individually (see Creating each map variable separately).



When the trigger event occurs, a record is inserted into the table.

The map variable value will be used when executing the SQL Insert operation at runtime.

For more information on how to build a transport map based on an Insert operation, see Your first transport map with a database transport.

## IIoTA industrial IoT Platform: Inserting multiple rows of data (Batch Insert)

The **Batch Insert** option gives you the ability to insert multiple rows of data into a single database table when a triggered event occurs.

Using Batch Insert you can:

- Insert entire arrays in one transaction rather than one insert per array element.

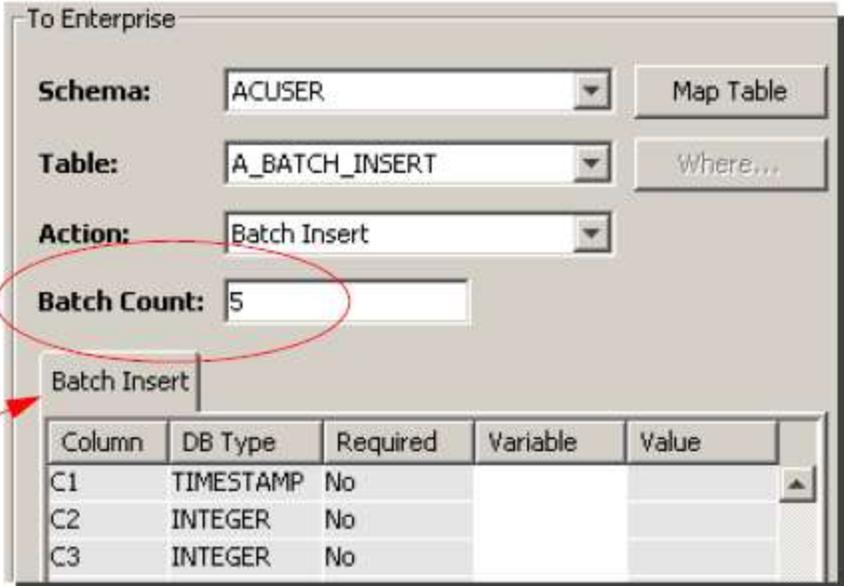
- Improve database performance. Most database systems offer increased performance for batch insert operations rather than individual inserts.

The following assumes that you have specified the database transport and selected a schema and table.

1. Click the **Action** down arrow, and then select **Batch Insert**.

The **Batch Count** box and the **Batch Insert** tab become available.

The **Batch Insert** tab is populated with the column information for the selected table. This is the table that will be used for the target of this batch insert.



The screenshot shows a dialog box titled "To Enterprise" with the following fields and controls:

- Schema:** ACUSER (dropdown menu)
- Table:** A\_BATCH\_INSERT (dropdown menu)
- Action:** Batch Insert (dropdown menu)
- Batch Count:** 5 (text input field, circled in red)
- Map Table** button
- Where...** button
- Batch Insert** tab (selected, indicated by a red arrow)

The "Batch Insert" tab contains a table with the following data:

Column	DB Type	Required	Variable	Value
C1	TIMESTAMP	No		
C2	INTEGER	No		
C3	INTEGER	No		

2. Type a numeric value in the **Batch Count** box. For this example, 5. The value indicates the number of inserts to batch. For example, if the size of an array equals 8, and you specify 5 in the **Batch Count** box, only the first 5 elements in the array will be used.

The next step is to create the map variables. You can automatically create the map variables using the **Map Table** button. When you use **Map Table**, the value typed in the **Batch Count** box is automatically added to the **Count** column of the **Input** tab.

Name	Logical	Count	Length
col_C1	TIMESTAMP	5	n/a
col_C2	INT4	5	n/a
col_C3	INT4	5	n/a

If you add each map variable separately, the value of the **Count** parameter (for the map variable) must match the value in the **Batch Count** box; otherwise, an error will occur. For more information, see *Creating map variables automatically* and *Creating each map variable separately*.

### Batch Insert and the Trigger window

When a transport map that contains a batch insert operation is added as a trigger action, a special **batchCount** variable appears on the **Input** tab of the Trigger window.

Name	Logical	Count	Value	Type
log1	INT2	1000	Local CPU 1.D[1000]	INT2
log2	INT4	1000	Local CPU 1.D[2000]	INT4
log3	INT4	1000	Local CPU 1.D[4000]	INT4
log4	FLOAT4	1000	Local CPU 1.D[6000]	FLOAT4
batchCount	INT4	1		

At runtime, the **batchCount** variable is used to control how many array elements to send as one batch.

You can map the **batchCount** variable to a device variable or local variable; and whenever the value of the device variable or local variable changes, the batch insert operation batches whatever the number is of the array element.

This accommodates a scenario whereby each time the trigger executes, the number of array elements to batch for inserts changes.

Otherwise, it will default to the number specified in the transport map (**Batch Count** box) and will always be the same number for each run.

It is important to know that the value for **batchCount** cannot be set to a value greater than the value in the **Count** column for the array variables.

Input				
Name	Logical	Count	Value	Type
log1	INT2	1000	Local CPU 1.D[1000]	INT2
log2	INT4	1000	Local CPU 1.D[2000]	INT4
log3	INT4	1000	Local CPU 1.D[4000]	INT4
log4	FLOAT4	1000	Local CPU 1.D[6000]	FLOAT4
batchCount	INT4	1	2	CONSTANT

value of **batchCount**

Also, if you set the value of **batchCount** to 1, the batch insert operation will behave like a regular insert operation. In other words, one row is batched for insert (a single insert statement is performed).

## IIoTA industrial IoT Platform: Changing data (Update)

The **Update** option allows you to change the value of a specific column whenever a triggered event occurs.

You select the schema and table that is used as the target for this operation.

The **Input** tab will consist of a collection of map variables whose values might be mapped to columns used in a **Where** clause.

Other items in the **Input** tab can be mapped to update the columns of the record that is selected as a result of the **Where** clause.

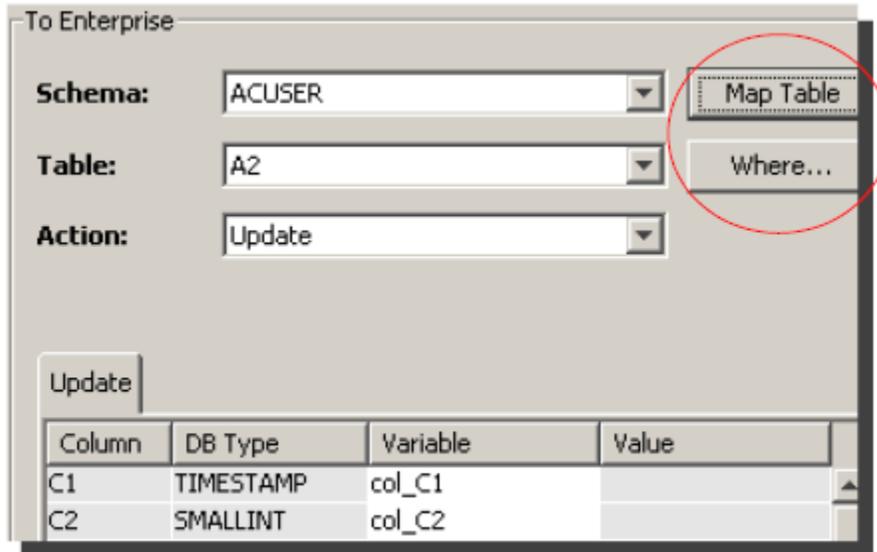
The following assumes that you have specified the database transport and selected a schema and table.

1. Click the **Action** down arrow, and then select **Update**.

The **Where** button becomes available.

The **Update** tab is populated with the column information for the selected table. This is the table that will be used for the target of this update operation.

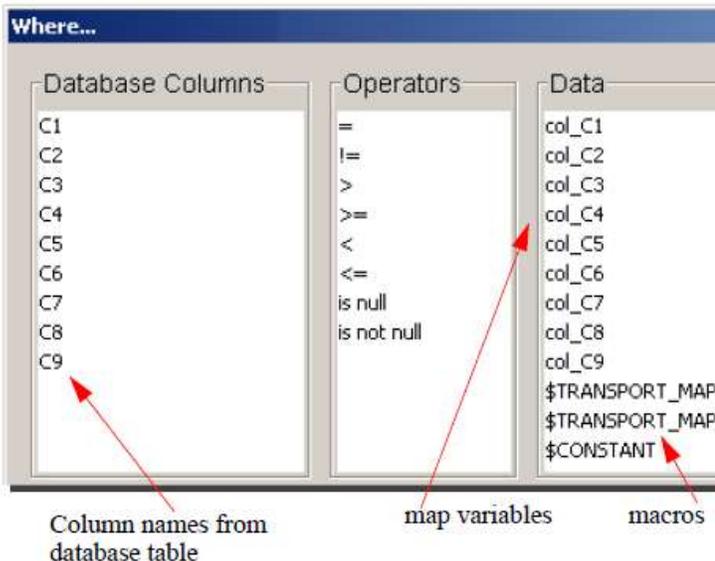
- The next step is to create the map variables. You can automatically create the map variables using the **Map Table** button. You can also create each map variable individually.



map variables from **Input** tab

For more information, see [Creating map variables automatically](#) and [Creating each map variable separately](#).

- Click the **Where** button.  
The **Where** window appears.

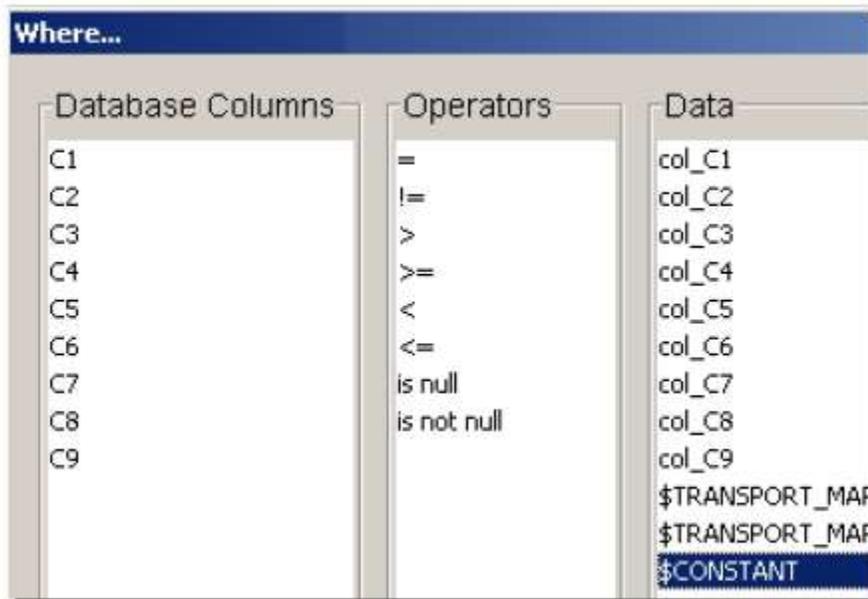


Names in the **Database Columns** in the Where window will match the first column of the table (in the **To Enterprise** section). The column contains column names from the selected database table. The **Data** column lists the map variables from the **Input** tab.

The Where window gives you the ability to construct a Where clause. You can select any column name, and then use a mathematical operator ( $=$ ,  $\neq$ ,  $>$ ,  $\geq$ ,  $<$ ,  $\leq$ ) to relate the operator to either a map variable or constant value. Each of these operators can be combined with other operators using an **AND** or **OR** statement.

Note if both an **AND** statement and an **OR** statement are used in a Where clause, the **AND** statement is evaluated first.

- To specify a value, select the database column, the mathematical operator, either a map variable or a macro, and then click **Add**.



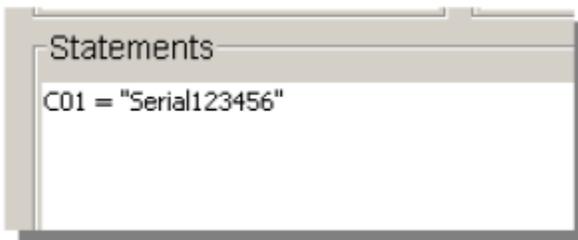
For this example, \$CONSTANT macro is selected.

The Constant window appears.

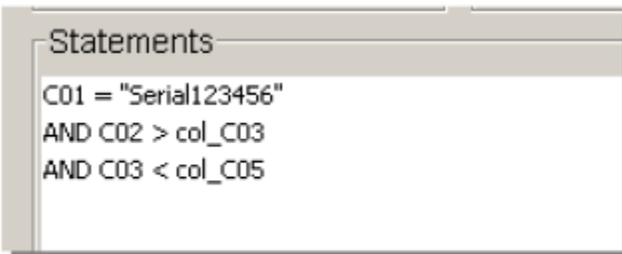


5. Type the string you want to have the column use at runtime, and then click **OK**.
6. A message will ask you to confirm the use of the Where clause for the column. Click **Continue**.

The string that you added will appear in the bottom portion of the Where window under **Statements**.



7. Continue to associate additional columns with map variables as appropriate.



8. When the Where clause is complete, click **OK**.

You are returned to the main Transport Map window.

9. Click **Validate**.

A window appears and displays the output of the transport map. If no error is received,

click **OK**.

10. Click **Save**.

## IIoTA industrial IoT Platform: Using a stored procedure

A database transport map can be defined to invoke a stored procedure with IN, OUT, and INOUT parameters and process data from multiple result sets.

When the stored procedure invocation is complete, the data from the OUT or INOUT parameters can be written to output variables.

If the stored procedure returns result sets, this data can also be written to output variables.

### Exceptions

A RDM transport does not support stored procedures. The **Actions** pick list will not show 'Stored Procedure' as a selection item for this database transport type.

Processing result set data from a stored procedure is allowed for the following database transport types:

- MSSQL
- Oracle (supported as an OUT parameter of type "REF CURSOR")
- DB2
- mySQL

## Assumptions

You are familiar with configuring and invoking a stored procedure in the database server.

## Defining the Stored Procedure transport map

1. Using the Enterprise -> Transport Map tab, select **New** to define a new transport map.
2. Select a previously defined database transport in the **Transport Name** parameter.
3. When the metadata is returned from the gateway, and the Workbench populates the **Schema** parameter, select the database schema you want to use.

4. In the **Action** parameter, select **Stored Procedure**.
5. In the **Procedure** parameter, select the stored procedure you want to invoke.  
For example:

From/To Enterprise

**Action:** Stored Procedure Map Params

**Schema:** dbo Where...

**Procedure:** dbo.getlargevarchar Order By...

Result Sets: 0  Allow in Store and Forward

Stored Procedure

Parameter	DB Type	Parm Type	Variable	Value
@RETURN_VALUE	INT	Return	col_out_RETURN_VALUE	
@in_string	VARCHAR	In	col_in_string	
@out_string	VARCHAR	In/Out	col_in_out_out_string	
@out_string_len	INT	In/Out	col_in_out_out_string_len	

6. The Workbench queries the stored procedure metadata from the database and populates the **Stored Procedure** tab with parameter information.
7. The **From/To Enterprise** title indicates the direction of the data flows both ways (bidirectional). In the case of a stored procedure, the transaction would start with data flowing to the stored procedure and would finish with data coming from the stored procedure.
8. The **Allow in Store and Forward** check box is used to specify whether to invoke the stored procedure when the transport is in the 'Store and Forward' state. If the box is unchecked a transport map with a stored procedure with INOUT or OUT parameters will not be added to the Store and Forward queue. This is because at some future point in time, when the transport is able to connect to the database server and execute the stored procedure, it will not be able to deliver data from the INOUT and OUT parameters of the stored procedure to a trigger action. Selecting the **Allow in Store and Forward** checkbox allows the transport map to be written to a Store and Forward queue with the understanding that when the procedure is eventually executed, the data sent back to the

map in the INOUT and OUT parameters will be dropped (not processed).

- The **Output** tab becomes available when a stored procedure with OUT or INOUT parameters is specified.

**Name:** TestSP\_RSET **Version Mismatch:** Pass

**Transport Type:** DB

**Transport Name:** SQLServer\_12\_58\_Login

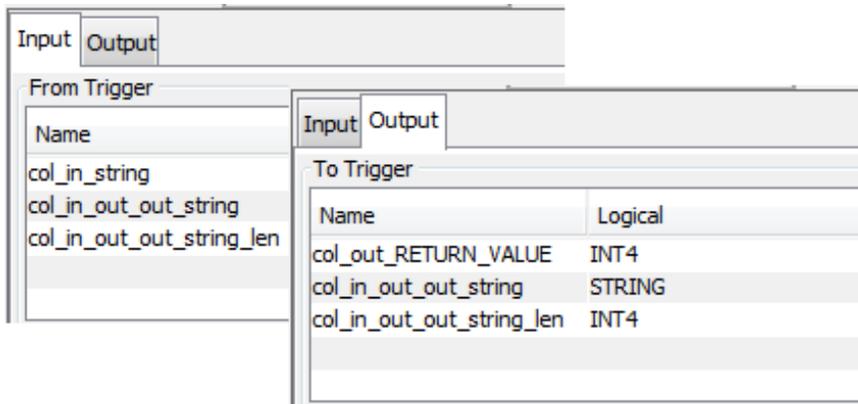
**Input** **Output**

To Trigger

Name	Logical	Count	Length
col_out_RETURN_VALUE	INT4	1	n/a
col_in_out_out_string	STRING	1	8000
col_in_out_out_string_len	INT4	1	n/a

Add Remove

- The **Input** tab will consist of a map variables whose values are mapped to IN or INOUT parameters of the stored procedure. The **Output** tab will consist of map variables that are the destination of data returned from the OUT or INOUT parameters of the stored procedure.
- To map a value to an IN or INOUT parameter, create a map variable using the **Input** tab.
- If the stored procedure is to return a value to an OUT parameter, create the map variable using the **Output** tab.
- When adding a map variable that will be used as an **INOUT** parameter by the stored procedure, you must create the variable on both the **Input** and **Output** tab. The variable names must exactly match; otherwise, the **Stored Procedure** tab will not recognize the variable as an INOUT parameter.



14. Now associate the map variables with the stored procedure parameters. If you need help with this task, go to [Associating map variables with table columns](#).

15. You can have the workbench automatically create map variables and associate them with stored procedure parameters by selecting the **Map Params** button. If the variable is mapped to an INOUT parameter, it will be added to both the Input and Output tabs.

## Configuring result sets

The **Result Sets** parameter is optional. It is used to specify the number of result sets the stored procedure is expected to return.

- If the **Result Sets** parameter is left blank, it indicates that you do not want to process result set information that may be returned from the stored procedure.
- If the **Result Sets** parameter is set to a value, the Workbench will create a set of tabs that corresponds to the number entered as shown in the example below.
  - These tabs can be used to specify column metadata information for the result set.
  - The order of the tabs represents the order in which the map is expecting the result sets to be returned by the stored procedure.

From/To Enterprise

**Action:** Stored Procedure Map Params

**Schema:** dbo Where...

**Procedure:** dbo.getManyWorkOrdersDirect Order By...

Result Sets: 2  Allow in Store and Forward

Stored Procedure | Result Set 1 | Result Set 2

Parameter	DB Type	Parm Type	Variable	Value
@RETURN_VALUE	INT	Return		
@startWOKey	BIGINT	In		

If you are working with an Oracle stored procedure that returns result sets, each result set is returned as an OUT parameter of type REF CURSOR.

This is indicated by the **Result Sets** parameter that is not editable. It will be populated with the number of OUT parameters that contain result set information.

The following example shows a transport map configured to invoke an Oracle stored procedure. Note the **Result Sets** parameter is not editable and the **DB Type** column of the procedure parameters that return a result set is "REF CURSOR".

The number of result set tabs corresponds to the number of "REF CURSOR" parameters.

From/To Enterprise

**Action:** Stored Procedure Map Params

**Schema:** M2MUSER Where...

**Procedure:** M2MUSER.DEMO\_MULTIPLE\_RESULTSETS.GET\_MULTIPLE\_WORKORDERS Order By...

Result Sets: 2  Allow in Store and Forward

Stored Procedure | Result Set 1 | Result Set 2

Parameter	DB Type	Parm Type	Variable	Value
I_WO_KEY	NUMBER	In		
O_WORKORDERS_RS1	REF CURSOR	Out		
O_ERROR_MSG1	VARCHAR2	Out		
O_WORKORDERS_RS2	REF CURSOR	Out		
O_ERROR_MSG2	VARCHAR2	Out		

## Specifying Result Set tab metadata

Navigate to a Result Set tab to specify metadata information about the result set.

- The **Max Rows** parameter is used to specify the maximum number of rows that will be processed from this result set.
- The **Name** parameter can be used to specify a prefix for output variables that are created to map content from this result set. It is populated with a default value.

From/To Enterprise

**Action:** Stored Procedure Map Params

**Schema:** dbo Where...

**Procedure:** dbo.getManyWorkOrdersDirect Order By...

Result Sets: 2  Allow in Store and Forward

Stored Procedure | **Result Set 1** | Result Set 2

**Max. Rows:** 1 **Name:** rs1

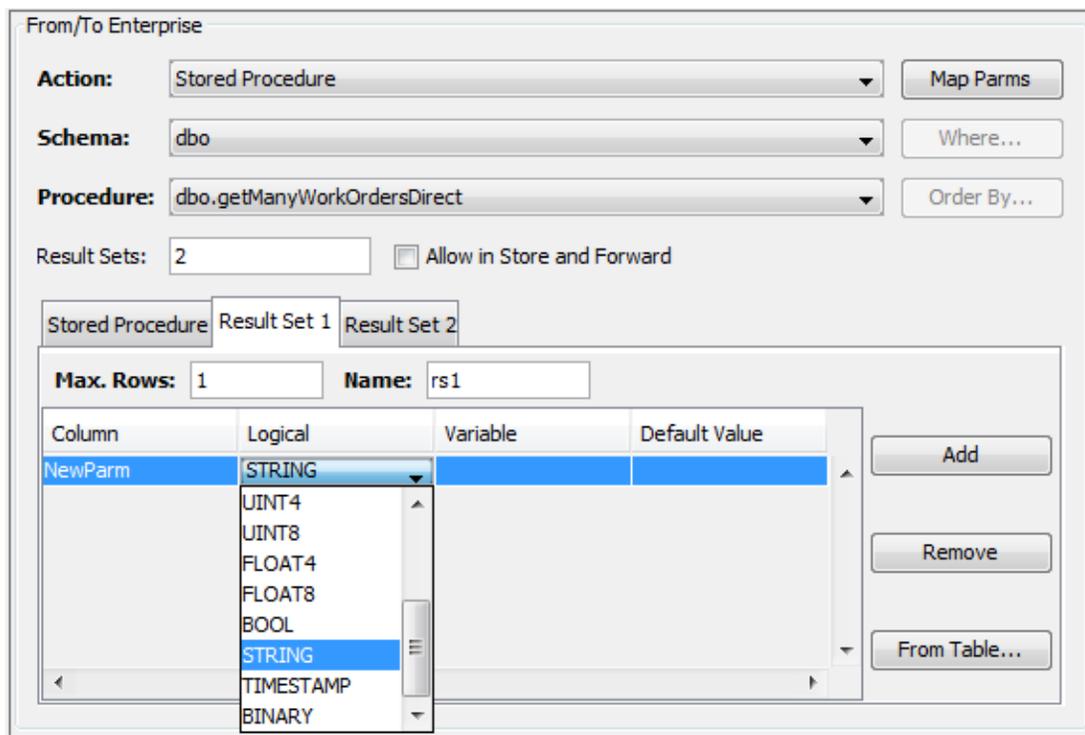
Column	Logical	Variable	Default Value

Add  
Remove  
From Table...

You can specify stored procedure metadata information by adding individual column information or by using the table lookup helper which is accessed by selecting the **From Table...** button.

## Specifying result set column data individually

Select the **Add** button and a row will be added to the Result Set tab as shown in the example below:



Each row in the Result Set tab has the following columns:

Parameter	Description
<b>Column</b>	The column name of the result set item that will be mapped.
<b>Logical</b>	The output variable data type that the column data will be converted to.
<b>Variable</b>	The output variable name that this result set column will be mapped to.
<b>Default Value</b>	A default value to be used when the column data is NULL.

You can use the **Remove** button to remove a row from the Result Set tab.

### Specifying result set column data based on a database table

Select the **From Table...** button to specify a table for the result set metadata.

A table selection panel is displayed, as shown in the example below.

Select the **Schema** and the **Table**.



From/To Enterprise

**Action:** Stored Procedure Map Params

**Schema:** dbo Where...

**Procedure:** dbo.getManyWorkOrdersDirect Order By...

Result Sets: 2  Allow in Store and Forward

Stored Procedure Result Set 1 Result Set 2

**Max. Rows:** 1 **Name:** rs1

Column	Logical	Variable	Default Value
wo_key	INT8		
site_num	INT4		
wo_name	STRING		
description	STRING		
category	STRING		
location	INT4		
create_ts	TIMESTAMP		
update_ts	TIMESTAMP		

Add Remove From Table...

This process can be used to specify result set metadata for each of the tabs.

### Mapping input and output variables

Select the **Map Params** button to have the workbench automatically create stored procedure and result set input and output variable data.

- The Input tab will contain variables for all IN and INOUT parameters.
- The Output tab will contain variables for INOUT and result set tab data. All variables that get mapped from result set data will have a result set name prefix.

For example, the output variable created for the 'site\_num' result set parameter will be called 'rs1\_site\_num'.

**Name:**  **Version Mismatch:**  ▾

**Transport Type:**  ▾

**Transport Name:**  ▾

**Input** **Output**

To Trigger

Name	Logical	Count	Length
col_out_RETURN_VALUE	INT4	1	n/a
rs1_wo_key	INT8	5	n/a
rs1_site_num	INT4	5	n/a
rs1_wo_name	STRING	5	32
rs1_description	STRING	5	32
rs1_category	STRING	5	32
rs1_location	INT4	5	n/a
rs1_create_ts	TIMESTAMP	5	n/a
rs1_update_ts	TIMESTAMP	5	n/a

---

From/To Enterprise

**Action:**  ▾

**Schema:**  ▾

**Procedure:**  ▾

Result Sets:   Allow in Store and Forward

**Stored Procedure** **Result Set 1** **Result Set 2**

**Max. Rows:**  **Name:**

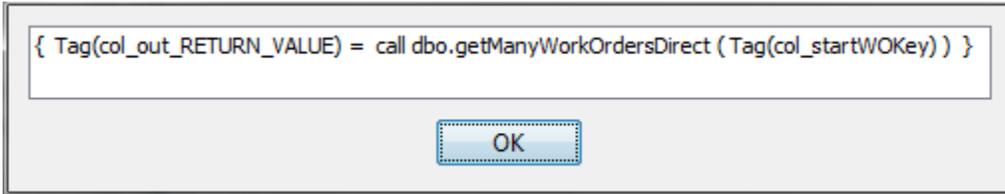
Column	Logical	Variable	Default Value
wo_key	INT8	rs1_wo_key	
site_num	INT4	rs1_site_num	
wo_name	STRING	rs1_wo_name	
description	STRING	rs1 description	

When you define a trigger that uses a Transaction action that references this Stored Procedure transport map, you associate the transport map's **Output** tab variables with the trigger's Transaction action **Output** tab.

## Validating and saving the transport map

1. Select **Validate**.

A window appears and displays a representation of the SQL statement that will be invoked when the transport map is processed.



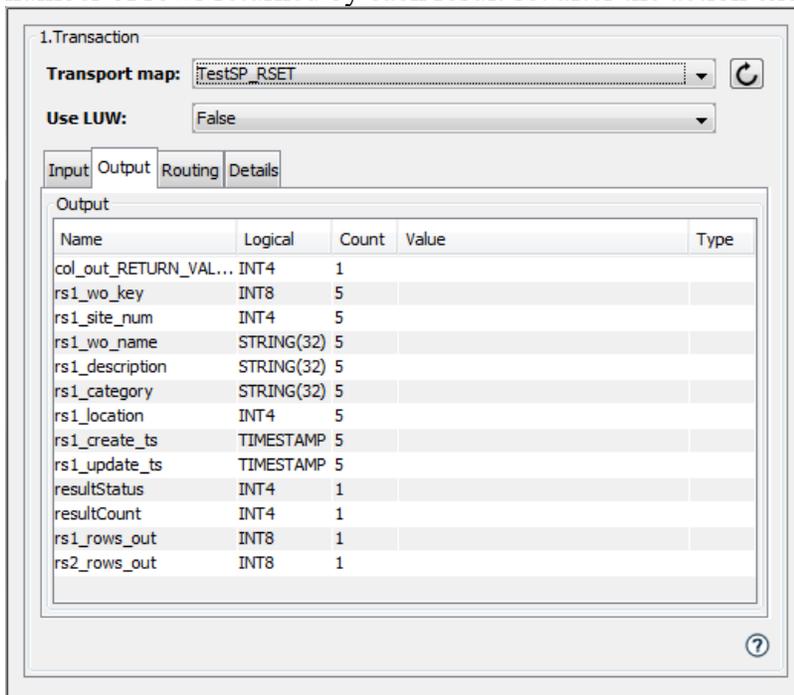
2. To close the window, select **OK**.
3. Select **Save** to save the transport map definition.

## Using the Store Procedure in a trigger's Transaction action

The Stored Procedure transport map can be referenced in a trigger's Transaction action. The Input and Output tabs of the Transaction action will display input and output variables that were configured in the Stored Procedure transport map.

If the Stored Procedure transport map is configured to process result sets you will see output variables that represent the number of rows that are returned in each result set.

In the example below the output variables 'rs1\_rows\_out' and 'rs2\_rows\_out' will contain the number of rows returned by each result set after the action executes successfully.



## IIoTA industrial IoT Platform: Using the distinct operator with Select

When creating a transport map using the **Select** operation, you can apply a distinct operator. The distinct operator provides the means to reduce the result set to unique rows in a database table.

For example, suppose you have a table with **C1** and **C2** columns that represent readings taken from various machines identified by the combination of a machine ID for column **C1**, and station ID for column **C2**. A reading is taken from the machine and written to the database in column **C3**, along with a timestamp, written to column **C4**. This is a routine process for various machines throughout the plant. Also, assume that a row is inserted into the database every 50 milliseconds causing numerous rows in the database table with the combination of a machine ID and station ID, each with unique timestamp values. A Select operation in which you are selecting columns **C1** and **C2** that does not use the distinct operator might fill output variables **log1** and **log2** with the column values that represent the same combination of a machine ID and station ID. The distinct operator is used to ensure you get only one **log1** and **log2** entry per combination of machine ID and station ID.

The following shows an example transport map with a **Select** operation and the **Distinct rows only** check box turned on.

The screenshot shows the 'From Enterprise' configuration window. The 'Action' dropdown is set to 'Select'. The 'Max. Rows' field is set to '200'. The 'Distinct rows only' checkbox is checked. Below the configuration fields is a table with columns: Column, DB Type, Nullable, Variable, and Default Value. The table contains three rows: STEP\_NO (CHAR, No, log1), HEAT\_TEMP (REAL, No, log2), and HEAT\_TIME (SMALLINT, No, log3).

Column	DB Type	Nullable	Variable	Default Value
STEP_NO	CHAR	No	log1	
HEAT_TEMP	REAL	No	log2	
HEAT_TIME	SMALLINT	No	log3	

Based on the example transport map, you can select all **C1** and **C2** columns, up to a maximum of 200, in which the machine reading in column **C3** is < 100 (as specified in the Where clause). The distinct operator ensures that you receive a result set that has only one entry per combination of machine ID and station ID. The result set can return up to 200 unique combinations of these two fields. Without the distinct operator, the results could contain only a few unique combinations of machine ID and station ID, intermixed with many duplicate entries.

## IIoTA industrial IoT Platform: Updating a set of columns (Select with Update)

The **Select** operation is used to configure a transport map that will select data from a database and then map the data to an output map variable. When the **Transaction** action is executed via a trigger, the output map variables can be written to PLC device variables.

The **Select with Update** operation is configured in much the same way as the **Select** operation but can also update a set of columns on a record after the row has been fetched from the table. **Select with Update** allows the rows to be updated in the table based on the values specified in the map variables on the **Input** tab. The input map variables are also used to specify values in a **Where** clause in the same way as they would have been used for a **Select** operation. The output map variables are used to return the fetched row data.

You will not see **Select with Update** in the **Actions** list for the following database transport types because the databases do not support this SQL operation

- RDM
- SAP HANA

When you specify a **Select with Update** operation, you can also specify how the table is locked when the transaction is executed by the trigger. There is pessimistic and optimistic locking as follows:

### Pessimistic locking

Pessimistic locking secures the database table that is the source of the data which prevents other database users from modifying data on that table while the **Select with Update** operation is being executed. When the transaction completes its execution, the lock on the table is released.

Use pessimistic locking to ensure that no rows are being added to the table while the **Select with Update** operation is executing. Pessimistic locking is not recommended when the target table has many users who want to access and update the data. In addition, pessimistic locking will prevent a phantom read phenomenon in the database concurrent access terms.

### Optimistic locking

Optimistic locking is less restrictive and allows other users to update data on the table (except for the rows that are being selected). Using optimistic locking allows rows to be added to the selected result set that match the **Where** clause criteria specified in the **Select with Update** operation while the result set (row set) is being processed. This will not prevent a phantom read phenomenon in database concurrent access terms.

The following shows an example transport map with a **Select with Update** operation that will select the data from the **WIP\_SCHEDULE** table.

Column	DB Type	Nullable	Variable
REC_TS	DATETIME	Yes	col_out_REC_TS
WIP_SEQ	INT	Yes	col_out_WIP_SEQ
WIP_ID	INT	No	col_out_WIP_ID
WIP_ENTRY	VARCHAR	Yes	col_out_WIP_ENTRY
WIP_SCHEDULE_NUM	VARCHAR	Yes	col_out_WIP_SCHEDULE
WIP_CLAIM_STATUS	CHAR	Yes	col_out_WIP_CLAIM_STA
WIP_SETTING X	INT	Yes	col out WIP SETTING X

In the example, the rows in the **WIP\_SCHEDULE** table represent scheduled entries that will be processed by a trigger. Rows that are available to be selected have a **WIP\_CLAIM\_STATUS** of **U**. After the rows have been selected, the **WIP\_CLAIM\_STATUS** column will be set to **C**.

The **WIP\_CLAIM\_STATUS** column is used to build the Where clause that will select rows from the table. After the rows have been selected, the **WIP\_CLAIM\_STATUS** column will be updated with a user defined status value (in this case **U**) indicating that rows have been fetched. This allows the example scheduling application to determine which rows have been fetched by a trigger. **Max Rows** is set to select 10 rows of data (this prevents more than 10 rows of data from being fetched). Pessimistic locking is turned off.

The **From/To Enterprise** section has a **Select** tab and **Update** tab. When the table is specified, both the **Select** and **Update** tabs become populated with the table information.

The first step is to create the map variables.

1. To automatically create the map variables, click the **Map Table** button.

Using **Map Table** with **Select with Update** automatically populates both the **Output** and **Input** tabs with map variables and also associates those map variables to the appropriate columns on the

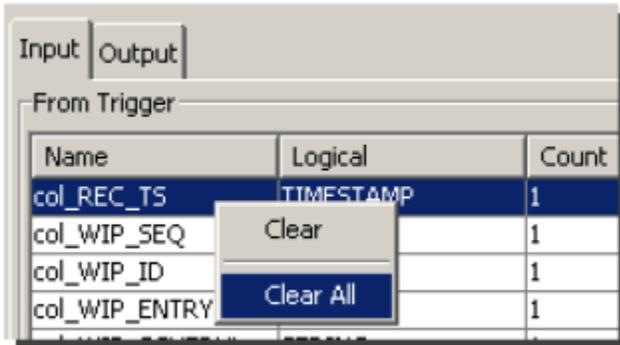
For this example, the map variables for the **Input** tab will be created separately. Therefore, the variables

**Select and Update tabs.**

When you add a value for **Max Rows**, that value is also automatically added to the **Count** column on the **Output** tab.

created from **Map Table** will be removed.

- From the **Input** tab, display its pop-up menu, and then click **Clear All**.



The **Input** tab becomes empty.

The next step is to create map variables for the **Input** tab.

- From the **Input** tab, click **Add**.

The New Item window appears.

- Type the name for the input map variable (for this example **col\_input\_WIP\_STATUS**), select string as the data type, and set the length of the string to 1. Accept the default value of 1 in the **Count** box.



5. Click **Add**. The input map variable is added to the **Input** tab.

6. Repeat the step for **col\_input\_set\_WIP\_STATUS**.

The completed **Input** tab will appear similar to the following:

Name	Logical	Count	Length
col_input_wip_status	STRING	1	1
col_input_set_wip_status	STRING	1	1

The **col\_input\_wip\_status** will be used in the **Where** clause of the Select that is executed on the WIP\_SCHEDULE table.

The **col\_input\_set\_wip\_status** will be used to set the **WIP\_CLAIM\_STATUS** column on the rows as they are being selected from the database.

The next step is to associate the **col\_input\_set\_wip\_status** map variable with the **WIP\_CLAIM\_STATUS** column on the **Update** tab.

7. From the **Update** tab, select the **WIP\_CLAIM\_STATUS** column, and then click the **Variable** column.

A list of variables appears.

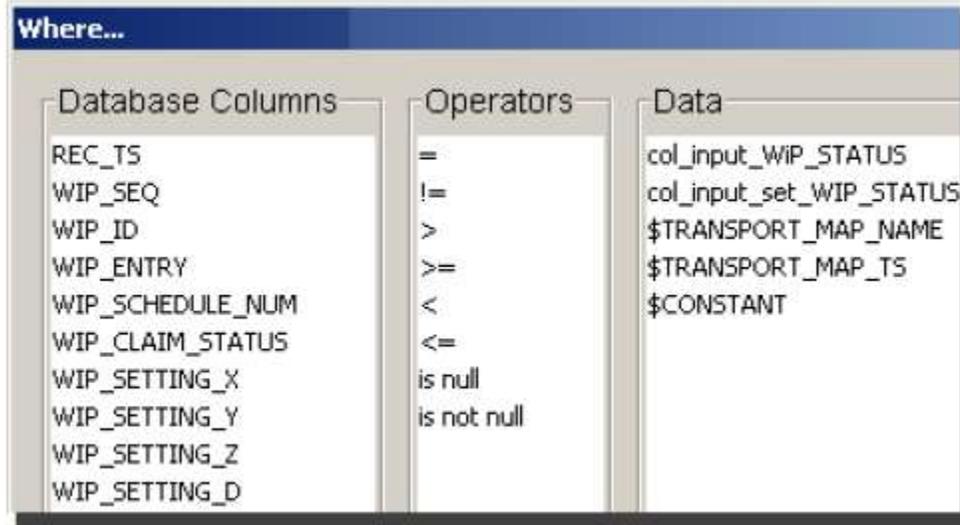
Column	DB Type	Variable
REC_TS	DATETIME	
WIP_SEQ	INT	
WIP_ID	INT	
WIP_ENTRY	VARCHAR	
WIP_SCHEDULE_NUM	VARCHAR	
WIP_CLAIM_STATUS	CHAR	
WIP_SETTING_X	INT	col_input_wip_status
WIP_SETTING_Y	INT	col_input_set_wip_status
WIP_SETTING_Z	INT	\$TRANSPORT_MAP_NAME
		\$TRANSPORT_MAP_TS
		\$CONSTANT
		\$NULL

- Select the **col\_input\_set\_WIP\_STATUS** map variable.

The next step is to add a Where clause.

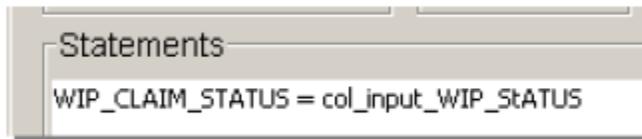
- Click the **Where** button.

The Where window appears.



- Select one or more database columns that you want updated (for this example **WIP\_CLAIM\_STATUS**), the operator (for this example =), and the variable whose value you want to use in the **Where** (for this example **col\_input\_WIP\_STATUS**).

The statement will be similar to the following:



- When the Where clause is complete, click **OK**.

You are returned to the main Transport Map window.

The final step is to create a trigger and assign the transport map as the action. You will associate the map variables in the **Transaction** action definition with the appropriate device variable or other variable.

When the trigger executes, the following will occur.

- Up to 10 rows will be selected from the table using the value specified in `col_input_WIP_STATUS` to build a Where clause.
- After the rows have been fetched, the `WIP_CLAIM_STATUS` column will be updated with the value specified in `col_input_set_WIP_STATUS` input variable.

The result is that the rows that were selected will have their `WIP_CLAIM_STATUS` column updated with the value specified in the `col_input_set_WIP_STATUS` map variable.

## IIoTA industrial IoT Platform: Determining the number of rows on a table (Count Rows)

A transport map can be configured to return the number of rows on a database table.

Using the **Count Rows** operation will return a value which represents the number of rows in the selected database table that meet the select condition.

The select condition is specified in a Where clause. If you do not specify a Where clause, you will get a count of the total number of rows in the database table.

When you specify a **Count Rows** operation, the **Count Rows** tab provides a read-only view of the table.

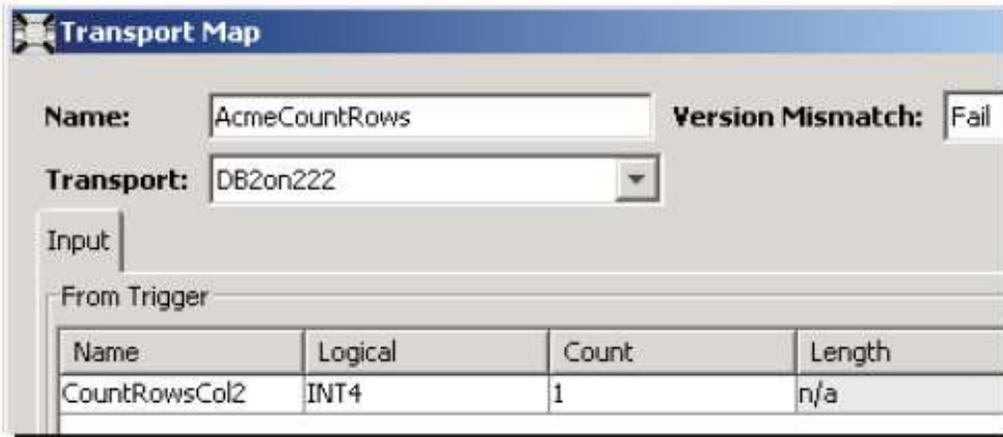
The screenshot shows a configuration window titled 'To Enterprise'. It has three main sections: Schema, Table, and Action. The Schema dropdown is set to 'ACUSER', the Table dropdown is set to 'TABLE1', and the Action dropdown is set to 'Count Rows'. To the right of these dropdowns are buttons for 'Map Table' and 'Where...'. Below these fields is a tab labeled 'Count Rows' which contains a table with the following data:

Column	DB Type	Required	Variable	Value
C1	CHAR	No		
C2	SMALLINT	No		

You cannot assign any map variables to any of the rows in the table.

A transport map that uses a **Count Rows** operation will not have any output variables.

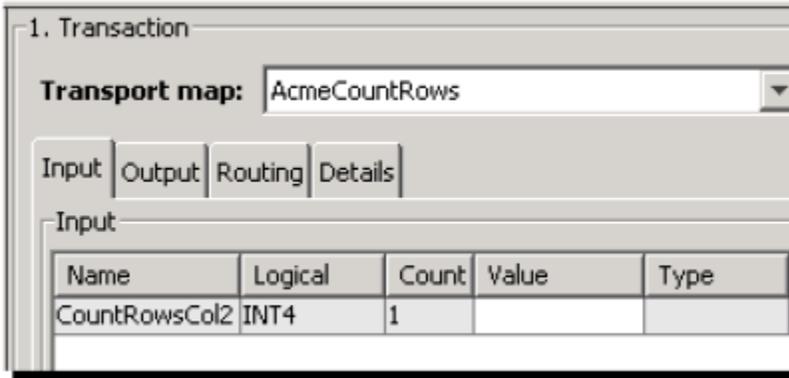
Only the **Input** tab becomes available on the Transport Map window. When you add one or more input variables to the **Input** tab, they are used solely to build a Where clause.



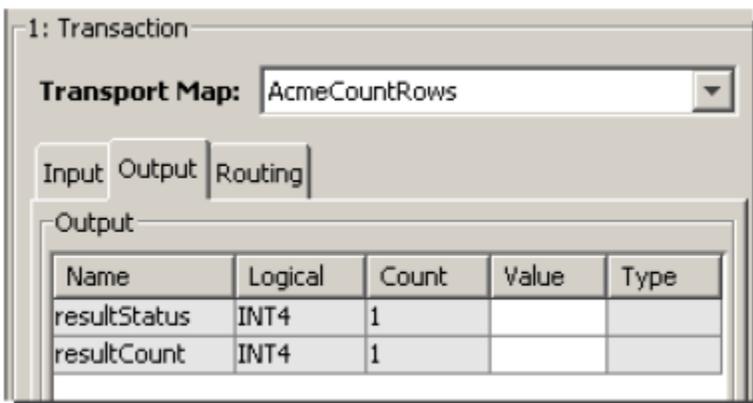
You can build a Where clause without input variables. For example, you can set a column to equal a constant.

### Count Rows and the Trigger window

When you assign the transport map as an action in a trigger, you will assign a value for any input map variable that were defined in the transport map for use in the Where clause.



The **Output** tab will contain the **resultStatus** and **resultCount** output variables.



The count of rows in the database that meet the select condition will be written to the **resultCount** variable.

The **resultStatus** variable will contain a code indicating whether or not the transaction completed successfully.

A zero indicates success.

A non-zero value translates to the error code returned either by the node or by the database vendor.

## IIoTA industrial IoT Platform: Deleting a set of rows after select (Select with Delete)

You can use a **Select with Delete** operation to delete a row after it has been fetched from the table.

The rows to be deleted in the table are based on the where clause associated with the transport map.

The input map variables are used to specify values in a **Where** clause.

The output map variables are used to return values from the column.

Using the Transport Map window, you can also specify an **Order by** clause. Order By determines the order in which rows will be returned by the database (either ascending or descending).

The following shows an example transport map with a Select with Delete operation that will select the data from the WIP\_SCHEDULE table.

From/To Enterprise

**Schema:**

**Table:**

**Action:**

**Max. Rows:**

Select

Column	DB Type	Nullable	Variable
REC_TS	DATETIME	Yes	COL_OUT_REC_TS
WIP_SEQ	INT	Yes	WIP_SEQUENCE_DELETE
WIP_ID	INT	No	col_out_WIP_ID
WIP_ENTRY	VARCHAR	Yes	col_out_WIP_ENTRY
WIP_SCHEDULE_...	VARCHAR	Yes	col_out_WIP_SCHEDULE_.

You will not see Select with Delete in the **Actions** list for the following database transport types because the specific databases do not support this SQL operation

- RDM
- SAP HANA.

## IIoTA industrial IoT Platform: Deleting rows in a table (Delete)

The **Delete** operation lets you set a transaction to delete a specific row in a table or all the rows in a table.

You must use the Where clause to select the rows to be deleted.

If you do not use the Where clause, all the rows in a table are deleted.

The **Delete** operation does not delete the table.

When you specify a **Delete** operation, the **Delete** tab provides a read-only view of the table.

To Enterprise

Schema: ACUSER

Table: A\_TABLE2

Action: Delete

Column	DB Type	Required	Variable	Value
C1	SMALLINT	No		
C2	INTEGER	No		
C3	BIGINT	No		

You cannot assign any map variables to any of the rows in the table.

A transport map that uses a **Delete** operation will not have any output variables.

Only the **Input** tab becomes available on the Transport Map window.

When you add one or more input variables to the **Input** tab, they are used solely to build a Where clause.

You can build a Where clause without input variables. For example, you can set a column to equal a constant.

## IIoTA industrial IoT Platform: Using SQL aggregate functions

A transport map can be configured to use an SQL aggregate function.

SQL aggregate functions return a single value, calculated from values in a column.

The following SQL aggregate functions are supported per database column:

- AVERAGE
- COUNT
- MINIMUM
- MAXIMUM

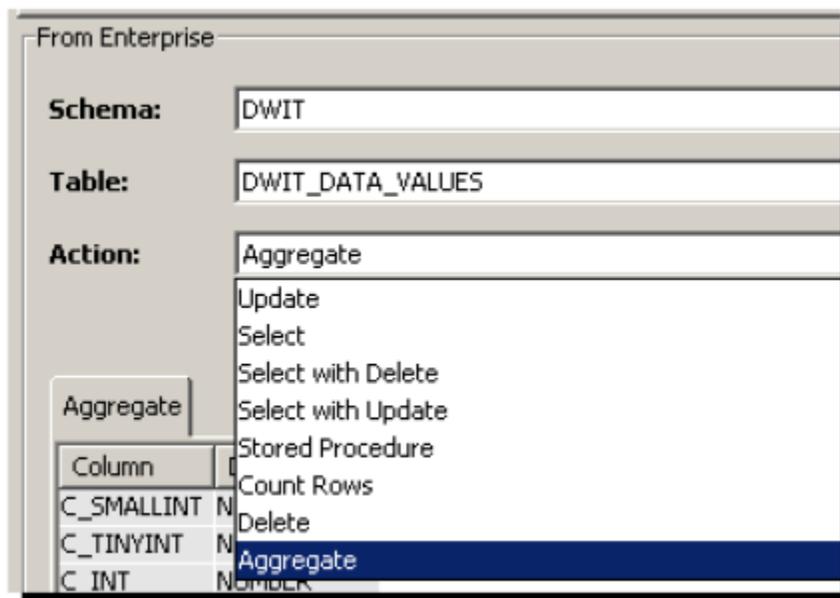
- STANDARD DEVIATION
- SUMMARY
- VARIANCE

**Before you begin, consider the following:**

- Only one SQL aggregate function can be applied to each table column. If more than one aggregate function is required for a single column an additional transport map will need to be created per SQL aggregate function for that one column. However, a transport map can be created with different columns specifying SQL aggregate functions.
- When using an SQL aggregate function, you can specify a Where clause to limit the range of column values where the SQL aggregate function is to be applied.
- SQL aggregate functions are not available with other database actions such as Select operations and no other database actions can be performed when selecting the Aggregate action.
- Transport maps using SQL aggregate functions are not placed in the store and forward queue if the transport enters the Store and Forward state.
- It is assumed that the underlying database product has implemented each SQL aggregate function.

Follow these steps to add an SQL aggregate function per database column:

1. From the **Enterprise** section of the transport map, select the schema and table to use.



2. From the **Action** drop-down list, select **Aggregate**.  
The **Aggregate** tab becomes available.

From Enterprise

**Schema:** DWIT

**Table:** DWIT\_DATA\_VALUES

**Action:** Aggregate

Aggregate

Column	DB Type	Function
C_SMALLINT	NUMBER	
C_TINYINT	NUMBER	

Rows under **Column** and **DB Type** become populated with data from the selected database table.

- From the **Aggregate** tab, select the first row under the **Function** column.

Aggregate

Column	DB Type	Function
C_SMALLINT	NUMBER	
C_TINYINT	NUMBER	
C_INT	NUMBER	AVG
C_BIGINT	NUMBER	COUNT
C_FLOAT	FLOAT	MIN
C_DOUBLE	FLOAT	MAX
C_DECIMAL	NUMBER	STDDEV
C_DATE	DATE	SUM
C_VARCHAR	VARCHAR	VARIANCE

The following SQL aggregate function are available:

Function	Description
AVG	Adds the values of all the rows in a table column and divides that value by the number of rows that match the criteria specified in a Where clause. If a Where clause is not used, the values of all the rows for that column are added together and then divided by the total number of rows in the table. The Local database does not support the <b>AVG</b> aggregate function.
COUNT	Counts the number of rows in a table column that match the criteria specified in a Where clause. If a Where clause is not used, then all the rows in the column are counted.
MIN	Selects the minimum value from the rows a table column that match the criteria specified in a Where clause. If a Where clause is not used, then the minimum value is selected from all the rows in the column.
MAX	Selects the maximum value from the rows in a table column that match the criteria specified in a Where clause. If a Where clause is not used, then the maximum value is selected from all the rows in the column.
STDDEV	Calculates the standard deviation for all the rows in a table column that match the criteria specified in a Where clause. If a Where clause is not used, then the standard deviation is calculated for all the values in the column. The Local database does not support the <b>STDDEV</b> aggregate function.
SUM	Adds the value of all the rows in a table column that match the criteria specified in a Where clause. If a Where clause is not used, then the values of all the rows for that column are added together.
VARIANCE	Calculates the variance for all the rows in a table column that match the criteria specified in a Where clause.If a Where clause is not used, then the variance is calculated for all the values in that column. The Local database does not support the <b>VARIANCE</b> aggregate function.

4. Select the aggregate function that should be applied to the column. To specify aggregate functions for other columns, navigate to the column and repeat the aggregate function selection step.

After you have mapped aggregate functions to columns your screen may look as shown below. Specify an upper limit on the number of rows to be processed by entering a numeric value in the **Max. Rows** field. The default value is 1.

From Enterprise

**Schema:** M2MUSER Map Table

**Table:** DWIT\_DATA\_VALUES Where...

**Action:** Aggregate Order By...

**Max. Rows:** 1 Group By...

Aggregate

Column	DB Type	Function	Variable
C_SMALLINT	NUMBER	AVG	col_out_C_SMALLINT
C_TINYINT	NUMBER	COUNT	col_out_C_TINYINT
C_INT	NUMBER	MIN	col_out_C_INT
C_BIGINT	NUMBER		col_out_C_BIGINT
C_FLOAT	FLOAT		col_out_C_FLOAT
C_DOUBLE	FLOAT		col_out_C_DOUBLE
C_DECIMAL	NUMBER		col out C DECIMAL

You can select the **Map Table** button to automatically create output map variables. These generated map variables will appear in the **Output** tab and on the **Variable** column on the **Aggregate** tab as shown below.

**Name:** SEL\_AGG\_IT\_DATA **Version Mismatch:** Pass

**Transport Type:** DB

**Transport Name:** ORA11G\_WITH\_TZ

Input Output

To Trigger

Name	Logical	Count	Length
col_out_C_SMALLINT	FLOAT8	1	n/a
col_out_C_TINYINT	FLOAT8	1	n/a
col_out_C_INT	FLOAT8	1	n/a
col_out_C_BIGINT	FLOAT8	1	n/a
col_out_C_FLOAT	FLOAT4	1	n/a

Add Remove

---

From Enterprise

**Schema:** M2MUSER **Map Table**

**Table:** DWIT\_DATA\_VALUES **Where...**

**Action:** Aggregate **Order By...**

**Max. Rows:** 1 **Group By...**

Aggregate

Column	DB Type	Function	Variable
C_SMALLINT	NUMBER	AVG	col_out_C_SMALLINT
C_TINYINT	NUMBER	COUNT	col_out_C_TINYINT
C_INT	NUMBER	MIN	col_out_C_INT
C_BIGINT	NUMBER		col_out_C_BIGINT
C_FLOAT	FLOAT		col_out_C_FLOAT
C_DOUBLE	FLOAT		col_out_C_DOUBLE
C_DECIMAL	NUMBER		col out C DECIMAL

The **Output** tab will consist of a collection of map variables whose values will be populated with the results of the execution of the select.

**Note:** You can change the logical variable names generated using the **Map Table** button. If you do so, you must manually edit the variable name and remap it to the column in the Aggregate tab.

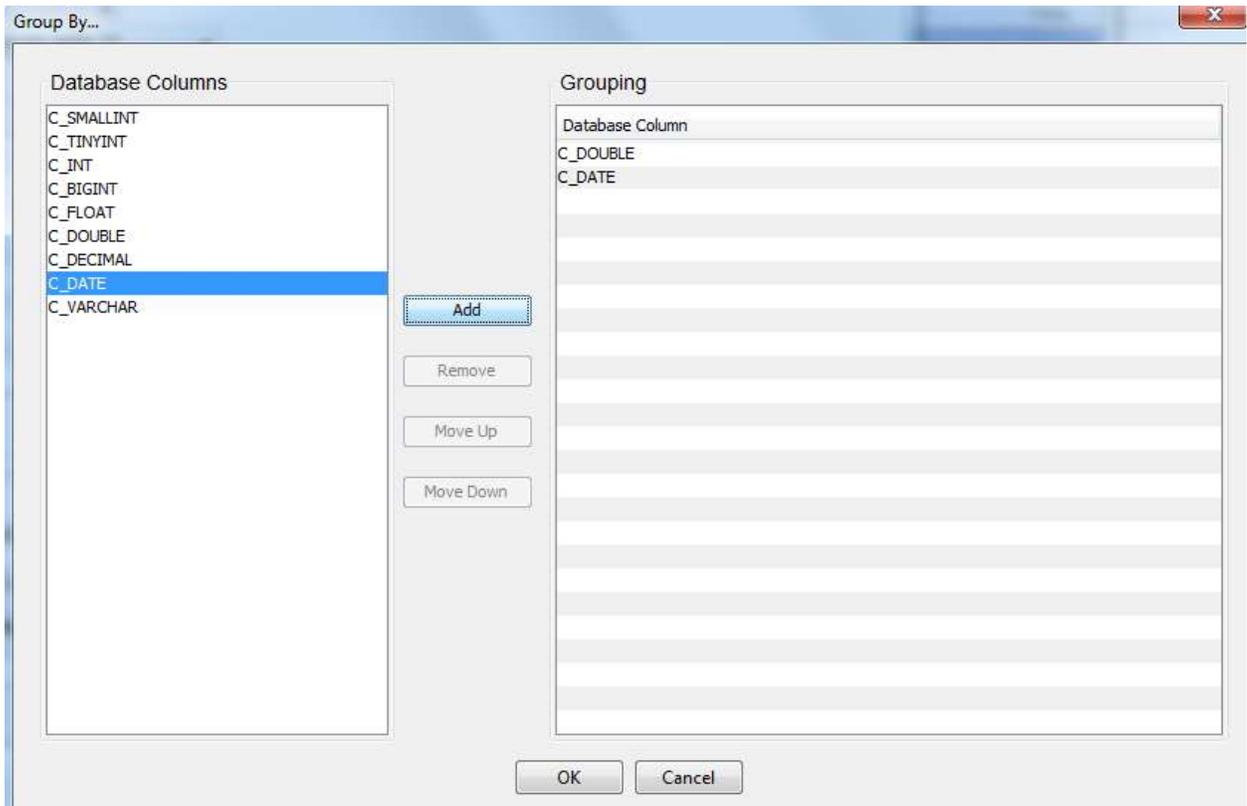
If you don't want to include columns in your Select Aggregate statement you can select the variable in the Output tab, right-click and choose **Clear**. This removes the output variable from the Output tab and clears the variable name from the column.

5. You can limit the number of rows being considered for aggregate operations by specifying a Where clause. When you select the **Where** button a popup window allows

you to select columns for the Where clause. This is like when you specify the Where clause for a Select Transport map. See Selecting columns (Select).

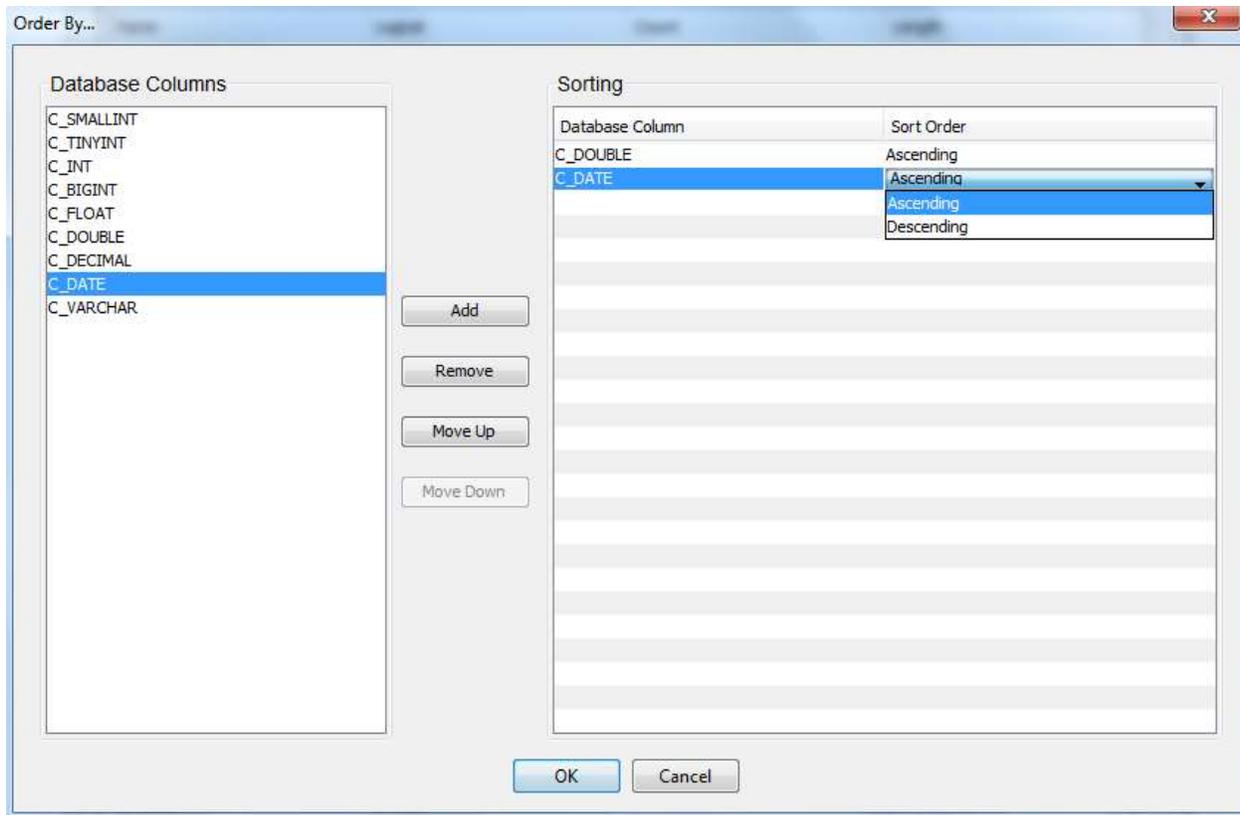
- You can group the aggregate data by columns that were not mapped to aggregate functions. To specify these columns, select the **Group By** button, and it will bring up a popup window as shown below.

Select the column and then select the **Add** button to add the column to the **Group By** by list. The selected columns appear in the **Grouping** table. Select **OK** when you are done.



- You can order the selected data by any of the columns. To specify these columns, select the **Order By** button, and it will bring up a popup window as shown below.

Select the column and then select the **Add** button to add it to the **Order By** list. You can further select the sort order of the column by selecting **Ascending** or **Descending** as shown in the figure. Select **OK** when you are done.



8. Select the **Save** button to save your transport map definition to the node.

## IIoTA industrial IoT Platform: Referencing a local database transport

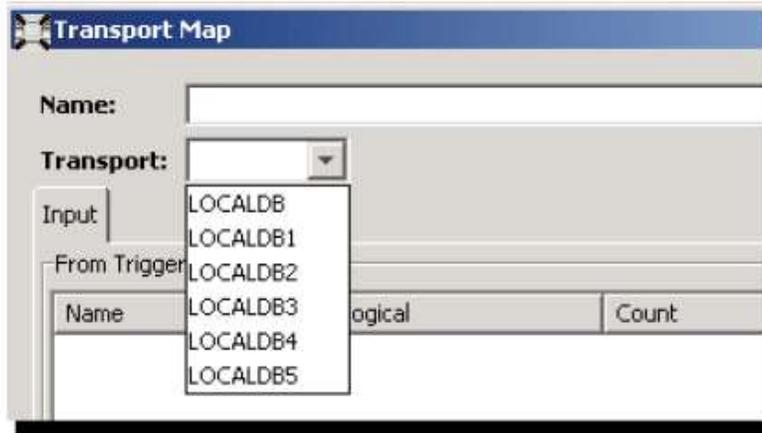
When creating a transport map, you can define a map that uses tables in the local database. The tables accessed in these maps must be predefined using the **Local Database** window. For more information, see Using a local database.

**Note:** The Transaction Server's support of Local Database tables is restricted to disk-based tables. Local Database tables created in memory are not supported by transport maps and the Transaction action.

To specify a local database, follow these steps:

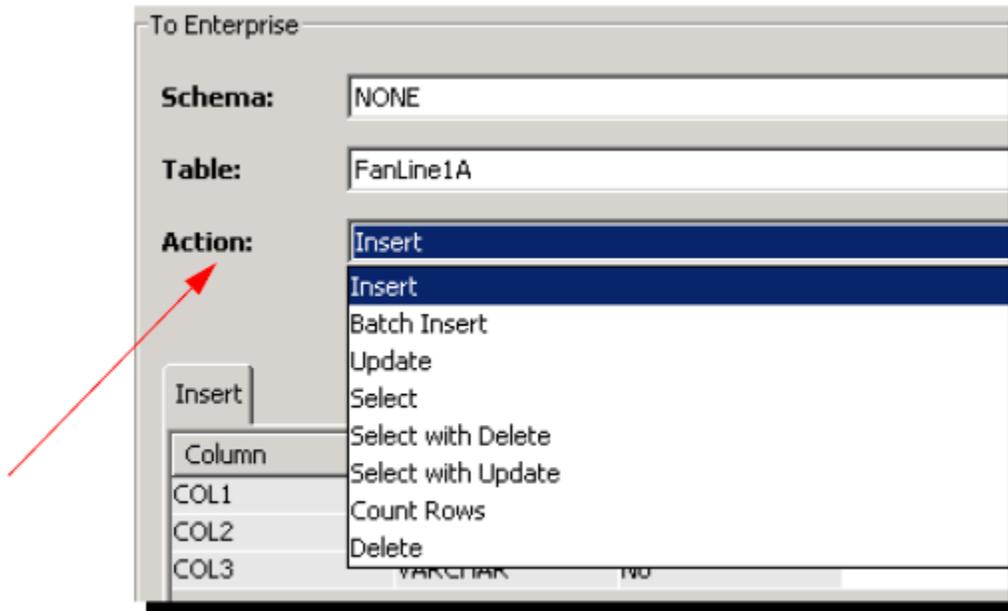
1. From the Transport Map window, click the **Transport** down arrow.

A list of default local database transports appears.



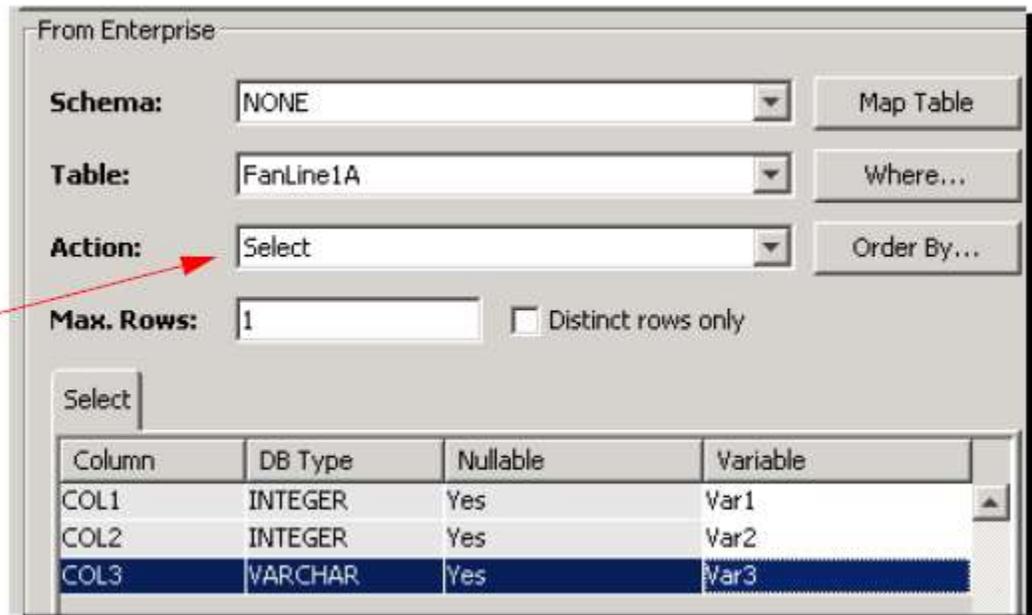
2. Select the database transport you want to use.

The bottom of the Transport Map window changes to accommodate the local database transport.



The table actions available for a **Local DB** table are the same as the actions for any of the supported commercial database products such as DB2, Oracle, and SQL Server.

Notice that there is no schema (database user) when adding a local database.



For the example transport map, a **Select** operation was specified. You can also use a Where clause and Order By statements.

## IIoTA industrial IoT Platform: Your first transport map with a database transport

Before you can create a transport map, you must have defined a transport.

It is also assumed that the Workbench is started, and you have logged on.

The following describes the steps to create a transport map that uses a database transport.

1. From the Workbench left pane, expand the node that you want to add the transport map to.
2. On the **Transport Maps** icon, right-click to display its pop-up menu, and then click **New**.



A Transport Map window appears.

Name	Logical	Count	Length

3. In the **Name** box, type a unique name for the transport map. A transport map name can be up to 64 characters and include letters, numbers, and the underscore character. Spaces are allowed.
4. From the **Version Mismatch** drop-down list, accept the default **Fail**. When the trigger event occurs, the transport map used by the trigger must match the current transport map; otherwise, the trigger event will fail.
5. From the **Transport** drop-down list, select the transport you want to use for the transport map. For this example, a database transport is selected. The **To Enterprise** section changes to accommodate the database transport. This is the data that will be stored on the enterprise system.

The next step is to configure the runtime payload. You will add a database table, define the map variables, and then associate the map variables with columns in the database table.

## IIoTA industrial IoT Platform: Adding a database table

When you selected the database transport, the **To Enterprise** section of the Transport Map window changed to accommodate a database table.

The first step is to select the table you want to route data to.

1. Go to the **To Enterprise** section. You will see parameters to select the schema, action, and database table.



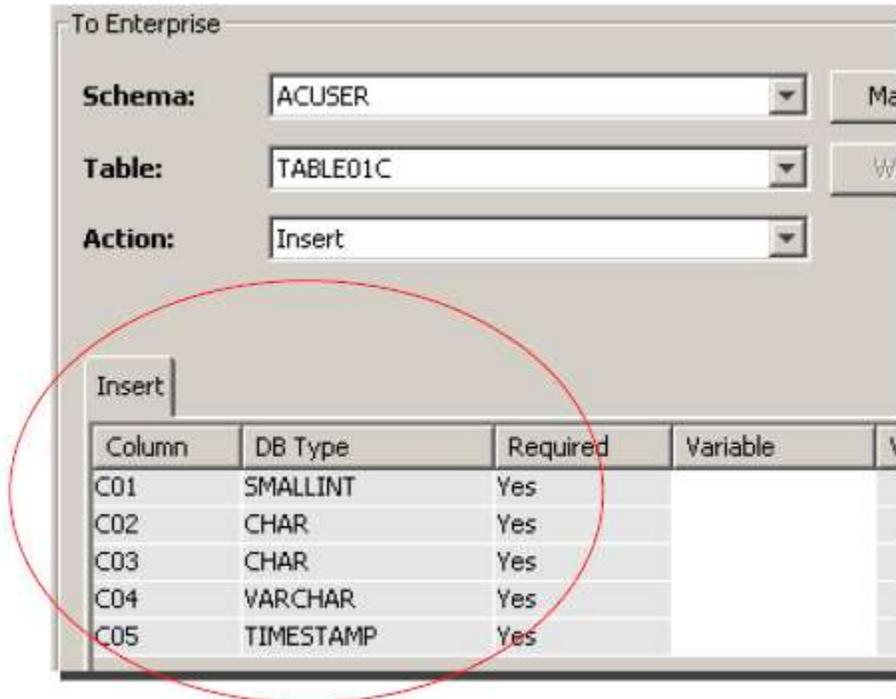
2. Click the **Schema** down-arrow. For this example, ACUSER is selected.

The schema will contain tables whose columns and rows you want to store values from map variables you set in the **Input** tab.

3. Click the **Action** down-arrow, and then select **Insert**.

**Insert** will add a new row of data into a database table whenever a trigger event occurs.

4. Click the **Table** down-arrow, and then select the appropriate table. For this example, **Table01C**.



The window becomes populated with the column information for that table. You can now create the map variables.

## IIoTA industrial IoT Platform: Defining map variables

Now that you have added the table, you can create the input map.

The input map will consist of a collection of map variables.

The plant floor engineer will see these variables when defining the trigger.

During runtime, the variables are mapped to physical PLC device variables (when a plant floor event occurs).

Since there are 5 required columns in the example table, you will want to map 5 data items (from the production line) into the table.

These data items will be defined by using 5 logical names (not the real PLC device variable or alias names).

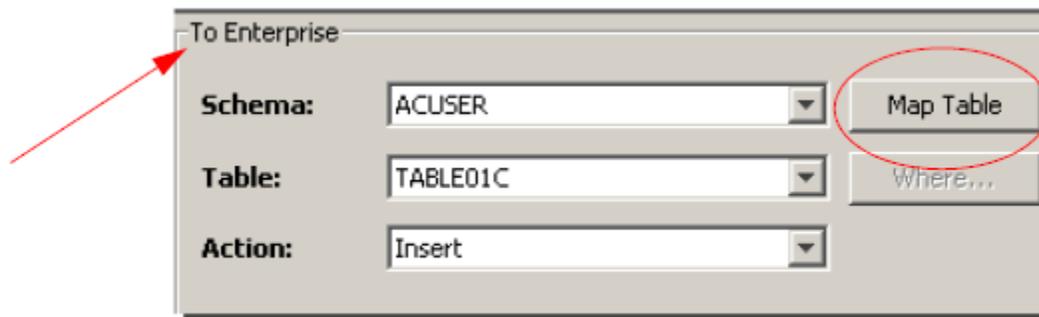
The resolution of these map variables to production line data will occur when you create the trigger.

There are two methods for defining the map variables:

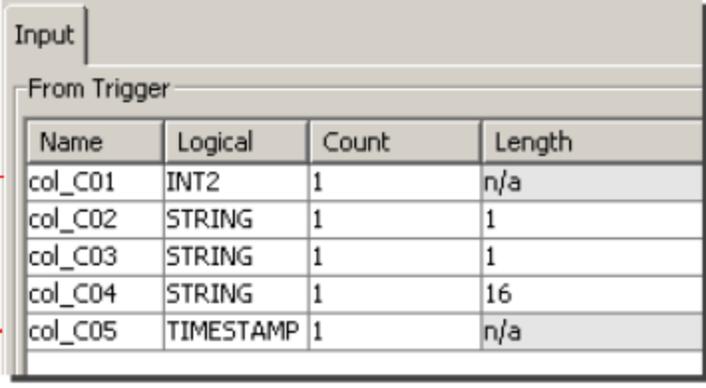
- Create each map variable separately.
- Create the map variables automatically at one time.

Since this example uses a database transport, the **Map Table** button can automatically create the map variables.

From the **To Enterprise** section, click **Map Table**.



The map variables are automatically added to the **Input** tab.



map variables

Name	Logical	Count	Length
col_C01	INT2	1	n/a
col_C02	STRING	1	1
col_C03	STRING	1	1
col_C04	STRING	1	16
col_C05	TIMESTAMP	1	n/a

**Output map:** Because the transport map is based on a database **Insert** operation, an output map is not applicable. If a transport map is bi-directional then the trigger will expect data that is queried from an enterprise system to be sent to it in the output map.

## IIoTA industrial IoT Platform: Associating map variables with table columns

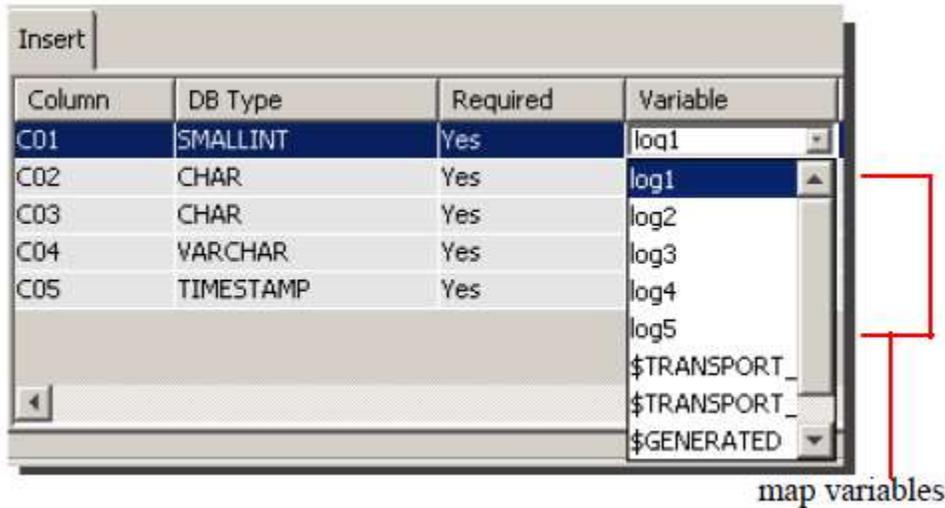
Now that you have created the map variables, the last step is to associate the map variables with the database table columns.

This is the payload that will be stored on the enterprise system.

For this example, the payload will facilitate an SQL Insert operation.

1. Go to the table at the bottom of the **To Enterprise** section.
2. Select the first database column. For this example, CO1.
3. Click the **Variable** column.

A list appears with the map variables that you just created.



4. Select the map variable that you want associated with the database column. For this example, log1.
5. Repeat steps 2 through 4 for each column in the table.

The completed **To Enterprise** section will be similar to the following.

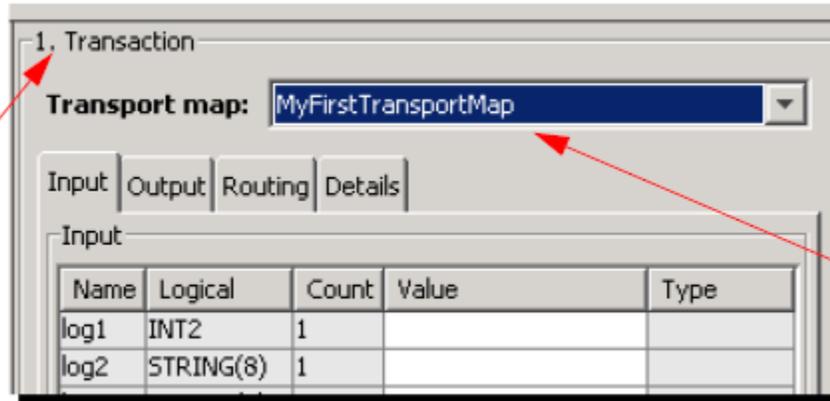
Column	DB Type	Required	Variable
C01	SMALLINT	Yes	log1
C02	CHAR	Yes	log2
C03	CHAR	Yes	log3
C04	VARCHAR	Yes	log4
C05	TIMESTAMP	Yes	log5

6. When you complete the transport map, click **Validate**. If no errors are received, click **Save**.

The new transport map is saved to the current node, and the name is added to the Transport Maps window.

The transport map also becomes available from the **Transport Map** drop-down list when

creating a trigger with a **Transaction** action.



The final step is to create the trigger and associate the map variables in the trigger **Transaction** action definition. For more information, see [Using a local database](#).

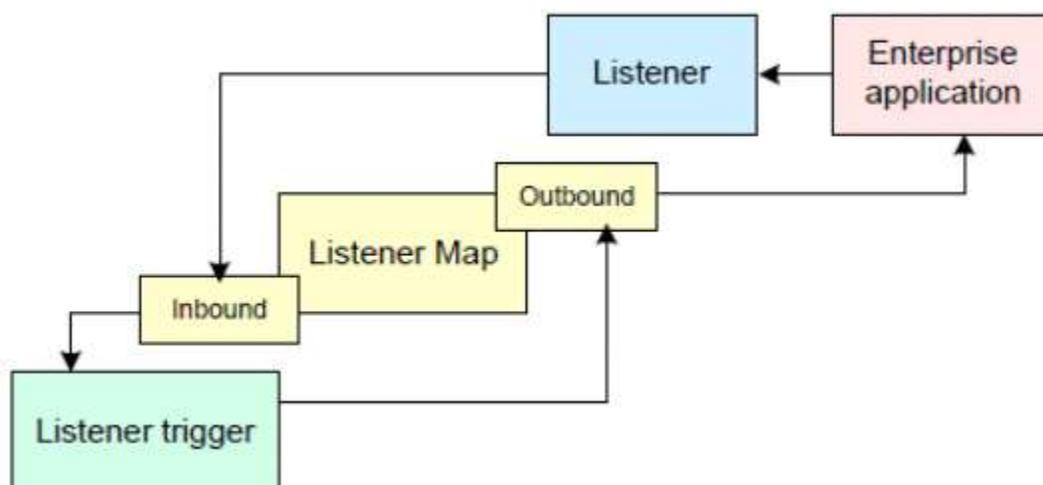
## IIoTA industrial IoT Platform: Listeners

A Listener defines the connectivity details and features, such as mapping logs, when a message (or request) is received from an enterprise application. The received message is processed by a trigger.

Listeners are defined to represent message and queuing applications, and TCP applications.

### Overview

A listener component is comprised of a listener, listener map, and a listener trigger.

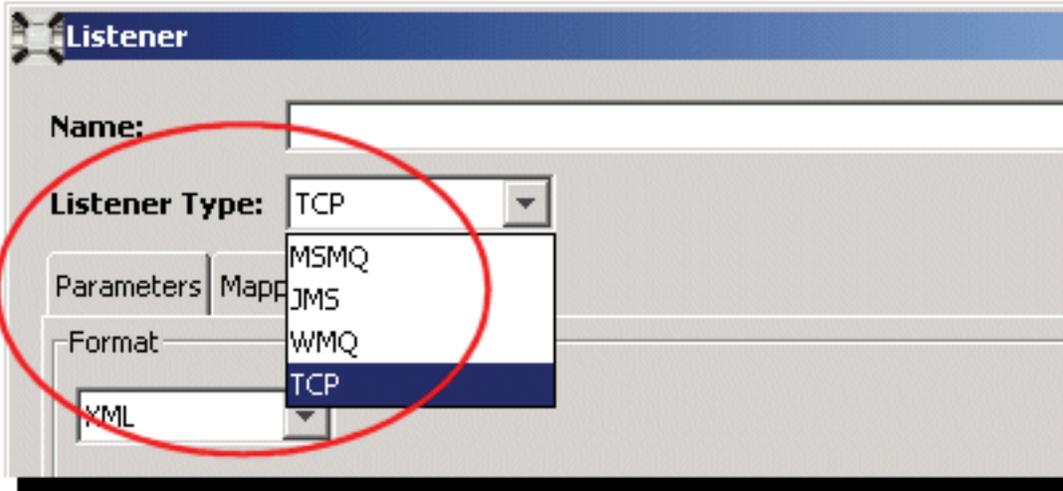


A listener can be configured to accept command requests from a remote enterprise application. The content of the request is mapped to an internal request using the listener map. When a value

request is received, the request will cause the listener trigger to execute a sequence of actions that were configured in the trigger definition.

## Listener types

Command requests can be sent to the node using the following interface points as shown:



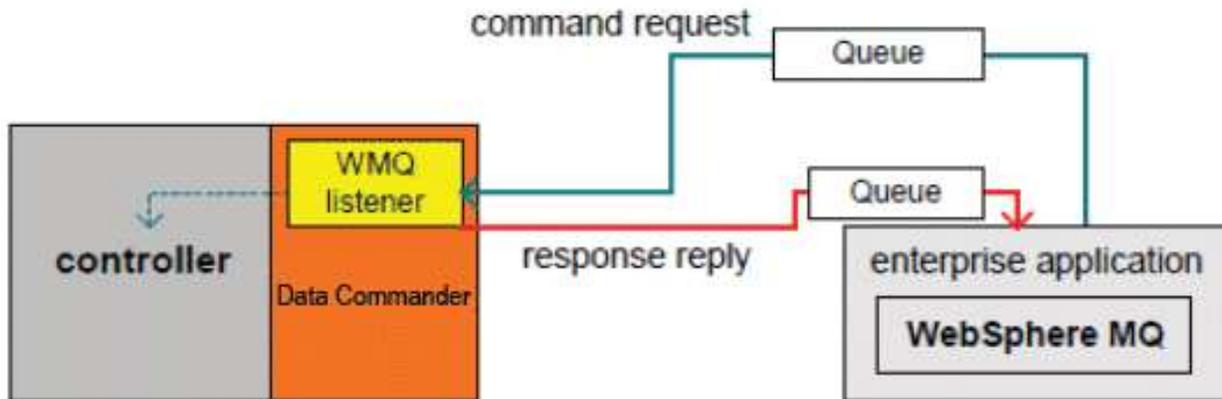
- A WebSphere MQ queue (for a **WMQ** listener types).
- A WebSphere SIB queue (for a **JMS** listener types). The listener is either a JMS WebSphere SIB listener or a JMS WebSphere SIB direct listener).
- A Microsoft Message Queuing queue (for **MSMQ** listener types).
- A TCP application (for **TCP** listener types).

A listener that is configured to interact with WebSphere MQ is referred to as a WMQ listener. You configure WMQ listeners to read requests messages on a remote WebSphere queue. The following sections will describe a WebSphere MQ listener. The concepts can also be applied to a WebSphere SIB integration point.

To summarize, the listener map works in combination with the listener and a special listener trigger. The listener trigger references the listener map. This allows an external enterprise application to execute a sequence of actions defined in the listener trigger that could effect changes on the node or on controller device variables.

A Listener can be optionally configured to send a response to every request it receives. The default message content format is XML.

The following illustrates how a WMQ listener works in conjunction with an enterprise application and the controller.



The WMQ listener supports an enterprise application-initiated request to the node such as start or stop a project, start or stop a trigger, write to or read from a controller variable using the action list configured in the listener trigger definition.

The enterprise application submits the request via a queue that the WMQ listener is monitoring. Upon completion of the enterprise application request, the listener sends the results in a response message that is put on a remote reply queue. This is a bidirectional operation. For more information, see the **Get Message From Controlled Listener** action.

## Controlled and non-controlled listeners

### Non-controlled listener

A non-controlled listener is always waiting for the next command request from the remote queue provided the following process has taken place:

- A listener map definition has been created.
- A listener trigger has been created that references that listener map definition as the triggering event condition.
- The listener trigger and its associated project has been started.

Having received a request, the non-controlled listener will complete the processing and send a response (if configured to do so) before getting the next command request.

The interval of time that the non-controlled listener waits before getting the next command is determined by how long the previous request takes to complete.

### Controlled listener

A controlled listener, on the other hand, is configured so that a **Get Message From Controlled Listener** trigger action determines when the listener should get the command request from the remote queue.

The result of getting a request from the queue is communicated to the action step.

The point in time at which the **Get Message From Controlled Listener** action is executed is determined by the event condition associated with the trigger definition.

The event condition could be on a certain schedule (schedule trigger) or associated with a controller variable change (data trigger). Note this trigger is not the listener trigger.

For more information, refer to [Creating a controlled listener](#).

## IIoTA industrial IoT Platform: Creating a TCP listener

A TCP listener provides TCP server socket support at a specific port within the node.

The socket will accept connections and receive messages from a TCP client application.

The TCP client application will send messages to the TCP listener in an XML format and ASCII delimited messages.

TCP listeners can receive the messages on the server socket, format the messages internally, and then use the contents of the message to execute a trigger.

The results of the trigger execution and the values of any requested device variables can optionally be returned to the TCP client application that sent the message.

A TCP listener is different than other listeners because the TCP listener acts as a server application accessible by various client applications.

Other listeners such as WMQ listeners and JMS listener types are clients that connect to message brokers who funnel messages to the listener component.

For TCP listeners, the Transaction Server will create a server application that will accept and deny connection requests from outside clients.

The TCP listener will receive and validate any messages that are sent to it once a connection is established.

A TCP listener is created much like other listeners but have parameters that support inbound messages sent across a TCP connection.

### TCP Client programming

If the TCP client you are using to send a request to the TCP Listener is using a Connect/Send/Disconnect pattern without waiting for a reply, you will need to ensure that you

call a shutdown/flush on the socket prior to closing it. This allows the TCP stack to send all the data frames before initiating the close of the connection. If this is not done the TCP listener may not receive the entire request in case the request payload is large enough to span multiple Ethernet frames.

To create a TCP listener, follow these steps:

1. From the Workbench left pane, expand the node that you want to add the listener to.
2. Expand **Enterprise**, right-click the **Listeners** icon to display its pop-up menu, and then click **New**.

The Listener window appears.

3. Use the **Name** box to type a unique name for the listener. The name can be up to 64 characters in length and can include letters, numbers, and the underscore character. Spaces are not allowed.
4. Click the **Listener Type** down-arrow, and then select **TCP**.

The Listener window changes to accommodate the TCP listener.

The following describes the parameter values to create a TCP listener.

Parameter	Description
<b>Port</b>	This is the port on the node that the TCP listener will be listening on.
<b>Max Msg Size (KB)</b>	The parameter indicates a size at which the listener should reject a message and report back an error to the TCP client application. The default setting for this parameter is 1 KB. The setting for this parameter will protect the system when a client application maliciously sends data or sends data that is missing termination characters.

<p><b>Max Connections</b></p>	<p>The default value is 1 connection. The TCP listener is designed to allow multiple client applications to connect to it at one time. This parameter will determine how many concurrent connections a particular TCP listener will allow. Subsequent client application connection requests received by the TCP listener will be rejected once the maximum number of connections has been reached. Connections will become available when client applications close their connection with the listener.</p>
<p><b>Msg Timeout (Sec)</b></p>	<p>This parameter provides a means for the system to close a socket that is waiting for the end of a transmission. For example, suppose a TCP listener began to receive a message from a client application and the transmission of the message stopped prior to receiving the termination characters. For that example, the TCP listener has received a partial message and could be caught waiting endlessly for additional characters that might not be coming. Rather than having the connection wait endlessly, a timeout value can be set that will cause the connection to the client application to close in this partial transmission situation. An error condition for this TCP listener will be reported if the timeout value is reached.</p> <p>The value for this parameter will be represented as the number of seconds between a message pause that the system will wait. The default value is 60 seconds. The smallest this value is 30 seconds and the largest this value should be is 300 seconds (5 minutes).</p>
<p><b>Filter Addresses</b></p>	
<p><b>New Subnet</b></p>	<p>In the <b>New Subnet</b> box, type the IP address, and then click <b>Add</b>. You can also type an optional subnet address.</p> <p>To delete an address, select it, and then click <b>Remove</b>.</p>
<p><b>Defined Subnets</b></p>	<p>This parameter will contain a list of allowable IP addresses and subnet masks that will provide the ability to restrict access to the TCP listener to only those addresses that are defined in this parameter.</p> <p>The TCP listener will create a server socket that will be available for the client application to connect to. The default behavior is to allow all connections to access this socket.</p>  <p>From the <b>Defined Subnets</b> pane, select the address, and then click <b>Allow</b> or <b>Deny</b>. For more information, see <a href="#">Allowing and Denying access to the subnet for the TCP listener</a>.</p>

<b>Send reply message</b>	This parameter indicates whether or not the TCP listener will be sending a reply message to the client application. Select the <b>Send reply message</b> check box to set the TCP listener to reply to all incoming messages. By default, the check box is selected Clear the check box to set the TCP listener to not send a reply message.
<b>Encoding</b>	This is the UTF encoding scheme used to translate the ASCII bytes received.  

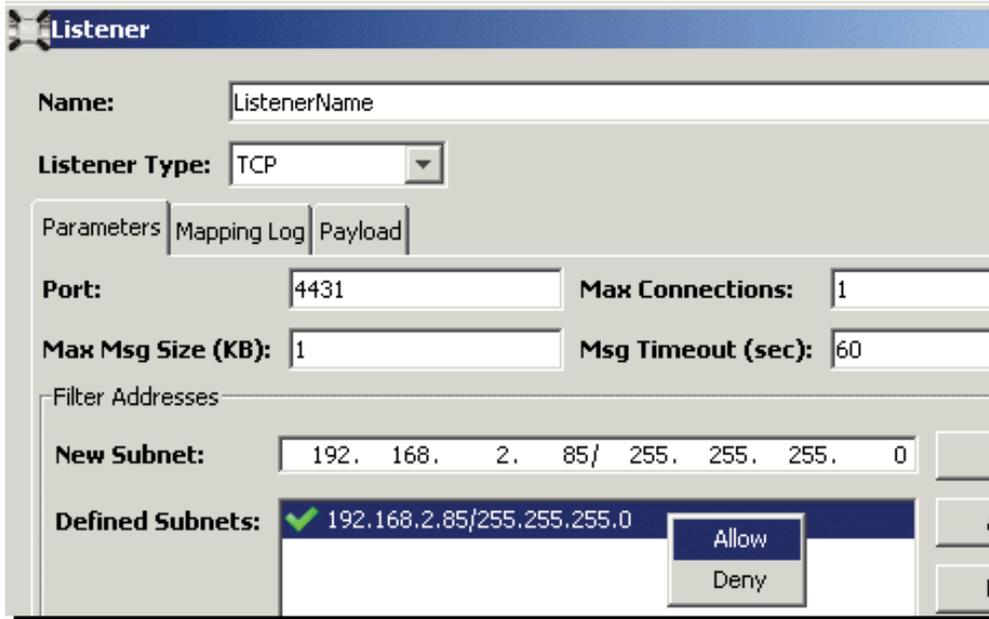
Related topics

Listener Mapping Log tab

Listener Payload tab

## IIoTA industrial IoT Platform: Allowing and Denying access to the subnet for the TCP listener

When creating (or editing) a TCP listener, the **Defined Subnets** parameter is used to add allowable IP addresses and subnet masks. By default, each subnet is set to allow access to the TCP listener.



**Listener**

Name:

Listener Type:

Parameters | Mapping Log | Payload

Port:  Max Connections:

Max Msg Size (KB):  Msg Timeout (sec):

Filter Addresses

New Subnet:

Defined Subnets:

<input checked="" type="checkbox"/>	192.168.2.85/255.255.255.0	Allow	Deny
-------------------------------------	----------------------------	-------	------

You can right click the subnet entry to display its pop-up menu to allow or restrict access to the TCP listener.

When you click **Deny**, the check mark will turn to an **X** and the subnet will not be available to the TCP listener.

## IIoTA industrial IoT Platform: ASCII payload considerations for the TCP listener

If a TCP listener is configured to not accept an ASCII payload, then any outbound reply message that is sent by the listener will not contain a termination character sequence. In this situation, the `</ListenerRequest>` XML tag acts as the termination sequence to delineate a complete request on the inbound side and the `</ListenerResponse>` XML tag acts as the termination sequence on the outbound side.

If a listener map using this listener is configured to send an ASCII reply, the payload that is delivered to the TCP application will not contain a termination sequence. If you want to use an ASCII outbound payload, you must append a character termination sequence to the ASCII payload definition in the listener map. For more information, see Listener maps.

## IIoTA industrial IoT Platform: TCP listener might not receive first message

This limitation affects all products that provide enterprise support.

### Problem scenario

A TCP listener might not receive the first message after the listener has been shut down and restarted.

When using a TCP transport to send messages to a TCP listener, the first message sent after the listener goes back to an active state (after being stopped and restarted) appears to be successful; however, the listener does not receive any message.

# IIoTA industrial IoT Platform: Example client application to interact with a TCP listener

## Overview

This section describes how to build and execute a simple example TCP client application to interact with a TCP listener, TCP listener map and listener trigger.

From this simple example, the basics of sending data to a TCP listener and receiving a reply message from a listener trigger can be expanded to meet your solution requirements.

## Prerequisites

- You have an Enterprise Gateway node where the example TCP listener project (listener, listener map, project and trigger) will be imported.
- You have installed JDK 1.6 or later on the computer that you will use to compile and execute the example TCP client application.

- 

## Step-by-step guide

1. Import the [Example TCP listener project](#) into your Enterprise Gateway node using the Workbench.
2. The imported items include:
  - A TCP listener, **ExampleTCPListener5050**:
    - Listens on Port 5050 for TCP clients to connect.
    - Mapping log is enabled
    - Payload format is ASCII
  - A TCP listener map, **ExampleDataFromAndReplyToTCPClient**:
    - Maps the data from the received message to be delivered to the listener trigger.
    - Maps reply data from the listener trigger to the reply message sent back to the example TCP client application
  - A project, **ExampleTCPListenerProject**, that contains a listener trigger, **ExampleTCPListenerTrigger**:
    - Max in Progress is set to 1
    - Trigger reporting is set to On

3. Access your Enterprise Gateway node with a Workbench and start the listener **ExampleTCPListener5050**, the project **ExampleTCPListenerProject** and listener trigger **ExampleTCPListenerTrigger**.

4. Copy the java source code included in the Info block below into a source file called **ExampleTCPClient.java**.

The example client sends an ASCII request payload that has the following fields delimited by a comma (,):

- ListenerMapIdentifier
- Message
- End-of-Message delimiter(\n).

For example: "DataFromTCPClient,Client1,HelloThere\n"

The listener forwards the request to the listener trigger which echoes the original message back.

Example Client Application Source

```
import java.io.BufferedInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.Socket;
import java.net.UnknownHostException;
```

```
public class ExampleTCPClient {
    public static void main(String[] args) {
        Socket sock;
        // Specify your Enterprise Gateway IP address here
        // or provide it as a command line argument
        String hostname = "127.0.0.1";
        byte[] readBuf = new byte[1024];
        if (args.length > 0) {
            hostname = args[0];
        }
        try {
            sock = new Socket(hostname, 5050);
            OutputStream output = sock.getOutputStream();
```

```

PrintWriter writer = new PrintWriter(new OutputStreamWriter(output,
"UTF-8"), true);
writer.append("ExampleDataFromAndReplyToTCPClient,Client1,HelloThere\n");
writer.flush();
InputStream input = sock.getInputStream();
BufferedInputStream bis = new BufferedInputStream(input);
sock.setSoTimeout(30000);
int bytesread = bis.read(readBuf);
System.out.println(" [" + bytesread + "] Bytes Received");
System.out.println(new String(readBuf, 0, bytesread));
sock.close();
} catch (UnknownHostException e) {
// TODO Auto-generated catch block
e.printStackTrace();
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
}
}
}

```

5. Compile the included java source using the following command:

```
javac ExampleTCPClient.java
```

6. Execute the example java client application using the following command:

```
java ExampleTCPClient <your listener host name or IP>
```

For example: *java ExampleTCPClient 127.0.0.1*

You should see the following output if all the components are configured correctly:

Example Client Application Output

[49] Bytes Received

0,Hello [Client1], your message was [HelloThere]

## Next steps

From this simple example TCP client application and example TCP listener (and listener map and listener trigger) build the application logic needed for your requirements.

The listener mapping log feature can be used to help understand the message format into the TCP listener and the reply message from the listener trigger.

The trigger reporting feature can be used to help understand the execution behavior of the listener trigger.

## IIoTA industrial IoT Platform: XML listener commands

The Transaction Server defines an XML command and response format that external enterprise applications can use to send and receive messages. The commands work in conjunction with the listener feature.

The enterprise application can use these commands to send parameters that will be forwarded to the listener trigger and receive parameters that are sent back to the enterprise after completion of the listener trigger.

Each command must begin with the following process instruction:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

### XML listener request payload

A listener request payload follows a predefined XML format. The request must specify a listener map identifier that the listener will use to locate the appropriate listener map definition. This definition will be used to map the items specified in the request payload to an internal request that will trigger the listener trigger.

For more information see **ListenerRequest** below.

The following is for the programmer who is experienced with message queuing software and wants to deliver an enterprise application that can interact with the node via these listener commands.

### ListenerRequest

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<ListenerRequest seq="100" >
<ListenerMap id="GetBinInformation" version="1" />
```

```

<Item name="PlantBinLocationId">
<Data>1254</Data>
</Item>
<Item name="PlantBinLevel" >
<Data>10 </Data>
</Item>
<Item name="PlantPriorBinLevel" >
<Data> 5</Data>
</Item></ListenerRequest>

```

The sample contains a **<ListenerRequest>** tag with these attributes and nested tags:

- **seq**= an attribute that specifies a unique string that will be returned in the reply to this request. The string is used by the enterprise application to differentiate a reply from several replies on the reply queue.
- **<ListenerMap>** a nested tag whose **id XML attributeid=** attribute specifies the name of the listener map definition that will be used to map this request.

For example:

```
<ListenerMap id="LEVT1" />
```

- **<Item>** a nested tag whose **name=** attribute specifies the name of the item element. This name should match the **PropertyName** column of an entry in the **From Enterprise** section of a Listener Map Definition window, in order for the data in the **<Data>** element to be sent to the listener trigger.
- **<Data>** a nested tag whose value determines what is sent to the Listener Trigger.

## ListenerReply

```

<?xml version="1.0" encoding="ISO-8859-1"?><ListenerReply seq="100" ><ListenerMap
id="GetBinInformation" /><Format encoded="no" delimiter="," /> <Item
name="PlantBinLevel"> <Data>100</Data> </Item> <Item name="PlantBinLocation">
<Data>25</Data> </Item></ListenerReply>

```

The sample contains a **<ListenerReply>** tag with these attributes and nested tags:

- **seq**= an attribute that specifies a unique string that was specified in the ListenerRequest. The value is returned in the reply and can be used by the enterprise application to differentiate a reply from several replies on the reply queue.
- **<ListenerMap>** a nested tag whose **id=** attribute specifies the name of the Listener Map definition that was used to map the response.

For example:

```
<ListenerMap id="GetBinInformation"
```

- **<Item>** a nested tag whose **name=** attribute specifies the name of the item element. This name is specified in the **PropertyName** column of an entry in the **To Enterprise** section of a Listener Map Definition window's **Output** tab.

- <Data> a nested tag that contains the value being returned from the listener trigger to the enterprise.

## IIoTA industrial IoT Platform: Creating a listener trigger

The listener map definition is surfaced at run time through a listener trigger.

A trigger is executed when an event occurs.

There are different types of events: schedule (or time based) events, data driven events, and listener maps.

The listener maps occur when the node receives a message from an enterprise application defined in a listener.

The contents of the message are defined by a listener map definition.

### Assumptions

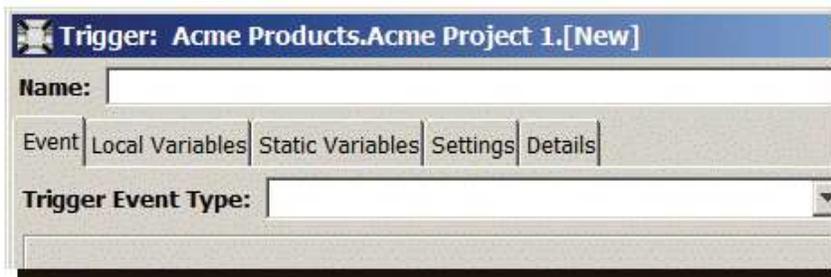
Before you begin to create the listener trigger, the following should have occurred:

- You reviewed Configuring listeners.
- You created a listener map and associated a predefined listener with it.

### Procedures

To create a listener trigger, follow these steps:

1. From Workbench left pane, go to the appropriate node, and then click the **Projects** icon.  
The Projects tab appears.
2. Double-click the project you want to add the trigger to.  
The project tab appears.
3. Click the **New** button from the bottom of the project tab.  
The Trigger window appears.

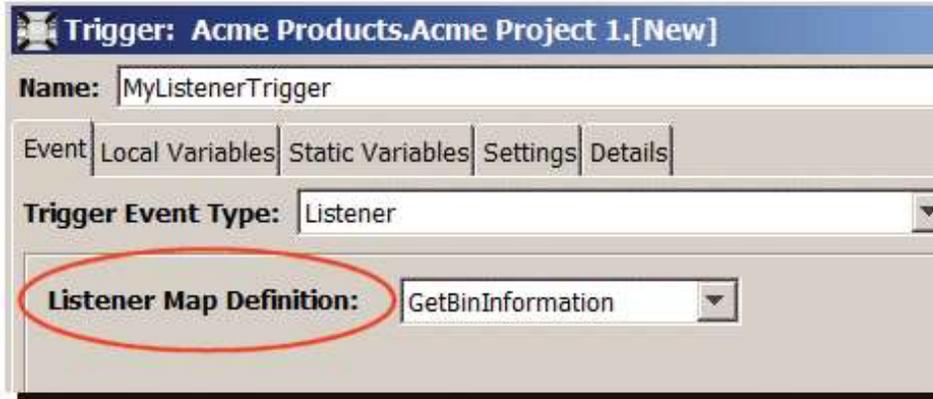


4. In the **Name** box, type a name for the trigger. A listener trigger name can be up to 64 characters and include letters, numbers, spaces, and hyphen and underscore characters. Only letters and numbers can be used for the first character of the name. Spaces are removed from the end of the name.
5. Click the **Settings** tab.

The screenshot shows a configuration window titled "Trigger: Acme Products.Acme Project 1.[New]". The "Name" field contains "MyListenerTrigger". Below the name field are five tabs: "Event", "Local Variables", "Static Variables", "Settings", and "Details". The "Settings" tab is active. It contains several input fields: "Max in Progress" with a value of 100, "Reporting" set to "Off", "Max Exec Time (ms)" with a value of 3000, "Editor mode" set to "User preference", and "Queue Size" with a value of 20.

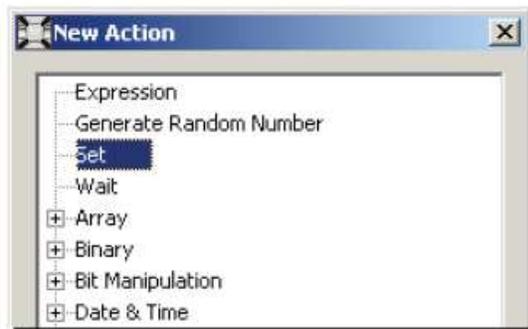
6. Set the parameters as follows:
  1. In the **Max in Progress** box, type the number of triggers that must complete processing before another trigger can execute. If the number of executing triggers exceeds the number specified, the triggers are stored in a pending queue if queue is enabled, otherwise they overflow.
  2. From the **Reporting** drop-down list accept the **Off** default value.
  3. In the **Max Exec Time (ms)** box, type a value in milliseconds for the maximum execution time for the trigger. If the trigger exceeds this value, a warning message is logged in the Exceptions Log.
  4. In the **Queue Size** box, type a value that will represent the number of items to hold in the trigger queue. For this example, 20.
  5. From the **Editor mode** list accept the default value. The **Editor mode** allows you to specify a method for creating actions for the trigger. You can drag and drop an action (Canvas) or select an action from a list (List).
7. Click the **Event** tab.

- From the **Trigger Event Type** drop-down list, select **Listener**.  
The window changes to accommodate a listener trigger.



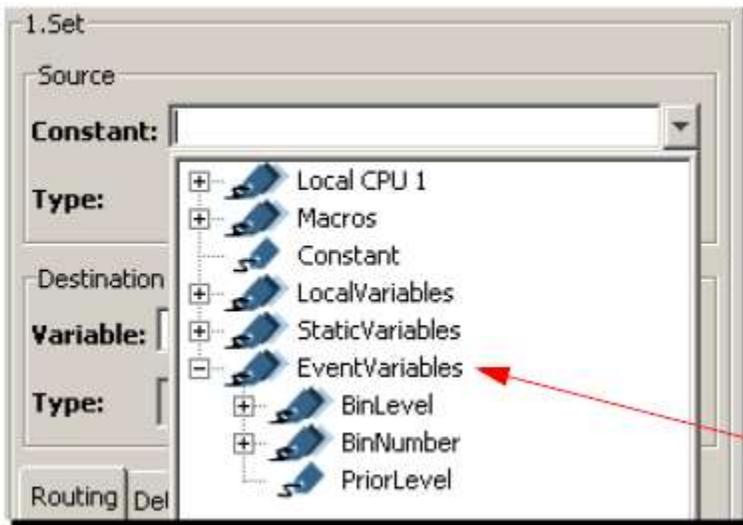
- From the **Listener Map Definition** drop-down list, select the listener map you want to use for the trigger. The list displays all the listener map definitions created on the current node. For this example, select *GetBinInformation*.  
The next step is to specify an action in the trigger. The following assumes that you have selected the action from a list.

- From the **Event** tab, under the **Actions** section, click **Add**.  
The New Action window appears.



- Select **Set**, and then click **Add**.  
The right pane changes to accommodate the **Set** action.  
Within the **Set** pane, there are **Set Source** and **Set Destination** sections.  
The next step is to specify a map variable that was defined in the **GetBinInformation** listener map as the source.
- Under **Source**, next to **Constant**, click the down-arrow to display a list of variables.  
A list of variables appears.

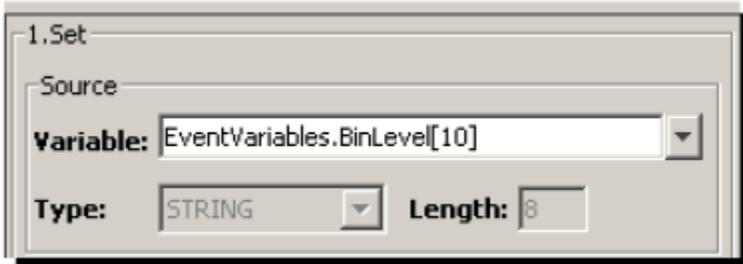
13. Expand the list of variables to locate **EventVariables**, and then expand **EventVariables**



For this example, the map variables that were defined in the **Input** tab of the **GetBinInformation** listener map definition appear.

For listener triggers, the values of these source map variables can be used as input for Transaction actions, Expression actions, or in Set actions as shown in this example.

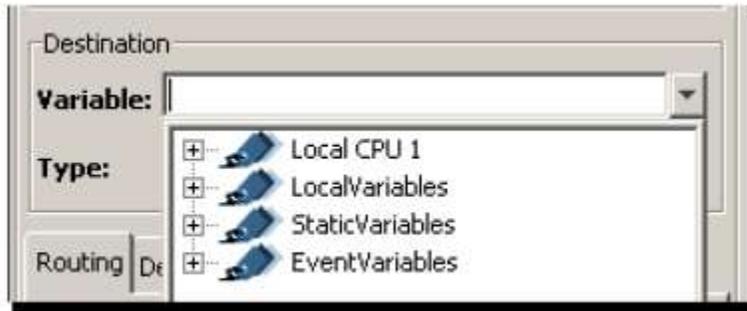
14. Select the appropriate variable to use as the source. For this example, *BinNumber*. The variable is added to the **Variable** box.



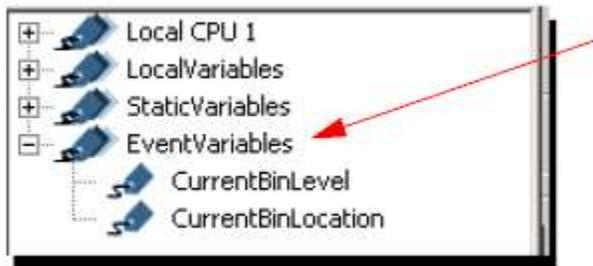
The next step is to specify an output map variable that was defined in the **Output** tab of the **GetBinInformation** listener map as the destination variable. Values can be written to these variables when the trigger executes.

15. Under **Destination**, next to the **Variable** box, click the down-arrow to display a list of variables.

A list of variables appears.



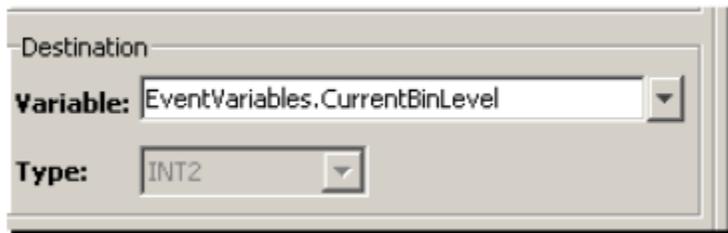
16. If necessary, expand the list of variables to locate the **EventVariables**, and then expand **EventVariables**



The variables created for the **GetBinInformation** listener map are listed.

17. For this example, select **CurrentBinLevel**.

The variable name is added to the **Variable** box.



The values written to the listener map output variables will become the contents of the Listener Reply message sent back to the enterprise system upon completion of the listener trigger.

The trigger is completed.

1. Click **Validate**.
2. A message will say whether or not the validation was successful. Click **OK**.

3. If no errors are received, click **Save**.
- 4.

The name of the trigger is added to the project tab.

Acme Products / Projects				
Projects <span>✓ Acme Project 1</span> <span>✕</span>				
Name ▾	Type	State	Status	Last Triggered
MyListenerTrigger	Listener	<input type="checkbox"/> Stopped	Unloaded	n/a

You must start the trigger and also start the listener associated with the trigger.

## IIoTA industrial IoT Platform: Using the Listeners tab

The **Listeners** tab provides a list of all saved listeners. In addition, the **Listeners** tab displays the status of each listener, the number of messages received by the listener and replied to, error messages and error codes, and more.

To display the **Listeners** tab:

From the Workbench left pane, expand the node whose listener you want to review, expand **Enterprise**, and then click the **Listeners** icon.

The Listeners tab appears in the right pane.

Acme Products / Enterprise				
Transport Maps   Transports   Listener Maps   Listeners				
Name ▲	Type	State	Dispatched	Rejected
AcmeProducts111	WMQ	Down	0	0
AcmeProducts112	WMQ	Down	0	0
AcmeProducts113	TCP	Started	0	0

The tab has a table format with these columns:

Column	Description
<b>Name</b>	This is the unique name of the listener.
<b>Type</b>	Listener types are: <b>JMS</b> (for WebSphere SIB queues) <b>MSMQ</b> (for Microsoft Message Queuing queues) <b>TCP</b> (for TCP server socket support) <b>WMQ</b> (for WebSphere MQ queues)
<b>State</b>	Can be: <b>Connecting</b> — Indicates the Transaction Server is connecting to the WebSphere queue manager. Should the state of the listener remain at <b>Connecting</b> , an error has occurred. You can view the error by selecting the listener. The error information will be displayed at the bottom of the <b>Listeners</b> tab. <b>Started</b> — Indicates the Transaction Server has started the listener; however, no listener triggers have been started that reference this listener. This state indicates that the listener is operational and has been initialized. <b>Active</b> — Indicates the listener is waiting on a remote command queue for a request from an external enterprise application and has processed at least one message. <b>Down</b> — Indicates that the listener has not been initialized. <b>Suspended</b> — Indicates that the listener has been initialized but processing on the node was stopped using the <b>Stop</b> button.
<b>Dispatched</b>	The number of messages successfully dispatched to the listener trigger.
<b>Rejected</b>	The number of messages that were not processed by the listener.
<b>Replied</b>	The number of replies sent by the listener. A listener can be configured to send a reply or to not send a reply upon the receipt of a message.

Various actions will cause the listener states to change. You can start or stop a listener from the Listeners tab. The act of starting a listener trigger will move listeners who are in a **Start** state into an **Active** state.

The **Listeners** tab also provides a pop-up menu that can be used to create, edit, delete, duplicate, start, and stop a listener.



You can also import previously exported listeners. For more information, see [Exporting and importing listeners](#).

The **Clear Counters** option allows you to reset to zero all columns that have counters such as **Dispatched**, **Rejected**, and **Replied**.

When the listener encounters a communication error, you can review the information from the lower portion of the **Listeners** tab.

## IIoTA industrial IoT Platform: Editing a listener

You can change parameters for the listener at any time. However, keep the following in mind:

- If the listener is in a **Connecting**, **Active**, or **Suspended** state, the Transaction Server will stop and restart the listener so that the changes will take effect.
- Any transactions that are currently in progress (the listener has taken the request off the command queue, and it is in the process of being executed) will be completed before the new definition changes take effect. If this is not the desired behavior, then you should use the **Listeners** tab to stop the listener prior to editing the listener definition.

## IIoTA industrial IoT Platform: Exporting and importing listeners

The IIoTA Workbench gives you the ability to export a listener and then import the listener into a different node. You can also export more than one listener at the same time.

### IIoTA industrial IoT Platform: Exporting a single listener

1. From the **Listeners** tab, select the listener you want to export, display the pop-up menu, and then click **Export**.

The Select Items to Export window appears.



2. From the bottom of the Select Items to Export window, click the browse button.

The Export File Location window similar to the following appears.



3. The listener name that you selected in Step 1 is automatically added to the **File name** box. The word **listener\_** is added to the front of the file name (for example Listener\_AcmeProducts111). The listener file name will be appended with a DWX file extension.
4. If necessary, navigate to the drive and folder that you want to save the listener to, and then double-click the folder. The folder name is added to the **Look in** box.
5. Click **Select**.

The Select Items to Export window reappears with the path and file name added to the **File** box.



6. Click **Export**.
7. A message will tell you the listener was successfully exported. Click **OK**.

## IIoTA industrial IoT Platform: Exporting multiple listeners

You can export more than one listener at the same time.

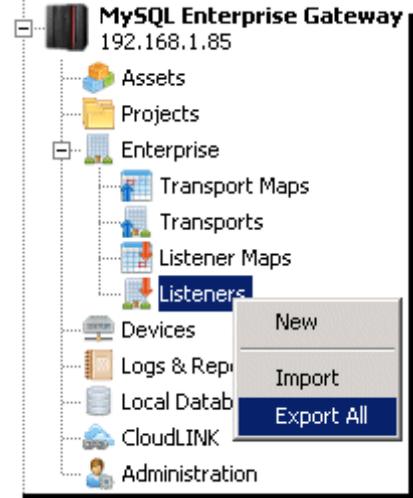
1. From anywhere in the Listeners tab, display its pop-up menu, and then click **Select All**.
2. Display the pop-up menu again, and then click **Export**.

The following describes three additional selection methods for exporting more than one listener from the **Listeners** tab.

- To select consecutive listeners: Click the first listener, press and hold down SHIFT, and then click the last listener. Anywhere on the **Listeners** tab, display its pop-up menu, and then click **Export**.
- To select listeners that are not consecutive: Press and hold down CTRL, and then click each listener. Anywhere on the **Listeners**

You can also export all listeners simultaneously using this method:

1. From the Workbench left pane, expand the node that contains the listeners you want to export.



2. Expand **Enterprise**, right-click the **Listeners** icon to display its pop-up menu, and then click **Export All**.
3. The Select Items to Export window appears with the appropriate listing of listeners.



All the listeners are marked to be exported.

If you decide to not export one or more of the listeners, simply select it, and it will not be exported.



1. From the bottom of the Select Items to Export window, click the browse button.

The Export File Location window appears. The word *listener* is automatically added to the **File name** box. The listeners file name will be appended with a .dwx file extension. All the exported listeners will be bundled within this file name.

1. Navigate to the drive and folder that you want to save the listeners to, and then double-click the folder. The folder name is added to the **Look in** box.
2. Click **Select**.

The Select Items to Export window reappears with the path and file name added to the **File** box.

1. Click **Export**.
2. A message will tell you the listeners were successfully exported. Click **OK**.

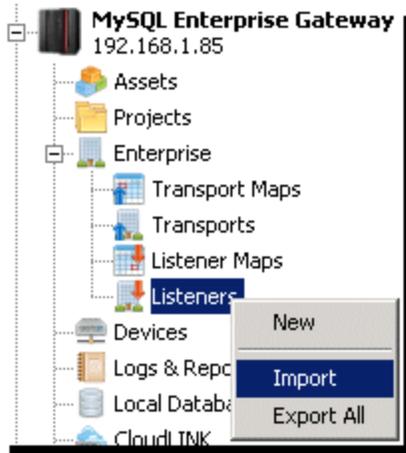
You are now ready to import one or more listeners.

## IIoTA industrial IoT Platform: Importing one or more listeners

It is assumed that you have previously exported a listener.

You can import a single listener or all listeners that were exported as a single exported file, the steps are the same.

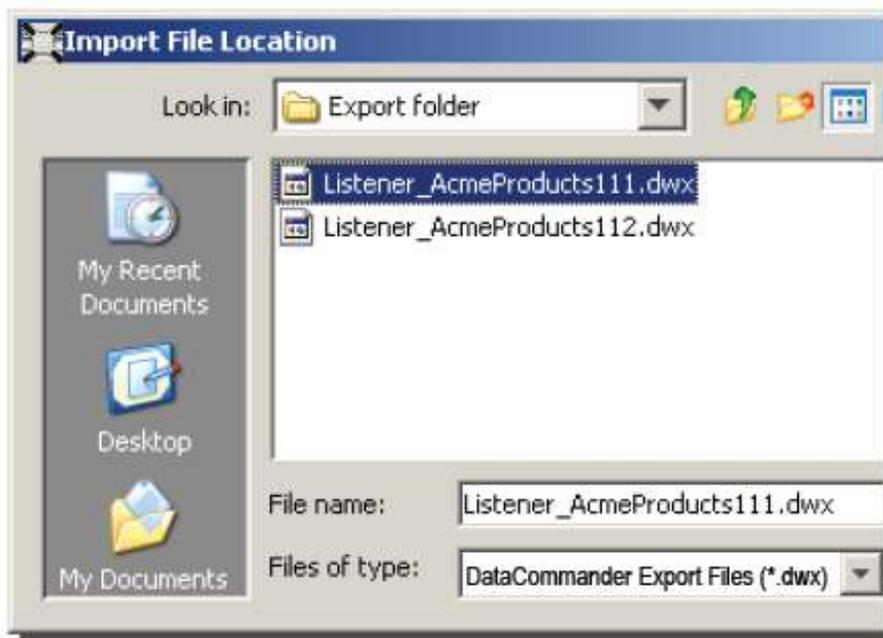
1. From the Workbench left pane, expand the node that you want to import one or more listeners into.



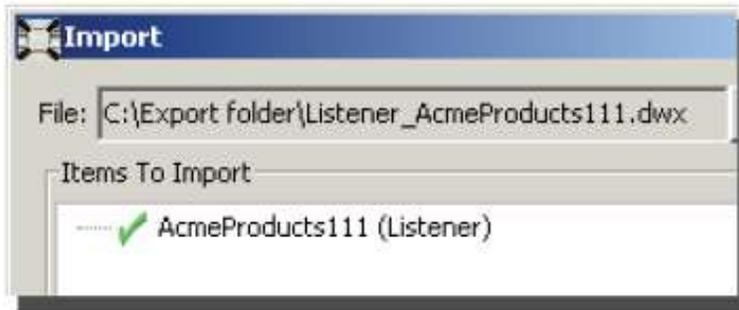
2. Expand **Enterprise**, right-click the **Listeners** icon to display its pop-up menu, and then click **Import**.

The Import File Location window appears.

3. Navigate to the drive and folder that contains the previously exported listener, and then double-click the folder. The folder name is added to the **Look in** box.
4. Select the listener file you want to import.
5. When the file name is added to the **File name** box, click **Select**.



The Import window appears.



- The **File** box shows the location and file name of the exported file. You can change the path specification by using the browse button.

The **Items in Import** section shows the component that is to be imported.

- Click **Import**.
- A message will tell you that the import completed. Click **OK**.

The listener appears in the **Listeners** tab.

## IIoTA industrial IoT Platform: Using the Listener Maps tab

The **Listener Maps** tab provides a list of saved listener maps. From this tab, you can obtain the name of the listener that the listener map is associated with, the type of listener, and other useful information.

To display the **Listener Maps** tab:

- From the Workbench left pane, expand the node whose listener you want to review, and click the **Listener Maps** icon.

The **Listener Maps** tab appears in the right pane.

Acme Products / Enterprise			
Transport Maps	Transports	Listener Maps	Listeners
Name ▲	Listener	Listener Type	Successes
GetBinInformation	AcmeProducts111	WMQ	

The tab has a table format with these columns:

Column	Description
<b>Name</b>	The name of the listener map.
<b>Listener</b>	The name of the listener that is associated with the listener map.
<b>Listener Type</b>	The listener type: <b>WMQ</b> (for WebSphere MQ queues) <b>JMS</b> (for WebSphere SIB queues) <b>TCP</b> (for TCP server socket support)
<b>Successes</b>	This column provides a count of messages that were successfully received and parsed according to an active listener event.
<b>Failures</b>	This column indicates a count of messages that the listener could not process. The cause of the failure could be due to one of the following reasons:  The message payload is missing from the message received. The message payload is missing the name of a listener event. The message payload could not be parsed according the specifications defined in the listener event. The listener event defined in the message payload is not defined or is not currently active.

The Listener Maps tab also provides a pop-up menu that can be used to create, edit, delete, and duplicate a listener map.



Using this menu, you can import a previously export listener map and clear data from the columns using the **Clear Counters** option.

## IIoTA industrial IoT Platform: Configuring a default listener map

### Overview

This page describes how to specify a default listener map for a listener.

A default listener map allows a user to specify a listener map that will be used to process a received payload when all configured methods of establishing a listener map have failed.

You can also choose to specify a default listener map instead of configuring a payload specific listener map identification mechanism like a **Listener Map Key ( JMS )** or a **Listener Map offset (ASCII)**.

All listener types support the concept of a default listener map.

#### Important

The default listener map that you specify should be able to process the payload type that the listener will receive. Otherwise the request will not be processed correctly. For example, If the default listener map expects an XML payload and the listener is receiving an ASCII delimited payload the request will be rejected as an incompatible payload.

### Assumptions

The following is assumed:

- You are familiar with configuring a Listener and a Listener Map.
- You are familiar with the Listener payload formats like XML, ASCII, Map, Custom and XSD.
- You have a workbench that is accessing a node

## Procedures

In order to specify a default listener map, follow these steps:

1. From the Workbench left pane, expand the node that you want to add the listener to.
2. Expand **Enterprise**, right-click the **Listeners** icon to display its pop-up menu, and then click **New**.  
The Listener window appears.
3. Use the **Name** box to type a unique name for the listener. The name can be up to 64 characters in length and can include letters, numbers, and the underscore character. Spaces are not allowed.
4. Click the **Listener Type** down-arrow, and then select a listener type (for this example a JMS listener).
5. Click the **Payload** tab.
6. Use the **Format** down-arrow, and then select the format type (for this example, **Map**).

The screenshot shows the 'Listener' configuration window with the following details:

- Name:** MyListener
- Listener Type:** JMS
- Format:** Map
- Map Key:** eventSource
- Sequence Key:** eventSeq
- Default Listener Map:** LMAPDefault

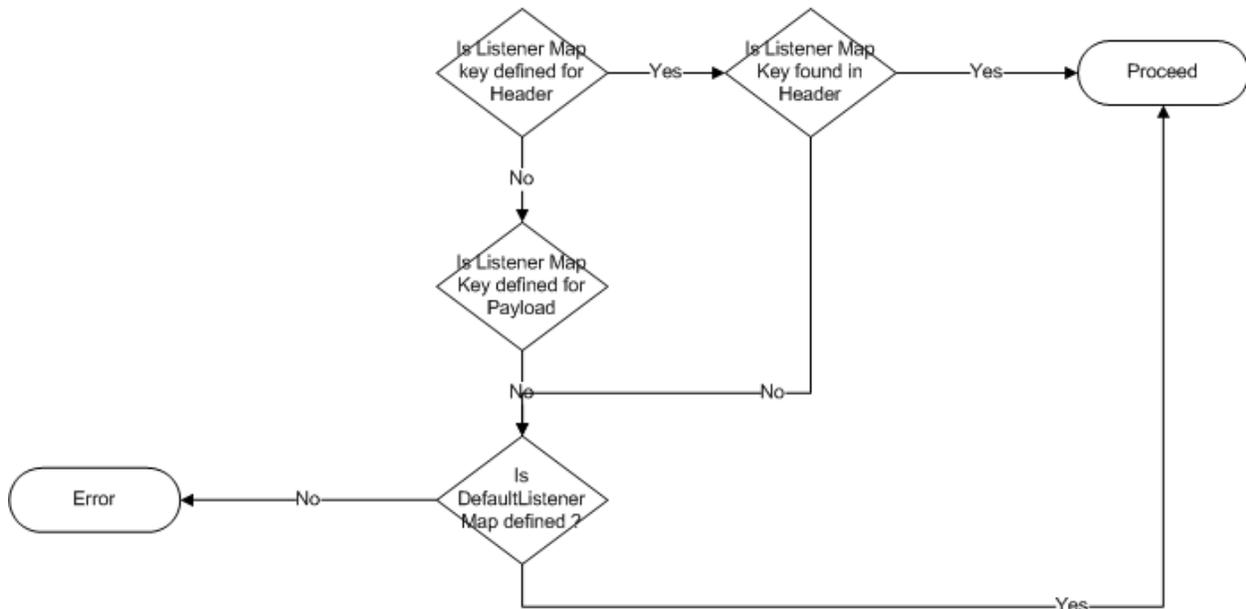
Red circles highlight the 'Payload' tab, the 'Map' format dropdown, and the 'eventSource' and 'LMAPDefault' text fields.

- In the **Default Listener Map** box, type the name of the listener map that you want to use.

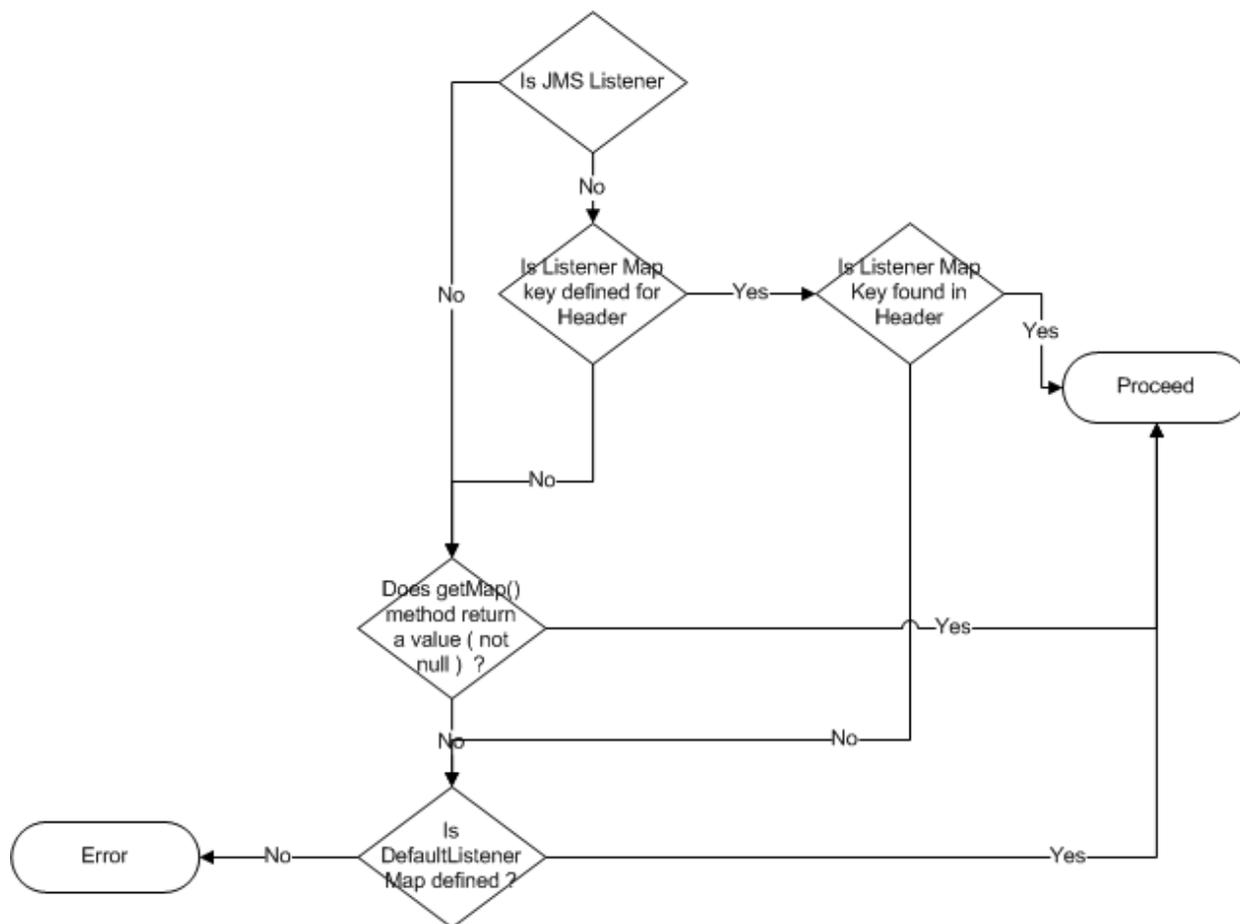
Parameter	Description
<b>Default Listener Map</b>	Specify a valid listener map name. The listener map should be able to process the payload type that is being received. For example, if you have configured a listener to receive a JMS map message then the default listener map should be configured to process a JMS <b>Map</b> payload.

The following flow charts illustrate how a default listener is used to process a payload.

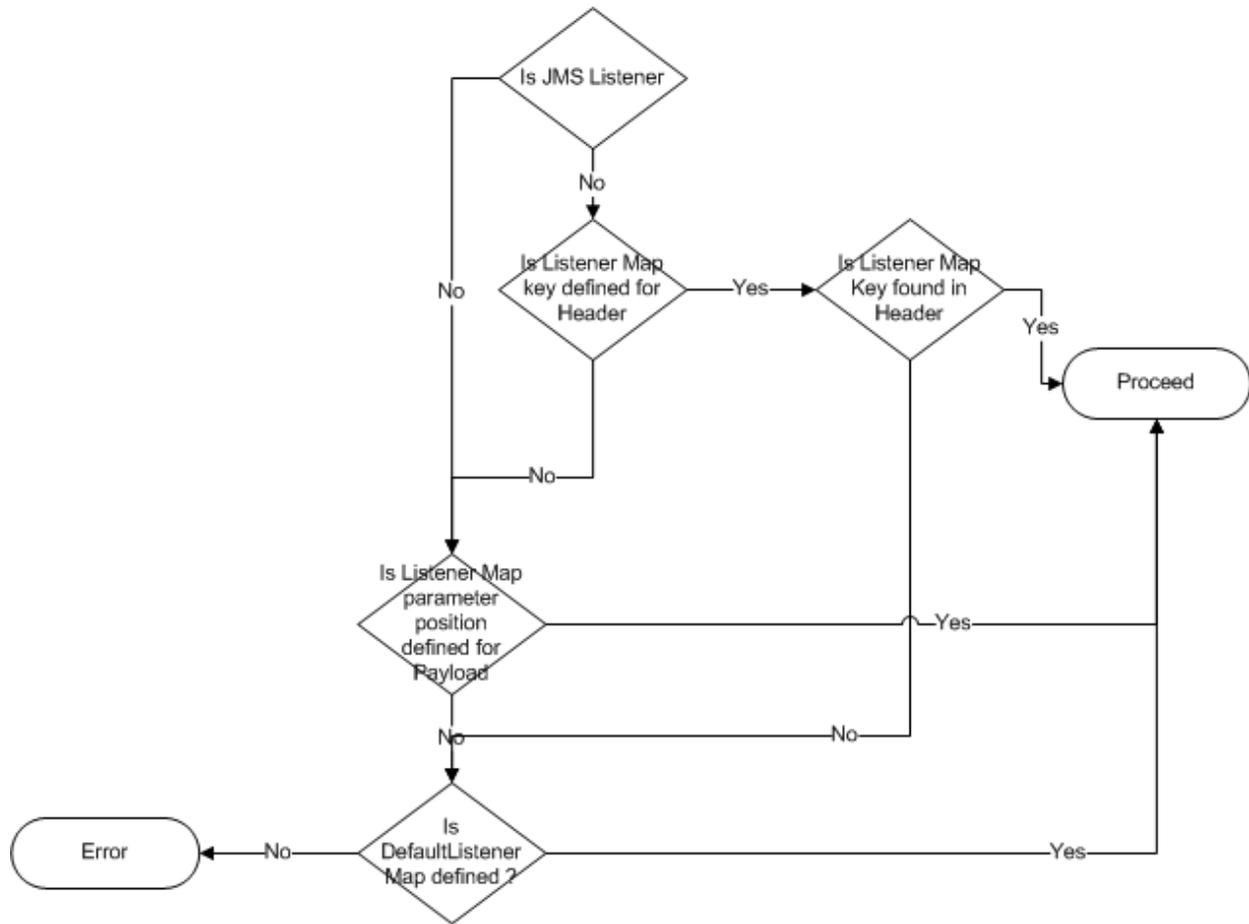
## JMS Mapped Message



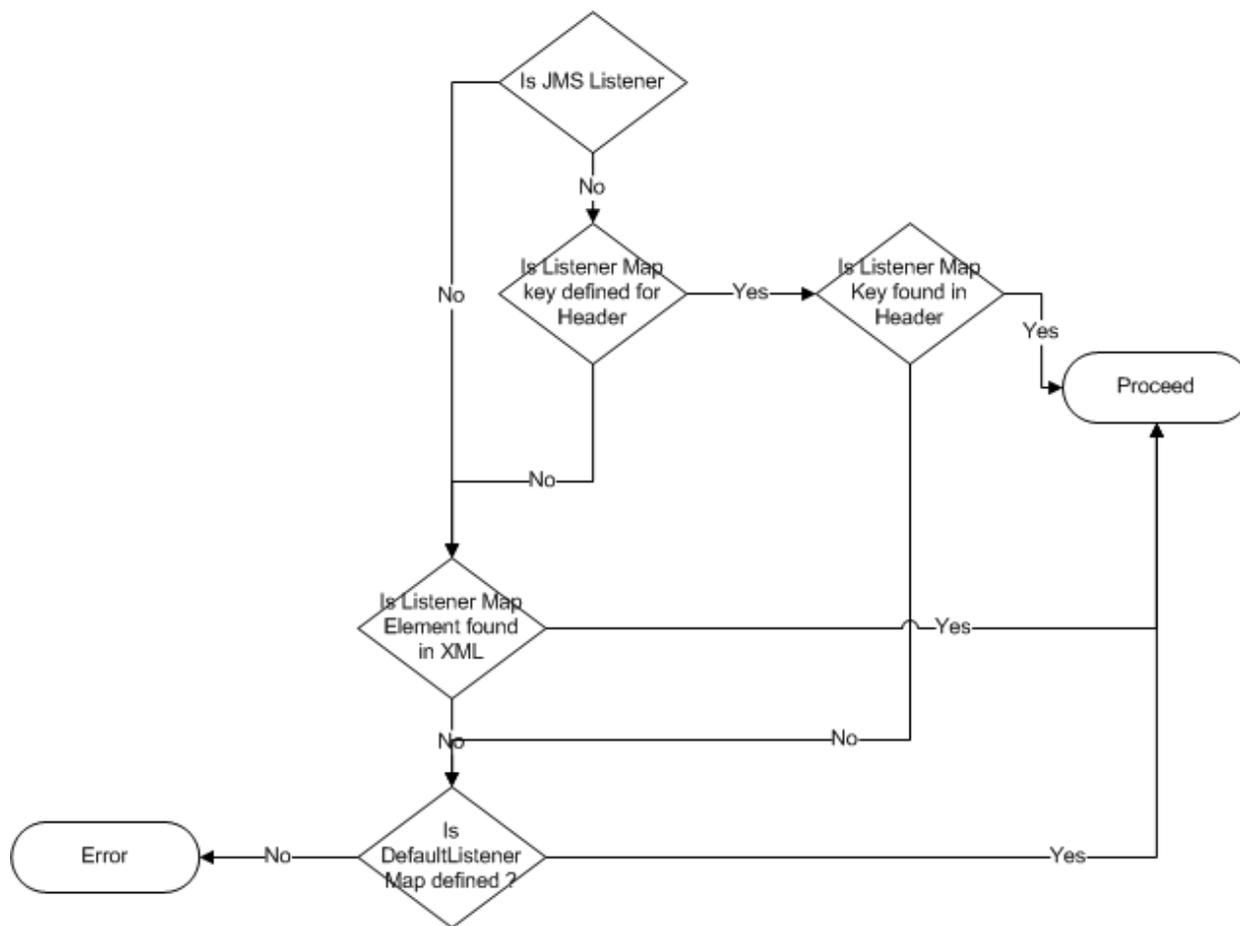
# Custom Payload



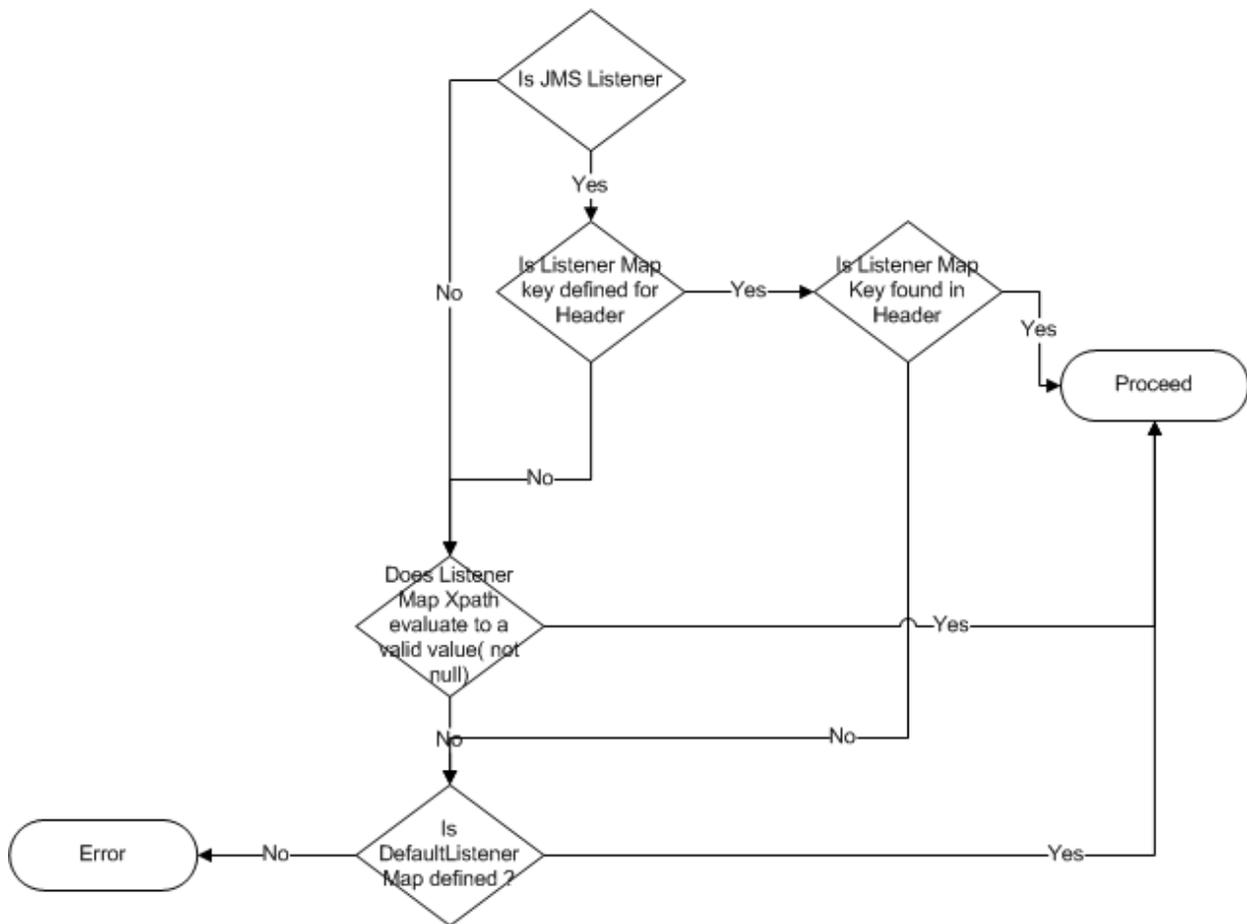
# Delimited Payload



# XML Payload



## XSD Payload



Related topics

[Creating a JMS listener](#)

[Listener Map format](#)

## IIoTA industrial IoT Platform: Tabs on the Listener window

When creating a listener and specifying a listener type, the Listener window provides a Listener Mapping Log tab and Payload tab.

**Listener**

Name: TCPLListener

Listener Type: TCP

Parameters | Mapping Log | Payload

Port:  Max Connections: 1

Max Msg Size (KB): 1 Msg Timeout (sec): 60

Filter Addresses

These two tabs and their parameters are common across the different types of Listener windows. The **Parameter** tab is unique based on the listener type selected.

## IIoTA industrial IoT Platform: Listener Mapping Log tab

The content of every incoming and outgoing transaction for the listener will be recorded in a mapping log.

**Listener**

Name: ListenerName

Listener Type: JMS

Parameters | Mapping Log | Payload

Mapping Log

Log Size (MB): 1 Number of Log Files: 1

Message Size (bytes): Entire Message  Copy rolled log to staging

The Mapping Log tab provides these parameters:

### Mapping Log

Select the check box to turn on map logging for the listener. When **Mapping Log** is selected other characteristics become available and must be specified as described in the following rows.

### Log Size (MB)

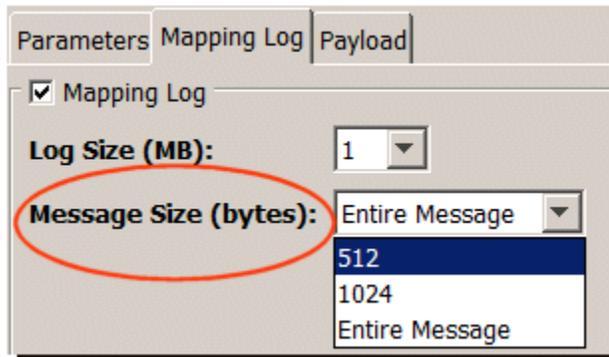
The value is the maximum size of the transaction log file for this listener. The size can be from 1 MB to 10 MB. Once the log file reaches the size specified, the file is archived. Subsequent transactions are recorded in the active log file.

### Number of Log Files

The value is the number of transaction log files that can be maintained for this listener.

### Message Size (byte)

The value controls the length of the line recorded in the transaction log for this listener.



You can specify that the entire message be recorded or limit the line to 512 or 1024 bytes.

### Copy rolled log to staging

Select this check box to send a copy of the log file to the root of the staging file system (the Staging Browser tab).

#### Related topics

Listener Payload tab

Creating a JMS listener

## IIoTA industrial IoT Platform: Listener Payload tab

A payload is the data that is delivered to the enterprise application.

**Listener**

**Name:** ListenerName

**Listener Type:** JMS

Parameters | Mapping Log | **Payload**

Format

XML  
XML  
ASCII  
Map  
Custom

Map ID Position: 0

Sequence ID Position:

The **Payload** tab provides formats that the payload can be delivered in:

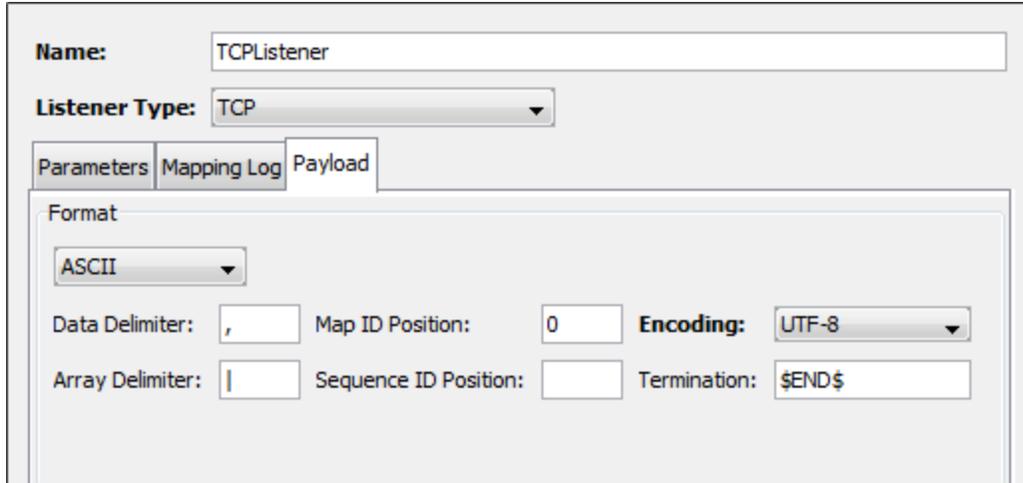
- XML — Requires a predefined external enterprise application program. For more information, see XML listener commands.

Related topics

Listener Mapping Log tab

## IIoTA industrial IoT Platform: Listener ASCII format

An ASCII format can be assigned to JMS, MSMQ, TCP, and WMQ listeners. When you enable a listener to receive ASCII character delimited payloads, other parameters become available and must be specified.



**Name:** TCPListener

**Listener Type:** TCP

Parameters Mapping Log **Payload**

Format

ASCII

Data Delimiter: , Map ID Position: 0 Encoding: UTF-8

Array Delimiter: | Sequence ID Position: Termination: \$END\$

The following describes the parameters on the **Payload** tab when **ASCII** is selected as a format option.

### Data Delimiter

One or more characters used to delimit the data. These characters must be different from the characters used to delimit array elements. The default is a comma ( , ) character.

### Map ID Position

The offset position in the message where the listener map definition can be found. The default is zero.

### Array Delimiter

One or more characters used to delimit array elements. These characters must be different from the characters used to delimit data. The default is a bar ( | ) character.

### Sequence ID Position

The offset position in the message where the message sequence number can be found. The default value is zero.

### Encoding

The UTF encoding scheme used to translate the ASCII bytes received.

### Termination

This field is only displayed for TCP Listeners. The set of characters that delineate the end of a message sent to the TCP listener. This set of characters is expected to be appended to every message sent to the TCP Listener.

Example termination characters appended to a message:

- \$END\$
- ===END===
- EOM

If you need to specify control characters as a termination, then specify the characters with a preceding hex notation (0x). For example:

- To specify the line feed control character as the message termination, enter 0x0a.
- To specify the carriage return and line feed characters as the message termination, enter 0x0d0x0a.

*Example TCP Listener message with termination characters*

myListenerMapId,10,20,\$END\$

**Related topics**

Listener Payload tab

## IIoTA industrial IoT Platform: Listener Custom format

This section assumes that you understand how to create a listener and are ready to develop a custom payload format. MSMQ, JMS, TCP, and WMQ listener types support custom formats for their payload data.

The screenshot shows the 'Listener' configuration window. The 'Name' field is 'MyFirstListener'. The 'Listener Type' dropdown is open, showing 'WMQ' selected. The 'Format' dropdown is set to 'Custom'. The 'Jar Name' and 'Transform Class Name' fields are empty.

A custom payload format requires the use of a client created java application program that defines method(s) to retrieve the values for each listener event input and output map variables.

## Methods

The java application program must implement the following methods:

The `getMap(String)` method custom format `getMap(String)` method from the the Transaction Server interface to retrieve the listener map name from the custom payload.

The `formatErrorMessage` custom payload `formatErrorMessage(String, int, String)` method to report errors that might be encountered while processing the listener request.

## Signatures

The two methods have the following signatures:

```
public String getMap(String sourceData )
```

where:

sourceData is the content of the message sent to the listener.

```
public String formatErrorMessage(String mapName, int errCode, String data )
```

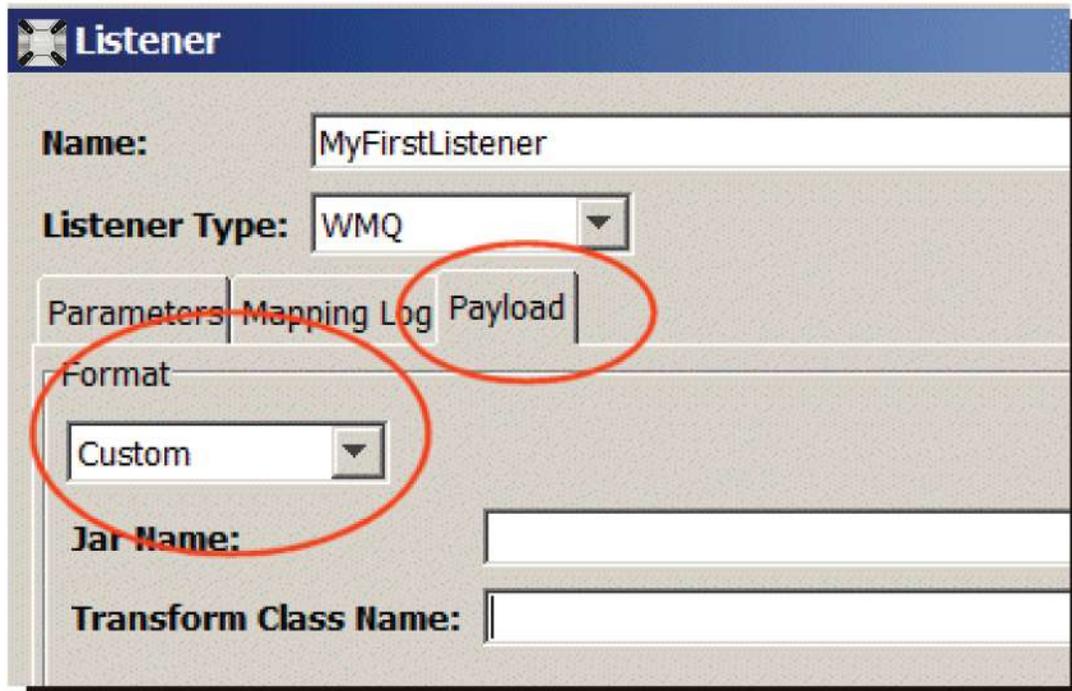
where:

mapName is the name of listener map (if available), errCode the error number associated with the error, and data is the error message.

## Adding the jar file and class name

When creating a listener that will use a custom format payload, you must specify a jar file and class name.

The following assumes you are creating a listener and have specified **WMQ** as the listener type.



1. From the Listener window, click the **Payload** tab.

- Use the **Format** down-arrow, and then select **Custom**.  
The **Payload** section changes to accommodate the **Custom** option.
- In the **Jar Name** box, type the name of the jar file that contains the client java application program.  
The jar file must be available in staging file system (the **Staging Browser** tab) on the node that contains the listener.
- In the **Transform Class Name** box, type the name of the class where the methods are defined.

The screenshot shows a configuration window with three tabs: Parameters, Mapping Log, and Payload. The Payload tab is active. Under the 'Format' section, a dropdown menu is set to 'Custom'. Below this, there are two text input fields: 'Jar Name' with the value 'dwcustom.jar' and 'Transform Class Name' with the value 'com.ils\_tech.custom.DWTransformRequest'. The 'Transform Class Name' field is circled in red.

For the example, the client java application program is packaged in **dwcustom.jar**. The class name is **DWTransformRequest**.

- Click **Validate**. If no errors are reported, a message will say the listener validated.
- Click **Save**. The name of the listener is added to the Listeners tab with the **State** column shown as **Down**.

## Termination character for TCP listener custom payload format

If you are creating a TCP listener, there is one additional parameter when you select the **Custom** option. This parameter identifies one or more characters that will indicate the end of an ASCII message. The characters set in the **Termination** parameter can be printable and unprintable characters.

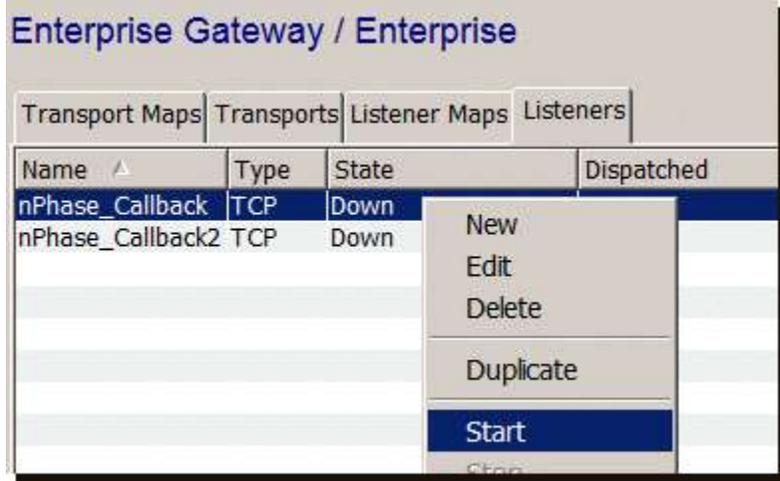
The screenshot shows the same configuration window as above, but with an additional field. The 'Jar Name' field now contains 'dwcustomXML.jar' and the 'Transform Class Name' field contains 'com.ils\_tech.customXML.DWCustomXML'. A new 'Termination' field is present at the bottom, containing the value '{end}', which is circled in red.

Unprintable ASCII characters are represented by their hexadecimal code. For example, 0x01 would represent the use of a hex 01 ASCII character as a termination value.

## Problems starting a listener with a custom format

Once the listener is saved, you can start the listener using the Listeners tab. To start a listener with a custom format:

1. Select the listener, and right click to display its pop-up menu.



2. Click **Start**.

If the **State** column remains as **Down**, the problem could be as follows:

- The name of the jar file specified in the **Jar Name** box is incorrect.
- The jar file is not present in the staging file system (the **Staging Browser** tab) on the node.
- The name specified in the **Transform Class Name** box is not found in the jar file.
- The `getMap(String)` method is not found in the transform class name.
- The `formatErrorMessage (String, int, String)` method is not found in the transform class name.

Related Topics

Staging Browser

Listener Payload tab

## IIoTA industrial IoT Platform: Listener Map format

The **Map** format enables customized names for map message attributes. The names that you specify will override the default names. The names will be case sensitive.

The screenshot shows the configuration interface for a listener. The 'Name' field contains 'JMSListener' and the 'Listener Type' is set to 'JMS'. There are three tabs: 'Parameters', 'Mapping Log', and 'Payload'. The 'Parameters' tab is active. Under the 'Format' section, a dropdown menu is set to 'Map', which is circled in red. Below this, there are four input fields: 'Map Key', 'Sequence Key', 'Error Code', and 'Error Message'.

The **Map** format is only available for JMS listener types.

The following describes the parameters available for a Map format:

### Map Key

A JMS property key name used to retrieve the value of the Listener Map name from the header or body of the received **JMS** payload. When processing a request message, the JMS listener will use the **Map Key** parameter as a key to identify value which is the listener map name. If the property key is not found in the header or payload, the default listener map is used (if specified). If the value yields a name that is not a predefined listener map, then the request is returned as an error.

### Sequence Key

A JMS property key name used to retrieve the sequence value from the header or body of the received JMS payload.

The following two parameters are only used in the Listener reply.

### Error Code

A name to use to retrieve internal error codes. If this parameter is left empty, the default is **ErrorCode**.

### Error Message

A name to use to retrieve internal error messages. If this parameter is left empty, the default is **ErrorMessage**.

### Related topics

Listener Payload tab

Creating a JMS listener

## IIoTA industrial IoT Platform: Listener XSD format

To allow the Transaction Server to receive and process XML described by XML Schema Definition Language (XSD), a listener must be created with the XSD payload type.

The screenshot shows the 'Listener' configuration window. The 'Name' field contains 'ListenerName'. The 'Listener Type' dropdown is set to 'JMS'. Below this are three tabs: 'Parameters', 'Mapping Log', and 'Payload'. The 'Format' dropdown menu is circled in red and shows 'XSD' selected. Below the 'Format' field are two empty text boxes labeled 'Map XPath:' and 'Sequence XPath:'.

Once the payload type is selected for a listener, the listener will only process messages of that type. Any other message type causes the listener to reject the message and log an error. The XSD payload is supported by all listeners:

- TCP
- MSMQ
- JMS
- WMQ

### Assumptions

- You know how to create a listener.
- You understand XSD and knowledge of the application program that will be sending the XML messages to the node. In addition, you can identify an XML tag that will be used to contain the listener map name.

## Specifying the listener map ID

For all the listeners, the listener map identifier can be specified as:

- a default listener map
- an xpath expression
- a JMS message header property (JMS listeners only)

The listener map is selected in the following order:

### 1) JMS message header property (JMS listener only)

If the listener map identifier is specified as a JMS message header property:

- The JMS message contains the JMS header property specified in the listener and the value points to a valid listener map, the listener map identifier is used to retrieve the listener map to process the XML.
- The JMS message contains the JMS header property specified in the listener and the value does not point to a valid listener map, the listener logs an error and no data is processed.
- The JMS message does not contain the JMS header property specified in the listener and there is a default listener map, the default listener map identifier is used to retrieve the listener map to process the XML.
- The JMS message does not contain the JMS header property specified in the listener and there is no default listener map, the listener logs an error and no data is processed.

### 2) XPath expression (all listeners, including JMS)

- The XML contains the tag specified in the listener xpath expression and the tag value points to a valid listener map ID. The listener map ID is used to retrieve the listener map to process the XML.
- The XML contains the tag specified in the listener xpath expression and the tag value does not point to a valid listener map ID, and there is a default listener map ID specified. The default listener map ID is used to retrieve the listener map to process the XML.
- The XML contains the tag specified in the listener xpath expression and the tag value does not point to a valid listener map id, and there is no default listener map id specified. The listener logs an error and no data is processed.

### 3) Default Listener Map Id

The default listener map ID is used when:

- There is no other listener map ID specified (no JMS message header properties nor xpath expression).
- There is a JMS message header property specified but the property does not exist in the JMS message being processed.

- The xpath expression does not resolve to a valid listener map ID.

## Using the default listener map

When providing only a default listener map on the listener XSD **Payload** tab, all messages received by the listener will be processed by the listener map name specified in the **Default Listener Map** parameter.

The screenshot shows a configuration window with three tabs: 'Parameters', 'Mapping Log', and 'Payload'. The 'Payload' tab is active. Under the 'Format' section, a dropdown menu is set to 'XSD'. Below this, there are three input fields: 'Map XPath:' with the value '/MachineMeasured/TransportNumber', 'Sequence XPath:' which is empty, and 'Termination:' with the value 'xxxx'. At the bottom, the 'Default Listener Map:' field is highlighted with a red circle and contains the text 'map-3694-v2'.

### Restriction

Using the default listener map implies that all XML messages received by the listener must comply with the XSD used to create the listener map.

There can only be one listener map per listener when using the default listener map.

## Using the xpath expression

For more flexibility and allowing the listener to process multiple XML formats, you can use the xpath expression. The xpath expression is used to locate the listener map ID in the XML.

The xpath expression must be the absolute path to the XML element containing the listener map ID value. It must start with the forward slash (/) character followed by the root element as in /root/tag1/tag2 for instance. The xpath expression cannot refer to an attribute.

The following examples show the XSD and corresponding XML:

### Example XSD:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="EmployeeRecord">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="employee" type="fullpersoninfo"/>
      <xs:element name="employeeId" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```

</xs:complexType>
</xs:element>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="fullpersoninfo">
  <xs:complexContent>
    <xs:extension base="personinfo">
      <xs:sequence>
        <xs:element name="address" type="xs:string"/>
        <xs:element name="city" type="xs:string"/>
        <xs:element name="country" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

</xs:schema>

```

**Example XML:**

```

|<?xml version="1.0" encoding="UTF-8"?>
|<EmployeeRecord>
|  <employee>
|    <firstname>James</firstname>
|    <lastname>Bond</lastname>
|    <address>MGM Studio</address>
|    <city>Hollywood</city>
|    <country>USA</country>
|  </employee>
|  <employeeId>ID-007</employeeId>
|</EmployeeRecord>

```

Using msgId as the listener map id, the xpath expression to specify is:  
/EmployeeRecord/msgId.

There must be a listener map named **ID-007** created for the current node.

## Using the JMS message header property

JMS listeners can have message header properties, and you can use this property to define the listener map ID. In this case, you must select the **Specify Map and Sequence Keys in Header** check box, and then type the name of the JMS message header property in the **Map Key** box.



The listener map ID is the value associated with the JMS message header property. The JMS text message body is expected to contain an XML.

Example:

A JMS Message is received with the following properties

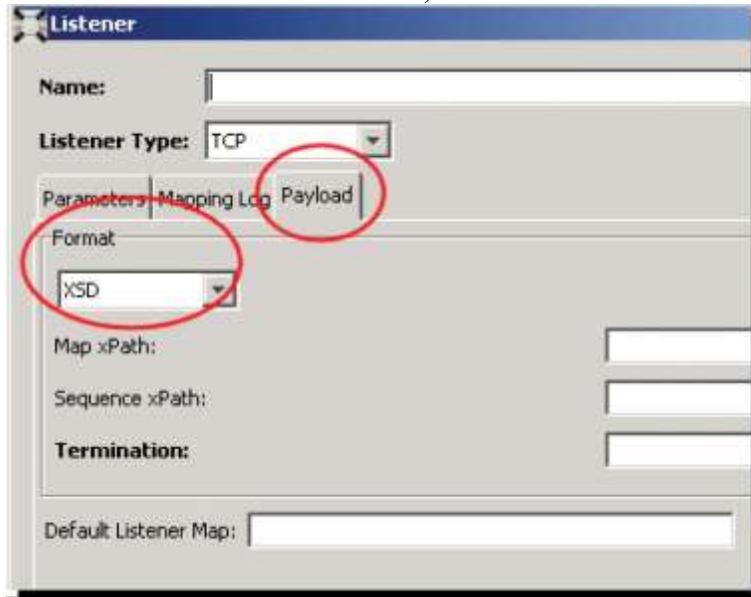
- customMsgId: ID-0009
- operationCode: CRK01
- destination: internal

and the listener **Map Key** parameter contains: customMsgId, there must be a listener map with the name of ID-0009 which contains the mapping based on the XSD matching the XML received as the JMS message body.

## Parameter descriptions

To add an XSD format to a listener, follow these steps:

1. From the Listener window, click the **Payload** tab.
2. Use the **Format** down-arrow, and then select **XSD**.



The **Payload** section changes to accommodate the **XSD** option. The **XSD Payload** tab has the following parameters.

### Map XPath

An XML absolute XPath expression, starting at the root element and pointing to an element tag whose value will be used to identify the listener map id.

## Sequence XPath

Currently not used.

## Default Listener Map

The name of the listener map id to use when a message is received by the listener.

## Termination

(TCP Listeners only)

Required. A character or string or control characters that indicate the end of the message (may or may not coincide with the end XML tag).

**Listener**

Name: ATCPListener

Listener Type: TCP

Parameters | Mapping Log | Payload

Format

XSD

Map XPath:

Sequence XPath:

Termination:

## Specify Map and Sequence Keys in Header

(JMS Listeners only)

Indicates if the listener map name should be located in the JMS message header properties.

**Listener**

Name: JMSListener

Listener Type: JMS

Parameters | Mapping Log | Payload

Format

XSD

Map XPath:

Sequence XPath:

Specify Map and Sequence keys in header

Map Key: Sequence Key:

Default Listener Map:

## Map Key

(JMS Listeners only)

Required when **Specify Map and Sequence Keys in Header** check box is selected.

The JMS message header property whose value will be used to identify the listener map id.

## Sequence Key

(JMS Listeners only)

Currently not used.

Related topics

[Creating a TCP listener](#)

[Creating a listener map with an XSD payload](#)

# IIoTA industrial IoT Platform: Listener maps

A Listener Map defines the data format of the data received from the enterprise application and, when there is response data, the format of the response data sent from the trigger back to the enterprise application.

## Overview

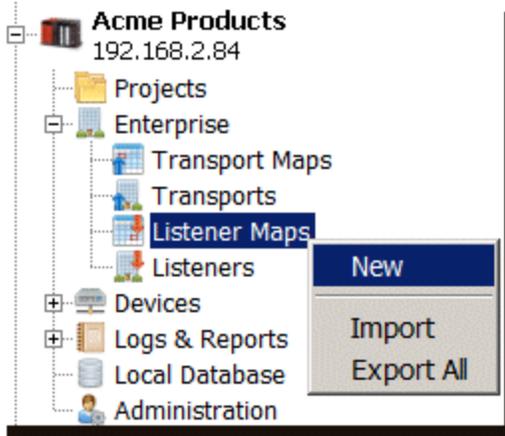
A listener map is used by the listener for the following actions:

- Maps the input request received by the listener to an internal request that is the triggering condition for the listener trigger.
- Maps the reply coming back from the trigger to an XML, ASCII, or map message response that is then put on a specified reply queue.

## Procedures

You can create a listener map using the **Listener Maps** feature of the Workbench. Follow these steps:

1. From the Workbench left pane, expand the node that you want to add the listener map to.
2. Expand **Enterprise**, right-click the **Listener Maps** icon to display its pop-up menu, and then click **New**.



The Listener Map window appears.

The 'Listener Map' window has the following fields:
 

- Name:** [Empty text box]
- Listener Type:** [All ▼]
- Listener Name:** [Empty dropdown menu]

 Below these fields are two tabs: 'Input' and 'Output'. Under the 'Input' tab, there is a section titled 'To Trigger' containing a table with the following columns:

Name	Logical	Count	Length

3. In the **Name** box, type a unique name for the listener map. The name can be up to 64 characters in length and can include letters, numbers, and the underscore character. Spaces are allowed.
4. Use the **Listener Type** down arrow to filter the listeners displayed by type in the **Listener Name** list. For example, the list might include TCP, JMS, XSD, and so forth.
5. Click the **Listener Name** down arrow to display a list of predefined listeners, and then select the appropriate listener.

The 'Listener Map' window is now populated:
 

- Name:** My Listener Map
- Listener Type:** All ▼
- Listener Name:** AcmeProducts111 ▼

 The 'Input' tab is selected. The 'To Trigger' table now contains the following data:

Name	Logical	Count	Length
AcmeProducts111			
AcmeProductsTCPXSD			
TCPListenerLocalHost			

Consider the following: A listener map can be associated with only one listener. However, a listener can be re-used any number of times by different listener maps. For example, ListenerMap1 can use the AcmeProducts111 listener; likewise, ListenerMap2 can also use AcmeProducts111.

The top of the Listener Map window provides one or two tabs:

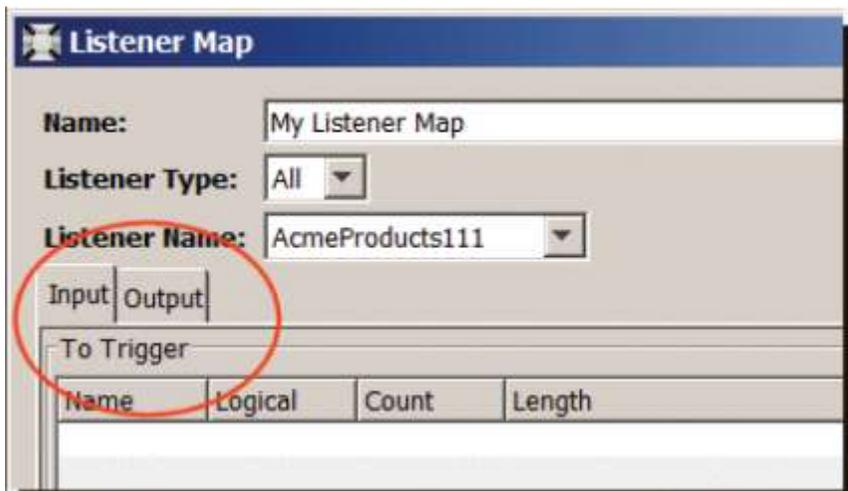
- An **Input** tab to define data coming from the enterprise request to mapping elements that will be sent to the listener trigger. For more information, see Using the Input tab below
- An **Output** tab to define a reply message based on mapping elements that are sent from the trigger execution after completion of all the actions in the trigger's action list. For more information, see Using the Output tab below.

### Using the Input tab

The **Input** tab will consist of map variables. The values for the variables will come from the message received from the enterprise system at run time. These values will be passed to the trigger that is associated with the listener map.

To add a map variable to the **Input** tab:

1. From the **Input** tab, under **To Trigger**, click **Add**.



The New Item window appears.

2. Type a name for the variable, select a type, and the length (for string types). In the **Count** box, type a value that specifies the dimension of the map variable such as a scalar or an array.
3. Click **Add**.

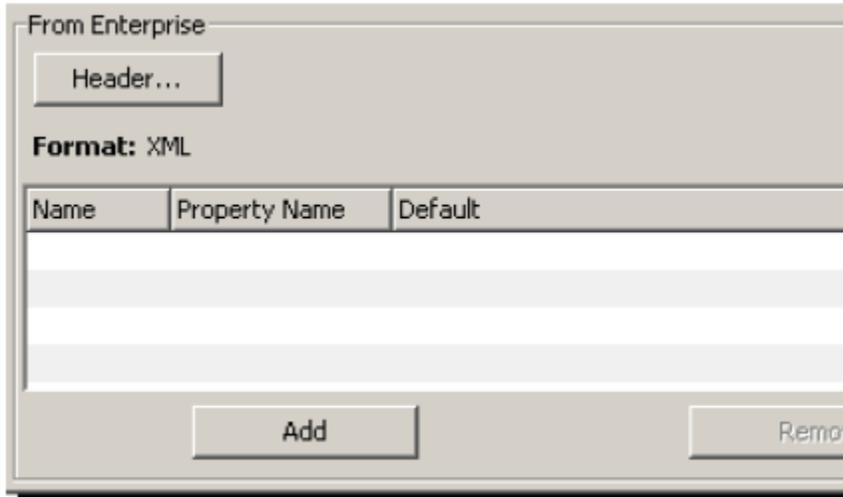
The map variable is added under the **To Trigger** section.

4. Repeat steps 1 through 3 as necessary.

The completed **Input** tab might look like this:

Name	Logical	Count	Length
BinNumber	STRING	12	8
PriorLevel	INT2	1	n/a
BinLevel	STRING	12	8

The next step is to associate the map variables you just created with a map variable in the **From Enterprise** section. You must create the map variables for the **From Enterprise** section.



- Under the **From Enterprise** section, click **Add**. Notice the **Format** for this example is XML.

The New Item window appears.



- Click the **Name** down-arrow to display a list of the map variables from the **To Trigger** section, and then select a variable. For this example, **BinNumber**.
- In the **Property** box, type a name to be used as a source for associating the data type string with the map variable **BinNumber**. For this example, type *PlantBinLocationId*.

The example is based on a message received from an enterprise application with the following XML <Item> element identified with a name attribute value of **PlantBinLocationId**.

```
<Item name="PlantBinLocationId" ><Data>1254</Data></Item>
```

- In the **Default Value** box, type a value to be used in the notification map that is sent to the trigger when a corresponding <Item> element is not found in the listener request. For this example, type 100.

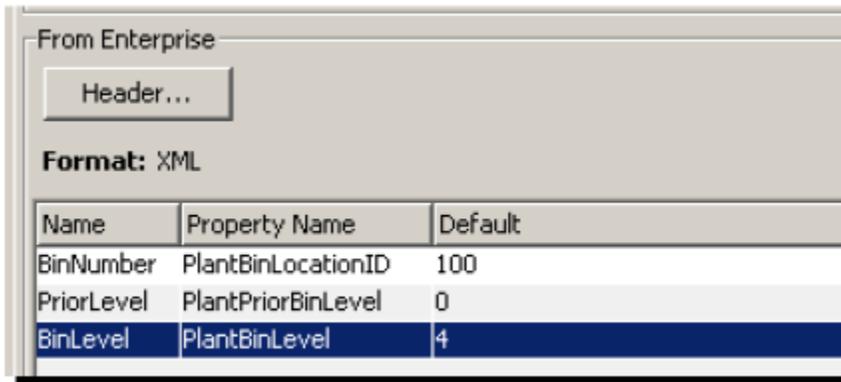
**Default behavior:** If the Item XML tagdefault value<Item> element is present in the XML but the <Data> element is missing, then the default value is used. However, a <Data> element with no value (for example <Data></Data>) will not cause the default value to be used but will generate an error if the data type is numeric and will write null to a variable of type String.

- Click **Add**.

The information is added under the **From Enterprise** section.

- Repeat the steps 7 through 10 for the remaining map variables.

The completed **From Enterprise** section might look like this:



The screenshot shows a window titled "From Enterprise" with a "Header..." button and a "Format: XML" label. Below is a table with three columns: Name, Property Name, and Default.

Name	Property Name	Default
BinNumber	PlantBinLocationID	100
PriorLevel	PlantPriorBinLevel	0
BinLevel	PlantBinLevel	4

The example shows the data mapping for the GetBinInformation listener map. A trigger defined to execute upon receipt of the GetBinInformation listener map will receive as input three variables, a String named **BinNumber**, an INT2 named **PriorLevel**, and a second String named **BinLevel**. The values for these three variables will come from information found in a message retrieved from an enterprise system. The message will contain three XML <Item> elements, with name attribute values of **PlantBinLocationId**, **PlantBinLevel**, and **PlantPriorBinLevel**.

### Using the Output tab

The **Output** tab will also consist of map variables. The trigger associated with a listener map will have write access to the map variables defined in the From Trigger section **From Trigger**

section. The trigger can pass information back to the enterprise system by setting the values in these map variables.

To add a map variable to the **Output** tab:

1. From the **Output** tab, under **From Trigger**, click **Add**.

The New Item window appears.

2. Type a name for the variable, select a type, and the length (for string types). Currently, the **Count** parameter is limited to one.
3. Click **Add**.

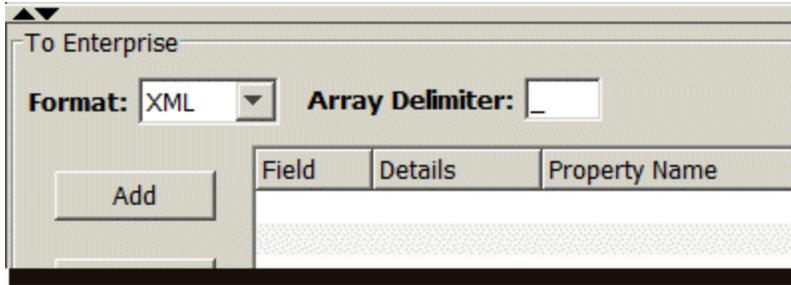
The map variable is added under the **From Trigger** section.

4. Repeat steps 1 through 3 as appropriate.

The completed **Output** tab might look like this:

Name	Logical	Count	Length
CurrentBinLevel	INT2	1	n/a
CurrentBinLocation	INT2	1	n/a

The last step is to assign the map variables created in the **From Trigger** section to a Listener Reply message that will be written to the enterprise application. You do this using the **To Enterprise** section.



The message will be written to the enterprise system upon completion of the trigger associated with the listener map. The message format can be either ASCII delimited or XML. This section will concentrate on an XML message format.

5. Click the **Format** down arrow, and then select **XML**.

When **XML** is selected as the format, the payload is UTF-8 encoded. When **ASCII** is selected as the format, then the payload is ASCII encoded.

### TCP listener - ASCII payload

If you are using a TCP listener that does not accept an ASCII payload, but you want an ASCII outbound payload, you must manually add the character termination sequence. From the **To Enterprise** section of the listener map, make sure ASCII is the format, click the **Add** button, and then add the character termination sequence.

### TCP listener - XML payload

If you configure a TCP listener to accept only XML payloads, then you should specify XML as the outbound payload on all listener maps defined for this Listener. If you choose to use an ASCII outbound payload in this situation, then you must append a character termination sequence (End Of Message delimiter) as the last parameter in the outbound payload map definition.

## Using the To Enterprise section

Under the **To Enterprise** section, click **Add**.

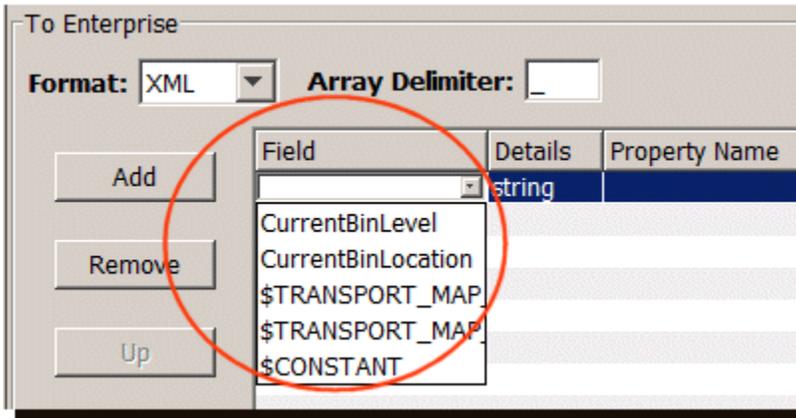
When you click **Add**, you create a new <Item> element for the Listener Reply XML message that will be generated by the Transaction Server. For each <Item> element, a map variable name from the **From Trigger** list will be selected. The value of the map variable will be placed in a <Data> element within the <Item> element. A **Property Name** is then specified and becomes the name attribute of the <Item> element.

For example, to map the **From Trigger** map variable named *CurrentBinLevel* to an XML <Item> element named *PlantBinLevel*:

```
<Item name="PlantBinLevel" > <Data>1254</Data> </Item>
```

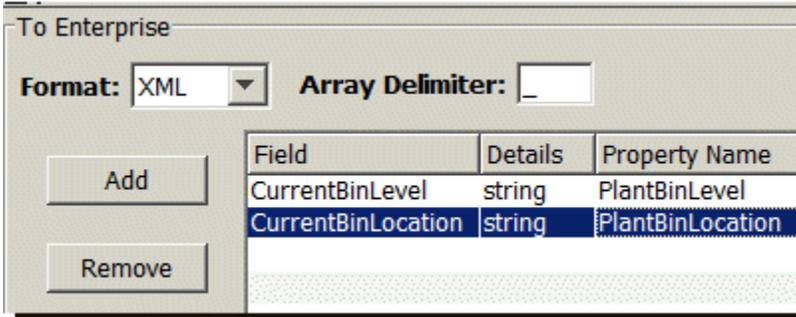
For more information, see XML listener commands.

The first row in the table becomes active.



1. Under **Field**, click the column to display a drop-down list, and then select the appropriate map variable (for this example, **CurrentBinLevel**).  
The **Details** column can be ignored as all data written in an XML and ASCII format is represented as a string.
2. Under **Property Name**, click to activate the text box, and then type a name. For this example, *PlantBinLevel*. The map variable named **CurrentBinLevel** is now associated with this property name.
3. Repeat the steps as necessary to associate all the map variables with a data type.

The completed **To Enterprise** section might look like this:



The example provides the data mapping for the Listener Reply message that will be sent once the trigger associated with the **GetBinInformation** listener map completes its execution. Upon trigger completion, the values that were set in the CurrentBinLevel and CurrentBinLocation map variables by the trigger, will be packaged into an XML message. The message will contain two <Item> elements, named PlantBinLevel and PlantBinLocation. The message will be written to the enterprise application identified in the listener (that was specified in the listener map definition).

### Saving the listener map

1. To test the listener map, click **Validate**.
2. If no errors are received, click **Save**. The new listener map is saved to the node and added to the **Listener Maps** tab. For more information, see Using the Listener Maps tab.

# IIoTA industrial IoT Platform: Defining JMS Header attributes

In the listener map defined for a JMS listener, you can specify how JMS Header properties can be handled on the request and the reply. The protocol specific context information that can be read from or updated in the message header includes:

- MessageID
- CorrelationID
- Priority, Persistence, Expiry
- Application specific header properties that may be sent by the enterprise application or required by the enterprise application receiving the message.

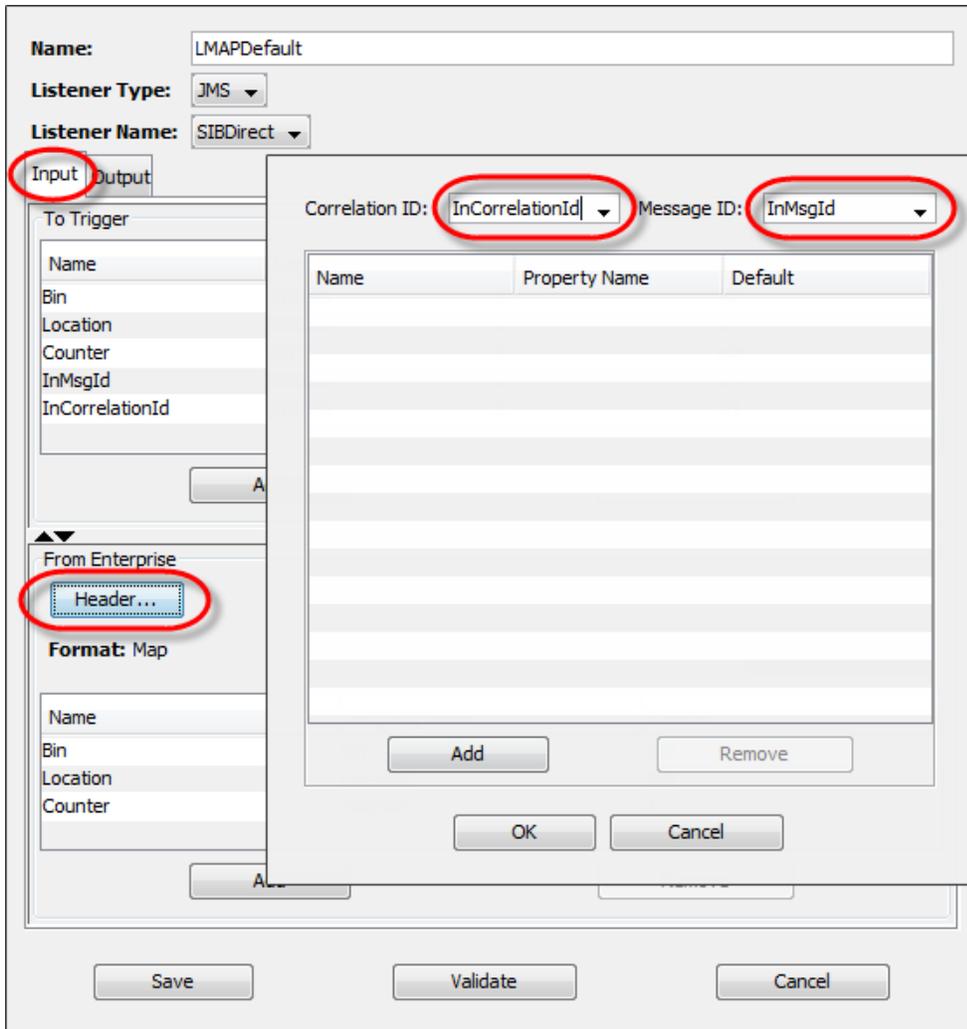
Any JMS Header information that needs to be mapped from a JMS Message received by a listener needs to be specified on the listener map.

Any JMS Header information that needs to be specified on the JMS Message that is sent as a reply can be specified either in the listener map or the listener. You can specify extended attributes in the listener map to have more control over the JMS header content of messages sent to the endpoint enterprise application. The header attribute configuration in the map overrides any configuration specified on the listener.

## Processing of JMS Header properties from the request

To access the JMS header configuration for the request, edit the listener map panel, select the Input tab (default), and then select the **Header** button. This brings up a popup panel as shown.

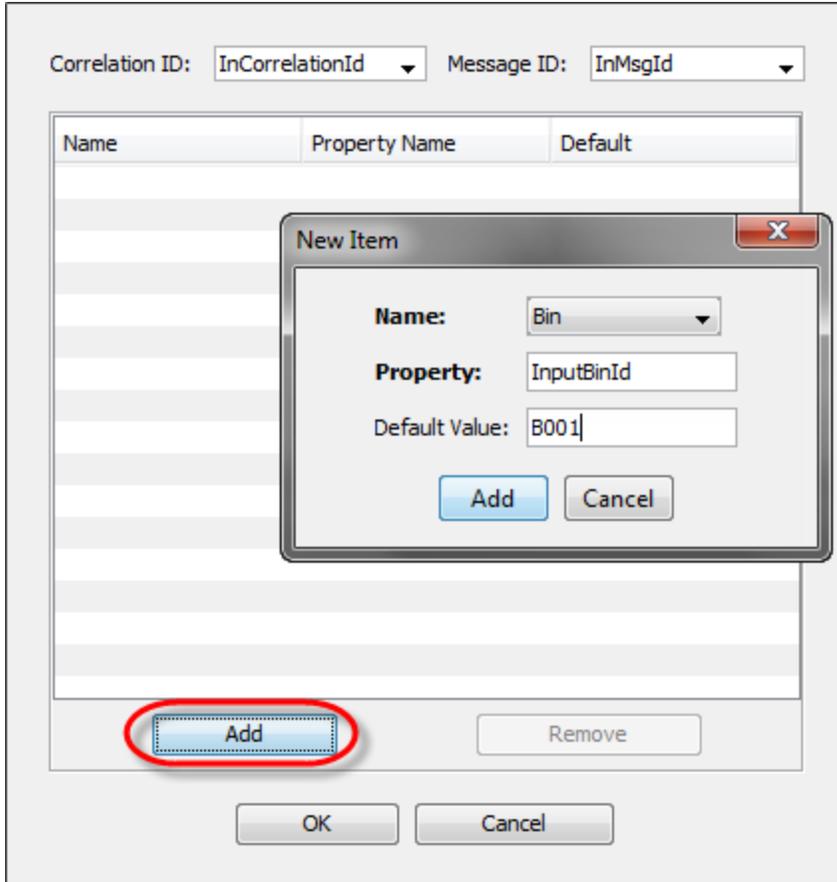
Using this panel, you can associate listener map variables defined in the **Input** tab with the MessageID, CorrelationID and application specific header property values that are received in the JMS Message.



Parameter	Description
<b>Correlation ID</b>	Select a variable from the <b>Input</b> tab that will be set to the CorrelationId of the JMS message received by the listener. The default is to leave this field unspecified.
<b>Message ID</b>	Select a variable from the <b>Input</b> tab that will be set to the MessageId of the JMS message received by the listener. The default is to leave this field unspecified.

### Application Defined Properties

Use this section of the Header panel if you want to map data from user defined JMS Header properties to variables in the **Input** tab of the listener map. You specify the value one property at a time by selecting the **Add** button and entering the property information in the popup panel.



Parameter	Description
<b>Name</b>	Specify the <b>Input</b> tab variable that will be set to the JMS header property value.
<b>Property</b>	Specify the key value of the enterprise application defined JMS Header property. This JMS Header property is expected to be present in the JMS message received by the listener and processed by this listener map.
<b>Default Value</b>	The value specified in this field will be used to set the <b>Input</b> tab variable in case the JMS Header property identified by <b>Property</b> is not found in the JMS message received by the listener.

**Specifying JMS Header properties on the reply**

To access the JMS Header configuration for the response, edit the listener map panel, select the **Output** tab, and then select the **Header** button. This brings up a popup panel as shown. Using this panel, you can specify the following attributes and metadata of the reply JMS Message:

- How listener map variables defined in the **Output** tab can be mapped to CorrelationID and application specific header property values

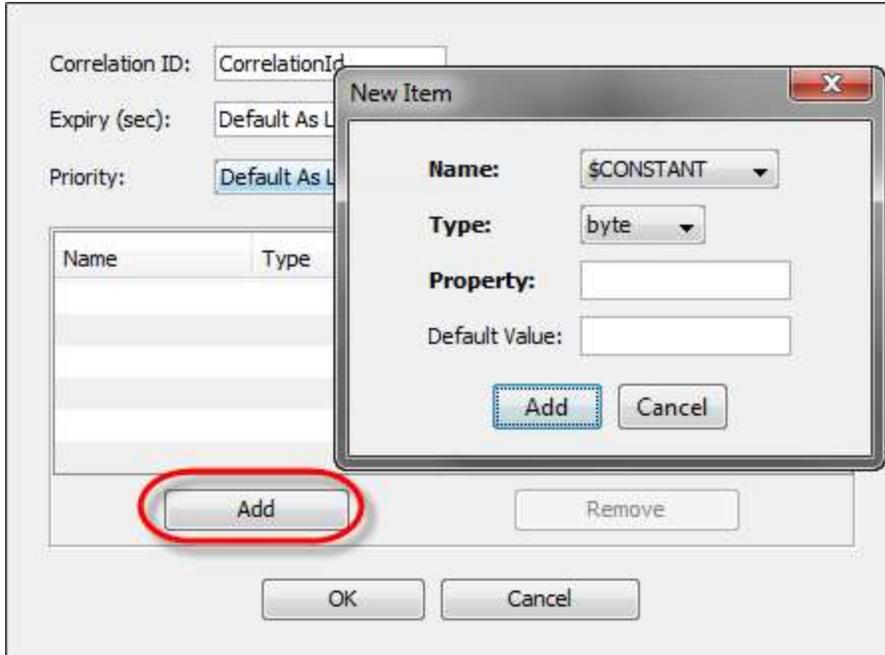
- Override Expiry, Persistence, Priority settings that will override what is specified on the listener
- Specify an alternate JMS ReplyTo queue that will override what is specified on the listener.

Parameter	Description
<b>Correlation ID</b>	Select a variable from the <b>Output</b> tab that will be set to the CorrelationId of the JMS Message reply message sent by the listener. The default is to leave this field unspecified.

<b>Expiry</b>	This is a combination field and list box. If left unspecified, as indicated by <b>Default As Listener</b> , then the JMS reply created by this listener map will be initialized with the expiration specified on the listener. Otherwise enter a numerical value that represents the time to live for the message in seconds. You can select <b>Unlimited</b> from the list to indicate that the message has an infinite time-to-live. The default is <b>Default As Listener</b> .
<b>Persistence</b>	This pick list can be used to specify the JMS delivery mode (persistence) of the JMS message reply sent by the listener. If left unspecified, as indicated by <b>Default As Listener</b> , the JMS reply message created by this map will be initialized with the <b>Persistence</b> specified on the listener. Choose from:  <b>Persistent</b> <b>Non-Persistent</b> <b>Default As Listener</b>
<b>Priority</b>	This pick list specifies the priority of the JMS message reply defined by this listener map. Choose a value from 0 (Lowest) through 9 (Highest). If left unspecified, as indicated by <b>Default As Listener</b> , then this JMS message will have a priority as specified on the listener.
<b>Reply Queue</b>	You can specify a replyTo Queue to be set in the JMS message reply defined by this listener map. If left unspecified, the replyTo Queue will not be set in the JMS reply.

### Application Defined Properties

Use this section of the Extended Attributes panel if your enterprise application expects user defined JMS Header properties to be added to the JMS reply message defined by this listener map. You specify the value one property at a time by selecting the **Add** button and entering the property information in the popup panel.



Parameter	Description
<b>Name</b>	This field allows you to specify the content of the JMS header property. The options are:  An input variable whose value will be used to set the value of the JMS header property \$CONSTANT if you will be entering a constant value.
<b>Property</b>	Specify the key value of the JMS Header property in this field. This JMS Header property key will be present in the JMS message defined by this listener map with a corresponding value as specified in the <b>Name</b> parameter.
<b>Type</b>	This pick list specifies the data type of the JMS Header property. The data type is typically dictated by the target enterprise application consuming this JMS Message. The type information is used to call the appropriate JMS Header property setter when adding the JMS Header property to the JMS Message defined by this listener map.
<b>Default Value</b>	Enter a text value in this field if you selected \$CONSTANT in the <b>Name</b> field. The value entered here will be the value of the JMS Header property added to the JMS Message defined by this listener map. Enter a value appropriate for the type specified. If an error occurs when setting the JMS Header property, the listener will fail the reply.

**Related topics**

Extended attributes for the JMS listener

# IIoTA industrial IoT Platform: Creating a listener map with an XSD payload

This page describes the following:

- How to use XML Schema Definition (XSD) files to create a listener map. A sample XSD file is provided to help you get started with creating a listener map with an XSD payload.
- How to create a trigger that uses a listener event and writes the XML tag values to variables.

## Overview

The XSD listener map is used when XML messages received by a listener need to be parsed using an XML Schema.

The XSD payload type is available to be selected only when the corresponding listener is defined to accept XSD. For more information, see Listener XSD format.

## XSD limitations

Make sure you review XSD limitations before you begin creating a listener map.

## Assumptions

The following is assumed:

- You are familiar with the XML Schema Definition language.
- You know how to use the Workbench and create triggers.
- There is a listener with an XSD payload defined on the node. For more information, see Listener XSD format
- If the XML Schema has import statements, then all referenced XSDs are available to be accessed from the Workbench.

## Downloading the sample XSD file

Download this [sample file](#), and note the folder where it is stored.

## Sample file

The file will look like this:

```
<?xml version="1.0" encoding="utf-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="urn:books"
  xmlns:bks="urn:books">

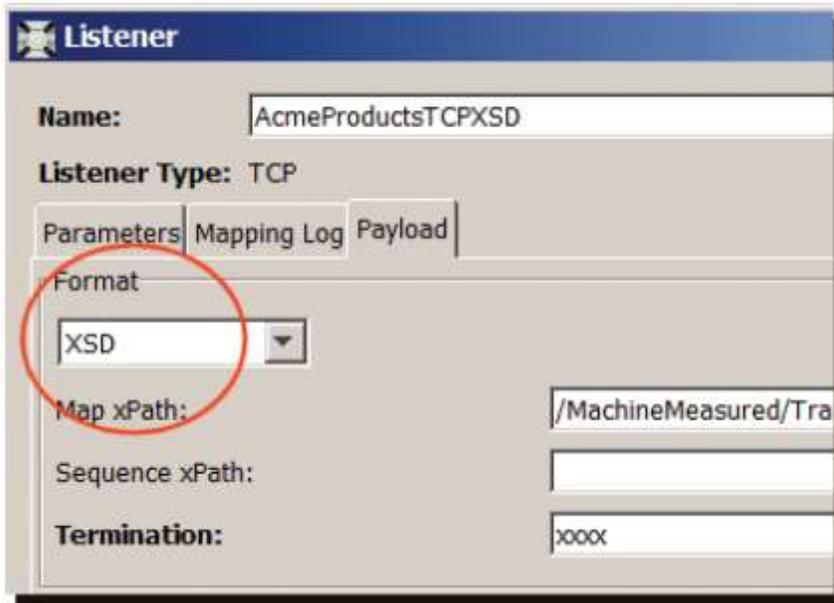
  <xsd:element name="books" type="bks:BooksForm"/>

  <xsd:complexType name="BooksForm">
    <xsd:sequence>
      <xsd:element name="book"
        type="bks:BookForm"
        minOccurs="0"
        maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="BookForm">
    <xsd:sequence>
      <xsd:element name="author" type="xsd:string"/>
      <xsd:element name="title" type="xsd:string"/>
      <xsd:element name="genre" type="xsd:string"/>
      <xsd:element name="price" type="xsd:float"/>
      <xsd:element name="pub_date" type="xsd:date"/>
      <xsd:element name="review" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

## Example listener with XSD payload

The following shows a completed Listener window with TCP as the listener type and XSD as the payload format.



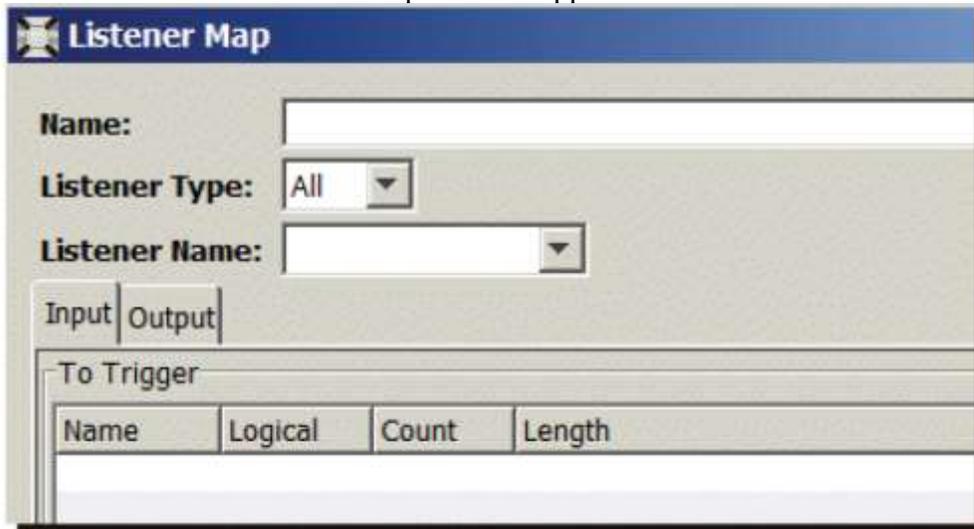
Once the listener is saved and available on the node, you can create the listener map.

## Rendering the XSD in the listener map

The following will walk you through creating a listener map to process XML data that will be written to variables. The sample XSD file will be referenced within the instructions.

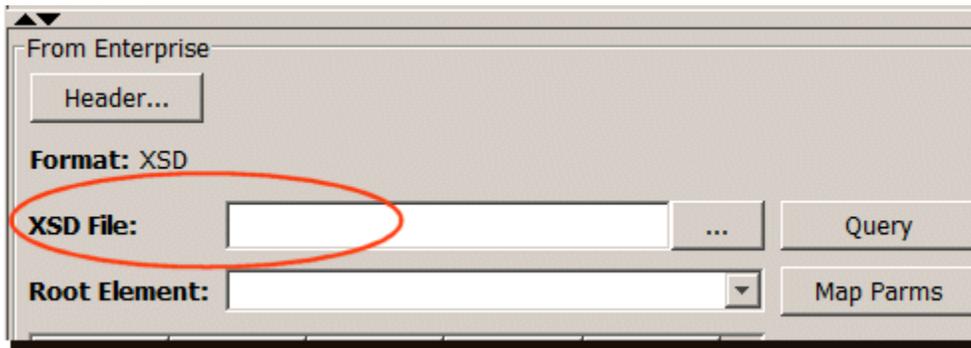
1. From the Workbench left pane, expand the node that you want to add the listener map to.
2. Expand **Enterprise**, right-click the **Listener Maps** icon to display its pop-up menu, and then click **New**.

The default Listener Map window appears.



3. In the **Name** field, type a unique name for the listener map.
4. Use the **Listener Type** down arrow to filter the listeners displayed by type in the **Listener Name** list. For example, the list might include TCP, JMS, or XSD.
5. Select the **Listener Name** down arrow to display a list of previously defined listeners, and then select the appropriate listener that supports the XSD payload.

The **From Enterprise** section of the listener map window changes to accommodate an XSD format.

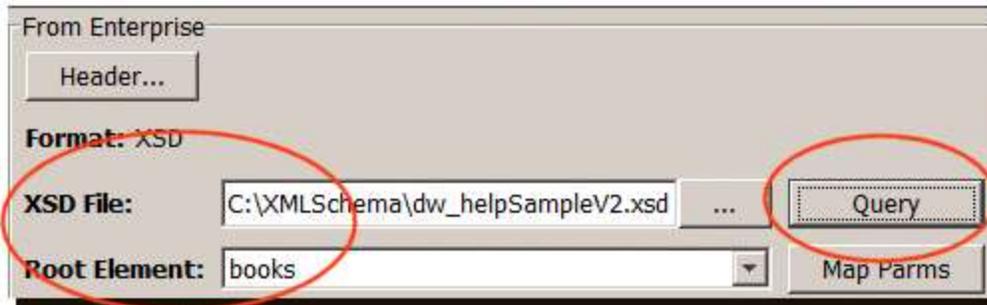


### Step 1: Specifying the location of the XSD file

Consider the following when specifying a value for the **XSD File** text field:

- If the XSD is located on a web server, type a URL starting with http(s) in the **XSD File** field.

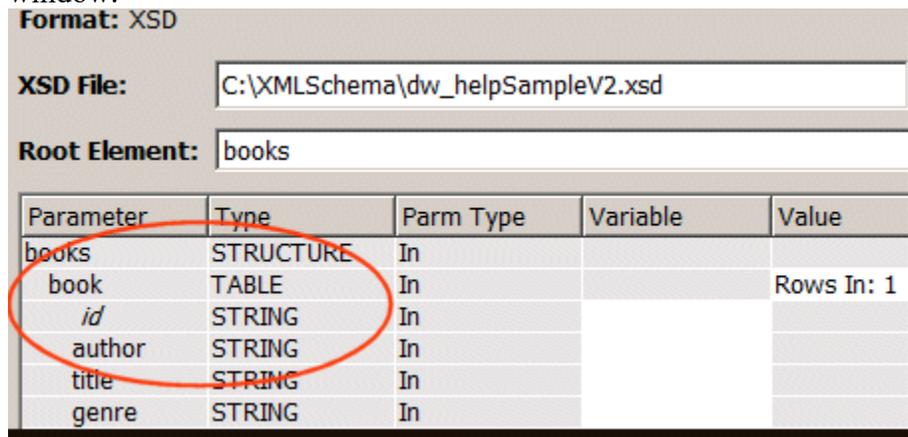
- If the XSD file is available on the local file system and you know the location and name of the file, you can type the full path and name into the **XSD File** field. For example, C:/path/xsdfilename.xsd.
  - If you do not know the exact location of the file, select the browse button. This brings up a file chooser dialog to help you navigate to the XSD file. Locate your XSD file and select it. The **XSD File** field is populated with the name.
- Once you have added a value in the **XSD File** field, select the **Query** button.



Assuming the XSD is valid and contains at least one element, the **Root Element** parameter becomes populated with elements found in the XSD file.

## Step 2: Displaying the XSD element structure

Use the **Root Element** down-arrow, and then select the appropriate element from the list. The XSD structure is displayed at the bottom of the From Enterprise section of the listener map window.



The XSD structure appears as follows:

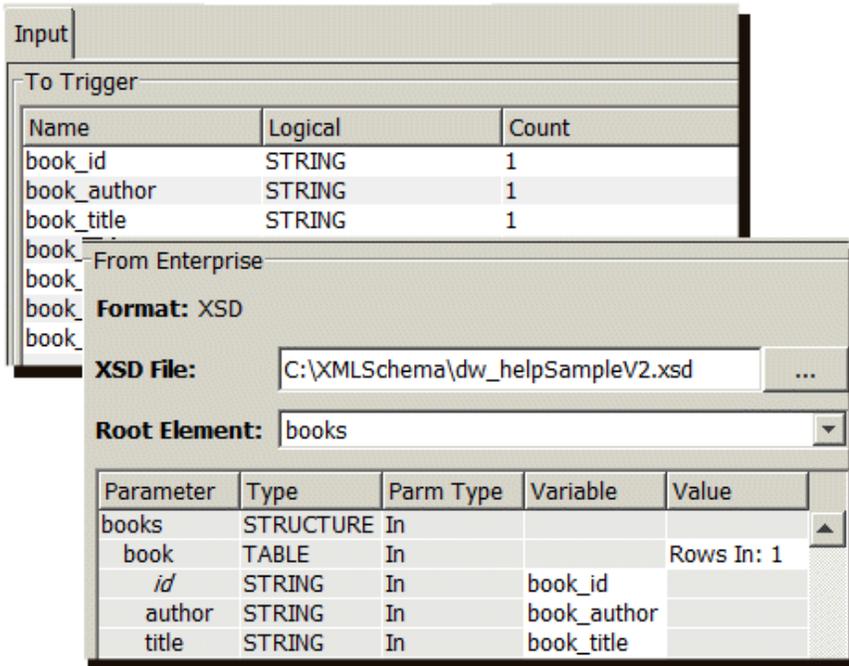
- XSD elements nested in other XSD elements, such as elements belonging to complex types, appear indented below their parent element.
- Attributes appear indented under the element they belong to and are *italicized*.
- Arrays are identified by the **Table** keyword in the **Type** column.

Array handling

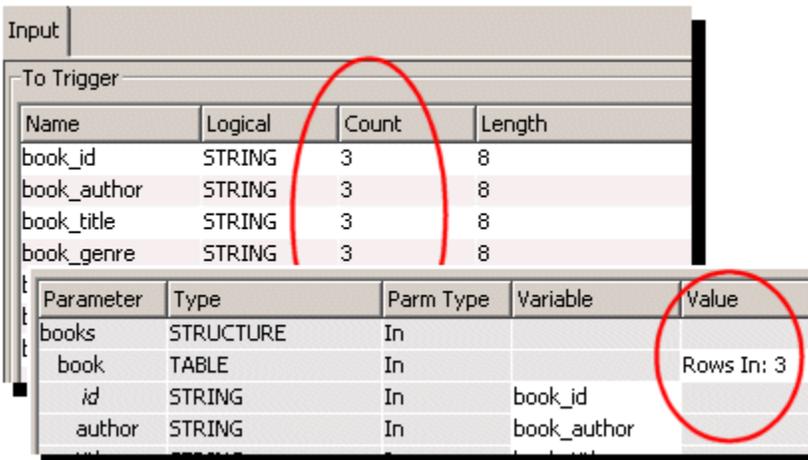
In the sample XSD, the **book** element has maxOccurs="unbounded" which indicates that there could be an array of books. Therefore, the **book** element is identified as a **TABLE** type in the 'From Enterprise' section. The **Rows In:** field can be used to specify the depth of the array. This will represent the maximum number of rows that can be processed and sent to the listener trigger.

### Step 3: Mapping the XML tags to logical variables

To automatically generate logical variables with the correct count (in the case of arrays) and data type, select the **Map Parm** button.



The map variables are automatically added to the listener map **Input** tab and the XSD element structure **Variable** column. It is recommended that you use the **Map Parm** button to ensure the best mappings between the XML Schema variable types and data types. Also notice that all elements defining a book have their **Count** set to the number specified in the **Rows In** parameter.



For this example, assuming a maximum of 3 books, change the value for the **Rows In** field to 3, and then select **Map Parm**s. All the input variables defining a book will have their **Count** set to the number specified in the **Rows In** parameter. The array size specified in the **Rows In** parameter should be based on the maximum number of rows that the trigger developer wants to process in the trigger.

If mapping an XML schema tag of type *time*, manually adjust the variable associated with the XML element to **STRING** instead of **TIMESTAMP**, otherwise the mapping will cause an error when the listener processes the XML.

Alternatively, logical variables can be created manually and then mapped to XML tags in the 'From Enterprise' section.

## Parameter descriptions

The following provides additional details for the columns at the bottom of the **From Enterprise** section of the listener map.

Column name	Description
<b>Parameter</b>	Required. Automatically populated after selecting a root element (from the <b>Root Element</b> parameter). XSD elements can be identified as <b>Structures</b> or <b>Tables</b> where they are complex. When an element is identified as <b>Table</b> by the Transaction Server, it is in fact an array, where you can specify the maximum size of the array (see <b>Value</b> description below). Elements belonging to a structure or table are indented below their parent tag.
<b>Type</b>	Required. The <b>Type</b> column shows the data type of the parameter. Unless the XSD element is complex, the type is the XSD type.
<b>Parm Type</b>	Required. Refers to incoming or outgoing elements. For this listener map, the <b>Parm Type</b> is always <b>In</b> .
<b>Variable</b>	Optional. This is the name of the map variables associated with the XML tag. The map variable are selected from the list of variables defined in the Listener Map window <b>Input</b> tab. You can manually add each input variable when you select the <b>Variable</b> column, or you can select <b>Map Parm</b> s to automatically

generate one or more variables.

Parameter	Type	Parm Type	Variable	Value
books	STRUCTURE	In		
book	TABLE	In		Rows In: 1
<i>id</i>	STRING	In	book_id	
author	STRING	In	book_author	
title	STRING	In	book_title	
genre	STRING	In	book_genre	
price	FLOAT	In	book_price	

**Value / Rows In**

Optional. An input field appears only when the parameter type is set to **TABLE** when XML Schema elements are defined with `maxOccurs="unbounded"`.

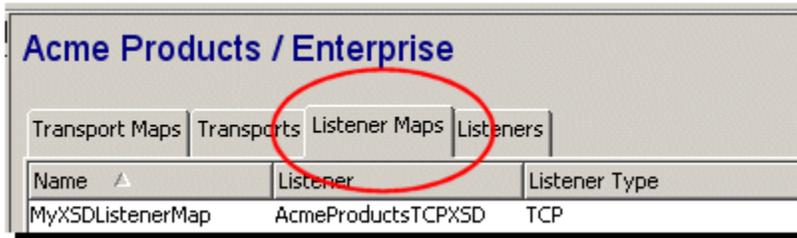
Parameter	Type	Parm Type	Variable	Value
books	STRUCTURE	In		
book	TABLE	In		Rows In: 1
<i>id</i>	STRING	In		
author	STRING	In		

The default value is 1 and has the same effect as if it were defined as a structure.

The value must be a positive integer.

If you select the **Map Params** button to generate the map variables, the map variables associated with table elements will automatically have their count updated with the value specified in **Rows In**.

When you complete the listener map, click **Save**.  
The name of the listener map is added to the **Listener Maps** tab.



The next step is to create a trigger that can write the XML tag values to variables.

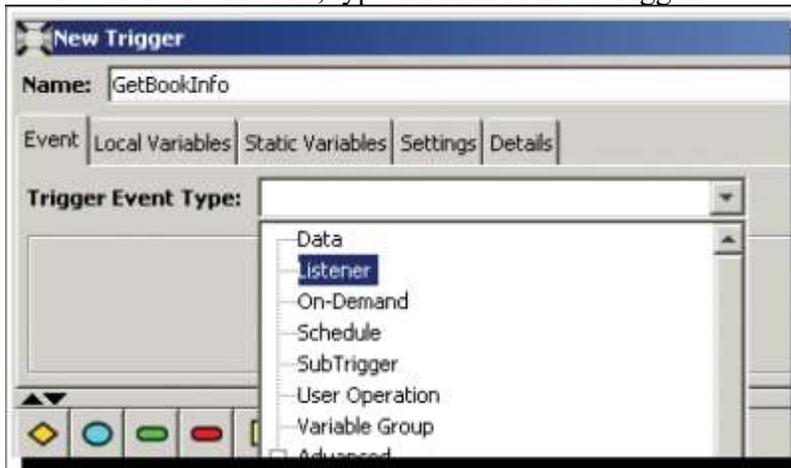
## Creating a trigger that uses a listener event

This section will describe how to create a trigger that uses a listener event. A listener event provides the means to generate events based upon the receipt of a message from an enterprise system.

Follow these steps:

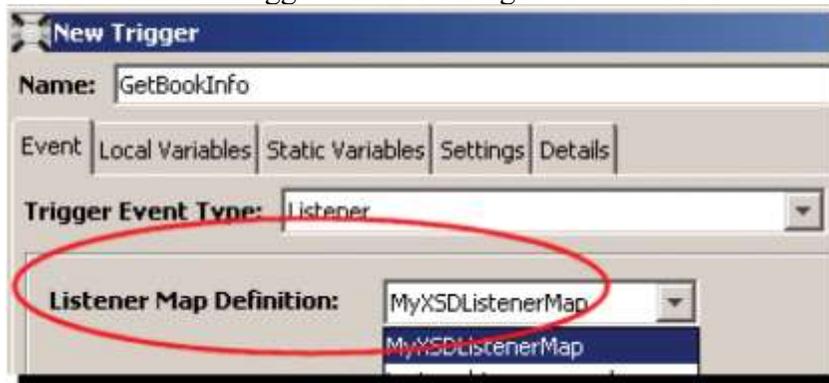
1. From the bottom of the appropriate project tab, click **New**.  
For this example the New Trigger window opens in Canvas Editor mode.

2. In the **Name** box, type the name for the trigger.

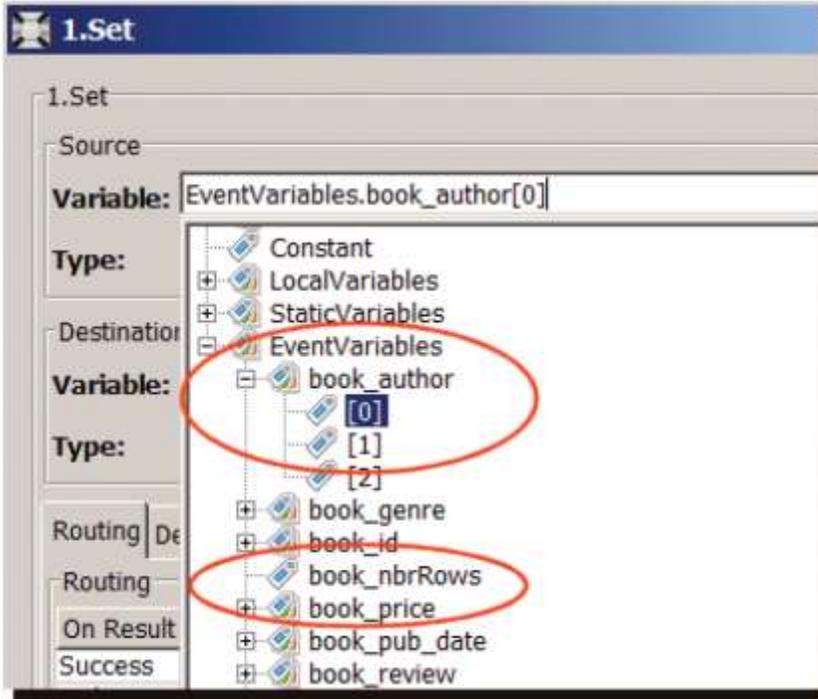


3. Use the **Trigger Event Type** down-arrow, and then select **Listener**.

The New Trigger window changes to accommodate the listener event.



4. From the **Listener Map Definition** drop-down list, select the listener map you just created.
- The next step is to add an action.
5. From the left pane of the New Trigger window, locate the **Set** action, and then drag the action on to the canvas area.
  6. Double-click the **Set** action to display its parameters.



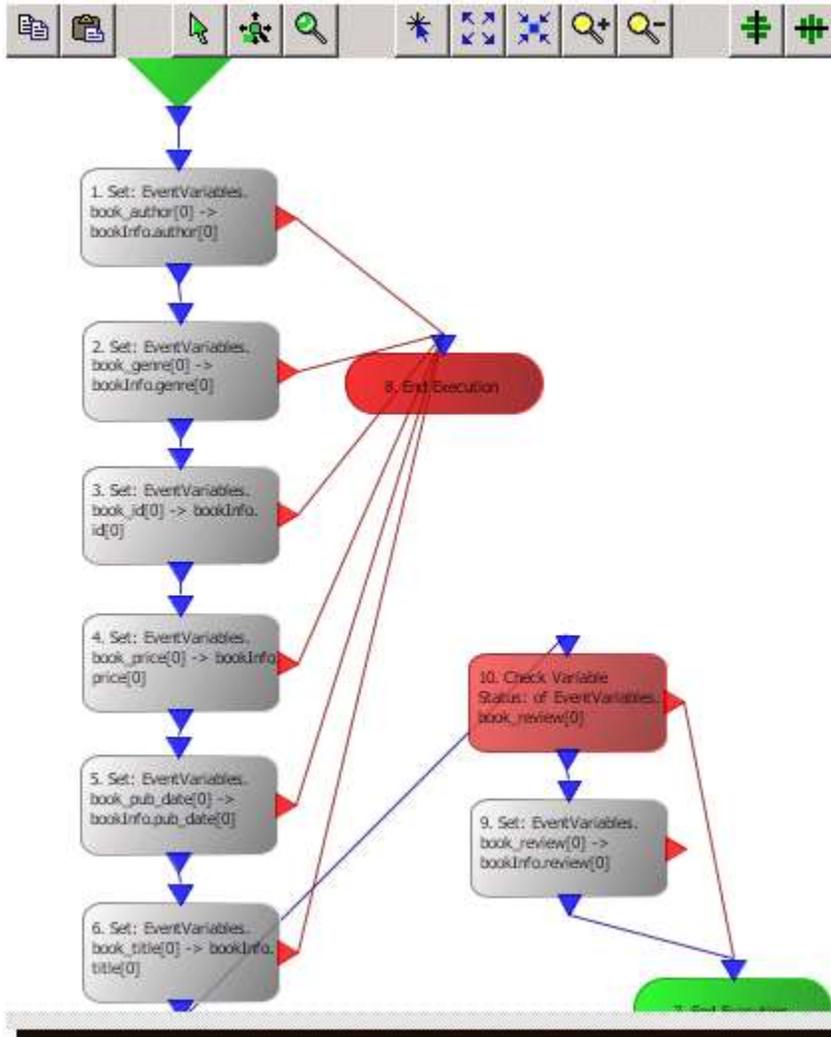
7. Using the **Set** action, expand the variables under **Event Variables**. Since **books** was defined as an array of 3 elements, the event variables are displayed as array of 3 elements.

Notice the Event Variables **book\_nbrRows** which will contain the exact number of elements received in the XML by the Listener.

8. Complete the **Set** action as follows for the **book** tags:



- Repeat the **Set** action as needed for the other event variables: `book_genre`, `book_id`, `book price`, `book pub date`, `book title`, and `book review`.



Assuming the XML received by the listener contains the following information:

```
<?xml version="1.0" encoding="UTF-8"?>
<books xmlns="urn:books" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <book id="F-132" >
    <author>author</author>
    <title>Future</title>
    <genre>fiction</genre>
    <price>23.99</price>
    <pub_date>2009-07-02</pub_date>
    <review> None </review>
  </book>
  <book id="N-335" >
    <author>writer</author>
    <title>Romance</title>
    <genre>novel</genre>
    <price>11.78</price>
    <pub_date>2003-05-05</pub_date>
    <review> None </review>
  </book>
  <book id="D-321" >
    <author>scribe</author>
    <title>Dream</title>
    <genre>fantasy</genre>
    <price>32.99</price>
    <pub_date>2010-08-02</pub_date>
    <review> None </review>
  </book>
</books>
```

Then the values written to the **bookInfo** variables are:

Devices			
Variables			
Data Mapping			
Variable Groups			
Name	Type	Value	
bookInfo	Global Variables		
id	STRING(8)[5]		
[0]	STRING(8)	F-132	
[1]	STRING(8)	N-335	
[2]	STRING(8)	D-321	
[3]	STRING(8)		
[4]	STRING(8)		
author	STRING(8)[5]		
[0]	STRING(8)	author	
[1]	STRING(8)	writer	
[2]	STRING(8)	scribe	
[3]	STRING(8)		
[4]	STRING(8)		
title	STRING(8)[5]		
[0]	STRING(8)	Future	
[1]	STRING(8)	Romance	
[2]	STRING(8)	Dream	
[3]	STRING(8)		
[4]	STRING(8)		
genre	STRING(8)[5]		
[0]	STRING(8)	fiction	
[1]	STRING(8)	novel	
[2]	STRING(8)	fantasy	
[3]	STRING(8)		
[4]	STRING(8)		
price	FLOAT4[5]		
[0]	FLOAT4	23.99	
[1]	FLOAT4	11.78	
[2]	FLOAT4	32.99	
[3]	FLOAT4		
[4]	FLOAT4		
pub_date	TIMESTAMP[5]		
[0]	TIMESTAMP	Thu Jul 02 00:00:0	
[1]	TIMESTAMP	Mon May 05 00:00	
[2]	TIMESTAMP	Mon Aug 02 00:00:	
[3]	TIMESTAMP		
[4]	TIMESTAMP		
review	STRING(10)[5]		

The XSD defines the **book** element with a minOccurs="0". So, the XML could be sent without any **book** elements. In this situation an attempt to use the '**book\_price[ ]**' event variable in a **Set** action would result in an -5111 ( Event variable does not exist) error. The value of **book\_nbrRows** will be set to 0. You can use the **If** action to test the value of the **book\_nbrRows** variable before proceeding with accessing event variables associated with the **book** element.

- Related topics
- XSD limitations
- Listener XSD format

## IIoTA industrial IoT Platform: XSD limitations

The current implementation of the XSD support is limited to tags, arrays, and attributes. Attributes are not supported as xpath expressions to locate the listener map id (for more information, see Listener XSD format).

The Transaction Server does not use an XML validating parser and will not validate the XML received against the XML Schema. Therefore, the XML must reflect the XSD structures to ensure the correct match of XML tag values to variables.

The only facet implemented is **maxLength** which is used to determine the size of the string in a logical variable.

### XSD constructs not supported by the Transaction Server

Currently, the Transaction Server does not process and ignores the following XML Schema constructs:

- **Choice**
  - Unless all the elements in the choice selection are strings, the XML Schema choice construct will result in an error when processing the XML.
- **Array Size**
  - When processing arrays in the XML (for example those elements specified with a `maxOccurs="unbounded"`), the array size specified in the 'Rows In' field represents an upper limit on the number of array XML elements that will be handled by the trigger. If the number of rows received in the XML is less than the value specified in 'Rows In', it does not result in an error. For every XML Schema element identified with `maxOccurs="unbounded"`, there will be an event variable (Listener Map) or input/output variable (Transport Map) created that ends with **\_nbrRows**. It stores the actual number of rows received capped by the 'Rows In' value. You should reference this value to determine how many rows to process when assigning the listener trigger event variables or Transaction action output variables to device, local or static variables. If the array of XML elements is not present, then the corresponding **\_nbrRows** variable will be set to 0. When working with a Transaction action input variable you can set this field to specify the number of array items that should be processed by the transport map. Set this value to 0 to indicate that you do not want to specify any to be processed.
- **Namespace**
  - The Transaction Server XSD Schema handling is not namespace aware.

The following shows an example of the XSD constructs that are not supported by the Transaction Server:

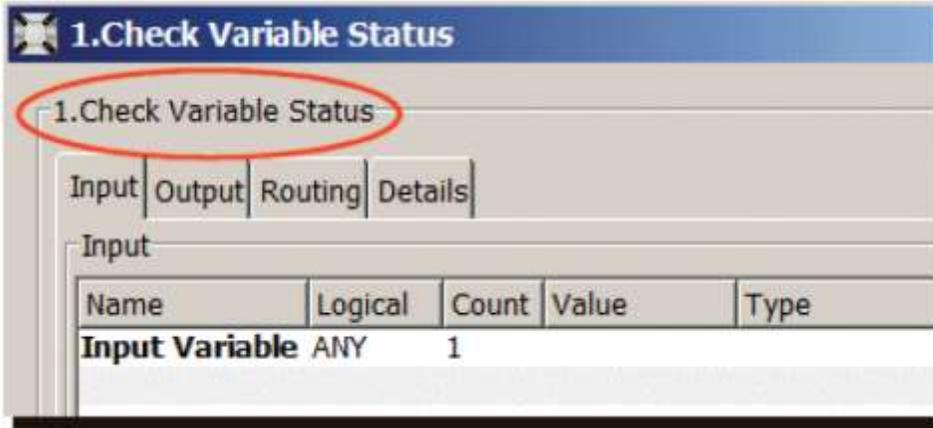
XSD element	XSD example
choice	<pre> &lt;xsd:group name="priceGroup"&gt; &lt;xsd:choice id="pg.choice"&gt; &lt;xsd:element name="fullPrice" type="fullPriceType"/&gt; &lt;xsd:element name="salePrice" type="salePriceType"/&gt; &lt;xsd:element name="clearancePrice" type="clearancePriceType"/&gt; &lt;xsd:element name="freePrice" type="freePriceType"/&gt; &lt;/xsd:choice&gt; &lt;/xsd:group&gt; </pre>
Attributes	<p>The Transaction Server supports attributes but not the following options on attributes:</p> <ul style="list-style-type: none"> <li>use="prohibited" is ignored.</li> <li>form="qualified" is ignored and is sent with the XML Schema default of form="unqualified".</li> </ul>

## Nil and missing element handling

- If an XSD Schema defines an element with minOccurs="0" and the XML does not contain the tag, the XML is accepted but any reference to the corresponding trigger event variables will result in a -5111(Event Variable Not Defined) or a -5209(Action variable is not serviced) error. To avoid this, you should test the corresponding **\_nbrRows** variable for a value greater than 0 before accessing the corresponding trigger event or transaction action output variable.
- If the XML contains a non-null empty tag (such as <tag></tag>) the value of the data mapped to a logical variable will depend on the logical variable datatype.
  - a logical variable of type String will get assigned an empty string (such as "").
  - a logical variable of type numeric ( INT2, INT4 etc) is rejected unless the XSD defines the element as nillable="true" and the XML element contains the "nil" attribute. For example, <tag nil="true"></tag>. In this case since null values cannot be represented in the trigger, the status code associated with the variable will be -5111 or -5209. You can use the 'Check Variable Status' action to test the status of the variable. As an alternative you can change the logical variable type to String and perform a data type conversion in trigger logic.

### Error codes

A variable with an error code such as Variable Not Serviced (-5209) or Event Variable Not Defined (-5111) cannot be manipulated or assigned to another variable without generating an error in the trigger. To check a variable error code, use the trigger Check Variable Status action from the Device category.



If one element in the array has an error such as Variable Not Serviced or Event Variable Not Defined, the entire array is invalidated. If the XSD contains default or fixed values, then those are used if the element is non nillable.

## Unsupported XML schema types

The following XML schema types are not supported by the Transaction Server:

- base64Binary
- hexBinary
- duration
- NOTATION

If the XSD contains elements of the above types, they must be optional or set with nillable="true" to be handled by the Transaction Server; otherwise, it will not be possible for the Transaction Server to generate a valid XML from the transport map XSD or successfully process an incoming XML in the listener.

Data types are mapped as follows to XML Schema types:

XML schema type	Data Type
boolean	BOOL
byte	INT1
date	TIMESTAMP
dateTime	TIMESTAMP
decimal	FLOAT8
double	FLOAT8
float	FLOAT4
gDay	UINT1

gMonth	UINT1
gMonthDay	STRING
gYear	UINT4
gYearMonth	STRING
int	INT4
integer	INT8
long	INT8
short	INT2
string	STRING
token	STRING
normalizedString	STRING
language	STRING
Time	STRING(8)
unsignedByte	UINT1
unsignedInt	UINT4
unsignedLong	UINT8
unsignedShort	UINT2
positiveInteger	UINT8
negativeInteger	INT8
nonPositiveInteger	INT8
nonNegativeInteger	UINT8
anyURI	STRING
QName	STRING
NCName	STRING
Name	STRING

**Related topics**

Creating a listener map with an XSD payload

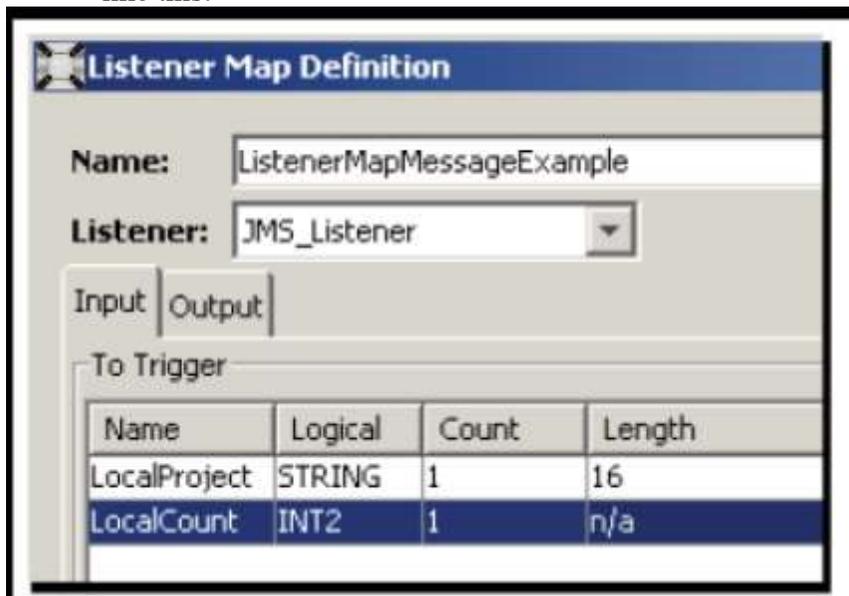
Listener XSD format

## IIoTA industrial IoT Platform: Creating a listener map with a map message payload

You can also create a listener map that uses a map message format. The map message payload will be comprised of name value pairs that have a type associated with them. If your enterprise application program uses map messages, you will want to add a map message format to the payload listener map message listener map. Unlike an XML format (which is always a String), specifying Map as the format will allow you to specify a data type for the variable such as integer, String, float, and so forth.

Follow these steps:

1. On the Listener Map Definition Map window, click the **Listener** down-arrow, and then select the appropriate listener. The map message format is only available for JMS-based listeners.
2. Using the **Input** tab, create the input map variables. The completed **Input** tab might look like this:



The next step is to create the payload. You will associate the Input map variables with **Property Name** values.

**Arrays:** You cannot use arrays with map message. Input and Output map variables must have a count of 1. The Transaction Server will not allow a transport map definition with Input and Output map variables with a count greater than one when the format is **Map**.

- Under the **From Enterprise** section, click **Add**. Notice the **Format** for this example is **Map**.

The New Item window appears.

The values from these parameters will appear in the map message:

**Name** — This parameter provides a list of the map variables from the **Input** tab. When the listener map is specified for use by the trigger, the map variable is associated with a PLC device variable or constant value.

**Property** — A name you specify as the property identifier.

**Default Value** — A value that will be used whenever the incoming request does not have a **Property** value specified.

- Click **Add**.

The information is added under the **From Enterprise** section.

Name	Property Name	Default
LocalProject	Project101	100

- Repeat steps 4 through 5 for the remaining map variable. For this example, **LocalCount**. Set **Property** as **Count**. For this example, no default value is set.

## Output tab

The next step is to use the **Output** tab.

1. From the **Output** tab, under **From Trigger**, click **Add**.

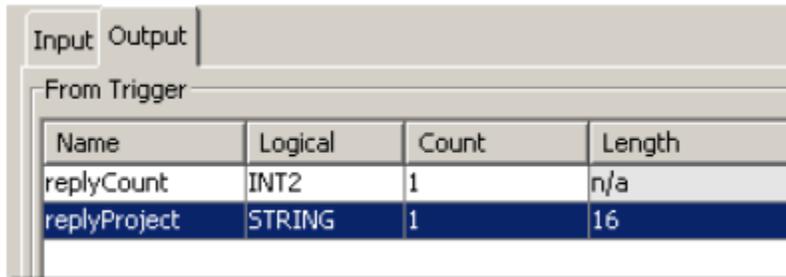
The New Item window appears.

2. Type a name for the output map variable, select INT2 as the type, and accept the count as one.
3. Click **Add**.

The map variable is added under the **From Trigger** section.

4. Repeat steps 7 through 9 as appropriate for the second output map variable.

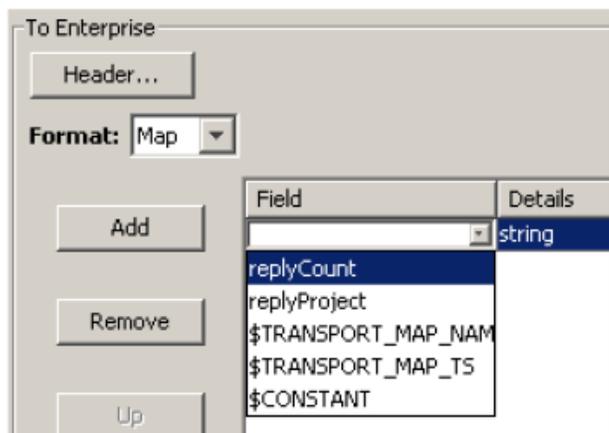
A row is inserted under the second row. The completed **Output** tab might look like this:



Name	Logical	Count	Length
replyCount	INT2	1	n/a
replyProject	STRING	1	16

5. Under the **To Enterprise** section, click the **Format** down-arrow, and then select **Map**.
6. Click **Add**.

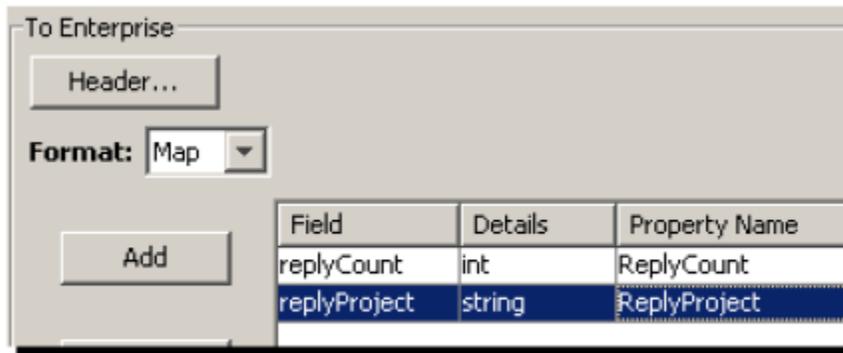
The first row in the table becomes active.



Field	Details
replyCount	string
replyProject	
\$TRANSPORT_MAP_NAME	
\$TRANSPORT_MAP_TS	
\$CONSTANT	

7. Under **Field**, click the column to display a drop-down list, and then select the appropriate map variable (for this example, **replyCount**).
8. Set the data type as an integer and type **ReplyCount** for **Property Name**.
9. Repeat steps 12 through 14 for the remaining map variable (for this example, **replyProject**).
10. Accept the default value of String for the **Details** column and type **ReplyProject** for **Property Name**.

A row is inserted under the second row. The completed **To Enterprise** section might look like this:



11. Click **Validate**.



12. A message will indicate whether or not the listener map definition is correct. Click **OK**.

13. Click **Save**.

The name of the listener map is added to the **Listener Maps** tab.

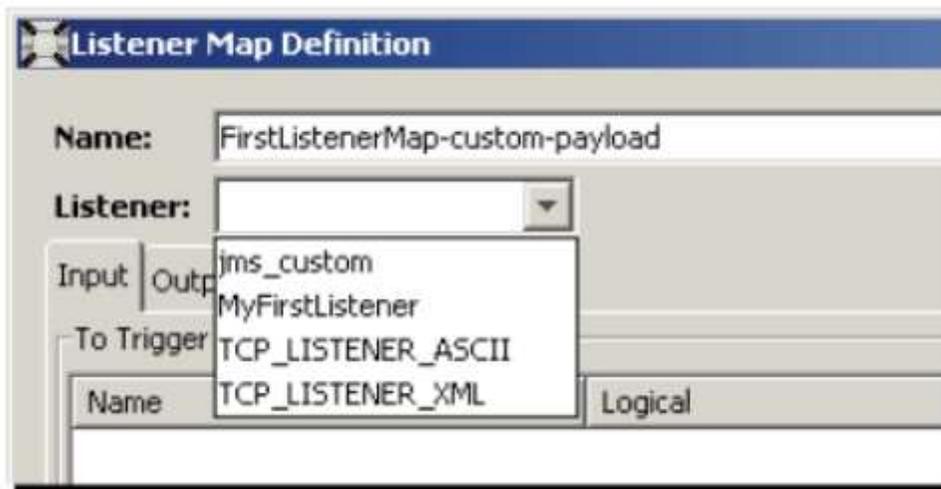
## IIoTA industrial IoT Platform: Creating a listener map with a custom format

This section assumes that you understand how to create a listener map and the listener (with the custom payload format) to use is available on the node. Follow these steps:

1. From the Workbench left pane, expand the node that you want to add the listener map to.
2. Expand Enterprise, right-click the **Listener Maps** icon to display its pop-up menu, and then click **New**.

The Listener Map Definition window appears.

3. Name the listener map.



4. From the **Listener** drop-down list, select the listener that has the custom payload format.

Name	Logical	Count	Length
ModelYear	INT2	1	n/a
Source	STRING	1	32
SpecBook	STRING	1	32

- On the **Input** tab, add the input variables and their values.

The bottom of the Listener Map Definition window accommodates the custom format.

From Enterprise

Header...

**Format:** Custom

**Custom Method Name:** transformCustomOptionRequest

- In the **Custom Method Name** box, type the name of the method from the specified class name on the listener window that implements the mapping method between the listener map input variables and their values.

Input Output

From Trigger

Name	Logical	Count	Length
responseItemCount	INT4	1	n/a
ItemKey	STRING	1	32
ItemValue	STRING	1	128
ModelYear	INT2	1	n/a
Source	STRING	1	32

- On the **Output** tab, add the output variables and their values.
- From the bottom of the Listener Map Definition window, use the **Format** down arrow, and then select **Custom**

To Enterprise

Header...

**Format:** Custom

**Custom Method Name:** transformCustomOptionRequest

- In the **Custom Method Name** box, you can type a method name to map the output variables to their values.

For both input and output map variables, the transform method is expected to use the

input and output map variable names. An error will be generated at runtime, if the value specified in **Custom Method Name** in both the **Input** and **Output** tabs cannot be found.

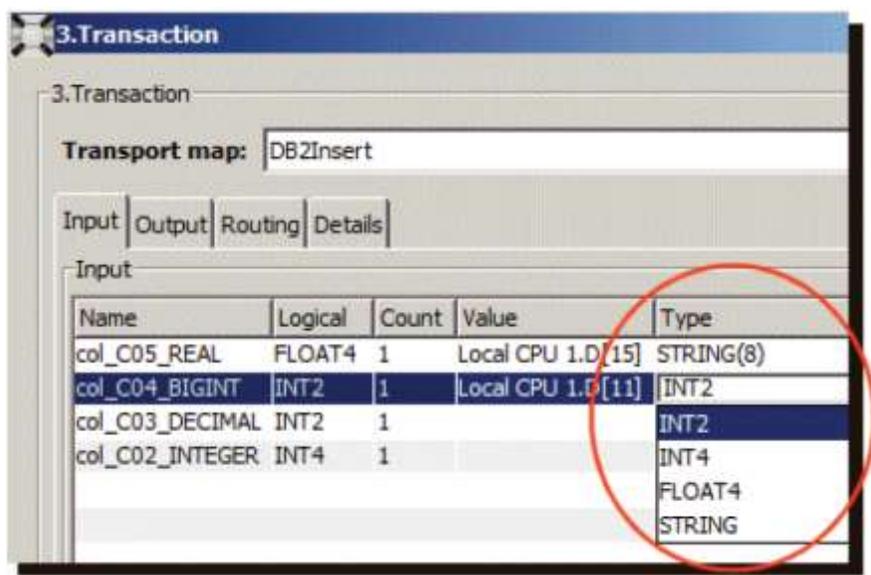
## IIoTA industrial IoT Platform: Understanding data types

When creating a transport map or listener map, you will encounter different data types. That is because the data types support different solution layers:

- The device or sensor layer. These data types are specific to the device, sensor or other objects that have device variables.
- The map layer. These data types represent the map variables that you define using the Workbench.
- The enterprise layer. These data types are specific to the enterprise applications.

### Device variable data types

These data types are specific to the device that contains the device variable. You will see these data types when the specific device and device variable are selected.



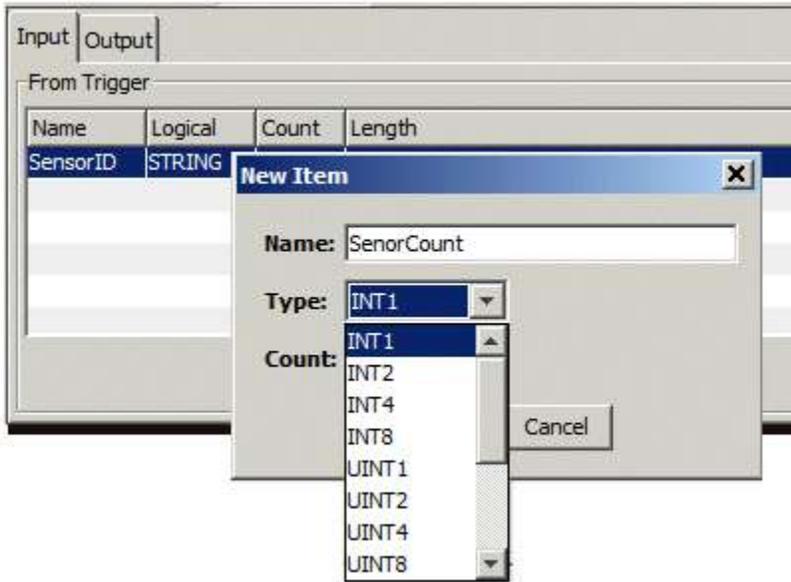
Name	Logical	Count	Value	Type
col_C05_REAL	FLOAT4	1	Local CPU 1.D [15]	STRING(8)
col_C04_BIGINT	INT2	1	Local CPU 1.D [11]	INT2
col_C03_DECIMAL	INT2	1		INT2
col_C02_INTEGER	INT4	1		INT4
				FLOAT4
				STRING

The following table lists a subset of the available device variable data types and their ranges, additional information is provided in the specific device section of Device types.

Data type	Description	Range
BOOLEAN	Boolean data is used in one-bit units.	zero to 1
INT2	2 bytes. Word data is 16-bit numeric data used by basic and application instructions.	-32768 to 32767
INT4	4 bytes. Double word data is 32-bit numerical data used by basic and application instructions.	-2147483648 to 2147483647
FLOAT4	4 bytes. Real number data is 32-bit floating decimal point data used with basic and application instructions.	3.4028234663852886 E 38 to 1.401298464324817 E -45 both positive and negative
STRING	String data is character data used by basic and application instructions.	A sequence of contiguous WORDs up to a user specified length that contains character data.

### Transport map and listener map data types

Map data types support map variables that you define. The data types translate all device variable data types to enterprise application data types.



You will see these data types when creating map variables from the Input or Output tab on the Transport Map and Listener Map windows.

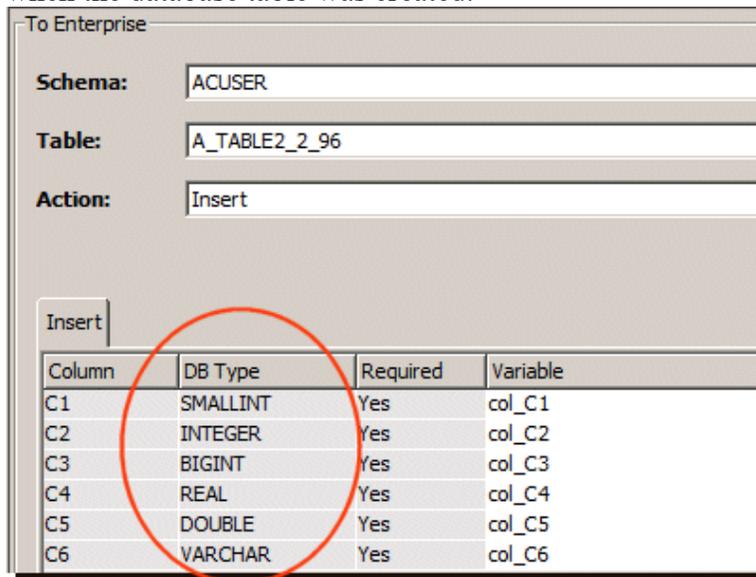
The following table lists available map data types and their ranges:

Data type	Description	Range
INT1	1-byte integer	-128 to 127
INT2	2-byte integer	-32768 to 32767
INT4	4-byte integer	-2147483648 to 2147483647
INT8	8-byte integer	- 9223372036854775808 to 9223372036854775807
UINT1	1-byte unsigned byte	0 to 255
UINT2	2-byte unsigned word	0 to 65535
UINT4	4-byte unsigned double word	0 to 4294967295
UINT8	8-byte unsigned long word	0 to 18,446,744,073,709,551,615

### Enterprise application data types

There are data types associated with each supported enterprise application product including IBM DB2, DB2 400, Oracle, Microsoft SQL server, and MySQL. You will see these data types when defining the map variables to the enterprise payload.

In the following example, the **DB Type** column shows IBM DB2 data types that were specified when the database table was created.

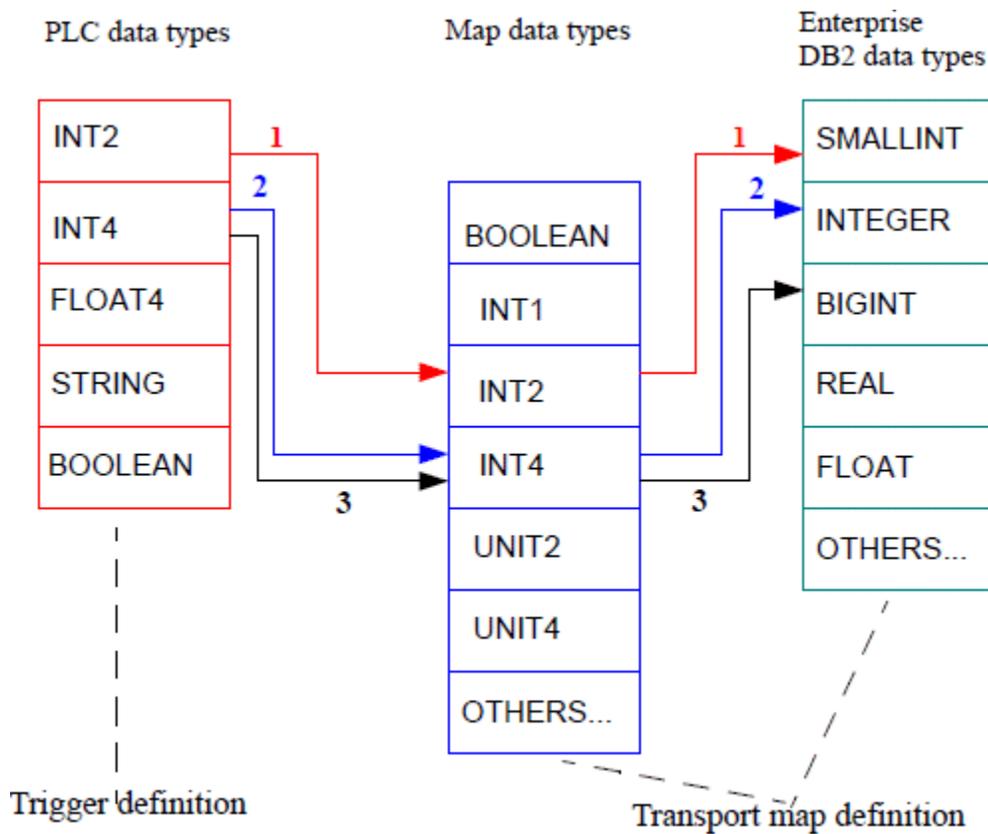


In the following example, the **DB Type** column shows data types from Oracle.

Column	DB Type	Required	Variable
COL1	NUMBER	Yes	col_COL1
COL2	NUMBER	No	col_COL2
COL3	FLOAT	No	col_COL3
COL4	FLOAT	No	col_COL4
COL5	FLOAT	No	col_COL5
COL6	TIMESTAMP	No	col_COL6

### Example data conversion

The following shows how data types from a PLC are mapped to DB2 database data types.



Related Topics  
 Device types

## IIOA industrial IoT Platform: Device connectivity

## Overview

A *device* is a representation of a group of variables and possible communication interface information. This representation can be for:

- A physical device, such as a programmable logic controller (PLC), a RFID tag reader or a sensor.  
When representing a physical device, the communication interface information could include properties such as the IP address, the TCP port, the communication timeout and the size and type of the variables within the device.
- A logical device that only resides within the system's runtime, such as a global variables device.

Each device is supported by a device driver, that handles the device specific:

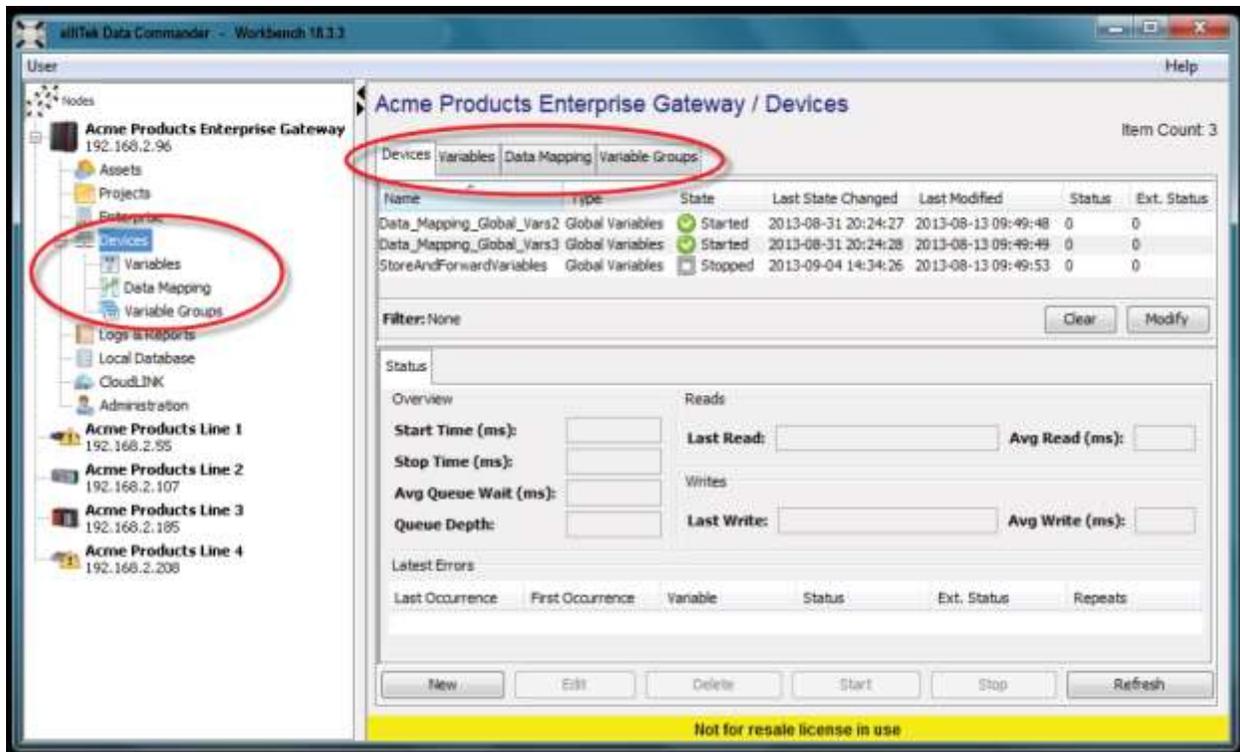
- Communication protocol
- Variables
- Data type conversions
- Commands
- Unsolicited message support.

This allows you to create your M2M solution using your devices in a vendor neutral methodology. The triggers (your application logic) and your enterprise applications can be shielded from most of the device connectivity details, while still providing the two-way device access functions required by your M2M solution.

The **Devices** feature from the Workbench is used to:

- Define, view and control devices
- Access device variables
- Define, view and control data mappings
- Define, view and control variable groups

Each of these is available as a sub icon that can be expanded from the Devices icon. They also are available as a separate tab when the Devices icon is selected.



## Assumptions

The information in this section assumes that you have:

- Activated the appropriate licenses.
- Installed the driver package that contains the protocol specific logic to communicate with the device.
- Installed the physical device and configured the hardware using the vendor's specific programming tool.

## Highlights

The first several sections of this guide describe the general tasks that apply to all device types.

The sections under Device types describe the specific features and definition details for each of the device types, both base product provided device types and installable device driver provided device types.

# IIoTA industrial IoT Platform: Defining, viewing, and controlling devices

## Overview

Devices can be defined, deleted, viewed, and have their state controlled.

The following sections provide general information that applies to all device types. Device type specification information is provided in Device types.

## Defining devices

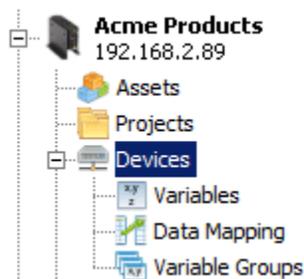
Devices are defined on the node where the device driver is installed and where the device specific communication to the physical device will take place. Physical devices types include PLCs, controllers, sensors, and bar code readers.

In the case of logical devices, the device is defined on the node where the memory representation of the device and its variables will take place. There is no physical device to connect to and no communication protocol to support. Logical device types include global variable device, aliases device, and property file reader device.

Once a device is defined and **Started**, the device's variables are available to triggers, on the same node, for read and write access.

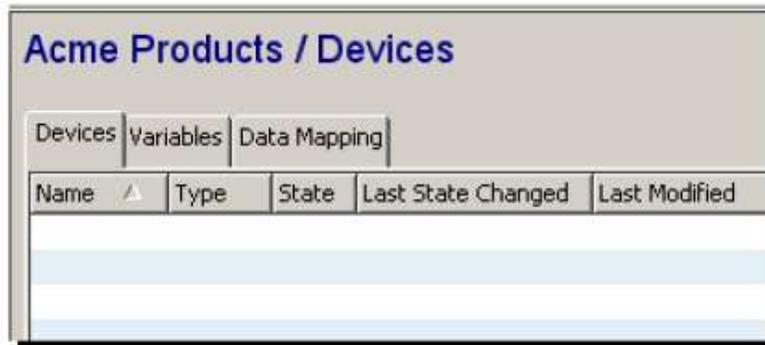
Follow these general steps to add a device definition to a node, each device type has its own specific parameters:

1. From the left pane, expand the node that you want to add a device definition to.
2. Select the **Devices** icon.

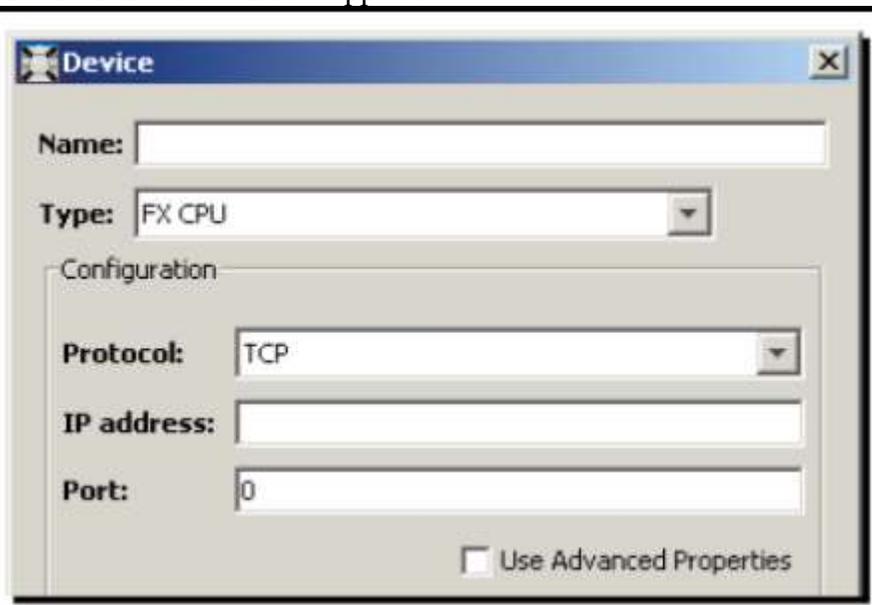


The Devices window appears as the right pane.

For this example, the **Devices** tab is empty because no device has been defined.

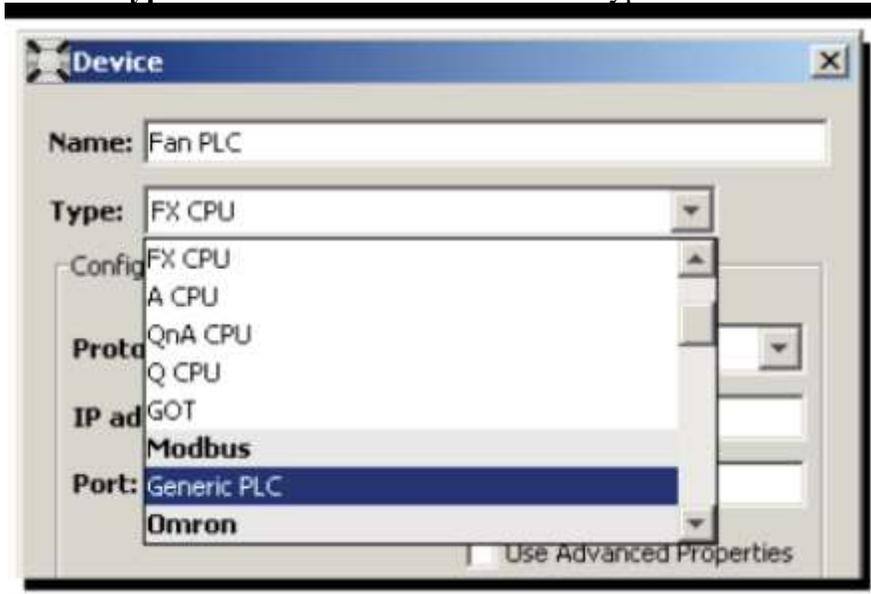


3. From the bottom of the **Devices** tab, select **New**.  
A default Device window appears.



4. In the **Name** box, type a name for the local device. A device name can be up to 64 characters and include letters, numbers, and the underscore character. Spaces are allowed.

- Use the **Type** down-arrow to select the device type.



The available device types are determined by the support available on the node. Some device support is installed using the Administration Packages tab.

The Device window changes to accommodate the selected device type.

- Fill in the device's parameters, based on the communication information and in some cases the variable information.

This is device type specific and is covered in Device types.

- Use the **Validate** button to validate the parameters and, in the case of a physical device, establish a connection to the device.

If the parameters are not correct or if a connection cannot be established an appropriate reason is displayed.

- Use the **Save** button to save the definition of the device.

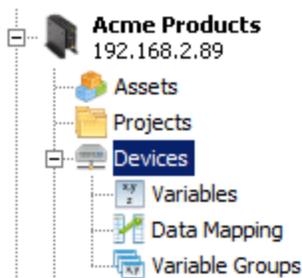
The device will appear in the list of devices for the node, in a **Stopped** state.

## Viewing devices

The **Devices** tab provides a list of devices defined on the node.

To use the **Devices** tab, follow these steps:

- From Workbench left pane, expand the node whose devices you want to view.



2. Select the **Devices** icon.

The **Devices** tab appears on the right pane.

Acme Products / Devices			
Devices   Variables   Data Mapping   Variable Groups			
Name	Type	State	Last State Changed
CIP Device	ControlLogix CPU	<input type="checkbox"/> Stopped	2012-07-24
CIP Device 1_97	ControlLogix CPU	<input type="checkbox"/> Stopped	2012-07-24
Global Modem Settings	Global Variables	<input checked="" type="checkbox"/> Started	2012-08-24
Global ROC History	Global Variables	<input checked="" type="checkbox"/> Started	2012-08-24

3. The **Devices** tab provides a table format that lists the devices defined on the node.

The top section of the **Devices** tab provides these columns:

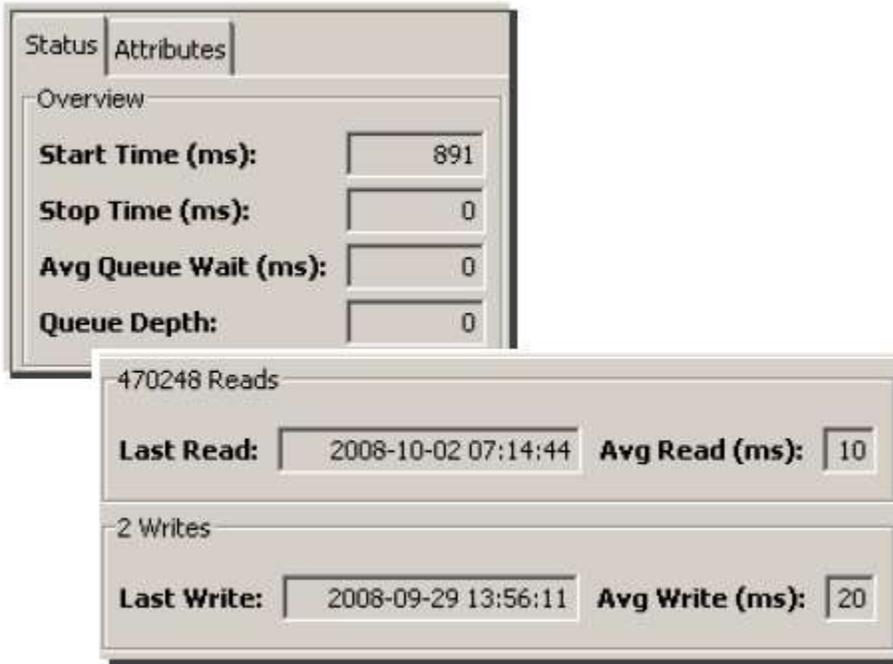
Column name	Description
Name	The name of the device. The device name must be unique on the node.
Type	The device type or model.
State	The device state, which can be: <b>Started</b> - the device is active. Its variables can be accessed (read and write) by triggers on this node. <b>Stopped</b> - the device is not active. Its variables cannot be accessed. <b>Disabled</b> - indicating there is a problem communicating with the physical device. <b>Starting</b> and <b>Stopping</b> - states that are temporary transition states.
Last State Changed	Date when the state of the device was last changed.
Last Modified	Date when the definition of the device was last changed.

Status	Indicates the health of the device. Zero indicates the device is operating normally. Non-zero indicates a communication error status. An error occurred between the node and the physical device.
Ext. Status	The Extended Status. When the <b>Status</b> column contains a non-zero value, the <b>Ext. Status</b> column contains the manufacturer specific error code that provides additional detail into what caused the error.

**Status tab**

When a device row in the table is selected, the middle portion of right-hand pane provides a Status and Attributes tab.

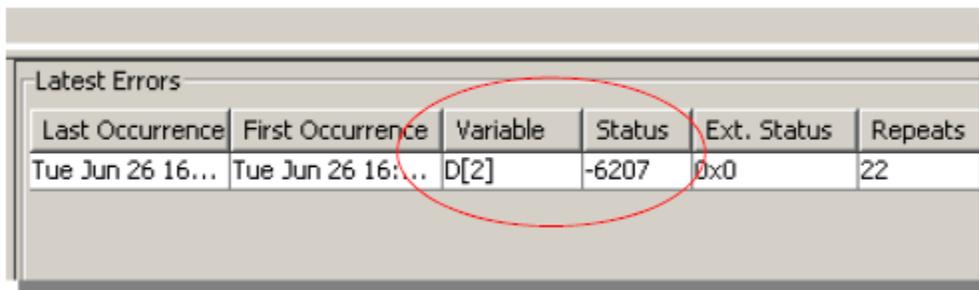
For Started devices, the Status tab displays read-only runtime statistics for the selected device.



The **Overview** and **Reads** panes provide information concerning the performance of the selected device as follows:

Parameter	Description
Start Time	The amount of time needed to start the device. The time shown is in milliseconds.
Stop Time	The amount of time needed to stop the device. The time shown is in milliseconds. For devices that respond quickly, this stop time can be listed as zero.
Avg Queue Wait	The average time a read or write device access request waits on the device's pending queue before it is processed.
Queue Depth	The current number of read or write device access requests that are on the device's pending queue waiting to be processed.
nnnn Reads	nnnn indicates the number of read requests made against the variables on the module. The read requests could have come from either the execution of a trigger accessing the variables, or a manual read request from the Variables tab. Last Read: Indicates the time of the last read request. Avg Read (ms): Indicates the average amount of time needed to read a variable from the device. The count continues to increment as the device is accessed.
nnnn Writes	nnnn indicates the number of write requests made against the variables on the device. Last Write: Indicates the time of the last write request. Avg Write (ms): Indicates the average amount of time needed to write a variable to the device. The count continues to increment as the device is accessed.

The **Latest Errors** section provides error information on any errors that were encountered when interacting with the device.



Latest Errors					
Last Occurrence	First Occurrence	Variable	Status	Ext. Status	Repeats
Tue Jun 26 16:...	Tue Jun 26 16:...	D[2]	-6207	0x0	22

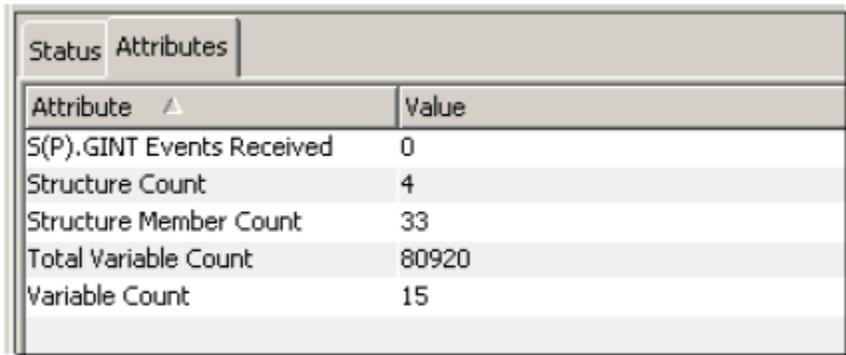
In the example, an error occurred when attempting to access a device variable name D[2] on a device that was stopped. The **Status** column shows the first occurrence of the -6207 error and the time associated with the most recent occurrence of the -6207 error.

Column name	Description
Last Occurrence	Time associated with the most recent occurrence of the error.
First Occurrence	Time associated with the first occurrence of the error.
Variable	Name of the device variable.
Status	The status code for the error.
Ext. Status	The extended status which indicates the manufacturer specific error code.
Repeats	Indicates the number of times the error occurred between the times of the first occurrence and last occurrence

The **Latest Errors** right pane provides columns with information concerning errors for the selected device as follows:

**Attributes tab**

For Started devices, the Attributes tab is displays additional information. Some of the attributes displayed are device vendor specific.



Attribute	Value
S(P).GINT Events Received	0
Structure Count	4
Structure Member Count	33
Total Variable Count	80920
Variable Count	15

The **Attributes** tab has two columns:

Column name	Description
Attribute	The name of an attribute supported by the device.
Value	The value for the attribute.

## Controlling devices

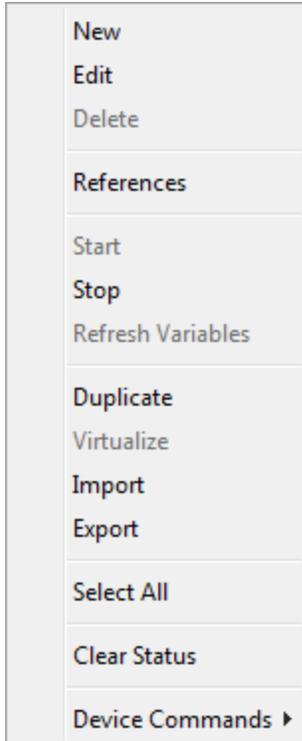
When a device row is selected in the table, the buttons at the bottom of the Devices tab become enabled or disabled. This is based on the current state of the device and the function of the button.

A single device row or multiple device rows can be selected and then the function buttons used, but the state of each device will determine if the function can be performed.

Button	Description
<b>New</b>	Define a new device.
<b>Edit</b>	Edit the device definition. This can be used when the device is in the <b>Started</b> or <b>Stopped</b> state, but care should be taken when modifying the definition of a <b>Started</b> device. This is only available for a single device row selection.
<b>Start</b>	Available when the device is in a <b>Stopped</b> state. Change the device to the <b>Started</b> state.
<b>Stop</b>	Available when the device is in the <b>Started</b> or <b>Disabled</b> state. Change the device to the <b>Stopped</b> state.
<b>Delete</b>	Available when the device is in the <b>Stopped</b> state. Delete the device definition from the node.
<b>Refresh</b>	Refresh the information displayed in the <b>Devices</b> tab. The Workbench will periodically refresh the information on its own without the Refresh button being used.

### A device's pop-up menu

If you right-click on an individual device in the device table, a pop-up menu with options is displayed:



Some menu options will not be available depending on the device's state and the device type. The following describes the menu options available from the pop-up menu:

Option	Description
<b>New</b>	Displays a New Device window to define a new device.
<b>Edit</b>	Displays a Device window to view and edit the device's definition. When editing a device, keep the following in mind:  When editing and then saving a started device, the device will be stopped and then saved. You will need to start the device after it has been saved. If the device is being edited by another person, you will receive a message indicating the device is locked but still available for view only access.
<b>Delete</b>	Deletes the device. The device must be in a Stopped state to allow deleting.
<b>References</b>	Displays a list of items that have a reference to the device and a list of items that are referenced by the device.
<b>Start</b>	Starts the device. The device must be in a Stopped state to allowing starting.
<b>Stop</b>	Stops the device. The device must be in a Started or Disabled state to allowing stopping.

<b>Refresh Variables</b>	Re-queries the device's variable structure.
<b>Duplicate</b>	Duplicates (copies) the device definition. The option allows entering a new name for the copy of the device.
<b>Virtualize</b>	Duplicates (copies) the device definition into a virtual copy of the device's variable structure. The option allows entering a new name for the virtual copy of the device.
<b>Import</b>	Displays the Import window, allowing the selection of a previously exported export file.
<b>Export</b>	Displays the Export window with the selected device and allows the device and its dependencies to be selected for export to a file on the Workbench's computer. For more information on the Export function, see Exporting a project or trigger.
<b>Select All</b>	Selects all the devices. If you want to exclude one or more devices from your selection, after using select all, press and hold down the Ctrl key, and then click on the device's row in the table.
<b>Clear Status</b>	Clears the execution counters and status from the device table and the Status tab for the device.
<b>Device Commands</b>	If the device supports device commands, they will be available for selection. The device commands are specific to the device driver.

## Device table and option considerations

- When editing and then saving a started device, the device will be stopped and then saved. You will need to start the device after it has been saved.
- If the device is being edited by another person, you will receive a message indicating the device is locked but still available for view only access.
- Some of the options can be used when multiple devices are selected. You can use Ctrl-A, Select All, Shift-click, Ctrl-click, or press and hold the mouse button while swiping multiple rows to select multiple devices. Then right-click to display the pop-up menu and selecting an option will attempt to apply the option to the multiple selected items. If the option cannot be applied, a message window will indicate the selections that will be applied.  
For example, select three stopped devices and then select **Start**.

### Related topics

The Device Control action can be used to start, stop or delete a device.

## IIoTA industrial IoT Platform: Starting a device

When a device definition is added to a node, the connection and variable information is available for the appropriate device driver that will support the device.

The device must be started to indicate to the device driver that:

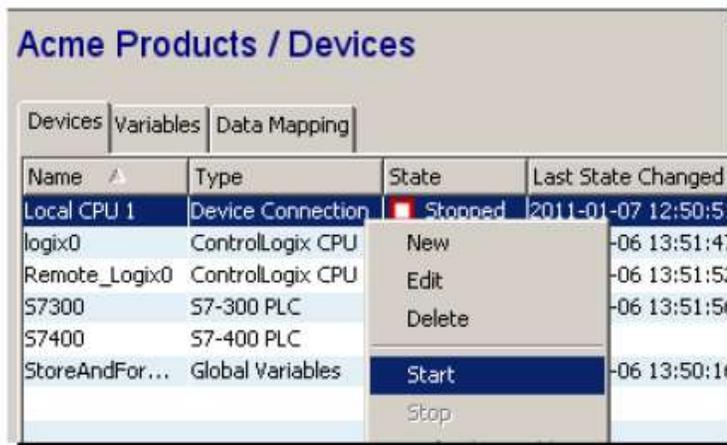
- A connection to the device should be established
- The device's variables should be listed and available for read/write access
- The device and its variables' individual security access permission should be determined
- Any supported device commands should be listed
- Any unsolicited message support that the device provides will be enabled.

Follow these steps to start a device.

1. From the **Devices** tab, select the appropriate device.

Right-click to display its pop-up menu, and then select **Start**.

Or when the device row is selected, select the **Start** button at the bottom of the panel.



2. The appropriate device driver will establish a connection to the device.

The **State** column for the device is changed to **Started**.



If there was a problem establishing a connection to the device, the state will change to Stopped or Disabled.

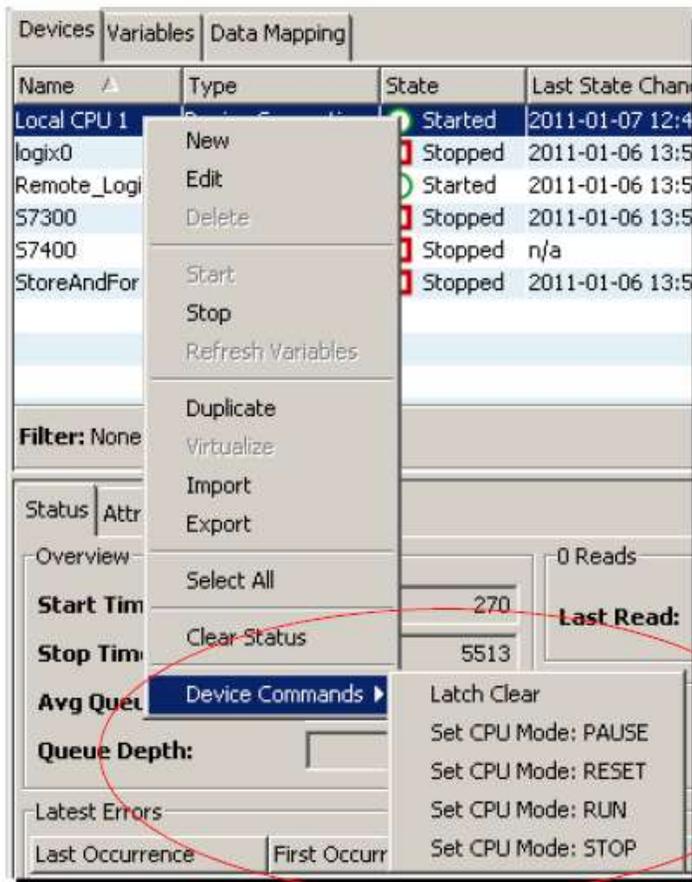
An appropriate error will be returned and logged in the exception log.

- All devices in the **Started** state will be listed in the **Variables** tab, where the device variables, including structures, can be viewed.

The individual device variables can be accessed for Read and Write functions.

The read and write permission supported by the device, and the read and write access configured for the user will determine the permitted access. For more information, see System administration: Security.

- If the device has device commands, they will become available.



From the device row, right-click to display its pop-up menu, and then select **Device Commands**.

You must have permission to use these device commands. You can see the corresponding resources on the System administration - Security tab Policy window.

5. If the device supports unsolicited messaging concepts, the corresponding trigger event type will be available under the PLC Logic Events category.

## IIoTA industrial IoT Platform: Enabling per device variable security

Some device drivers support the configuring of individual device variable security. In order to support this, the device driver will allocate internal memory for the necessary control structures for each variable that the device contains.

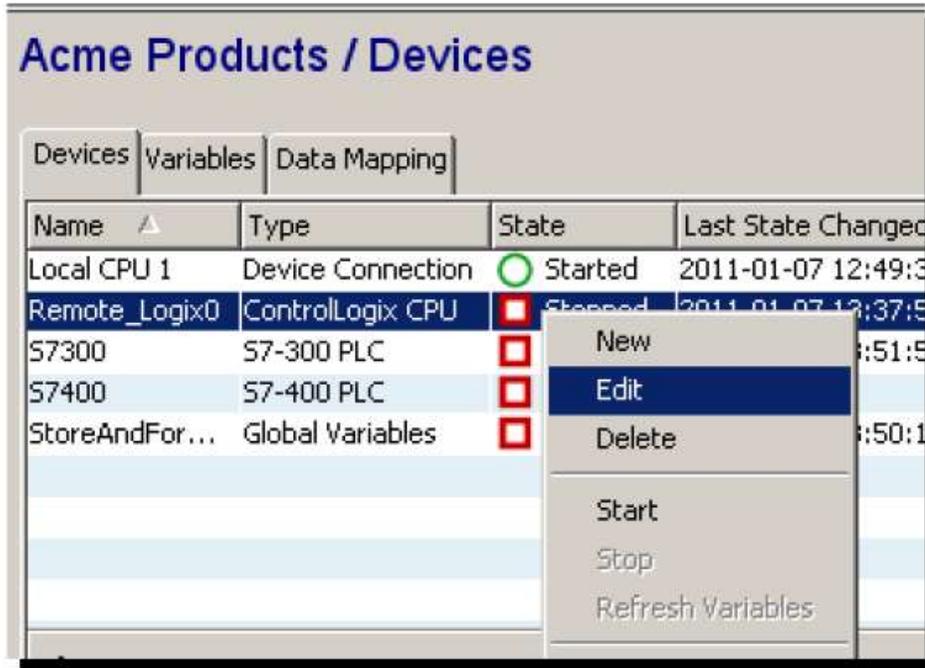
On memory resource constrained nodes, this internal memory may take more of the available memory than the application developer wishes to allocate. In this case, the support of individual variable security and the required memory can be disabled. This will result in all variables being treated the same, as far as access control, for all users that have access to the node. This enabling or disabling of individual variable security is done for each device, so the appropriate level of access control and memory usage is flexible.

The default setting for a device's per variable security parameter, either True (enabled) or False (disabled) will vary based on the node type.

The following assumes that the device is already created and **Stopped**.

Follow these steps:

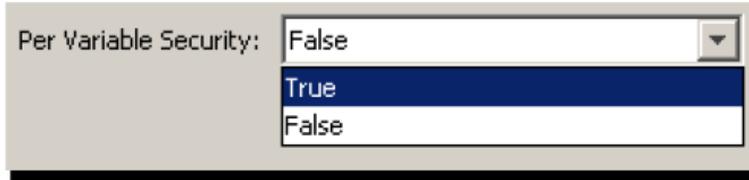
1. From the Workbench left pane, expand the node that contains the device whose per variable security setting you wish to change.
2. Select the **Devices** icon.  
The **Devices** tab appears on the right pane.
3. Locate the device, and make sure the device is stopped.



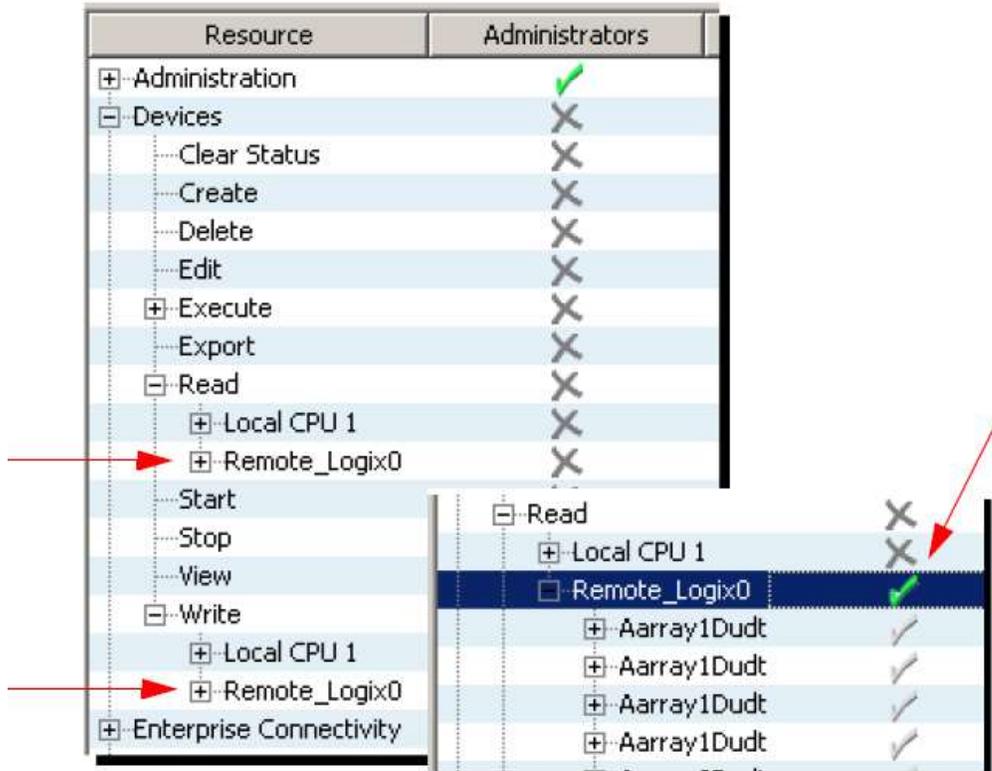
4. Select the device, display its pop-up menu, and then select **Edit**.  
Or when the device row is selected, select the **Edit** button at the bottom of the panel.  
The Device window appears.



5. Under **Configuration**, make sure **Use Advanced Properties** is selected.
6. Select the **Per Variable Security** down arrow, and then select **True**.



7. Select **Save**.
8. **Start** the device.
9. The next step is to go to the Administration window and use the **Security** tab to allow read and write access for each device variable for this device. Expand the device name listed under the **Devices** group for the Read and Write resource. You will see each device variable.



For this example, you would select the symbol next to the device to allow or deny access. For more information, refer to System administration Security.

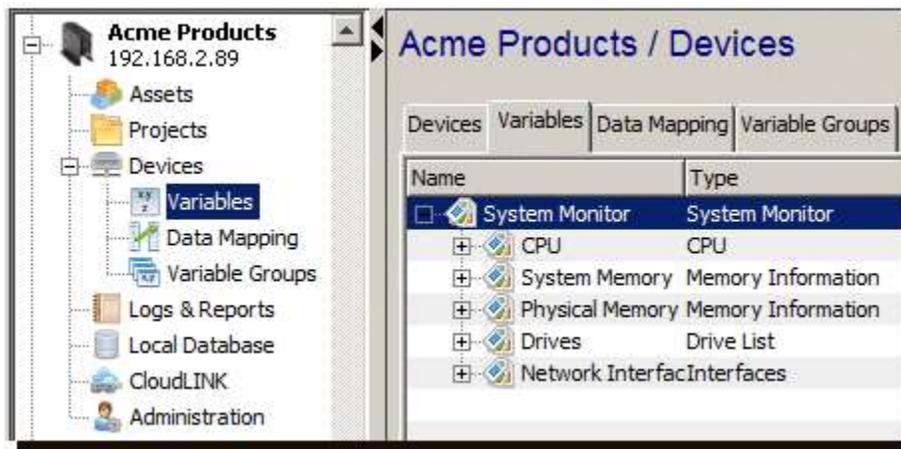
## IIoTA industrial IoT Platform: Accessing device variables

### Overview

Device variables can be accessed by the Workbench, triggers, and other features when the device is in a **Started** state.

The Workbench **Variables** tab displays all started devices, with controls to expand and collapse the device's internal structure and variables. When one or more variable rows are selected, the **Read** and **Write** function buttons are enabled. The Security feature can be used to control a user's access to features, including devices and device variables.

For physical devices, the device and its programming tools may have their own security access control features to limit read and/or write access to its variables.



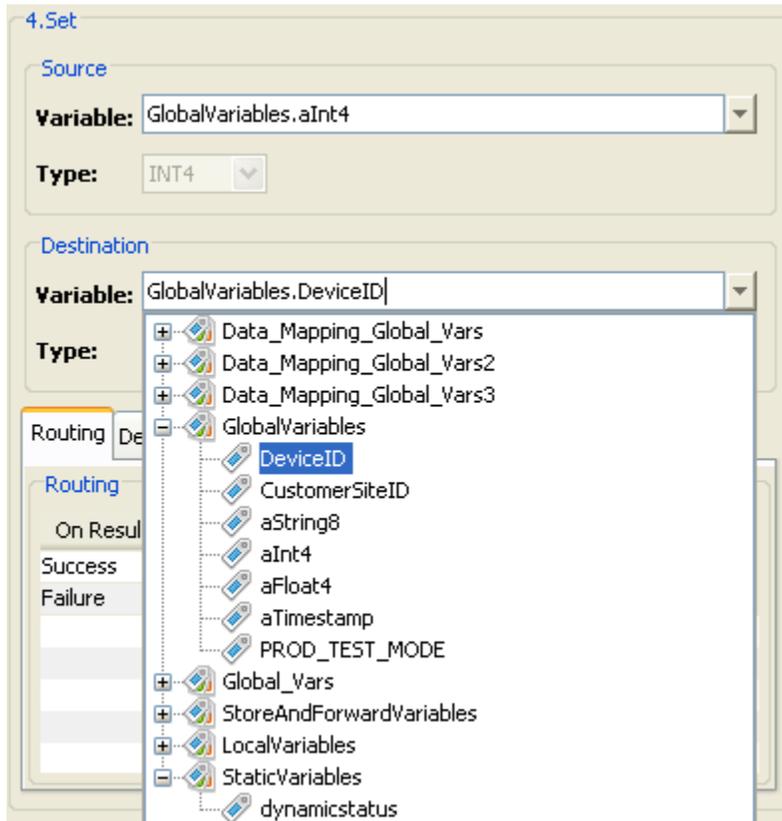
Triggers that execute on the same node as the device definition have access to all started devices' variables for use in the trigger actions that make up the application logic of the M2M solution.

Trigger actions have a concept of a *source* variable(s) and a *destination* variable(s). The source variables are read from the device and the destination variables are written to the device.

Using a Set action as an example:

- The **Source** variable, aInt4, is an integer (INT4) in the device named GlobalVariables

- The **Destination** variable, DeviceID, is an integer (INT8) in the same device.



In addition to trigger actions, device variables are also accessed by the features that are part of the application logic, including:

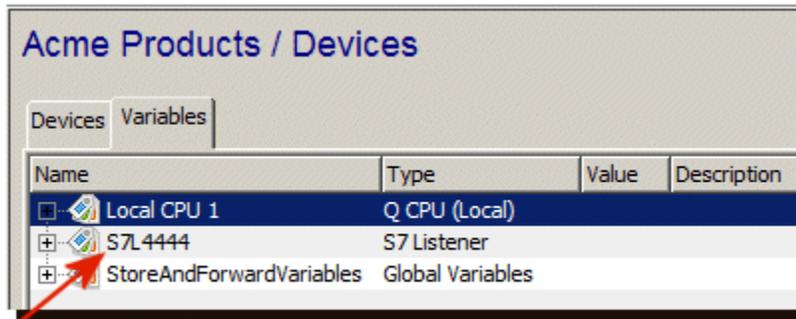
- Some trigger event types
- Data mappings
- Variable groups

### Using the Variables tab

The **Variables** tab lists the started devices for the current node. A device must be in the **Started** state to be included in the Variables tab.

To use the Variables tab, follow these steps:

1. From Workbench left pane, expand the node whose device variables you want to view or change.
2. Select the **Devices** icon.
3. From the right pane, select the **Variables** tab.  
The Variables tab appears as the right pane.



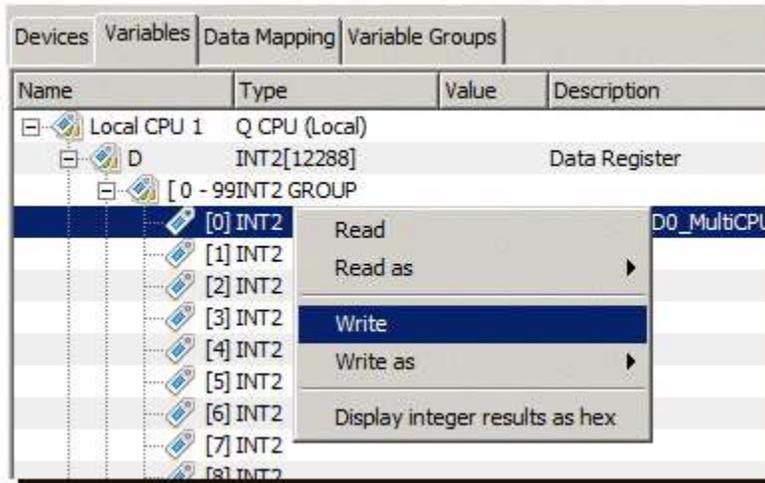
Name of device.

- For any device, use the plus sign and minus sign to expand and collapse the device's internal structure and variables.

The **Variables** tab provides these columns:

Column name	Description
Name	This is the name of the device. You will not see devices listed on the <b>Variables</b> tab if they are not in a <b>Started</b> state. When the device's structure is expanded, the internal structure and variables are displayed.
Type	When a device row is selected, this is the type of device. When a device variable row is selected, this is the type of the device variable. For devices that support structures and user defined types, this is the type of the structure or the user defined type.
Value	The current value of the device variable. If you have write access, you can select the device variable and then change (write) its value.
Description	The description of the device variable as set in the device's programming tool.

You can also right-click on a row to display a pop-up menu with options according to the row type and the user's access.



### Device variable data types

Device variables data types that appear on Workbench windows are always device vendor specific data types.

### Variable length

A variable's location name can be a maximum of 128 characters long. The variable location name consists of the parent structure names plus the variable name. Renaming a variable to fit the 128-character limit will allow you to read and write to the variable. A variable with a location name greater than 128 characters long will not be able to be accessed and will return an "Error: Variable does not exist" error.

### Related Topics

Device types

Trigger actions reference

Defining, viewing, and controlling data mappings

Defining, viewing, and controlling variable groups

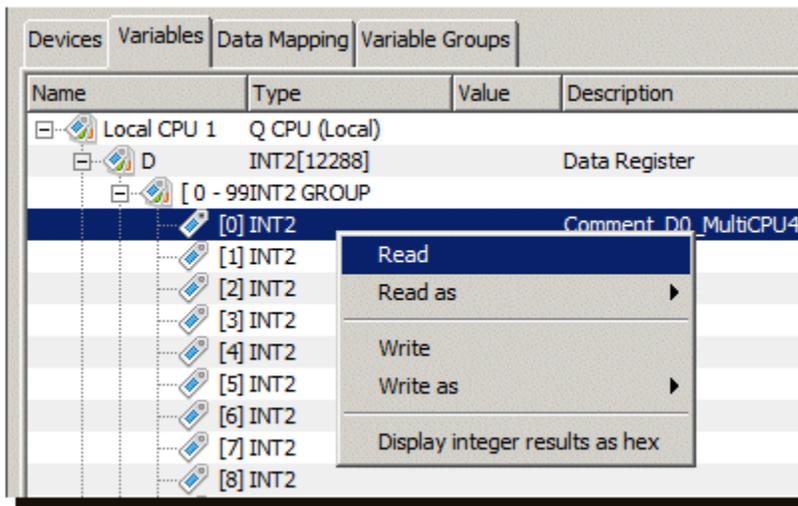
## IIoTA industrial IoT Platform: Reading the value of a device variable

The **Variables** tab provides the ability to **Read** the current value of device variables. You must have read access to view the value of a device variable.

To read the value of a device variable, follow these steps:

1. From Workbench left pane, expand the node whose device's variables you want to read.

2. Select **Devices**.
3. From the right pane, select the **Variables** tab.  
The **Variables** tab appears as the right pane.
4. From the appropriate device, select the plus sign to expand the device's internal structure and variables to locate the device variable whose value you want to read.
5. Select the device variable, display its pop-up menu, and then select **Read**.



For this example, the device variable D[0] with an INT2 data type is read. The current value of the device variable is displayed in the Value column. If the variable has comments associated with it, they are displayed in the comment column.

Alternatively, instead of displaying the pop-up menu and selecting **Read**, you can:

- Double click the device variable row to read the device variable's value
- Use the Read button at the bottom of the panel

### Reading multiple device variables at one time

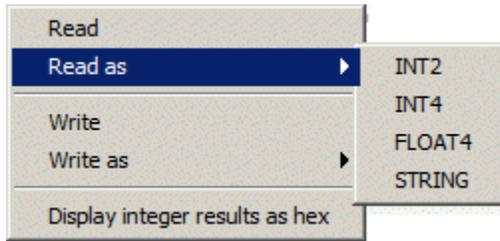
Multiple device variable rows can be selected and read at one time by using the pop-up menu **Read** option or the **Read** button.

If non readable rows are selected, such as a Device row or a structure row, the rows are ignored and only the readable rows are actually read.

### Reading device variables as a different data type

The device and its variables shown in the example support the reading of device variables as a different data type. All devices do not support this function.

In the pop-up menu, you can also select **Read as** to have the supported alternative data types displayed. In this example the data types are: INT2, INT4, FLOAT4 and STRING.



When you use the **Read as** function, the device driver reads the values of variables starting at the current variable address for the length implied by the selected data type.

For example, using **Read as** INT4 on device variable D[0] would read 2 2-byte WORDS and display the value as a 4-byte INT4. The variables read would be D[0] and D[1], each of which is a 2-byte INT2.

Understanding of the device's variable types (registers, tags, coils, inputs, and outputs are some examples of device specific terminology), the device's variables data types, and the device's variable addressing concepts is imperative when using the **Read as** and **Write as** functions of the Workbench.

### Displaying integer results as hex

The **Variables** tab supports displaying integer values as hex values. This option is toggled on or off and applies to all device's variables' values.

When writing values, the value must be entered in decimal (base 10) format.

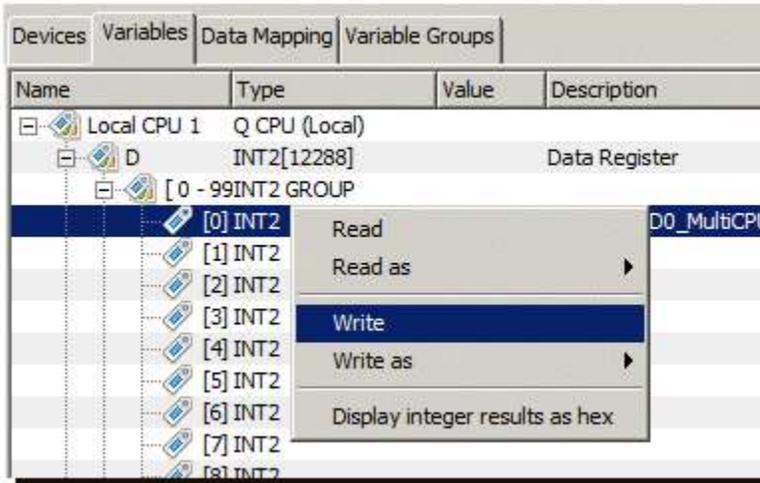
## IIoTA industrial IoT Platform: Writing the value of a device variable

The **Variables** tab provides the ability to **Write** the current value of device variables. You must have write access to change the value of a device variable.

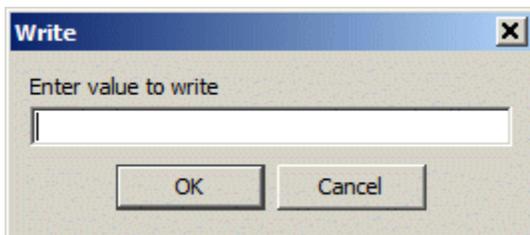
To write the value of a device variable, follow these steps:

1. From Workbench left pane, expand the node whose device's variables you want to write.
2. Select **Devices**.

- From the right pane, select the **Variables** tab.  
The **Variables** tab appears as the right pane.
- From the appropriate device, select the plus sign to expand the device's internal structure and variables to locate the device variable whose value you want to write.
- Select the device variable, display its pop-up menu, and then select **Write**.



- Select the device variable, display its pop-up menu, and then select **Write**.  
For this example, an INT based device variable is selected.  
The Write window appears.



- Type the new value, and then select **OK**.  
The device driver writes the value to the device and, if successful, displays the new value in the **Value** column.

### Writing multiple device variables at one time

Multiple device variable rows can be selected and written at one time by using the pop-up menu **Write** option or the **Write** button.

If non writable rows are selected, such as a Device row or a structure row, a warning messages is displayed and asks if you want to continue.

Care should be taken when writing multiple device variables of different data types.

### Writing device variables as a different data type

The device and its variables shown in the example support the writing of device variables as a different data type. All devices do not support this function.

In the pop-up menu, you can also select **Write as** to have the supported alternative data types displayed. In this example the data types are: INT2, INT4, FLOAT4 and STRING.

When you use the **Write as** function, the device driver writes the values of variables starting at the current variable address for the length implied by the selected data type.

For example, using **Write as** INT4 on device variable D[0] would write 2 2-byte WORDS. The variables written would be D[0] and D[1], each of which is a 2-byte INT2.

Understanding of the device's variable types (registers, tags, coils, inputs, and outputs are some examples of device specific terminology), the device's variables data types, and the device's variable addressing concepts is imperative when using the **Read as** and **Write as** functions of the Workbench.

### Displaying integer results as hex

The **Variables** tab supports displaying integer values as hex values. This option is toggled on or off and applies to all device's variables' values.

When writing values, the value must be entered in decimal (base 10) format.

## IIoTA industrial IoT Platform: Watching the value of a device variable

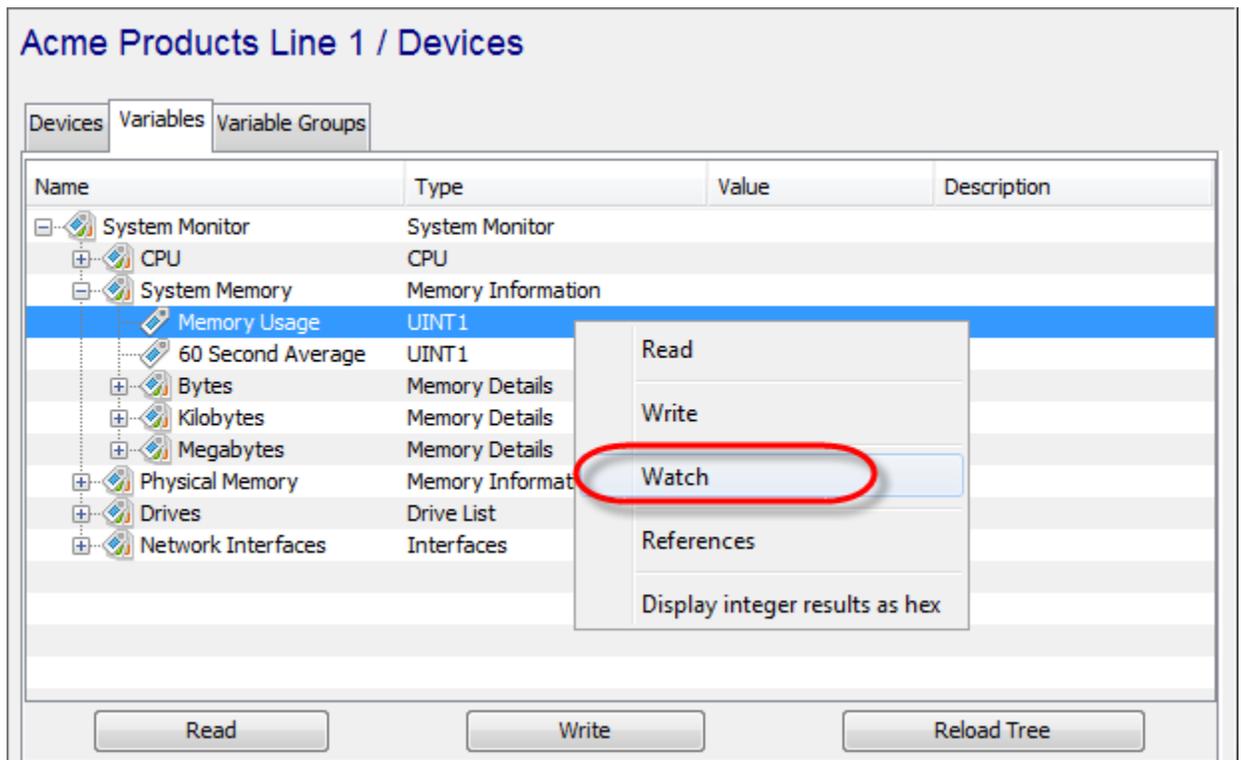
The **Variables** tab provides the ability to **Watch** the current value of device variables. You must have read access to watch the value of a device variable.

A watched variable is displayed in the **Variable Watch** window, where the displayed value will be updated whenever the variable is updated.

To watch the value of a device variable, follow these steps:

1. From Workbench left pane, expand the node whose device's variables you want to watch.

2. Select **Devices**.
3. From the right pane, select the **Variables** tab.  
The **Variables** tab appears as the right pane.
4. From the appropriate device, select the plus sign to expand the device's internal structure and variables to locate the device variable whose value you want to read.
5. Select the device variable, display its pop-up menu, and then select **Watch**.



6. The **Variable Watch** window will be displayed, showing the current value of the variable and when it was last updated. Value changes to the variable will be automatically displayed in this window.

For this example, the device variable **Memory Usage** with an UINT1 data type is watched. The current value of the device variable is displayed in the Value column in the variable panel as well as the watch window (see below).

### Watching multiple device variables at one time

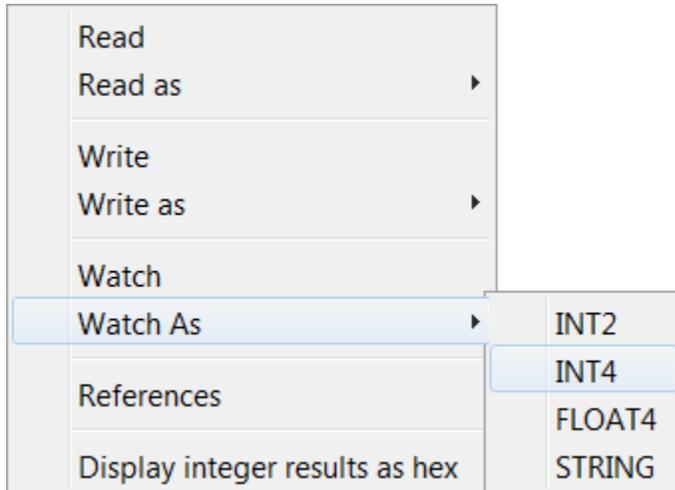
Multiple device variable rows can be selected and watched at one time by using the pop-up menu **Watch** option.

If non readable rows are selected, such as a Device row or a structure row, the rows are ignored and only the readable rows are actually added to the watch list.

### Watching device variables as a different data type

The device and its variables shown in the example support the watching of device variables as a different data type. All devices do not support this function.

In the pop-up menu, you can also select **Watch as** to have the supported alternative data types displayed. In this example the data types are: INT2, INT4, FLOAT4 and STRING.



When you use the **Watch as** function, the device driver reads the values of variables starting at the current variable address for the length implied by the selected data type.

For example, using **Watch as** INT4 on device variable INT2 would read 2 2-byte WORDS and display the value as a single 4-byte INT4.

Understanding of the device's variable types (registers, tags, coils, inputs, and outputs are some examples of device specific terminology), the device's variables data types, and the device's variable addressing concepts is imperative when using the **Watch as** function.

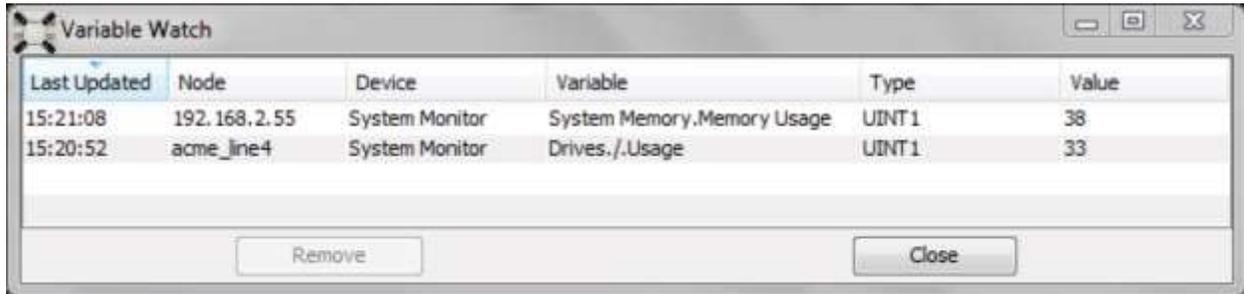
### Watch Window

A variable displayed in the Variable Watch window will show:

- The last time it was updated
- The node where the variable and device reside. The value in the Node column gives an indication of the scan type used to access the node. For a network scan an IP address is displayed. For a TR50 scan, a thing key is displayed.
- The device name
- The Variable name

- The type being watched, which can differ from the base type if **Watch as** had been selected
- The current value.

Watched variables do not have to all be on the same node or device and may show a different value than what is displayed in the device variable window due to being updated when the value changes.



Last Updated	Node	Device	Variable	Type	Value
15:21:08	192.168.2.55	System Monitor	System Memory.Memory Usage	UINT1	38
15:20:52	acme_line4	System Monitor	Drives./Usage	UINT1	33

The screenshot shows a window titled "Variable Watch" with a table containing two rows of data. Below the table are two buttons: "Remove" and "Close".

The window can remain open in the background while other tasks are performed in the Workbench. Updates to the variable values will occur regardless of the window being active. If the window is closed all watches will be stopped.

Entries can be removed from the list by selecting them and then selecting **the Remove** button or by right-clicking to display the pop-up menu and then selecting **Remove**.

The **Watch Window** supports displaying integer values as hex values. This option is toggled in the pop-up menu and applies to all variables being watched.

## IIoTA industrial IoT Platform: Defining, viewing, and controlling data mappings

### Overview

Data mappings can be defined, deleted, viewed, and have their state controlled.

The **Data Mapping** feature provides the function to:

- Read a source device variable at a defined frequency
- If the device variable has changed since the last read, write it to a destination device variable
- Optionally, include transformation from one data type at the source device variable to a different data type for the destination device variable.

Data mappings are defined on the node where the source device driver and destination device driver are installed and where the device specific communication to the physical devices will take place. Physical devices types include PLCs, controllers, sensors, and bar code readers.

In the case of logical devices, the device is defined on the node where the memory representation of the device and its variables will take place. There is no physical device to connect to and no communication protocol to support. Logical device types include global variable device, aliases device, and property file reader device.

Once a data mapping is defined and **Started**, the runtime handles the reading of the source device variable, processing, and the writing of the destination device variable.

There is no need for a separate trigger to be defined to do this device variable access (for example using the **Set** action).

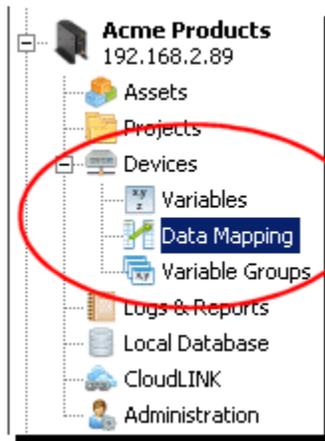
The Data Mapping feature can be used to compliment, and be a part of, the M2M solution application logic that users define in triggers.

## Defining data mappings

Data mappings are defined after the devices referenced for the source device variable and destination device variable are defined and **Started**.

Follow these steps to add a data mapping definition to a node:

1. From the left pane, expand the node that you want to add a data mapping definition to.
2. Select the **Devices** icon.  
Then select the **Data Mapping** tab  
Or, alternatively, select the **Data Mapping** sub icon.



The **Data Mapping** window appears as the right pane.

For this example, the Data Mapping window has previously defined data mappings.

Priority (ms) ▲	Source	Destination	State
50	Data_Mapping_Global_Vars3.Read_5	Data_Mapping_Global_Vars2.Write_5	✔ Started
500	Data_Mapping_Global_Vars3.Read_2	Data_Mapping_Global_Vars2.Write_2	✔ Started
1000	Data_Mapping_Global_Vars.Read_2	Data_Mapping_Global_Vars.Write_2	✔ Started
1000	Data_Mapping_Global_Vars.Read_3	Data_Mapping_Global_Vars.Write_3	✔ Started
1000	Data_Mapping_Global_Vars.Read_1	Data_Mapping_Global_Vars.Write_1	✔ Started
1000	Data_Mapping_Global_Vars.Read_4	Data_Mapping_Global_Vars.Write_4	✔ Started
1000	Data_Mapping_Global_Vars.Read_5	Data_Mapping_Global_Vars.Write_5	✔ Started
1000	Data_Mapping_Global_Vars2.Read_1	Data_Mapping_Global_Vars3.Write_1	✔ Started
1000	Data_Mapping_Global_Vars2.Read_3	Data_Mapping_Global_Vars3.Write_3	✔ Started

- From the bottom of the **Data Mapping** tab, select **New**.  
A new Data Mapping window appears.

**Mapping**

Priority (ms): 50

Source

Variable: [dropdown]

Type: [dropdown]

Destination

Variable: [dropdown]

Type: [dropdown]

Details

Comment: [text area]

Save Cancel

- Select a value for the **Priority (ms)** parameter.  
This is the frequency that the runtime Device Publisher feature will read the source device variable.  
The units are in milliseconds (ms).

5. Select a Source Variable.  
The list of devices are the Started devices on this node.  
Expand the desired device and its internal structure until you can select the individual source device variable.
6. Select a Destination Variable.  
The list of devices are the Started devices on this node.  
Expand the desired device and its internal structure until you can select the individual destination device variable.
7. If the drivers support data transformation, the data type fields will become active and can be changed to a data type different than the variables' data type.
8. If the source device variable and the destination device variables are arrays, enter the Count of variables to transfer.
9. Use the **Save** button to save the definition of the data mapping.  
The data mapping will appear in the list of data mappings for the node, in a **Stopped** state.

## Viewing data mappings

The **Data Mapping** tab provides a list of data mappings defined on the node.

To use the **Data Mapping** tab, follow these steps:

1. From Workbench left pane, expand the node whose data mapping you want to view.
2. Select the **Devices** icon.  
Then select the **Data Mapping** tab  
Or, alternatively, select the **Data Mapping** sub icon.
3. The **Data Mapping** tab provides a table format that lists the data mappings defined on the node.  
The top section of the **Data mapping** tab provides these columns:

Column	Description
Priority	The priority of the data mapping specified in the millisecond frequency that the runtime Device Publisher will read the source device variable.
Source	The source device variable.

Destination	The destination device variable.
State	The data mapping state, which can be: <b>Started</b> - the data mapping is active. The runtime Device Publisher is reading the source device variable at the priority frequency. <b>Stopped</b> - the data mapping is not active. <b>Disabled</b> - indicating there is a problem communicating with the source device.
Last Executed	Date and time when the data mapping was last executed, including the write to the destination device variable.
Successes	The number of successful data mappings from the source device variable to the destination device variable.
Failures	The number of failed data mapping. This is usually caused by the destination device not being in a <b>Started</b> state.
Overflow	The number of times the runtime Device Publisher cannot process the data mapping in a timely manner. The entire node's definitions of data triggers, variable groups and data mappings defines the load on the runtime Device Publisher. Each of the Priority parameter values defines the list of device variables that need to read and processed that the Priority's millisecond frequency. You might want to change the value set in the Priority parameter.

### Data Mapping status information

The bottom portion of the **Data Mapping** pane provide information concerning the performance of the selected data mapping as follows:

Parameter	Description
Source Device Name	The source device name.
Source Name	The source variable name.
Source Type	The data type of the source device variable.
Source Count	When the source and destination device variables are an arrays, the number of source device variables from the source array to transfer to the destination device variable array.

Source Length/Bit	For String data types, the length of the source device variable. For BOOL data types, the number of bits being mapped.
Priority	The priority frequency in milliseconds that the source device variable is read by the runtime Device Publisher.
Status	A pair of status values. The left side value is the last result code for the read of the source variable or the write of the destination variable. This is normally zero. The right-side value is an internal data mapping status, which is different from the data mapping's <b>State (Started, Stopped, or Disabled)</b> .
Successes	The number of successful data mappings from the source device variable to the destination device variable.
Failures	The number of failed data mapping. This is usually caused by the destination device not being in a <b>Started</b> state.
Successes/Minute	The number of successful data mappings per minute. This is an approximation over the recent time period.
Comment	The comment that was defined with the data mapping.
Destination Device Name	The destination device name.
Destination Name	The destination variable name.
Destination Type	The data type of the destination device variable. This can be different from the source device variable type
Destination Count	When the source and destination device variables are an array, the number of source device variables from the source array to transfer to the destination device variable array.
Destination Length/Bit	For Bit data types, the number of destination device variable bits being mapped.
State	The data mapping state, which can be: <b>Started</b> - the data mapping is active. The runtime Device Publisher is reading the source device variable at the priority frequency. <b>Stopped</b> - the data mapping is not active. <b>Disabled</b> - indicating there is a problem communicating with the source device.
Last State Change	The date and time of the last state change of the data mapping

Last Fired	Date and time when the data mapping was last executed, including the write to the destination device variable.
Overflow	The number of times the runtime Device Publisher cannot process the data mapping in a timely manner. The entire node's definitions of data triggers, variable groups and data mappings defines the load on the runtime Device Publisher. Each of the Priority parameter values defines the list of device variables that need to read and processed that the Priority's millisecond frequency. You might want to change the value set in the Priority parameter.
User	The log in id of the user who started the data mapping. This user's security credentials are used to determine the access rights to the source and destination variables.

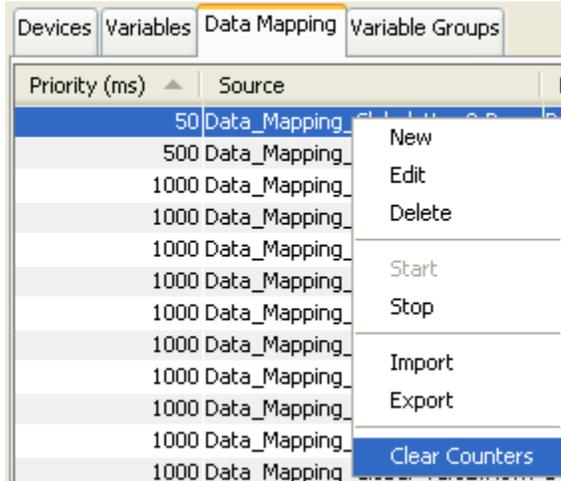
## Controlling data mappings

When a data mapping row is selected in the table, the buttons at the bottom of the **Data Mapping** tab become enabled or disabled. This is based on the current state of the data mapping and the function of the button.

A single data mapping row or multiple data mapping rows can be selected and then the function buttons used, but the state of each data mapping will determine if the function can be performed.

Button	Description
<b>New</b>	Define a new data mapping.
<b>Edit</b>	Edit the data mapping definition. This can be used when the data mapping is in the <b>Stopped</b> state. This is only available for a single data mapping row selection.
<b>Start</b>	Available when the data mapping is in a <b>Stopped</b> state. Change the data mapping to the <b>Started</b> state.
<b>Stop</b>	Available when the data mapping is in the <b>Started</b> or <b>Disabled</b> state. Change the data mapping to the <b>Stopped</b> state.
<b>Delete</b>	Available when the data mapping is in the <b>Stopped</b> state. Delete the data mapping definition from the node.
<b>Refresh</b>	Refresh the information displayed in the <b>Data Mapping</b> tab. The Workbench will periodically refresh the information on its own without the Refresh button being used.

A data mapping row's pop-up menu can be displayed, and the available options selected.



## IIoTA industrial IoT Platform: Defining, viewing, and controlling variable groups

### Overview

Variable groups can be defined, deleted, viewed, and have their state controlled.

The **Variable Groups** feature provides the function to:

- Define and start a named variable group which contains one or more device variables
- Read the device variables in the variable group at a defined frequency
- If any of the device variables has changed since their last read, schedule any triggers with the **Variable Group** trigger event type and the named variable group.

Variable groups are defined on the node where the device variables are accessible.

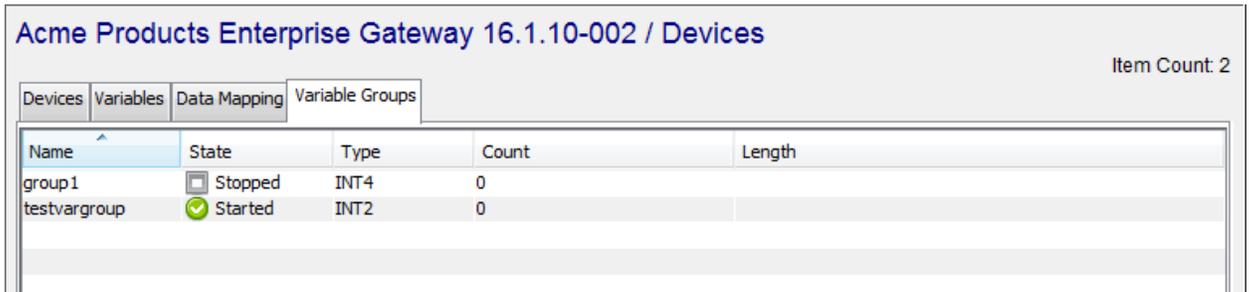
The Variable Group trigger event type and Data trigger event type are similar in that they monitor the change of the value of a device variable (or multiple device variables). The differences include the number of device variables (single or multiple) and the variable value change conditions.

### Defining variable groups

Variable groups are defined after the devices referenced for the device variables are defined and **Started**.

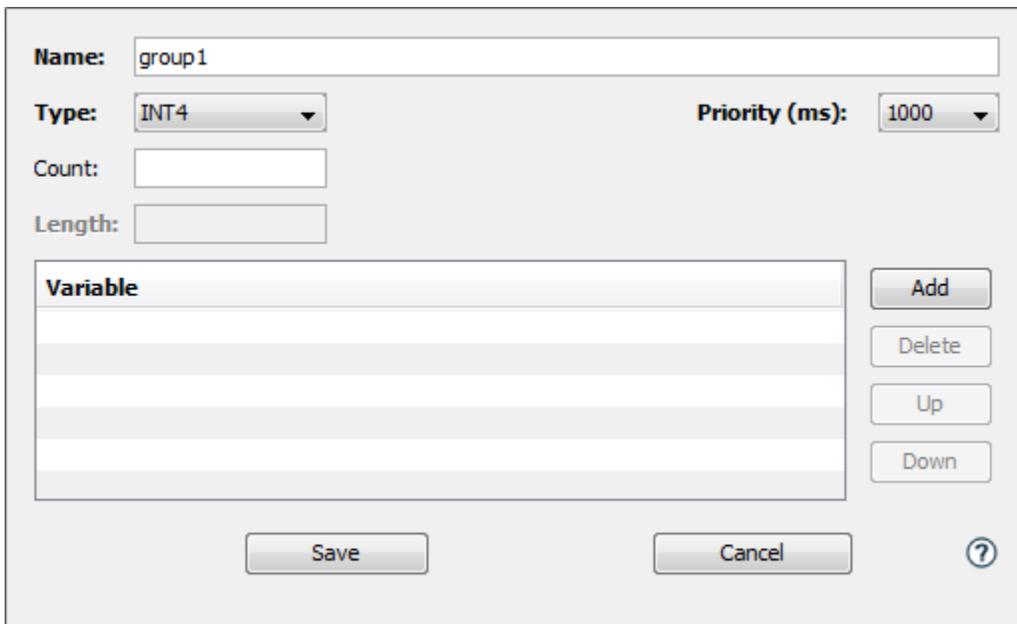
Follow these steps to add a variable group definition to a node:

1. From the left pane, expand the node that you want to add a variable group definition to.
2. Select the **Devices** icon.  
Then select the **Variable Groups** tab.  
Or, alternatively, select the **Variable Groups** sub icon.



The **Variable Groups** window appears as the right pane.  
For this example, the **Variable Groups** window has previously defined variable groups.

3. From the bottom of the **Variable Groups** tab, select **New**.  
A new Variable Group window appears.



4. Enter a name for the variable group.  
This will be used to identify the variable group in the Variable Group trigger event type.

5. Select a value for the **Priority (ms)** parameter.

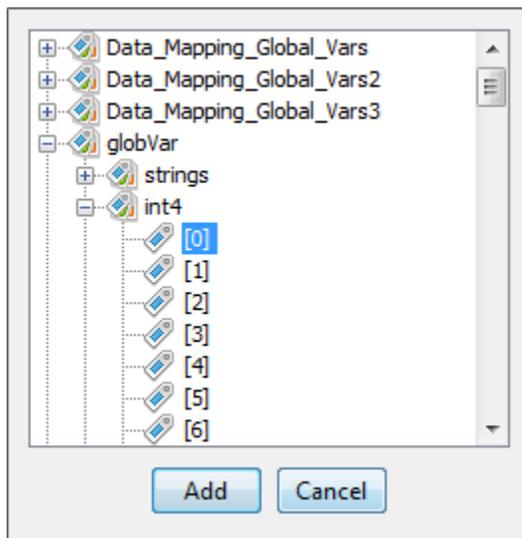
This is the frequency that the runtime Device Publisher feature will read the device variables.

The units are in milliseconds (ms).

6. Select the Add button to add device variables.

The list of devices are the Started devices on this node.

Expand the desired device and its internal structure until you can select the individual device variable.



7. Continue to add device variables to the variable group until you have the complete list desired.

Each variable has a **Key**, which will be available to the Variable Group event type trigger as an event variable. This is set to a default value when the variable is added to the group but can be edited. This variable key can be used as a reference ID, or correlation ID, to help identify which variable's value in the group has changed.

Variables can be deleted from this list, as well as moved up and down in the list. The location in the list has no impact on the evaluation of the variable group, it is solely for list readability for the user.

**Name:** group1

**Type:** INT4 **Priority (ms):** 1000

**Count:** 0

**Length:**

Variable	Key
globVar.int4[0]	Var_01
globVar.int4[12]	Var_02

Item Count: 2

Buttons: Add, Delete, Up, Down, Save, Cancel, ?

- Use the **Save** button to save the definition of the variable group.

The variable group will appear in the list of variable groups for the node, in a **Stopped** state.

## Viewing variable groups

The **Variable Groups** tab provides a list of variable groups defined on the node.

To use the **Variable Group** tab, follow these steps:

- From Workbench left pane, expand the node whose variable groups you want to view.
- Select the **Devices** icon.  
Then select the **Variable Groups** tab  
Or, alternatively, select the **Variable Groups** sub icon.
- The **Variable Groups** tab provides a table format that lists the variable groups defined on the node.

The top section of the **Variable Groups** tab provides these columns:

Column name	Description
Name	The name of the variable group. This will be used to identify the variable group in the Variable Group trigger event type.

State	The variable group state, which can be: <b>Started</b> - the variable group is active. The runtime Device Publisher is reading the device variables at the priority frequency. <b>Stopped</b> - the variable group is not active.
Type	The data type of the device variables.
Count	The number of device variables when the variable is an array.
Length	For String data types, the length of the string.

### Variable group status information

The bottom portion of the **Variable Groups** window provides information for the selected variable group as follows:

Parameter name	Description
Name	The name of the variable group.
Priority	The priority frequency in milliseconds that the device variables are read by the runtime Device Publisher.
Count	The number of device variables when the variable is an array.
Last Modified	The date and time that the variable group was last changed.
User	The log in ID of the user that started the variable group.
Total Runs	The number of device variables in the variable group.
State	The variable group state, which can be: <b>Started</b> - the variable group is active. The runtime Device Publisher is reading the device variables at the priority frequency. <b>Stopped</b> - the variable group is not active.
Type	The data type of the device variables.
Length	For device variables with a String data type, the length of the string.
Last State Change	The date and time of the last state change.
Inactivity	
Use Count	The number of Started Variable Group event type triggers that reference this variable group.

### Variable group Variables section

The Variables section provides information for each of the device variables in the selected variable group as follows:

Column name	Description
Device	The name of the device where the variable resides.
Name	The name of the device variable.
State	The state of the device variable: <b>Active</b> - the variable group is <b>Started</b> , and the device variable can be read. The runtime Device Publisher is reading the device variables at the priority frequency. <b>Inactive</b> - the variable group is not active. <b>Disabled</b> - the variable group is <b>Started</b> , but the device variable cannot be read. This is usually because the device is not in a <b>Started</b> state.
Error	The last error code encountered when reading the device variable.

## Controlling variable groups

When a variable group row is selected in the table, the buttons at the bottom of the **Variable Groups** tab become enabled or disabled. This is based on the current state of the variable group and the function of the button.

A single data variable group row or multiple variable group rows can be selected and then the function buttons used, but the state of each variable group will determine if the function can be performed.

Button	Description
<b>New</b>	Define a new variable group.
<b>Edit</b>	Edit the variable group definition. This can be used when the variable group is in the <b>Started</b> or <b>Stopped</b> state. This is only available for a single variable group row selection.
<b>Start</b>	Available when the variable group is in a <b>Stopped</b> state. Change the variable group to the <b>Started</b> state. The runtime Device Publisher component will read the device variables at the defined priority frequency.
<b>Stop</b>	Available when the variable group is in the <b>Started</b> state. Change the data mapping to the <b>Stopped</b> state. The runtime Device Publisher component will stop

	reading the device variables at the defined priority frequency for this variable group.
<b>Delete</b>	Available when the variable group is in the <b>Stopped</b> state. Delete the variable group definition from the node.
<b>Refresh</b>	Refresh the information displayed in the <b>Variable Groups</b> tab. The Workbench will periodically refresh the information on its own without the Refresh button being used.

A variable group row's pop-up menu can be displayed, and the available options selected.

Acme Products Enterprise Gateway 16.1.10-002 / Devices Item Count: 2

Devices Variables Data Mapping Variable Groups

Name	State	Type	Count	Length
group 1	<input type="checkbox"/> Stopped	INT4	0	
testvargroup	<input checked="" type="checkbox"/> Started	INT2	0	

Context menu options: New, Edit, Delete, References, Start, Stop, Import, Export

### Related Topics

Variable Group

Data

## IIoTA industrial IoT Platform: Device types

### Overview

Each device is supported by a device driver, that handles the device specific:

- Communication protocol
- Variables
- Data type conversions

- Commands
- Unsolicited message support.

This allows you to create your M2M solution using your devices in a vendor neutral methodology. The triggers (application logic) and your enterprise applications can be shielded from many of the device connectivity details, while still providing the two-way device access functions required by your M2M solution.

The information in these sections provides the details for the devices supported by the device drivers, organized by the device drivers.

## Assumptions

The general tasks that apply to all device types are documented in the first several sections of Device connectivity.

It is assumed that you are familiar with those general tasks.

The tasks that apply to building your M2M solution's application logic in triggers are documented in Projects and triggers.

Information related to using trigger actions to access devices and their variables is documented in those sections.

# IIoTA industrial IoT Platform: Aliases device

## Overview

An Aliases device is a *logical* device that references other device variables. The Aliases device type includes the option to:

- Reference multiple device variables that reside in other, multiple, devices defined on the same node.

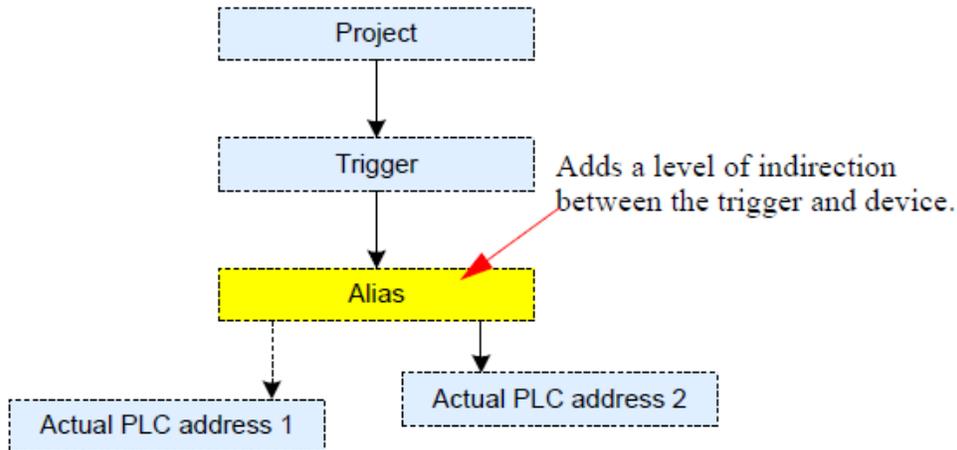
When defining the device variables in the Aliases device, you can define variable names that are different from the actual referenced device variables.

This is both a level of indirection and an aggregation feature.

- Reference a single device, defined on the same node, which then gives access to all of the device variables defined in that referenced device.

This is a level of indirection feature for the Aliases device.

The following illustrates the concept of indirection of a single device variable:



In this example, the trigger references the Aliases device variable name and not the **PLC address 1**. Should the application change and the referenced data move to **PLC address 2**, the trigger does not have to be modified. You simply re-map the Aliases device variable to the new PLC device variable.

Here are other advantages for using an Aliases device:

- If the actual device variable is referenced by many (perhaps tens or hundreds of triggers) and that device variable changes its name or location, the Aliases device variable only has to be re-mapped in a single location, and not hundreds of triggers.

This reduces application maintenance cost.

- An Aliases device can define variables that reference multiple actual devices. In other words, the triggers have a logical view of one device (the Aliases device), while underneath there may be multiple actual devices. Instead of applying security policies to multiple physical devices, you can just apply a security policy to the one Aliases device.

This simplifies the security access control policy definition and allows a straight forward way to provide different users different levels of access to device variables, if required.

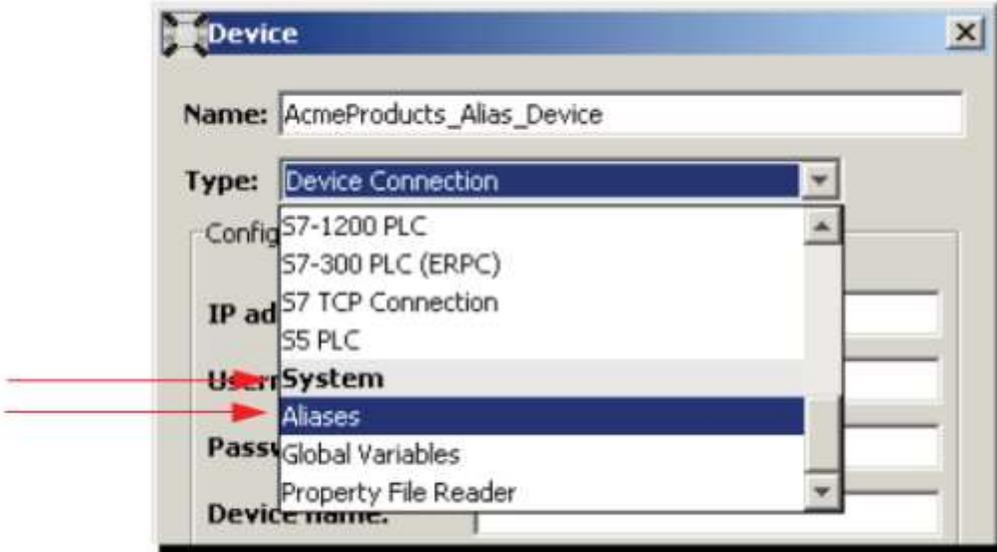
## Defining an Aliases device using the Variable Aliasing Mode

To define an Aliases device using the Variable Aliasing Mode, follow these steps:

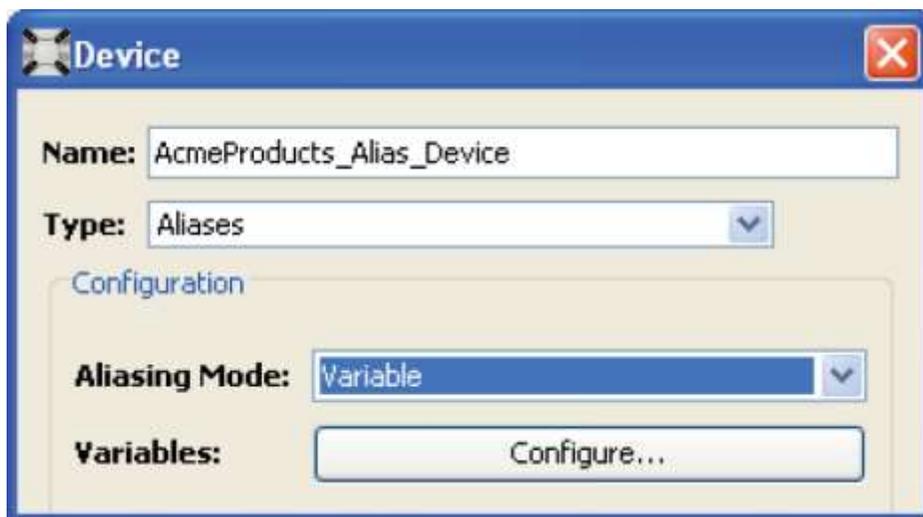
1. From the Workbench left pane, expand the node where you want to define the Aliases device.
2. Select the **Devices** icon to display the Devices panel, right-click the Devices icon to display its pop-up menu, and then select **New**.

You also can select the **New** button at the bottom of the Devices panel.

- Name the device. The device name can be up to 64 characters and include letters, numbers, and the underscore character. Spaces are allowed.
- Use the **Type** down-arrow, locate the **System** category, and then select **Aliases**.



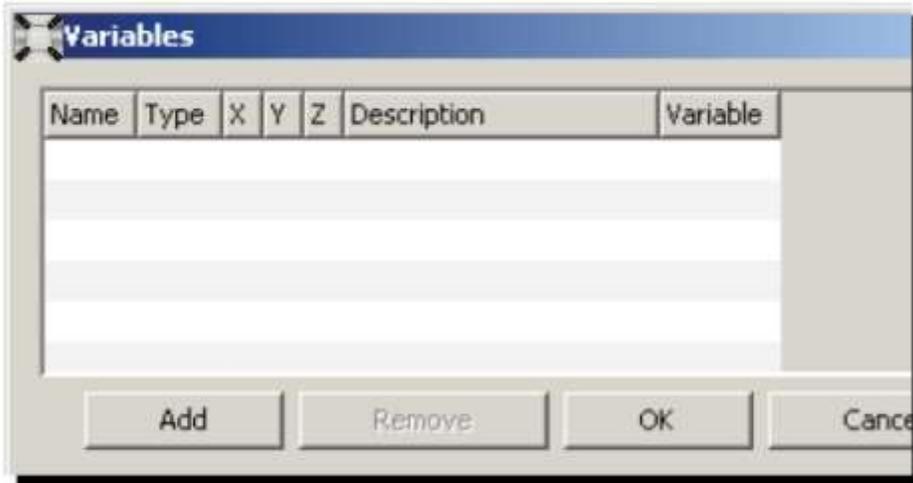
- The Device window changes to accommodate an Aliases device. In the **Configuration** section, select **Variable** for the **Aliasing Mode** parameter. The Device window changes to accommodate the **Variable** option for the **Aliasing Mode** parameter.



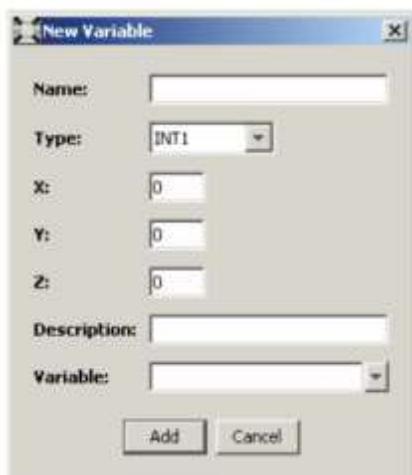
- In the **Configuration** section, next to **Variables**, select **Configure**.



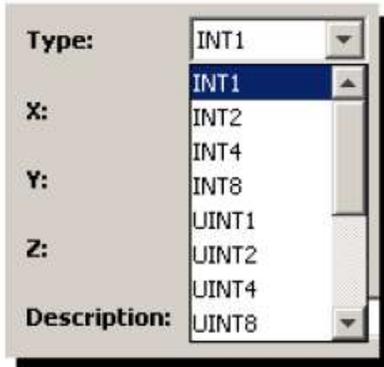
- A Variables window similar to the following appears.



- Select **Add**.  
The New Variable window appears.



- In the **Name** box, type a name for the Aliases device variable.



- Use the **Type** down-arrow to select the type of data the variable will represent. The standard data types are listed. For this example, **INT4** is selected.

The X, Y, and Z values represent the dimensions of an array variable:

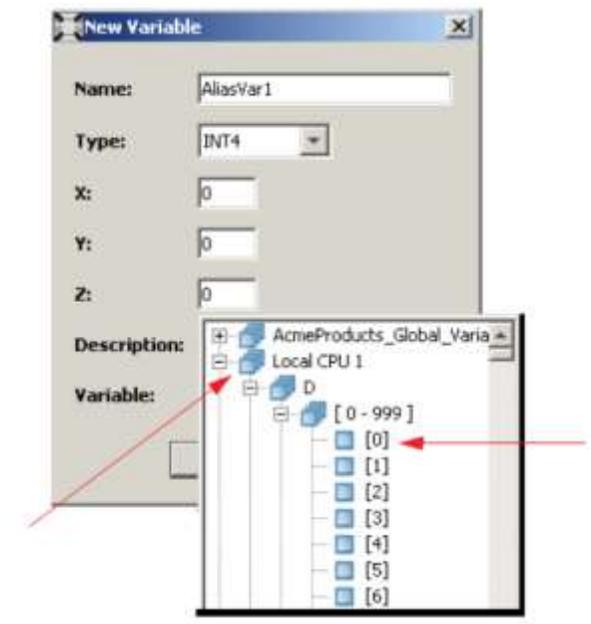
**X** is the first dimension

**Y** is the second dimension

**Z** is the third dimension

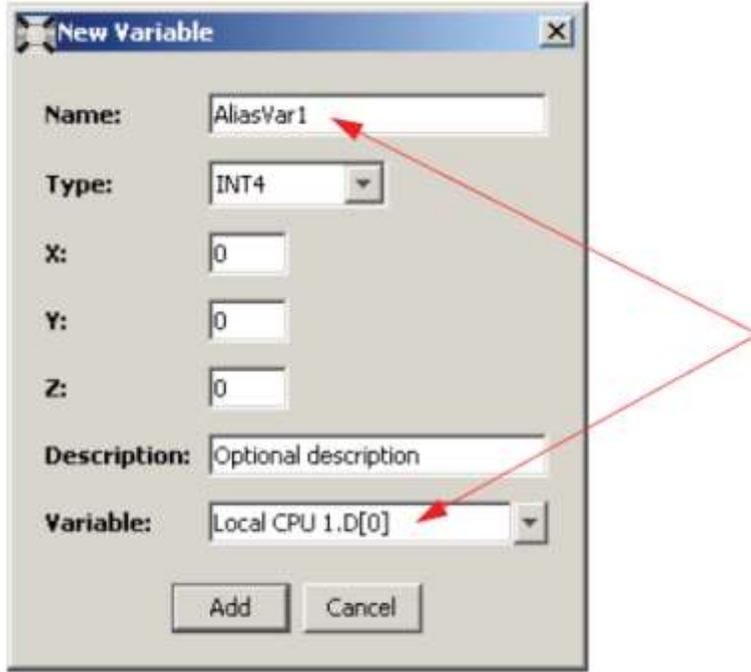
- Type an optional **Description** for the variable.

- Use the **Variable** down-arrow to display a list of the currently started devices on this node.



- Expand the appropriate device's internal structure and variables, and then select the device variable you want to use as the referenced device variable. For this example, Local CPU 1.D[0] is selected.

The value is added to the **Variable** box.



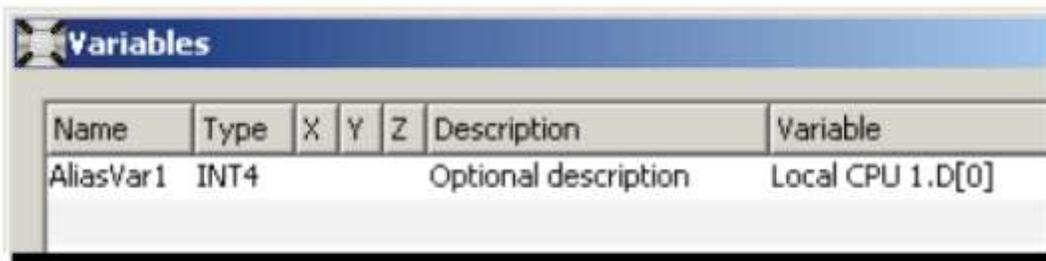
The New Variable window contains a variable called **AliasVar1** which is defined as an INT4.

**AliasVar1** references a device, Local CPU 1, and its variable at D[0]. This example is referencing a Mitsubishi PLC, so the D[0] variable name is using Mitsubishi PLC device variable naming concepts.

When a trigger references the Aliases device **AliasVar1** variable, it is actually referencing the 4 bytes in the Local CPU1 PLC device starting at D[0].

- When finished with this Aliases device variable, select **Add**.

The Variables window re-appears with the parameters you specified for the first variable.



The new variable is added to the Variables window and indicates that Local CPU 1.D[0] will

be accessed as an INT4 in a trigger when the trigger references this Aliases device variable. In this example, AcmeProducts\_Alias\_Device. AliasVar1.

15. Repeat the steps to configure any additional variables for this Aliases device.

The additional variables can reference any of the devices that are started on this node. This feature allows you to aggregate the actual device variables from multiple devices into this one single Aliases device.

16. When finished adding variables, select **OK** to return to the Device window.

17. Select the option for the Per Variable Security parameter:

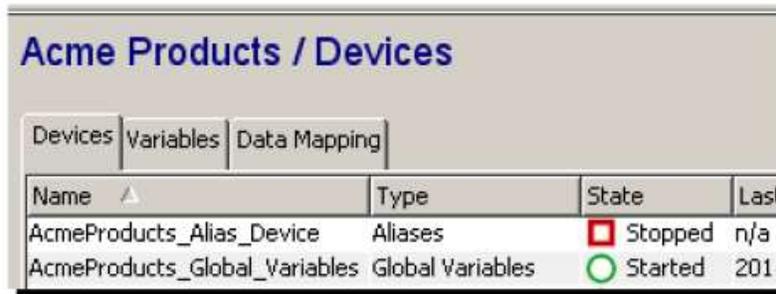
Select **False** to disable the allocation of additional memory to track User to Variable access for all Variables in this Device.

Select **True** to enable this feature if required. For more information, see **Setting up Read Write per device variable**.

18. Select **Save** to save the device definition. The device will appear in the Devices window list of devices.

19. You can now control the device (**Start, Stop**), access the device's variables by using the **Variables** window, and build solutions that use the device's resources.

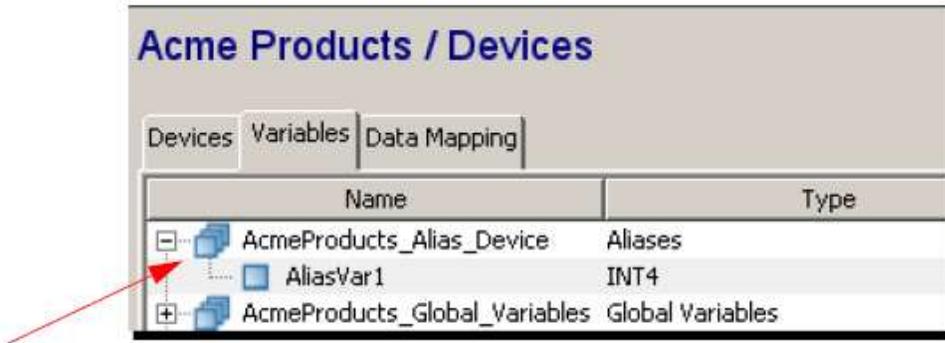
The name of the Aliases variable appears in the Devices tab.



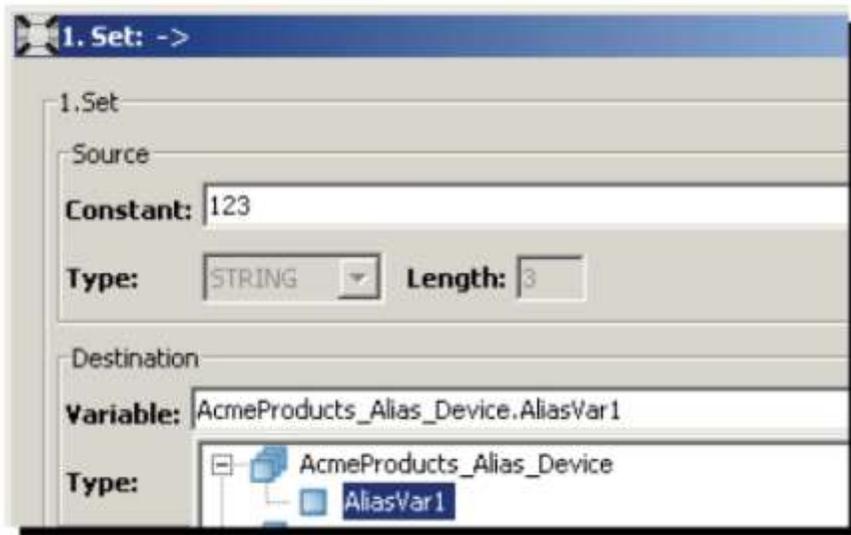
The screenshot shows a window titled "Acme Products / Devices" with three tabs: "Devices", "Variables", and "Data Mapping". The "Devices" tab is active, displaying a table with the following data:

Name	Type	State	Last
AcmeProducts_Alias_Device	Aliases	<input type="checkbox"/> Stopped	n/a
AcmeProducts_Global_Variables	Global Variables	<input checked="" type="checkbox"/> Started	2011

For triggers to be able to access these variables, you must start the Aliases device. When the device is started, you will see the device and its variables in the **Variables** tab.



The following is a partial view of a trigger that uses the Aliases device variable just defined.



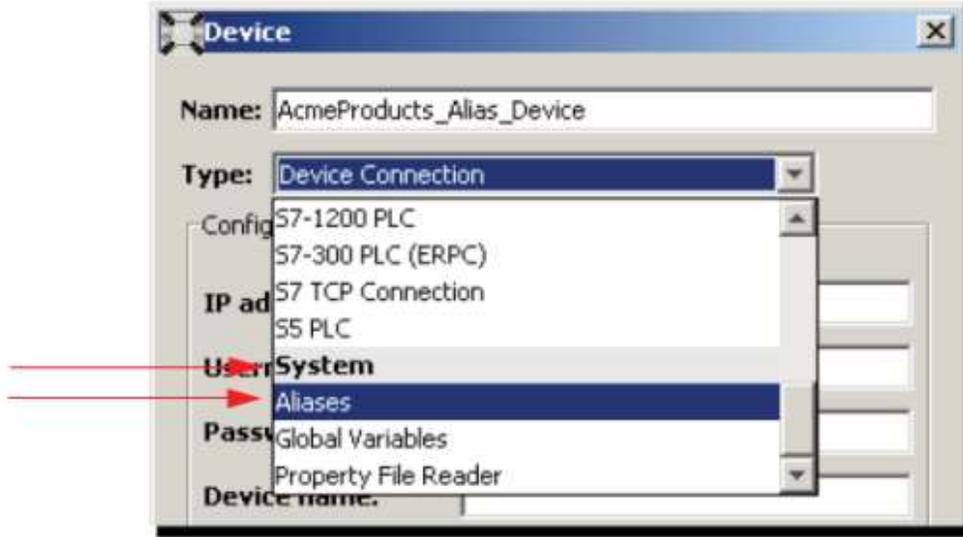
The trigger **Set** action assigns the value 123 to the Aliases device variable **AliasVar1**. This is assigning the value 123 as an INT4 data type to the device variable Local CPU 1.D[0].

## Defining an Aliases device using the Device Aliasing Mode

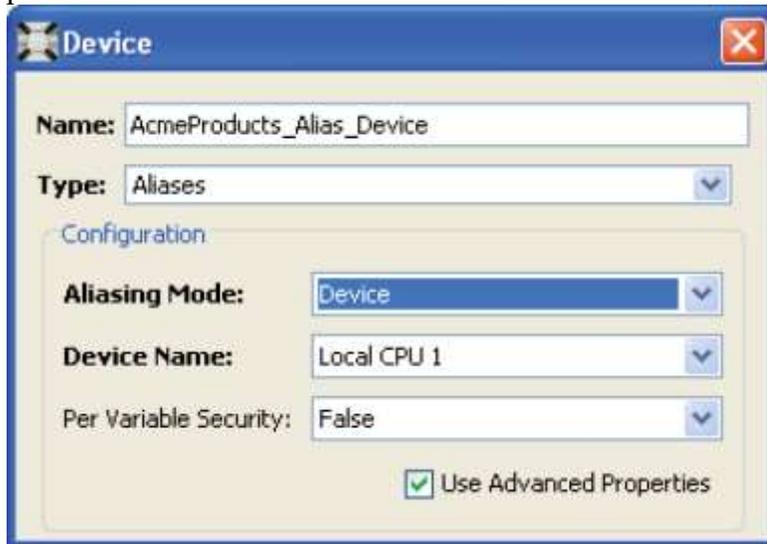
To define an Aliases device using the Device Aliasing Mode, follow these steps:

1. From the Workbench left pane, expand the node where you want to define the Aliases device.
2. Select the **Devices** icon to display the Devices panel, right-click the Devices icon to display its pop-up menu, and then select **New**.  
You also can select the **New** button at the bottom of the Devices panel.

- Name the device. The device name can be up to 64 characters and include letters, numbers, and the underscore character. Spaces are allowed.
- Use the **Type** down-arrow, locate the **System** category, and then select **Aliases**.



- The Device window changes to accommodate an Aliases device. In the **Configuration** section, select **Device** for the **Aliasing Mode** parameter. The Device window changes to accommodate the **Device** option for the **Aliasing Mode** parameter.



- In the **Configuration** section, in the **Device Name** parameter, select the device that will be the referenced device.

7. Select the option for the Per Variable Security parameter:  
 Select **False** to disable the allocation of additional memory to track User to Variable access for all Variables in this Device.  
 Select **True** to enable this feature if required. For more information, see **Setting up Read Write per device variable**.
8. Select **Save** to save the device definition. The device will appear in the Devices window list of devices.
9. You can now control the device (**Start, Stop**), access the device's variables by using the **Variables** window, and build solutions that use the device's resources.

The Aliases device, using Device Aliasing mode, will have reference to all of the device variables in the referenced device.

When a variable is referenced in the Aliases device, it will actually be referencing the corresponding variable in the referenced device.

## Editing and updating an Aliases device

One of the advantages when using Aliases device variable is that the triggers and other components that reference the Aliases device variable have a level of indirection from the actual device variable. Should the actual device variable, you only have to edit the Aliases device variable and not the triggers and other components.

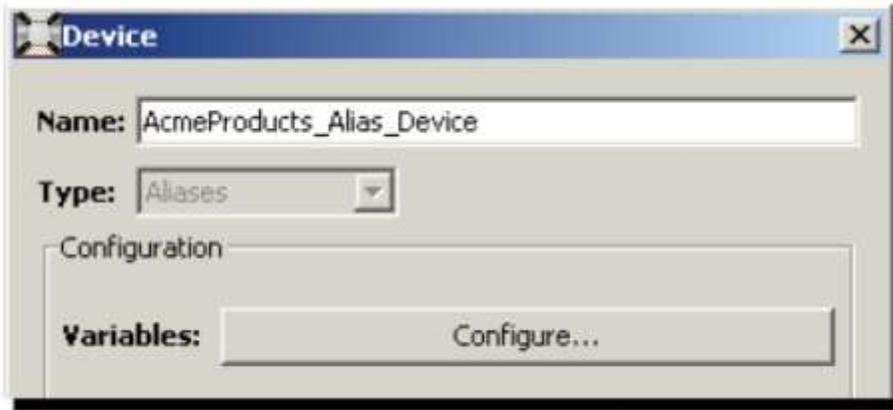
The following describes the steps to edit an Aliases device and have the updates take effect.

1. From the Devices tab, display the pop-up menu for Aliases device you want to update, and then select **Stop**.  
 Or select the Aliases device and then select the **Stop** button at the bottom of the panel.  
 The **State** column for the Aliases device changes to **Stopped**.

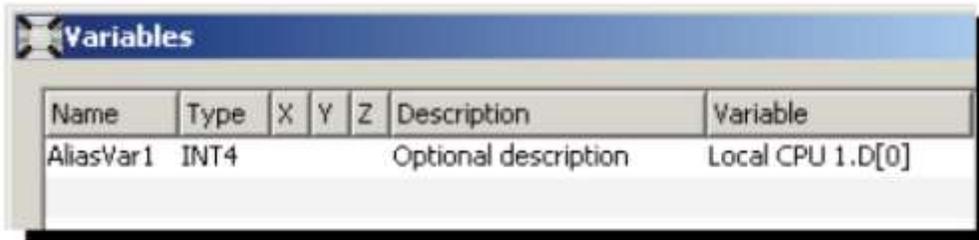


2. Select the Aliases device, display its pop-up menu, and then select **Edit**.  
 Or select the **Edit** button at the bottom of the panel.

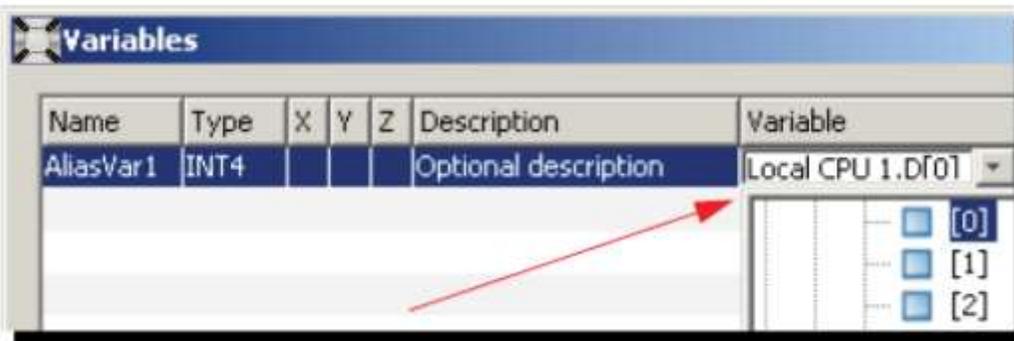
The Device window for the alias variable appears.



- From under **Configuration**, next to **Variables**, select **Configure**.  
A Variables window appears with the defined variables. In this example, there is the one variable we defined.



- Select inside the **Variable** column to display the list of started devices, expand the appropriate device's internal structure and variables, and then select the device variable you want to use as the referenced device variable.

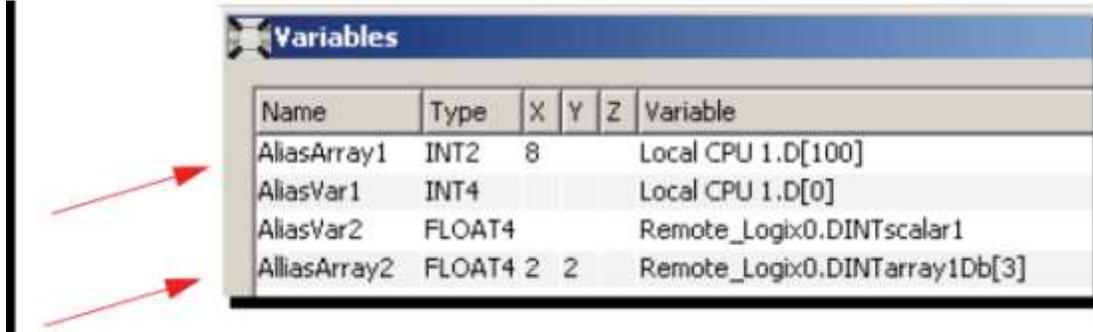


- When changes are complete, select **OK** to return to the Device window, and then select **Save** to save the device definition  
You are returned to the Devices tab. For triggers and other components to access the updated variables, you must start the Aliases device.

## IIoTA industrial IoT Platform: Aliases device example

The following example shows an Aliases device using Variables option for the Aliasing Mode parameter.

The Variables window shows the four variables defined.



The screenshot shows a 'Variables' window with a table of defined variables. Two red arrows point to the first and second rows of the table.

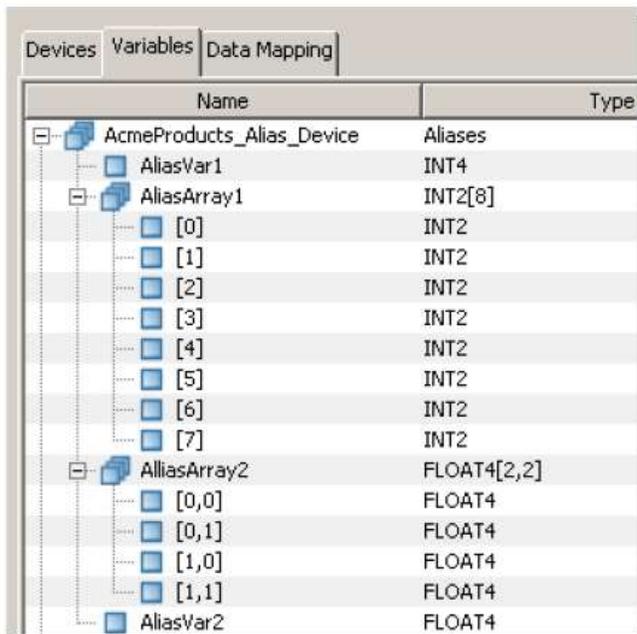
Name	Type	X	Y	Z	Variable
AliasArray1	INT2	8			Local CPU 1.D[100]
AliasVar1	INT4				Local CPU 1.D[0]
AliasVar2	FLOAT4				Remote_Logix0.DINTscalar1
AlliasArray2	FLOAT4 2 2	2	2		Remote_Logix0.DINTarray1Db[3]

The Aliases device is referencing device variables in multiple referenced devices.

- **AliasArray1** and **AliasVar1** are variables that reference two device variables that reside in the Local CPU 1 device, a Mitsubishi PLC.
- **AliasArray2** and **AliasVar2** are variables that reference two device variables that reside in the Remote\_Logix0 device, Rockwell ControlLogix PLC.

Notice the 8 element one dimensional array called **AliasArray1** on the Local CPU 1 device. Also notice the 4 element two dimensional array called **AliasArray2** on the Remote\_Logix0 device.

These Aliases device variables are available in the **Variables** tab as follows:



The screenshot shows the 'Variables' tab in the software interface. It displays a tree view of variables under the 'AcmeProducts\_Alias\_Device' folder. The variables are listed in a table format.

Name	Type
AcmeProducts_Alias_Device	Aliases
AliasVar1	INT4
AliasArray1	INT2[8]
[0]	INT2
[1]	INT2
[2]	INT2
[3]	INT2
[4]	INT2
[5]	INT2
[6]	INT2
[7]	INT2
AlliasArray2	FLOAT4[2,2]
[0,0]	FLOAT4
[0,1]	FLOAT4
[1,0]	FLOAT4
[1,1]	FLOAT4
AliasVar2	FLOAT4

## IIoTA industrial IoT Platform: Global Variables device

A Global Variables device is a *logical* device with no corresponding physical device. The Global Variables device type includes the features to:

- Define scalar (single) variables of a variety of data types
- Define array (3 dimensional) variables of a variety of data types
- Define structures that can be used as a data type to define variables
- Optionally, have the variables' values be memory resident only or have the variables value persisted to disk. A Global variables persisted variables retain their values across the stop and restart of the device.

### Trigger variables

The trigger features: local variables and static variables are also available:

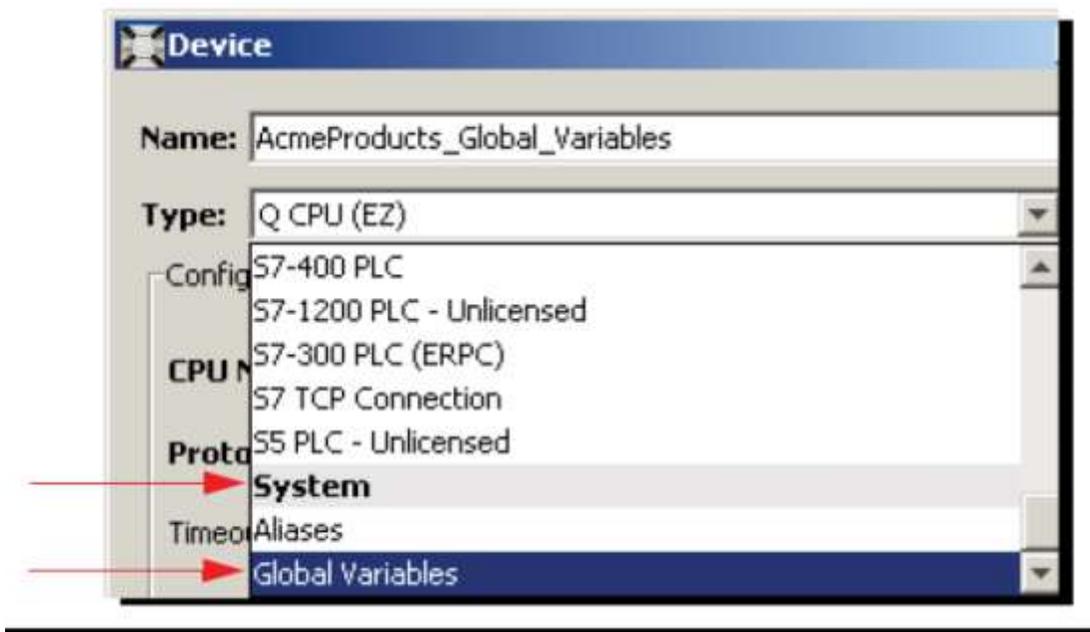
- A local variable is specific to a single execution of a trigger. Each execution instance of the trigger has its own copy of the local variable, including its value.
- A static variable can be used to share data between different executions of the same trigger. The static variable is created when the trigger is started and destroyed when the trigger stops.

For more information, see Trigger local variables, static variables, macros and event variables.

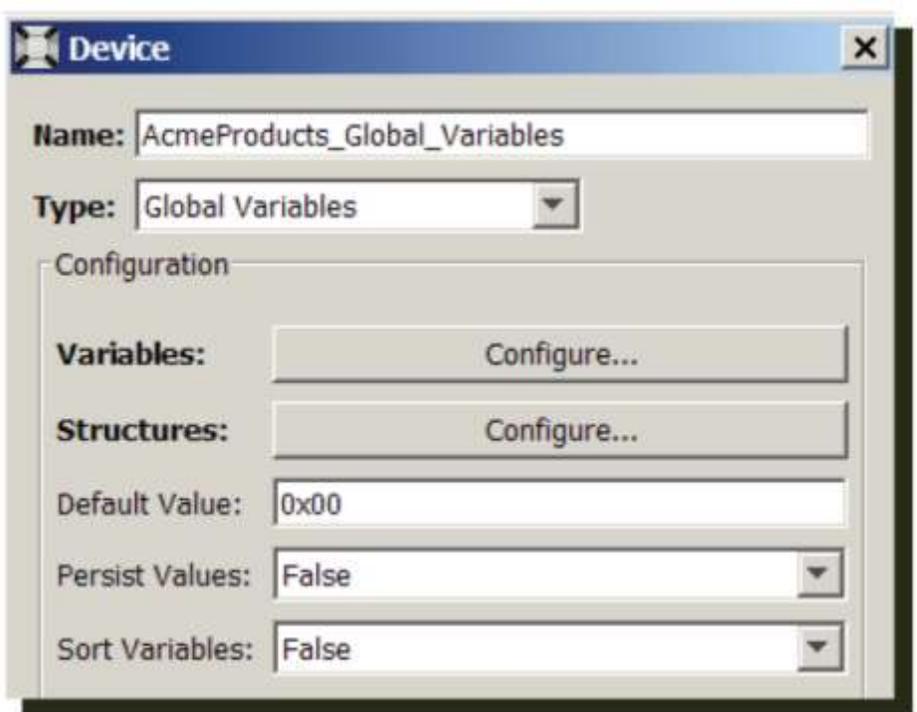
## Defining a Global Variables device

To define a Global Variables device, follow these steps:

1. From the Workbench left pane, expand the node where you want to define the Global Variables device.
2. Select the **Devices** icon to display the Devices panel, right-click the Devices icon to display its pop-up menu, and then select **New**.  
You also can select the **New** button at the bottom of the Devices panel.
3. Name the device. The device name can be up to 64 characters and include letters, numbers, and the underscore character. Spaces are allowed.
4. Use the **Type** down-arrow, locate the **System** category, and then select **Global Variables**.



The Device window changes to accommodate a Global Variables device.



The following parameters are available under the **Configuration** section:

Parameter	Description
<b>Variables</b>	Provides the means to define simple variables in the device (scalars and arrays). For more information, see <a href="#">Configuring Global Variables device variables</a> .
<b>Structures</b>	Provides the means to create complex structures of data. For more information, see <a href="#">Configuring Global Variables device structures</a> .
<b>Default Value</b>	This is the initial memory value of the variables. The default is hexadecimal, but you can use standard decimal notation.
<b>Persist Values</b>	Options are <b>False</b> and <b>True</b> . A value of <b>True</b> retains variable values across the stopping and starting of the Global Variables device and the starting and stopping of the system.
<b>Sort Variables</b>	Options are <b>False</b> and <b>True</b> . A value of <b>True</b> sorts the variables in alphabetic order on the Variables tab when the device is started; otherwise, they will appear in the order that they were created.
<b>Per Variable Security</b>	Select <b>False</b> to disable the allocation of additional memory to track User to Variable access for all Variables in this Device. Select <b>True</b> to enable this feature if required. For more information, see <a href="#">Setting up Read Write per device variable</a> .

5. After adding the desired Variables and Structures, select **Validate** to have the parameters validated.
6. Select **Save** to save the device definition. The device will appear in the Devices window list of devices.
7. You can now control the device (**Start, Stop**), access the device's variables by using the **Variables** window, and build solutions that use the device's resources.

## Exporting and Importing Global Variables device data

When a Global Variables device is exported, persisted values for its variables can be optionally exported. The data values are treated as a separate item that can be included.

When a Global Variables device is imported, its data values, if exported, can be optionally imported.

The default for a Global Variables device with the **Persists Values** parameter defined as **True** is to include the variable data with the export or import.

Data values

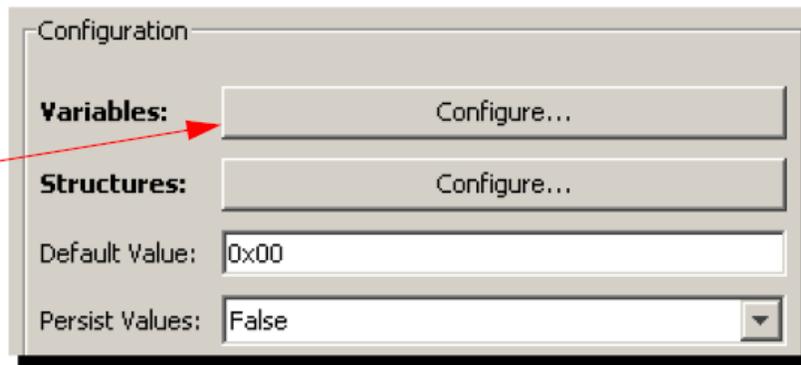
When exporting a Global Variables device with its data values from one node and then importing the Global Variables device and the data values into another node, the Endianness of the nodes must be taken into consideration. For example: exporting from a big endian node (Power PC) and then importing into a little endian node (X86). The numeric data values will not be correctly represented in the imported node.

After the import, the Global Variables device should be edited, and the data values reviewed and modified.

## IIoTA industrial IoT Platform: Configuring Global Variables device variables

When defining or editing a Global Variables device, you configure the variables that the device will contain.

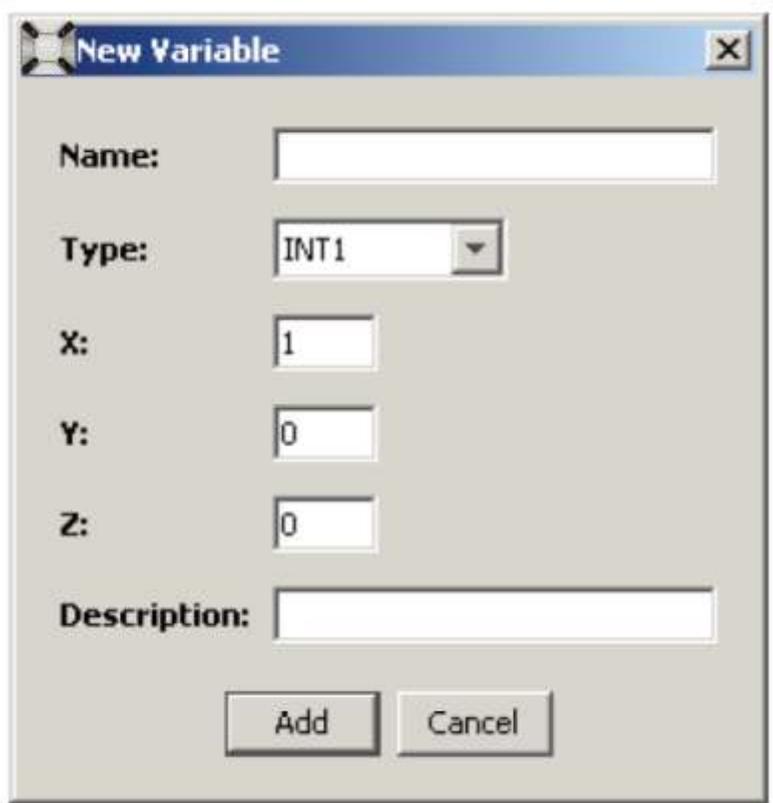
1. From the **Device** window being used to define or edit a Global variables device, next to **Variables**, select **Configure...**



2. A Variables window like the following appears. Select **Add**.



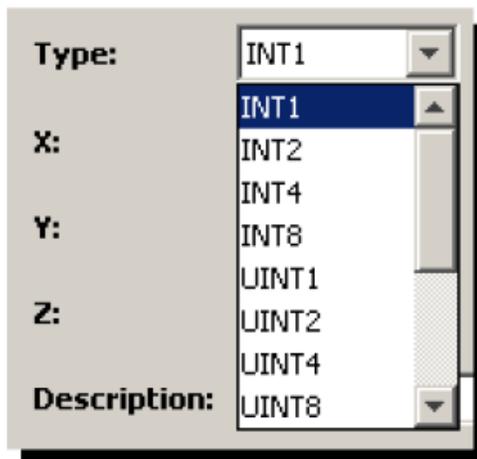
- The New Variable window appears.



The screenshot shows a dialog box titled "New Variable". It has a standard Windows-style title bar with a close button (X). The dialog contains the following fields and controls:

- Name:** An empty text input field.
- Type:** A dropdown menu currently showing "INT1".
- X:** A text input field containing the number "1".
- Y:** A text input field containing the number "0".
- Z:** A text input field containing the number "0".
- Description:** An empty text input field.
- At the bottom, there are two buttons: "Add" and "Cancel".

- In the **Name** box, type a name for the first variable.
- Use the **Type** down-arrow to select the data type of the variable. The standard data types are listed. For this example, **INT4** is selected.



This is a close-up view of the "Type" dropdown menu from the "New Variable" dialog. The menu is open, displaying a list of data types. The "INT1" option is highlighted in blue, indicating it is the currently selected item. The list of options includes:

- INT1
- INT2
- INT4
- INT8
- UINT1
- UINT2
- UINT4
- UINT8

6. The X, Y, and Z values represent the dimensions of the variable.  
For a scalar or simple variable, leave the dimension values blank  
For an array variable:
1. **X** is the first dimension
  2. **Y** is the second dimension
  3. **Z** is the third dimension
- For example:

The 'New Variable' dialog box shows the following configuration:

- Name: FirstGVar
- Type: INT4
- X: 2
- Y: 2
- Z: 2
- Description: Optional description

*FirstGVar* is a three-dimensional array of INT4s. Its size is 2 x 2 x 2 (or 8 elements).

7. Type an optional description for the variable, and then select **Add**.  
The Variables window re-appears with the parameters you specified for the first variable.

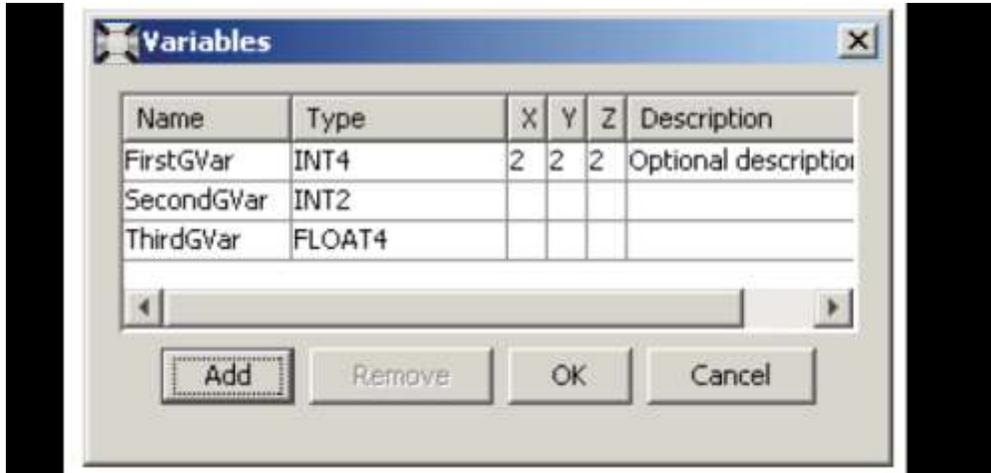
The 'Variables' dialog box displays the following table:

Name	Type	X	Y	Z	Description
FirstGVar	INT4	2	2	2	Optional description

The 'Add' button is highlighted with a red arrow.

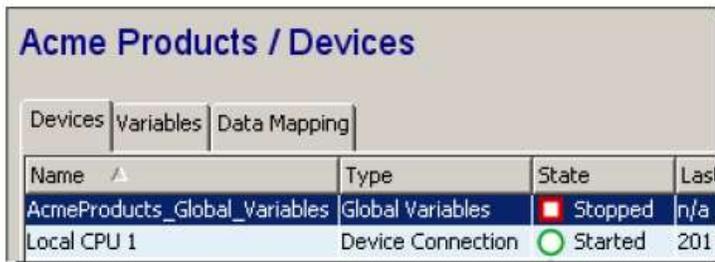
- Repeat the steps to configure any additional variables for this Global Variables device. For this example, two additional variables, an INT2 and FLOAT4 are configured.

The following shows the Variables window with three variables that reside in the device.



- Select **OK** to save the variables configuration.
- Select **Save** to save the device definition.

The Global Variables device is added to the **Devices** tab.



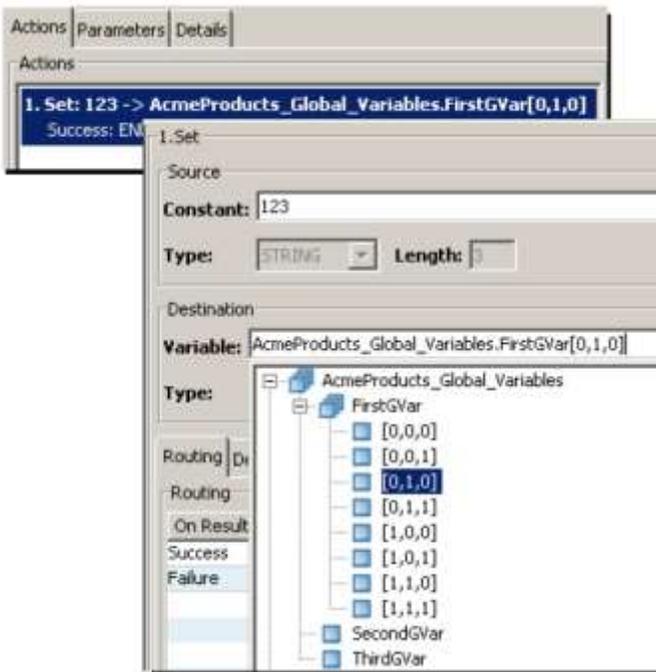
For triggers to be able to access these variables, you must start the Global Variables device. When the device is started, you will see the device and its variables in the **Variables** tab.



Name	Type
AcmeProducts_Global_Variables	Global Variables
FirstGVar	INT4[2,2,2]
[0,0,0]	INT4
[0,0,1]	INT4
[0,1,0]	INT4
[0,1,1]	INT4
[1,0,0]	INT4
[1,0,1]	INT4
[1,1,0]	INT4
[1,1,1]	INT4
SecondGVar	INT2
ThirdGVar	FLOAT4

Notice that FirstGVar is expanded to show the 8 elements of the array.

These variables can be accessed by triggers just as any physical device variables are. The following shows a partial view of a Trigger with a **Set** action being assigned the value of 123 to an element of the previously defined FirstGVar array variable.



The next section, Configuring Global Variables device structures, describes how to configure a user defined structure in a Global Variables device.

## IIoTA industrial IoT Platform: Configuring Global Variables device structures

When defining or editing a Global Variables device, in addition to being able to configure simple variables for the Global Variables device, you can also configure user defined structures. These structures can be a single level, or they can be a multilevel, complex structure.

Once the structure is configured for the Global Variables device, you can configure a variable to use the structure as a data type.

Follow these steps to configure a structure for a Global Variables device.

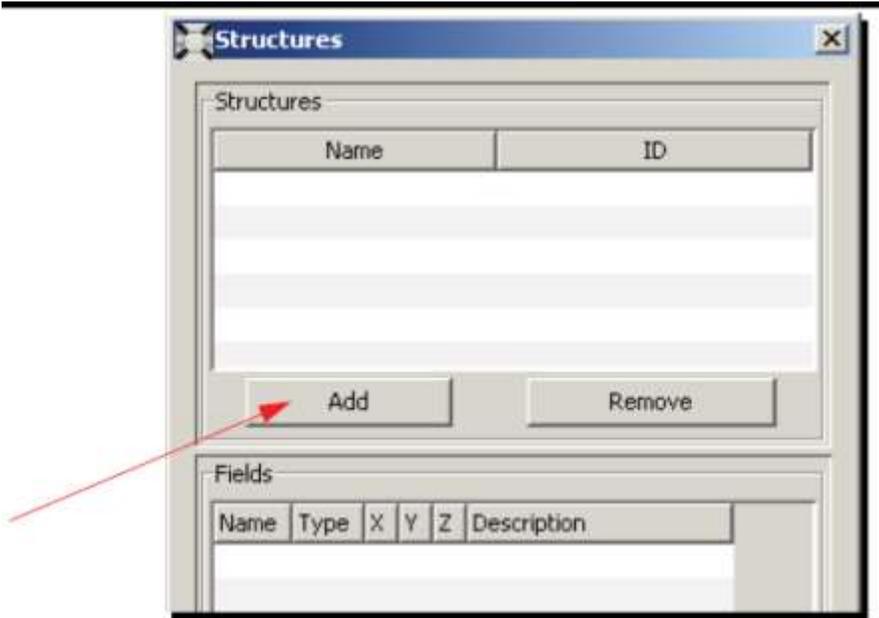
The steps use the example saved **AcmeProducts\_Global\_Variables** device used in the previous configuring variables section.

1. If you are editing a previously defined Global Variables device, make sure the Global Variables device is stopped.
2. From the **Devices** tab, select the appropriate Global Variables device, and then select **Edit**.

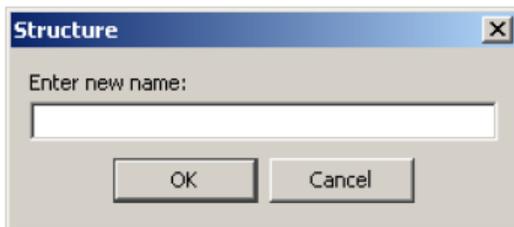
The Device window appears.



- Next to **Structures**, select **Configure**.  
The Structures window appears.



- Select **Add**.  
A window appears where you type the name of the structure.

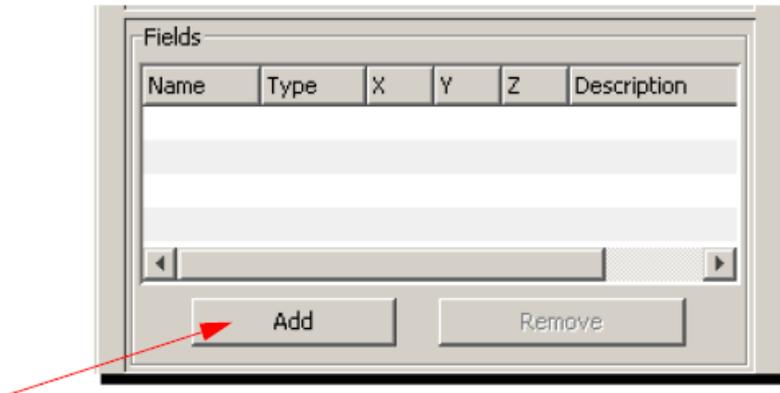


- Type the name (for this example *StructureA*), and then select **OK**.  
The name is added to the Structures window.

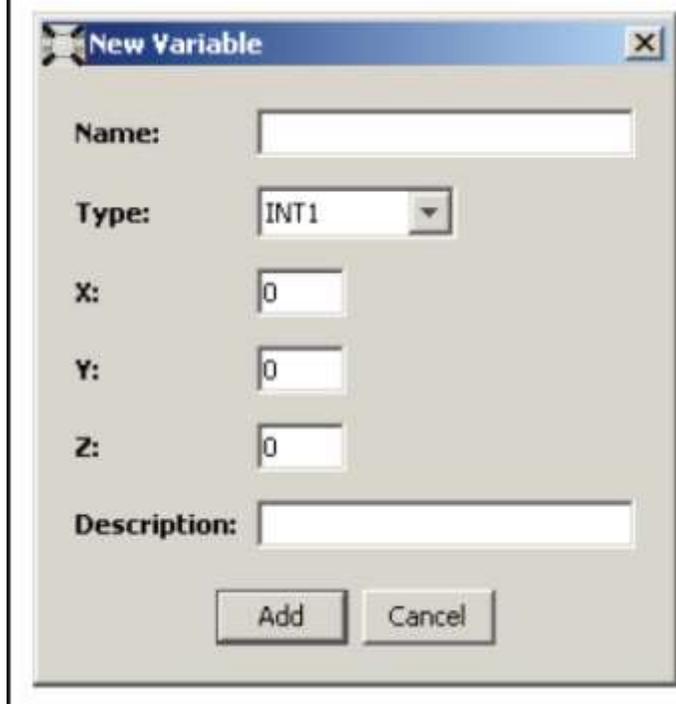


The next step is to define the fields within the structure.

- From the bottom of the Structures window under **Fields**, select **Add**.



The New Variable window appears.



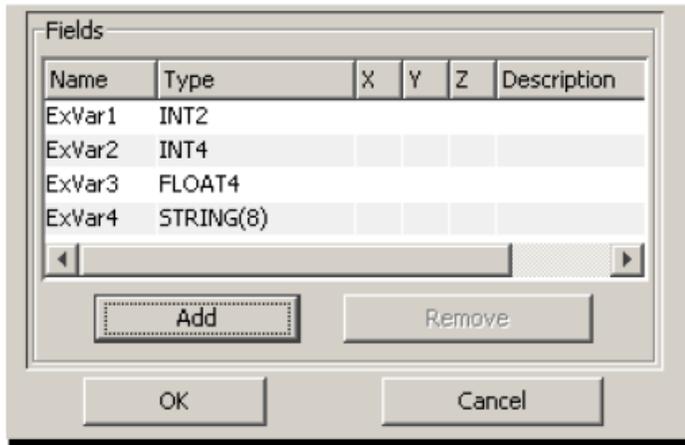
The window is the same as when defining a simple variable; however, now you will be creating a variable field that will be a member of a parent structure.

- Type the name of the variable, and then use the **Type** down-arrow to select the type of data the variable will represent. For this example, the following four fields will be created:

Variable name	Type
ExVar1	INT2
ExVar2	INT4
ExVar3	FLOAT4
ExVar4	STRING of length 8

8. For each field created, select **Add**.

For this example, the completed **Fields** section of the Structures window will appear as follows:



9. Select **OK** after configuring the desired variable fields in the structure.  
The Device window re-appears.

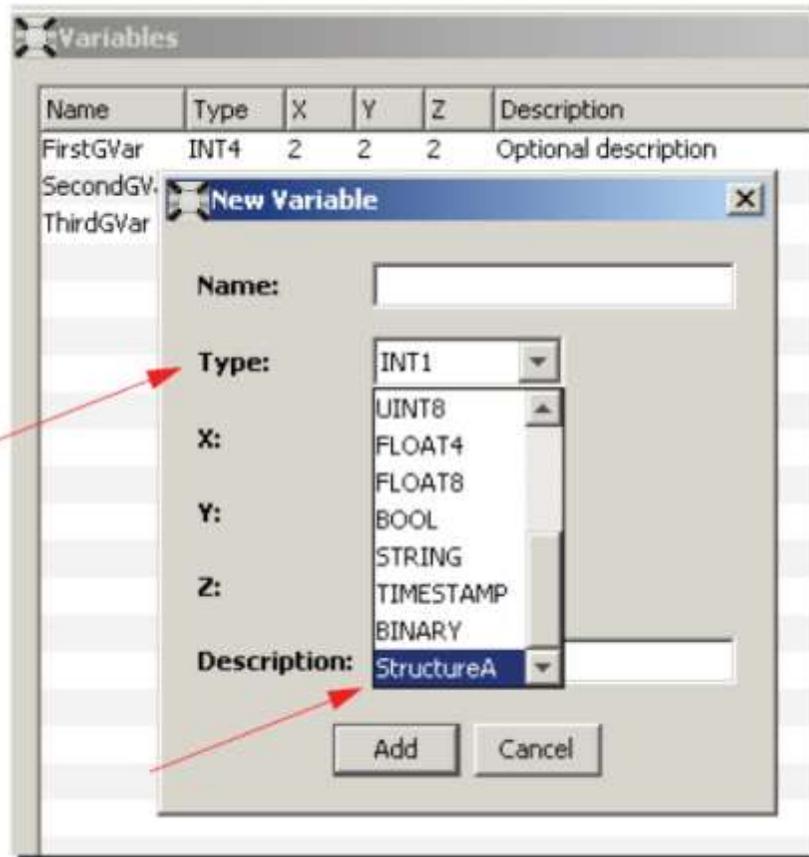
10. Select **Save** to save the definition of the Global Variables device.

Now that you have configured a structure (with four variables), you can configure a variable that uses the structure as its data type. See [Assigning a structure as a data type](#).

## IIoTA industrial IoT Platform: Assigning a structure as a data type

You can think of a user defined structure as a complex data type.

When you configure a structure in a Global Variables device, it becomes available from the New Variables window **Type** drop-down list.



When you configure a new variable, you can assign the structure as the data type.

To create a new variable, follow the steps in Configuring Global Variables device variables; however, the data type to select will be the new structure.

For this example, **StructureGVar** is the variable of type **StructureA**.

When the new variable is configured, and the Global Variables device started, the variable becomes available from the Variables tab.

Acme Products / Devices	
Variables	
Name	Type
AcmeProducts_Global_Variables	Global Variables
+ FirstGVar	INT4[2,2,2]
SecondGVar	INT2
ThirdGVar	FLOAT4
StructureGVar	StructureA
ExVar1	INT2
ExVar2	INT4
ExVar3	FLOAT4
ExVar4	STRING(8)

Notice that the **StructureGVar** variable is composed of four fields that comprise the **StructureA** data type (which was configured as a structure).

## IIoTA industrial IoT Platform: Understanding nested structures

Global Variables device configured structures can be nested. This means a structure can reference another structure.

Follow these steps to configure a structure that references another structure.

The steps use the example saved **AcmeProducts\_Global\_Variables** device used in the previous configuring structures section.

1. If you are editing a previously defined Global Variables device, make sure the Global Variables device is stopped.
2. To create a nested structure  
From the Devices tab, select the appropriate Global Variables device, and then select **Edit**.

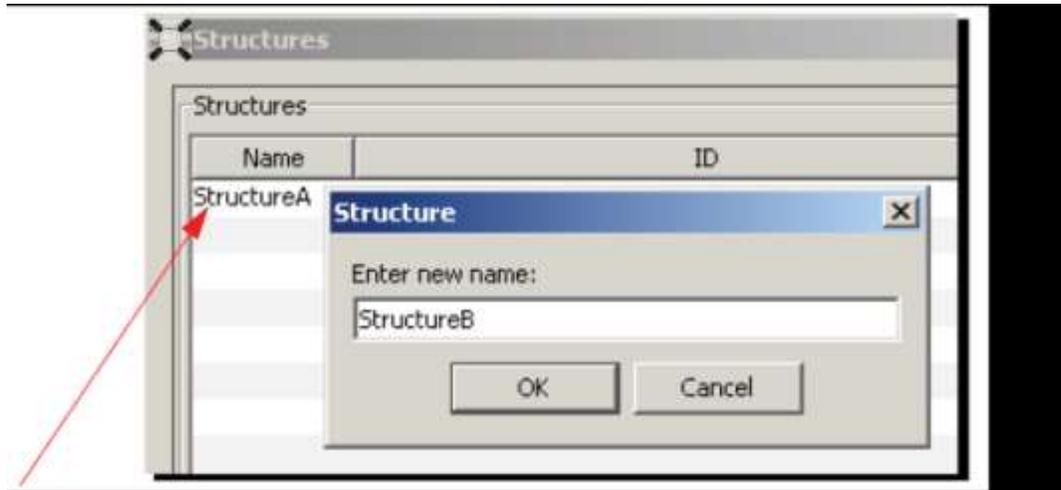
The Device window appears.



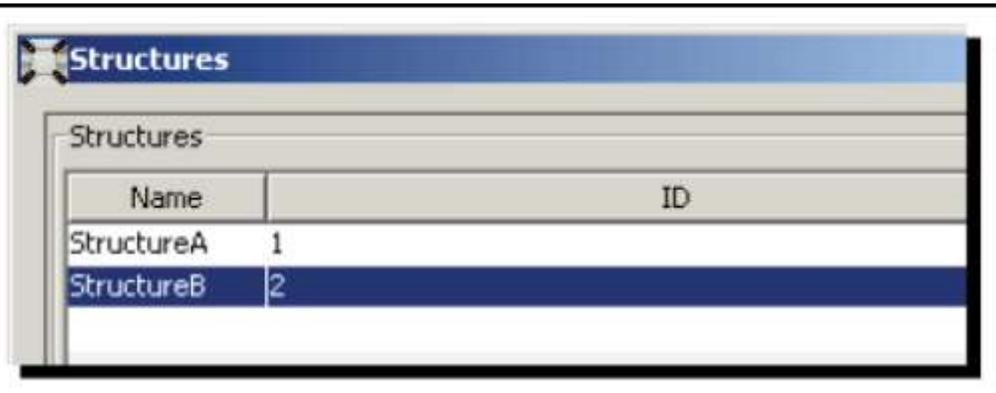
- Next to **Structures**, select **Configure**.

The Structures window appears.

- Select **Add**, type a name for the new structure (for this example, *StructureB*), and then select **OK**.



The name is added to the Structures list.



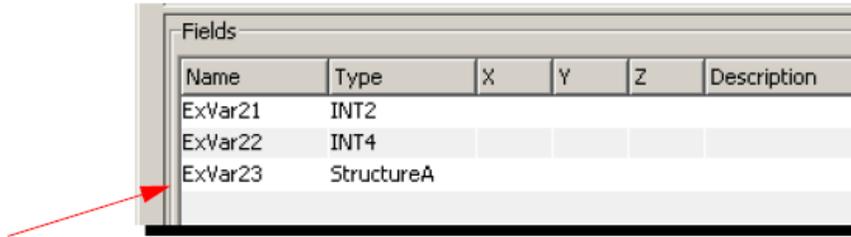
- From the bottom of the Structures window under **Fields**, select **Add**.  
The New Variable window appears.

- Type the name of the variable, and then use the **Type** down-arrow to select the type of data the variable will represent. For this example, the following three fields are added:

Variable name	Type
ExVar21	INT2
ExVar22	INT4
ExVar23	StructureA

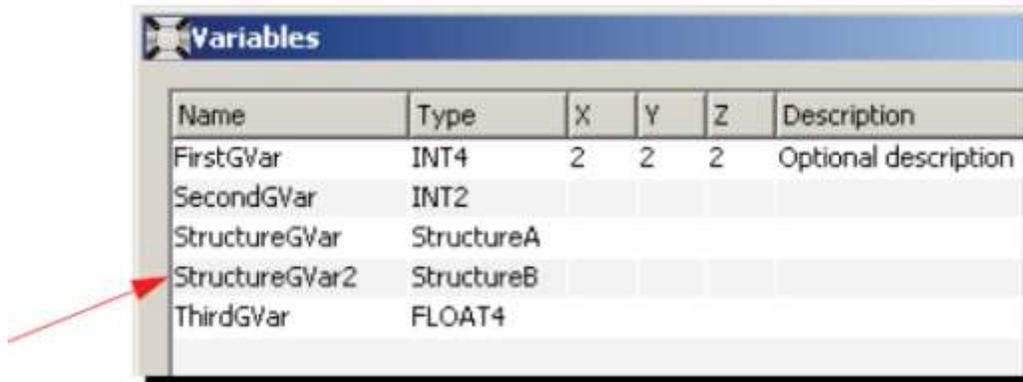
Notice that the data type for **ExVar23** is another structure.

- For each field created, select **Add**.  
For this example, the completed **Fields** section of the Structures window for **StructureB** will appear as follows:



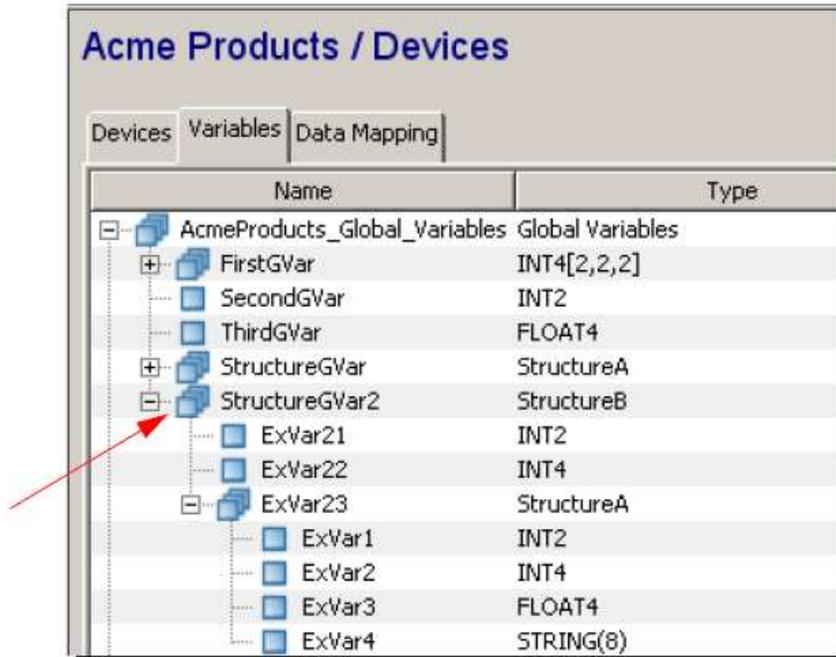
Name	Type	X	Y	Z	Description
ExVar21	INT2				
ExVar22	INT4				
ExVar23	StructureA				

8. Select **OK** to save the definition of the variable fields.  
The Device window re-appears.
9. Select **Save** to save the definition of the device.  
Now that you have configured a second structure, you can configure a variable called *StructureGVar2* that is of type **StructureB**.
10. To configure this variable, refer to Configuring Global Variables device variables; however, the data type to select will be the new structure, **StructureB**.  
The new variable will appear in the Variables window as shown.



Name	Type	X	Y	Z	Description
FirstGVar	INT4	2	2	2	Optional description
SecondGVar	INT2				
StructureGVar	StructureA				
StructureGVar2	StructureB				
ThirdGVar	FLOAT4				

When the Global Variables device is started, the structure becomes available from the **Variables** tab.



The **Variables** tab shows the new variable **StructureGVar2** of type **StructureB**, which is composed of two scalar variables and a variable of type **StructureA**.

## IIoTA industrial IoT Platform: TCP Listener

A TCP Listener device supports the receiving of TCP messages from partner TCP applications. The TCP Listener device type includes the features to:

- Define the port to be used when listener for inbound TCP connections from partner TCP applications.
- Define the maximum number of concurrent connections to support from partner TCP applications.
- Define the incoming data format.
- Define the mechanism to recognize a complete TCP message.

The TCP driver and the TCP Listener device does not support sending data back to the partner TCP application that sent the TCP message.

The trigger events that are supported by the TCP Listener device are:

- Receive TCP Message
- TCP Listener Status

## IIoTA industrial IoT Platform: TCP Listener device definition

The task of device definition fits into the overall process as follows:

1. Installing your hardware and the partner TCP applications that will send TCP messages to your IIoTA node.

The detailed installation of your partner TCP applications is beyond the scope of this documentation.

2. Installing the runtime software and Workbench.
3. Ensure the node has the TCP driver package installed and the required license to enable the TCP driver functions.

1. For information on how to add the TCP driver package, refer to System Administration

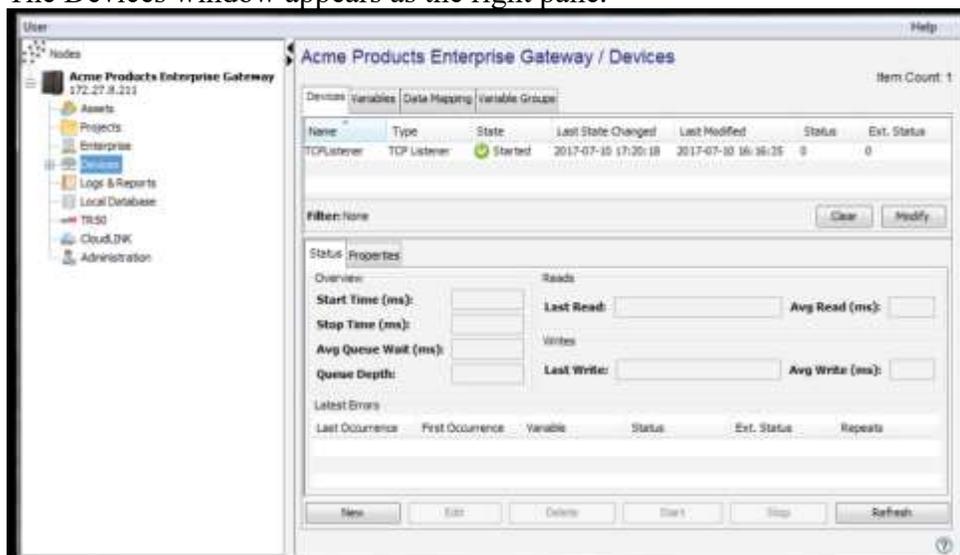
## IIoTA industrial IoT Platform: Using the Workbench to define a TCP Listener device

### Defining a TCP Listener device

To define a TCP Listener device, follow these steps:

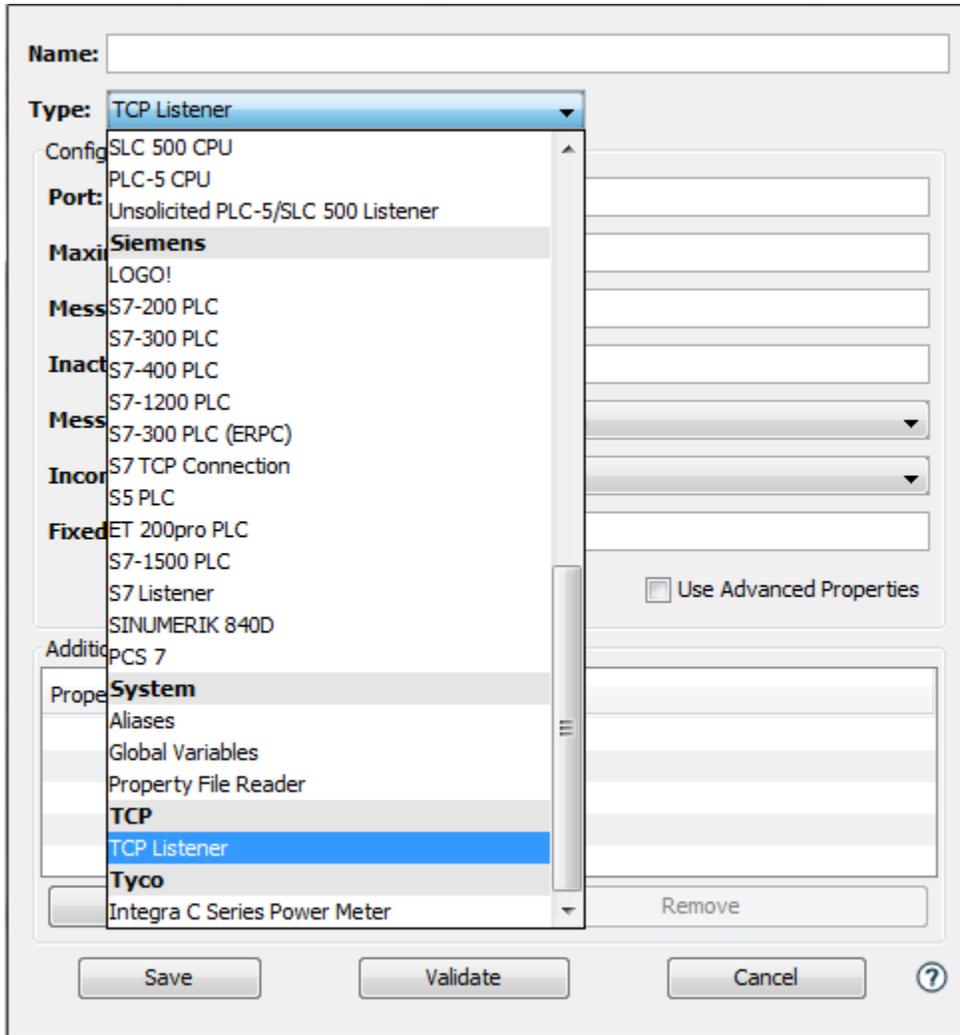
1. From the Workbench left pane, expand the node where you want to define the device.
2. Select the **Devices** icon.

The Devices window appears as the right pane.



The Devices window provides a table format that lists the previously defined devices.

3. To define a new device, select **New** at the bottom of the pane.  
The Device window appears. The available device types are determined by the device support that is installed in this node.
4. Use the **Type** down-arrow to select the TCP Listener device under the **TCP** group.



5. The Device window changes to accommodate the selected device type.

6. To define a TCP Listener device, set this new device’s parameters as follows:

Parameter	Description
<b>Name</b>	Enter a name for the TCP Listener device.
<b>Port</b>	The port on which to listen for TCP messages. The valid values are 1 - 65535. The default value is 6000.
<b>Maximum Concurrent Connections</b>	The maximum number of clients that will be connected concurrently. The default is 10.
<b>Message Timeout (sec)</b>	The timeout while waiting for the next chunk of a TCP message. The default is 30 seconds.
<b>Inactivity Timeout (sec)</b>	The timeout while waiting for the next TCP message. The default is 300 seconds.

<b>Message Type</b>	The Message Type indicates how to recognize the TCP data received as individual messages. - <b>Fixed Length</b> - <b>Variable length with Terminating Characters</b> - <b>Message Header</b>
<b>Incoming Data Format</b>	The type of the incoming TCP data - <b>Binary</b> - <b>String</b>
<b>Fixed Length (bytes)</b>	Available when the Message Type is <b>Fixed Length</b> . The size of data to receive to indicate a complete TCP message. The valid values are 1 - 64000.
<b>Terminating Characters</b>	Available when the Message Type is <b>Variable Length with Terminating Characters</b> . The character or multi-byte terminating string used to indicate a complete TCP message. The default value is blank.
<b>Fixed header Size (bytes)</b>	Available when the Message Type is <b>Message Header</b> . The size of the fixed header in bytes. The valid values are 1 - 64000. The default value is 1.
<b>Length Offset (bytes)</b>	Available when the Message Type is <b>Message Header</b> . The bit offset of the TCP message size field in the message header. It does not include the Fixed Header Size. The value values are 0 - 63999. The default value is 0.
<b>Length Size (bytes)</b>	Available when the Message Type is <b>Message Header</b> . The size of the TCP message size field in the message header. The valid values are 1 - 4. The default value is 1.
<b>Endianess</b>	Available when the Message Type is <b>Message Header</b> . The order of the bytes within the message header. The valid values are Little or Big. The default value is Little.
<b>Per Variable Security</b>	Select <b>False</b> to disable the allocation of additional memory to track User to Variable access for all Variables in this Device. Select <b>True</b> to enable this feature if required. For more information, see <b>Setting up Read Write per device variable</b> .

7. Select **Validate** to have the parameters validated and temporarily Bind to the TCP port. If there is a problem with the validation, an error code will be displayed.

8. Select **Save** to save the device definition. The device will appear in the Devices window list of devices.
9. You can now control the device (**Start, Stop**), access the device's variables by using the **Variables** window, and build solutions that use the device's resources.

## IIoTA industrial IoT Platform: Using the Variables window to access TCP Listener device variables

Once you have defined your devices, you can use the Workbench to access the variables that reside in those devices.

1. From the Workbench left pane, select the **Variables** icon. The **Variables** window appears as the right pane.



The **Variables** window provides a tree format that lists the started devices in the node. The devices must be in the **Started** state to be included in the **Variables** window. Each device can be expanded to show the variables that reside in that device.

TCP Listener device type variables display connection and total messages related information.

TCP Listener device variable	Description
Active Connections	The number of connections (partner TCP applications) currently connected to the TCP Listener device.
Last Connection Timestamp	The timestamp of the most recent connection.

Total Connections Accepted	The total number of connections that have been accepted since the TCP Listener device was started.
Total Connections Closed	The total number of connections that have been accepted since the TCP Listener device was started.
Total Connections Rejected	The total number of connections attempts that the TCP Listener device has rejected.
Total Messages Rejected	The total number of messages received by the TCP Listener device that could not be processed.
Total Messages Received	The total number of messages received by the TCP Listener device since it was started.

## IIoTA industrial IoT Platform: TCP Listener troubleshooting

For TCP Listener device troubleshooting, the following sections list common tasks and problems.

### Common TCP Listener tasks and problems

#### Verify the proper license keys are installed

The features and functions available in a node are controlled by the licenses that are installed in the node.

To verify that the appropriate device license is installed:

1. From the Workbench left pane, expand the node whose license you want to check, and then select the **Administration** icon.
2. From the Administration window, select the **Licenses** tab.
3. Select the appropriate license. Details of the license appear on the tab.
4. View the **Features** column to identify the device driver.

If you do not see the correct license, or if the license is expired, you must request a license from your license key provider.

For information on how to install a license, refer to System Administration Licenses.

### **Active device license count exceeded**

The number of active partner TCP application connections has exceeded the total active device license count. The connection from the partner TCP application has been rejected by the TCP Listener device.

A system generated Alert will be displayed when the number of active device licenses in use exceeds a threshold percentage of the total active device licenses for the node. The Alert is cleared when the usage drops below the threshold percentage. For example:

"The system is currently using 90% of the available 20 device licenses"

An Exceptions Log message is inserted for the first connection rejection. For example:

"Device connection rejected due to insufficient device licenses"

### **Unable to define a TCP Listener device**

The TCP Listener device support is not part of this node's installation. The TCP Listener device support is installed as a package separate from the base product installation.

You may be working with different levels of nodes, each with different levels of support for devices. Ensure that this node is the one with support for the TCP Listener device.

### **Unable to Validate or Start a TCP Listener device**

The default TCP port used by a TCP Listener device is 6000, which can be configured to use another port. If you have another TCP Listener device defined that is using the port, or if another service is using the port, the TCP Listener device will not be able to start. In this situation you will either need to free this port or configure the TCP Listener device to use a different port. If a different port is selected, ensure that the partner TCP applications that are connecting to TCP Listener device are aware of the new port designation.

### **The partner TCP application's connection to a TCP Listener device won't go away.**

The TCP Listener device allows you to define a maximum number of partner TCP applications that can be connected at one time. The TCP driver keeps a running count of how many partner TCP applications are actively connected while it is running. TCP partner applications that attempt to connect to a TCP Listener device once the maximum connection count has been reached will be rejected by the TCP driver. The active connection value is decremented when a partner TCP application closes its socket connection to the TCP Listener device, which will allow a new partner TCP application to connect, in the scenario where the maximum connection count has been reached.

The TCP driver, running on the Linux operating system, may not be notified of socket error conditions that originate from the partner TCP application. If the TCP driver is not notified of these error conditions, then the TCP Listener device will not decrement the active connection count when the partner TCP application error occurs. An example of this would be a partner TCP application whose Ethernet cable becomes disconnected after it had previously connected to the TCP driver on a Linux node. Linux does not raise this disconnected socket error condition to the TCP driver, so the TCP driver has no way of knowing this partner TCP application is no longer connected. If the TCP Listener device has reached its maximum number of connections limit, no further connections can be accepted until the Linux operating system notifies the TCP driver of

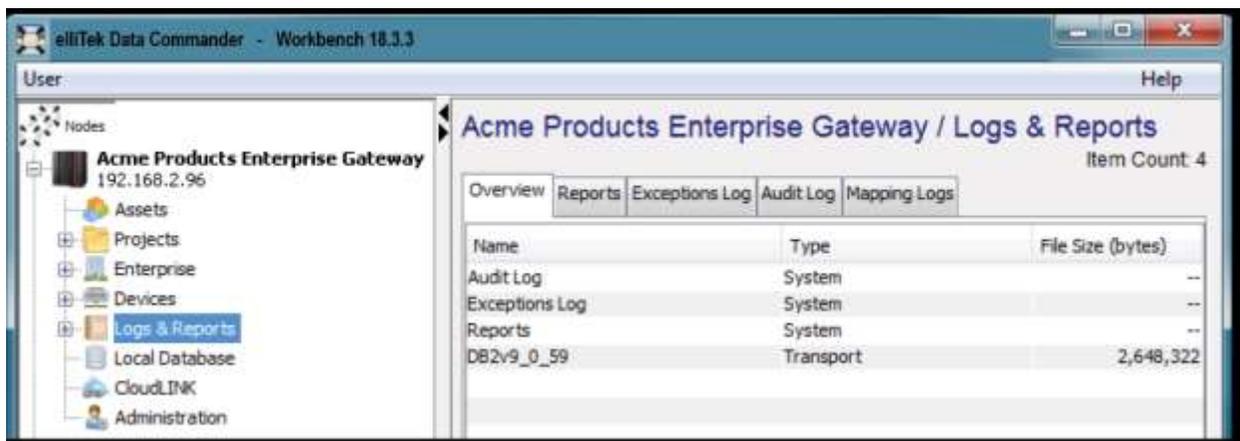
the invalid partner TCP application connection. Once the TCP driver has been notified, it will close the connection and decrement the active connections counter, which will make available an additional connection.

## IIoTA industrial IoT Platform: Logs & Reports

### Overview

The **Logs & Reports** feature provides access to the system audit and exception log files, transaction mapping log files, and trigger report file that are maintained on a node.

When you select the **Logs & Reports** icon from the Workbench left hand pane, the Logs & Reports window appears in the right-hand pane with individual tabs for each type of log file. All the log and report files are stored on the node. The Workbench provides filters when viewing the data to limit which log messages are displayed.



### Types of logs

The **Overview** tab provides a table with columns that indicate the name, type, and for transaction mapping logs the size of the log.

The Audit Log and Exceptions Log receive messages from the system components when events occur.

The Reports Log receives trigger reports when the reporting option is configured for a trigger.

The Transaction Server Mapping Logs option is configured in the transports and listeners definitions and only applies to an Enterprise Gateway.

### Highlights

This guide contains the following:

- Exceptions Log
- Audit Log
- Reports
- Mapping Logs

#### Related Topics

Automated Log Export

## IIoTA industrial IoT Platform: Exceptions Log

When a system component experiences an exception condition or event, it is recorded in the Exception Log.

These events can be informational or error in nature, for example:

- The node is started (with the node version level)
- A device has a communication problem
- A trigger is disabled
- A transport has a problem communicating with a database server.

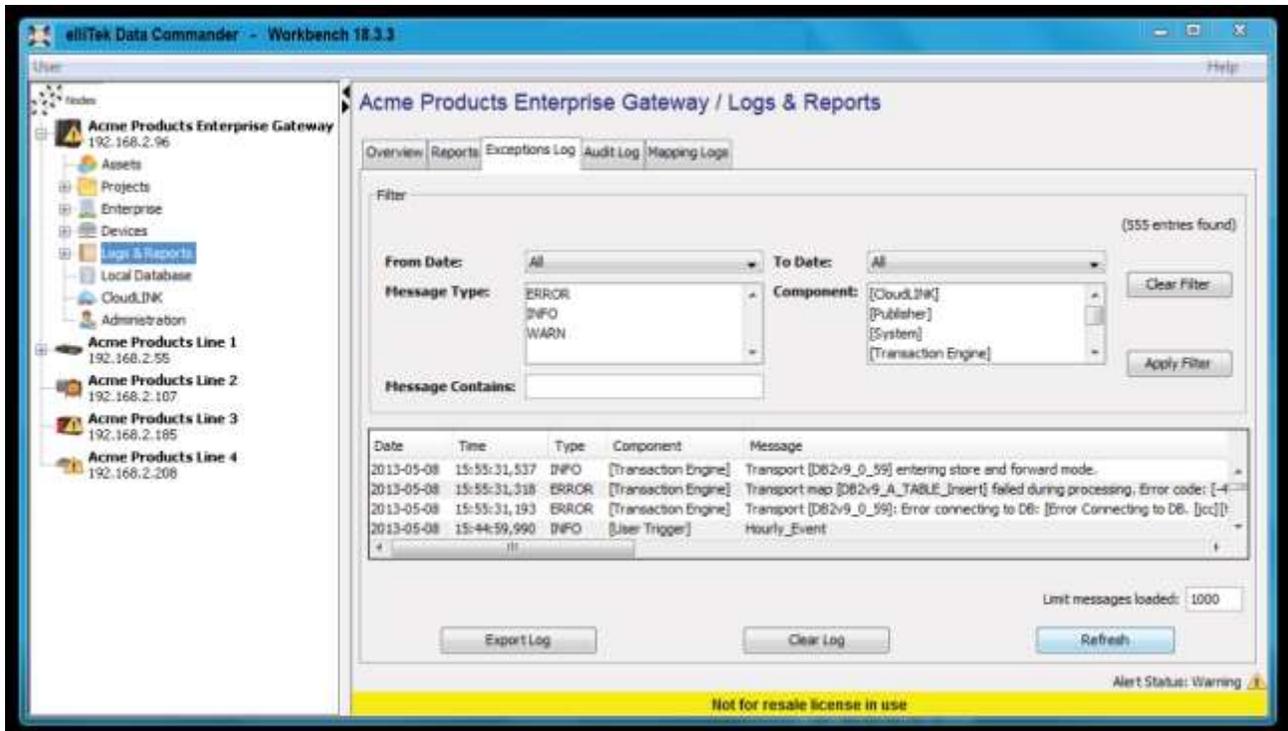
Triggers can add messages to the Exceptions Log file using the Log Message action.

The Exceptions Log and Audit Log are managed to a set size by the system. When that size is reached, older entries are automatically deleted.

### Viewing the Exceptions Log

You can access a node's Exceptions Log as follows.

1. From the Workbench left hand pane, expand the node whose log information you want to view.
2. Select the **Logs & Reports** icon.
3. From the right-hand pane, select the **Exceptions Log** tab.  
The **Exceptions Log** tab appears.



From the bottom of the **Exceptions Log** tab, select **Refresh** to view recent entries.

The **Date** and **Time** columns indicates the date and time, in the node's local time zone, that the log message was created.

The **Type** column indicates the level of the log message, which includes the following:

- FATAL
- WARN
- ERROR
- INFO

The **Message** column describes what caused the message to be written.

The columns can be reordered by dragging and dropping them to their new position. The entries can also be sorted by selecting a column heading.

You cannot delete individual messages from the Exceptions Log file.

## Filtering the Exceptions Log entries

The Filter section allows the selection of filter criteria to limit the number of entries that are displayed.

The **From Date**, **To Date**, **Message Type**, and **Component** parameters list the selections that are available from the entries in the Exceptions Log. The **Message Contains** parameter allows free form text filtering.

After making the selections from these parameters, select the **Apply Filter** button to apply the filter to display only the entries that match the filter criteria. The **Clear Filter** button clears all selections and displays all entries.

The **Limit messages loaded** parameter at the lower right portion of the window indicates the number of entries to display. This can be useful if the node is accessed over a slow connection and only a limited number of the most recent entries are of interest.

## Exceptions Log function buttons

The **Exceptions Log** tab buttons at the bottom of the window provide the following functions:

- **Export** - makes a copy of all of the entries for writing to a text file on the Workbench's computer.
- **Clear Log** - removes all entries from the Exceptions Log file.
- **Refresh** - refreshes the window to include any recent entries.

Related Topics

Log Message

Automated Log Export

## IIoTA industrial IoT Platform: Audit Log

When a Workbench user or system component performs an activity, it is recorded in the Audit Log.

These events are informational in nature, for example:

- The Workbench from an IP address connects to the node
- A project has been started
- A trigger has been started
- Messages have been pruned (deleted) from the Exceptions Log and Audit Log

The Exceptions Log and Audit Log are managed to a set size by the system. When that size is reached, older entries are automatically deleted.

## Viewing the Audit Log

You can access a node's Audit Log as follows.

1. From the Workbench left hand pane, expand the node whose log information you want to view.
2. Select the **Logs & Reports** icon.
3. From the right-hand pane, select the **Audit Log** tab.

The **Audit Log** tab appears.

The screenshot shows the 'Acme Products Enterprise Gateway / Logs & Reports' window. The 'Audit Log' tab is selected, showing a table of log entries. The table has the following data:

Date	Time	User	Component	Message
2013-05-08	08:52:13,427	[admin]	[handler]	User [admin] logged in from [192.168.0.61:49726] [Workbench].
2013-05-08	08:39:18,255	[admin]	[AttentionBit]	Attention Bit has been cleared because: [The user cleared the attention bit.]
2013-05-08	08:37:54,318	[admin]	[handler]	User [admin] logged in from [192.168.2.96:1263] [Workbench].
2013-05-08	08:20:25,584	[admin]	[handler]	User [admin] from [192.168.2.96:2047] [Workbench] has been disconnected.
2013-05-08	01:42:56,599	[System]	[AttentionBit]	Attention Bit has been set because: [Transport entered store and forward.]
2013-05-07	17:24:43,130	[admin]	[handler]	User [admin] from [192.168.0.61:5000] [Workbench] has been disconnected.

From the bottom of the **Audit Log** tab, select **Refresh** to view recent entries.

The **Date** and **Time** columns indicates the date and time, in the node's local time zone, that the log message was created.

The **User** column indicates the user logged on to the Workbench or the system component that initiated the activity.

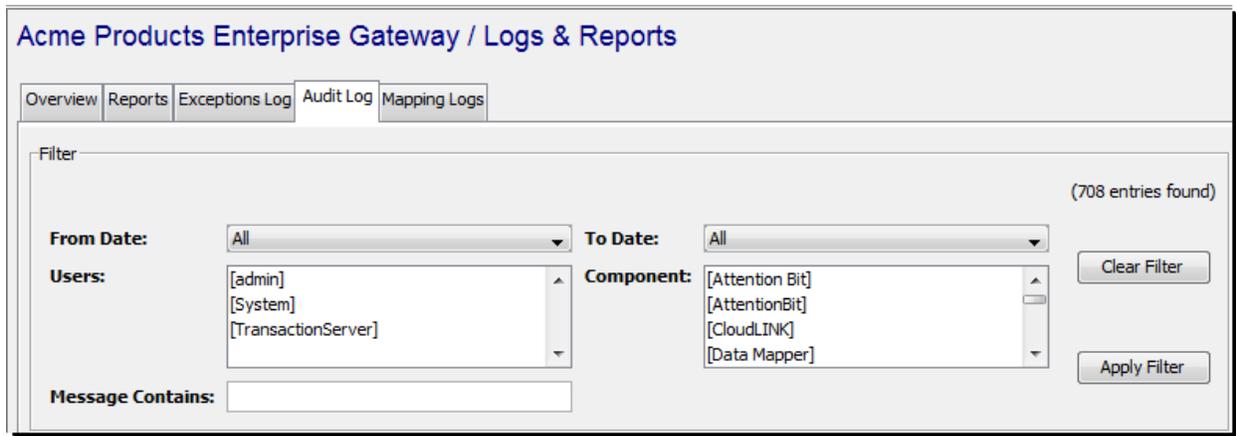
The **Component** column indicates the component that logged the entry.

The **Message** column describes the activity.

The columns can be reordered by dragging and dropping them to their new position. The entries can also be sorted by selecting a column heading.

## Filtering the Audit entries

The Filter section allows the selection of filter criteria to limit the number of entries that are displayed.



Acme Products Enterprise Gateway / Logs & Reports

Overview Reports Exceptions Log **Audit Log** Mapping Logs

Filter (708 entries found)

**From Date:** All **To Date:** All

**Users:** [admin], [System], [TransactionServer] **Component:** [Attention Bit], [AttentionBit], [CloudLINK], [Data Mapper]

**Message Contains:** [ ]

Clear Filter Apply Filter

The **From Date**, **To Date**, **Users**, and **Component** parameters list the selections that are available from the entries in the Audit Log. The **Message Contains** parameter allows free form text filtering.

After making the selections from these parameters, select the **Apply Filter** button to apply the filter to display only the entries that match the filter criteria. The **Clear Filter** button clears all selections and displays all entries.

The **Limit messages loaded** parameter at the lower right portion of the window indicates the number of entries to display. This can be useful if the node is accessed over a slow connection and only a limited number of the most recent entries are of interest.

## Audit Log function buttons

The **Audit Log** tab buttons at the bottom of the window provide the following functions:

- **Export** - makes a copy of all the entries for writing to a text file on the Workbench's computer.
- **Clear Log** - removes all entries from the Audit Log file.

- **Refresh** - refreshes the window to include any recent entries.

## Related Topics

### Automated Log Export

# IIoTA industrial IoT Platform: Reports

When a trigger execution meets the trigger settings reporting criteria, a trigger report is recorded in the Reports Log.

The **Reports** feature provides the detailed viewing of the trigger reports and the configuration and management of the Reports Log.

## Viewing trigger reports

You can access a node's trigger reports as follows.

1. From the Workbench left hand pane, expand the node whose trigger reports you want to view.
2. Select the **Logs & Reports** icon.
3. From the right-hand pane, select the **Reports** tab.

The **Reports** tab appears.

The screenshot displays the 'elliTek Data Commander - Workbench 18.3.3' interface. The left-hand pane shows a tree view of nodes under 'Acme Products Enterprise Gateway', including 'Acme Products Line 1' through 'Acme Products Line 4'. The 'Logs & Reports' icon is selected. The main right-hand pane is titled 'Acme Products Enterprise Gateway / Logs & Reports' and shows the 'Reports' tab selected. The 'Reports' tab contains a table with columns: Project, Trigger, Time, File Size (bytes), and Status. Below this table is a detailed view for a specific project, 'StoreAndForwardTest', showing a list of trigger events with columns: Seq, Name, ID, Route, Status / Extended Status, and Execution Time (ms). The 'Refresh' button is visible at the bottom right of the main pane.

Project	Trigger	Time	File Size (bytes)	Status
StoreAndForwardTest	Node Start MySQLv5_0_59 Triggers	2013-05-03 12:20:45	2106	Success
StoreAndForwardTest	Node Start DB2v9_0_59 Triggers	2013-05-03 12:23:44	2100	Success
StoreAndForwardTest	Node Start DB2v9_0_59 Triggers	2013-05-03 12:23:46	2100	-5223 (Action route is
StoreAndForwardTest	Node Start MySQLv5_0_59 Triggers	2013-05-03 12:32:45	2111	-5417 (Trigger ended
StoreAndForwardTest	Node Start DB2v9_0_59 Triggers	2013-05-03 12:17:15	2101	Success
StoreAndForwardTest	Node Start MySQLv5_0_59 Triggers	2013-05-03 12:17:15	2107	Success

Seq	Name	ID	Route	Status / Extended Status	Execution Time (ms)
1	Trigger Control	1	Success	0	891
2	Trigger Control	2	Success	0	734
3	Trigger Control	3	Failure	-5404 (Trigger is already started.)	0
4	Trigger Control	4	Failure	-5404 (Trigger is already started.)	0
5	Trigger Control	5	Failure	-5404 (Trigger is already started.)	0
6	Trigger Control	6	Failure	-5404 (Trigger is already started.)	0
7	Wait	7	Success	0	10016
8	Fire Trigger	8	Success	0	16
9	Fire Trigger	9	Success	0	422
10	Fire Trigger	10	Success	0	937
11	Log Message	11	Success	0	203

- From the bottom of the **Reports** tab, select **Refresh** to view recent entries.
- The **Project** and **Trigger** columns indicate the trigger that generated the report.
- The **Time** column indicates the date and time, in the node's local time zone, that the report was generated.
- The **File Size (bytes)** column indicates the size of the report.
- The **Status** column indicates the ending status of the trigger
- The columns can be reordered by dragging and dropping them to their new position. The entries can also be sorted by selecting a column heading.

### Viewing the details of a trigger report

When an individual report is selected from the top portion of the **Reports** tab, the bottom portion of the tab displays the details of the report.

Since a report is generated for each trigger execution instance that meets the trigger settings reporting criteria, there can be multiple reports generated for a given trigger.

Project: StoreAndForwardTest		Seq	Name	ID	Status / Route	Extended Status	Execution Time (ms)
Trigger: Node Start MySQLv5_0_59 Triggers		1	Trigger Control	1	Success		891
Execution Time: 13219 ms		2	Trigger Control	2	Success		734
+	Event Notification	3	Trigger Control	3	Failure	-5404 (Trigger is already started.)	0
+	Trigger Control	4	Trigger Control	4	Failure	-5404 (Trigger is already started.)	0
+	Trigger Control	5	Trigger Control	5	Failure	-5404 (Trigger is already started.)	0
+	Trigger Control	6	Trigger Control	6	Failure	-5404 (Trigger is already started.)	0
+	Wait	7	Wait	7	Success		10016
+	Fire Trigger	8	Fire Trigger	8	Success		16
+	Fire Trigger	9	Fire Trigger	9	Success		422
+	Fire Trigger	10	Fire Trigger	10	Success		937
+	Log Message	11	Log Message	11	Success		203
+	Event Completion						

The left portion of the report details displays the execution summary including the total trigger execution time and the sequence of actions that were executed.

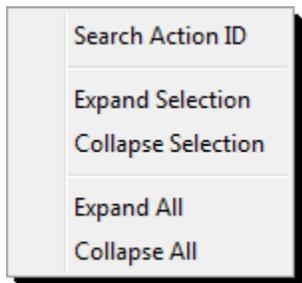
The right portion of the report details displays additional details about the actions. The columns are:

Column	Description
<b>Seq</b>	The execution sequence number of the action. This is a sequential list.
<b>Name</b>	The name of the action.
<b>ID</b>	The action identifier. When using the List Editor, the action IDs are included in the list of actions. When using the Canvas Editor, the action ID is displayed in each action block. While the <b>Seq</b> number is sequential, the action ID indicates the execution path and shows how the routing, including <b>For</b> loops, <b>If</b> actions, and other branching progressed through the trigger's actions.
<b>Status/Route</b>	The route that was taken from the action in determining the next action to execute.
<b>Extended Status</b>	The result of the action.
<b>Execution Time (ms)</b>	The time to execute the action in milliseconds. The sum of all of the action execution times will be the overall trigger execution time displayed in the left portion of the report details.

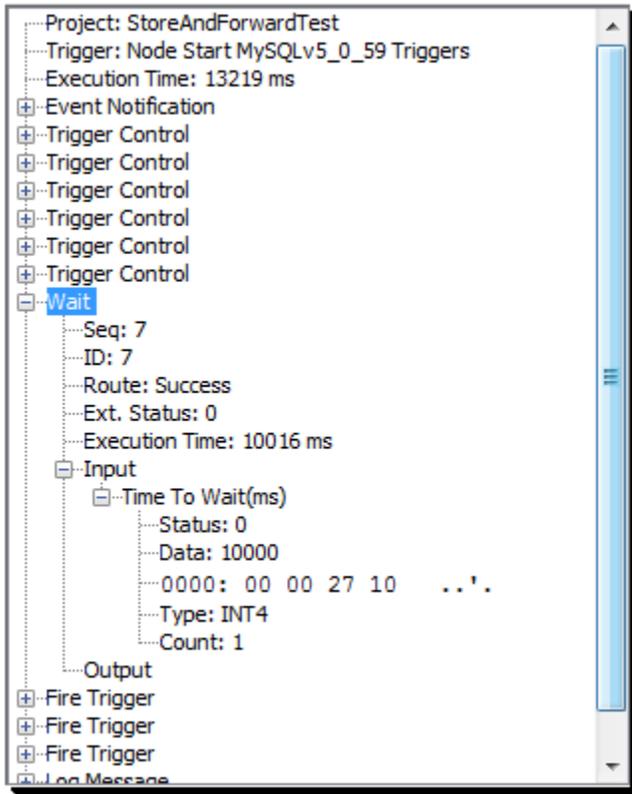
### Expanding the report execution summary

The actions listed in the left portion execution summary can be expanded to show additional details, including any input and output parameters. Individual actions and expanded or collapsed using the [+] and [-] icons next to the actions.

Other options are available by selecting an action in the left-hand pane, then right click to display a pop-up menu:



When an action and all its inputs and outputs are expanded, the details are displayed including the data values of the parameters. For the example of the **Wait** action, the input parameter for the time to wait in milliseconds was a constant with a value of 10000. This is displayed next to the **Data** item along with the display in individual bytes in hexadecimal. The byte values 00 00 27 10 in hex is 10000 in decimal.



Some input values that are selected from lists may not be have their value displayed in the expanded details for the input and output parameters.

As your trigger's are developed and debugged, the trigger report feature will aid in understanding the run time execution of a trigger, including the data values used, the success or failure of each action and the execution route.

## Report limits

While the trigger reporting feature is very useful in understanding the run time behavior of trigger execution instances, it does add overhead to the system's execution.

Since the outcome of a trigger's execution (success, failure, execution time) cannot be predetermined, the generation of the trigger report must take place as the trigger progresses through its actions. This means that for all trigger settings reporting options other than **Off**, the overhead to generate the report takes place before the final execution results determine whether to write the report to the Reports Log or discard the trigger report. If the report is written to the Reports Log, then that is additional overhead to the system's execution.

For nodes that have limited resources (CPU, memory, disk), this overhead can be significant.

The reporting feature does have restrictions built in to limit the number of trigger reports that are generated and written to the Reports Log. When these limits are reached, additional trigger reports are not generated until the pending reports are processed and written to the Reports Log.

Care should be taken when specifying the reporting option for triggers, based on the need for the additional information, the frequency of the execution of the triggers, and the availability of system resources on the node.

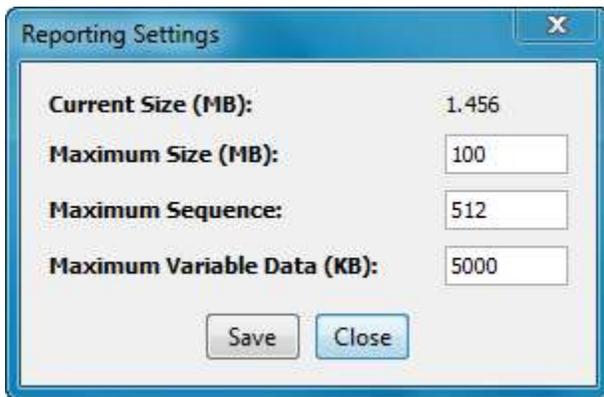
## Configuring and managing trigger reports

For each trigger, the Trigger settings **Reporting** parameter is used to configure the use of the trigger reporting feature. Some node types may not have all options available.

Independent of the **Reporting** parameter and the trigger's success or failure completion, a trigger report can be generated for the next trigger execution instance using the pop-up menu for the trigger. For more information, see the Generate Report option.

In addition to each trigger's setting of the **Reporting** parameter, there is a node wide **Trigger Reporting** parameter available in System administration > **Diagnostics** to disable the generation of all trigger reports. This setting can be used as an alternative to change all triggers' settings for the **Reporting** parameter to Off.

The **Reports** tab **Settings** button is used to configure the Reports Log:

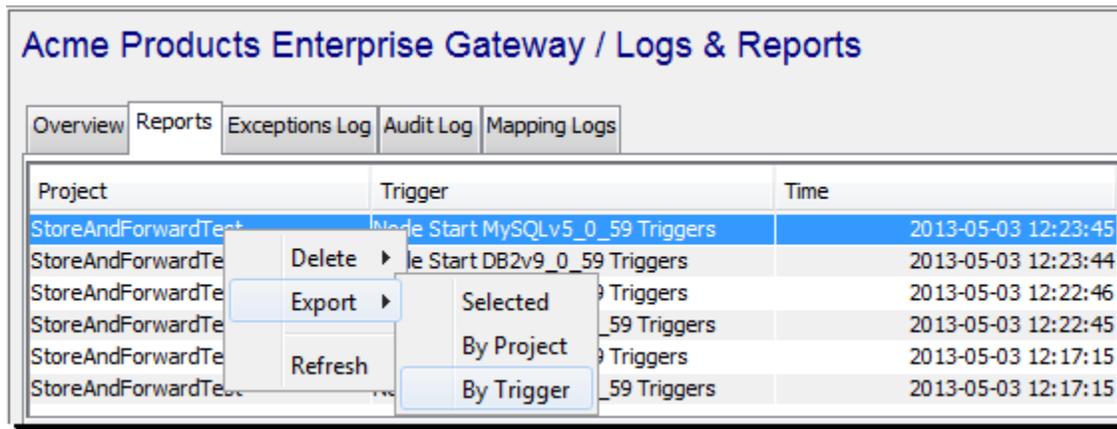


The parameters are:

Parameter	Description
<b>Current Size (MB)</b>	The current size of the Reports log in megabytes.
<b>Maximum Size (MB)</b>	The maximum size that the Reports Log will reach. Once this size is reached, older report entries are discarded.
<b>Maximum Sequence</b>	The maximum number of sequential actions that will be part of a report. The sequence of actions can reach large numbers when there are branching actions, such as <b>For</b> and <b>If</b> , and the routing criteria to end the execution of the trigger is not met.

<b>Maximum Variable Data (KB)</b>	The maximum amount of data, in kilobytes, that will be displayed for a variable within the report. For variables that are arrays, this value indicates the maximum number of elements in the array that will be written to the report. For variables that are String or Binary data types, this is the maximum number of bytes that will be written to the report for the variable.
-----------------------------------	---

When an individual report is selected from the top portion of the **Reports** tab, a popup menu can be display using a right click:



The options available are:

Option	Description
<b>Delete</b>	The options to delete report entries are: <b>Selected</b> - the selected triggers (Shift-click, Ctrl-click or Ctrl-A to select multiple entries). <b>By Project</b> - all report entries from triggers in the selected project. <b>By Trigger</b> - all report entries from the selected trigger. <b>All</b> - all report entries.
<b>Export</b>	The selected report entries are exported to a text file on the Workbench's computer. The options to export report entries are: <b>Selected</b> - the selected triggers (Shift-click, Ctrl-click or Ctrl-A to select multiple entries). <b>By Project</b> - all report entries from triggers in the selected project. <b>By Trigger</b> - all report entries from the selected trigger.
<b>Refresh</b>	Refresh the Reports tab with recent additions.

## Related Topics

Trigger settings

Using an individual project's tab

# IIoTA industrial IoT Platform: Mapping Logs

When a transport map or listener map processes data, a copy of the transaction can be recorded in a mapping log.

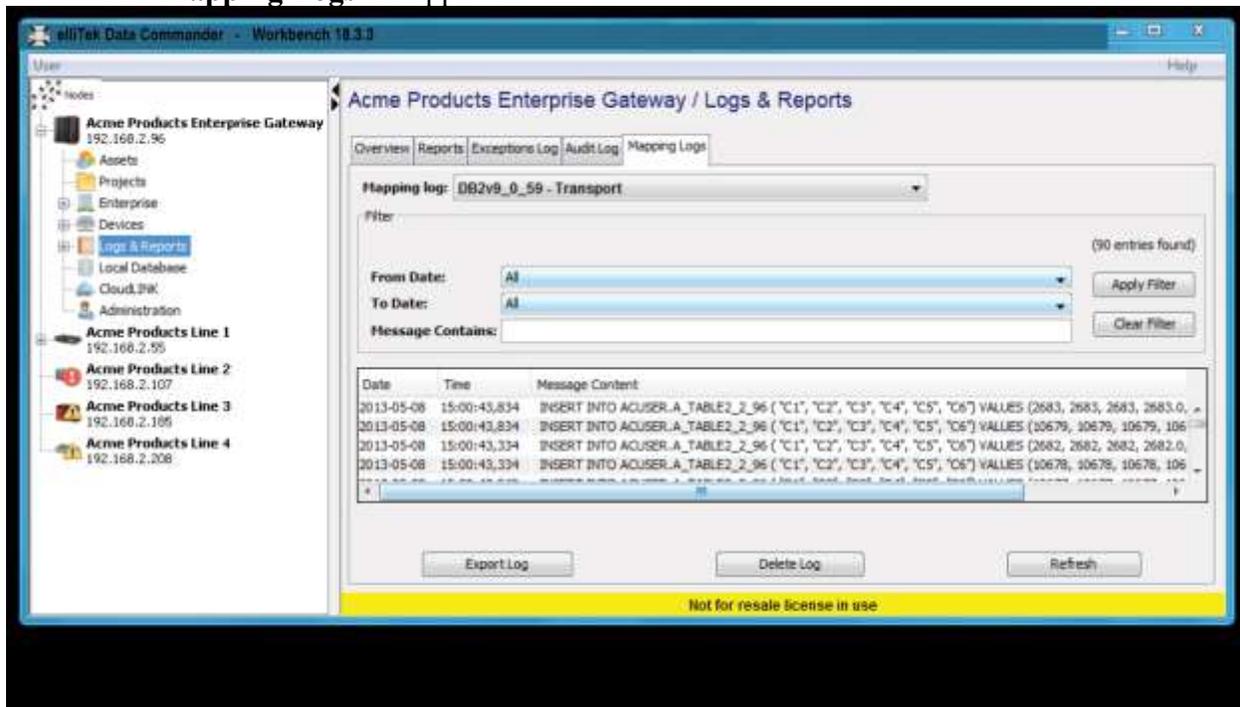
The **Mapping Logs** feature provides the viewing of the transport map or listener map mapping logs.

## Viewing mapping logs

You can access a node's mapping logs as follows.

1. From the Workbench left hand pane, expand the node whose mapping logs you want to view.
2. Select the **Logs & Reports** icon.
3. From the right-hand pane, select the **Mapping Logs** tab.

The **Mapping Logs** tab appears.



You may need to select the down arrow next to the **Mapping log** parameter to view all of the mapping logs available and select one of the items from the list.

From the bottom of the **Mapping Logs** tab, select **Refresh** to view recent entries.

The **Date** and **Time** columns indicate the date and time, in the node's local time zone, that the entry was generated.

The **Message Content** column contains a copy of the transport map or listener map transaction.

The columns can be reordered by dragging and dropping them to their new position. The entries can also be sorted by selecting a column heading.

## Filtering the mapping logs entries

The Filter section allows the selection of filter criteria to limit the number of entries that are displayed.

The screenshot shows the 'Acme Products Enterprise Gateway / Logs & Reports' interface. At the top, there are tabs for 'Overview', 'Reports', 'Exceptions Log', 'Audit Log', and 'Mapping Logs'. The 'Mapping Logs' tab is selected. Below the tabs, there is a 'Mapping log:' dropdown menu showing 'DB2v9\_0\_59 - Transport'. Underneath is a 'Filter' section with the text '(90 entries found)'. The filter section includes three input fields: 'From Date:' with a dropdown menu set to 'All', 'To Date:' with a dropdown menu set to 'All', and 'Message Contains:' with a text input field. To the right of these fields are two buttons: 'Apply Filter' and 'Clear Filter'.

The **From Date**, and **To Date** parameters list the selections that are available from the entries in the mapping Log. The **Message Contains** parameter allows free form text filtering.

After making the selections from these parameters, select the **Apply Filter** button to apply the filter to display only the entries that match the filter criteria. The **Clear Filter** button clears all selections and displays all entries.

## Configuring mapping logs

The mapping log parameters are configured on the **Mapping Log** tab of the transport or listener window. For more information, see Mapping Log tab.

## Mapping Logs function buttons

The **Mapping Logs** tab buttons at the bottom of the window provide the following functions:

- **Export Log** - makes a copy of all of the entries for writing to a text file on the Workbench's computer.
- **Delete Log** - removes all entries from the Mapping Log file(s).
- **Refresh** - refreshes the window to include any recent entries.
- **Navigation** - if multiple mapping log files have been configured and are in use, navigation controls are displayed to choose which file to display.

**Related Topics**  
Mapping Log tab

## IIoTA industrial IoT Platform: Local Database

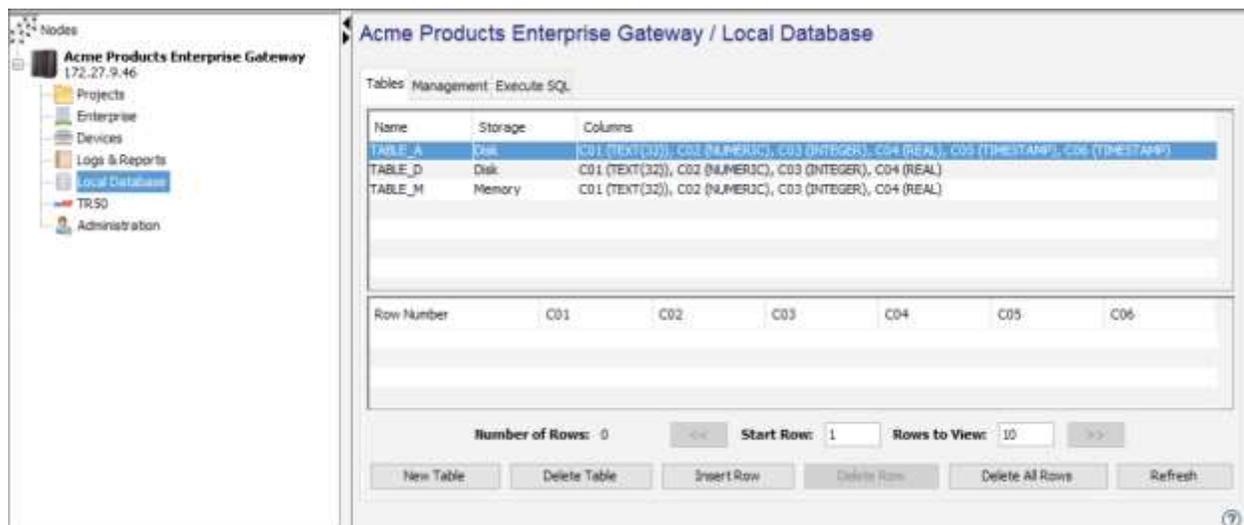
### Overview

The **Local Database** feature provides a light weight, self-contained SQL database engine. This SQL database engine can be used by the application logic in your triggers for a variety of tasks, for example:

- Temporary storing of data
- Reference table look ups
- Longer term collection of data

There is no restriction on the types of tasks your triggers can support using the Local Database feature.

When you select the **Local Database** icon from the Workbench left hand pane, the Local Database window appears in the right-hand pane with individual tabs for different Local Database functions. The Local Database files are stored on disk or in memory, depending on the selected Storage Option when creating the Local Database Table.



### Assumptions

Before using information in this section, the following is assumed:

- You are familiar with relational database concepts.
- You are familiar with structured query language (SQL) concepts and syntax.

For information about the SQL database engine, see [SQLite.org](https://www.sqlite.org).

## Local Database support in the Transaction Server

The actions that use the Local Database are documented in the Trigger action reference, Local Database actions section.

In addition to the Local Database actions supported on Enterprise Gateway, the Transaction Server component of the Enterprise Gateway products support the Local Database feature with the Transaction action, transport maps and transports. For more information on the Transaction Server component's support, see Referencing a local database transport.

**Note:** The Transaction Server's support of Local Database tables is restricted to disk-based tables. Local Database tables created in memory are not supported by transport maps and the Transaction action.

## Highlights

This guide contains the following:

- Local Database Tables
  - Inserting, updating, deleting, and viewing rows in a table
  - Exporting and Importing Local Database tables
  - Altering Existing Tables
- Local Database Management
- Local Database Execute SQL

Related Topics

Local Database actions

Transaction

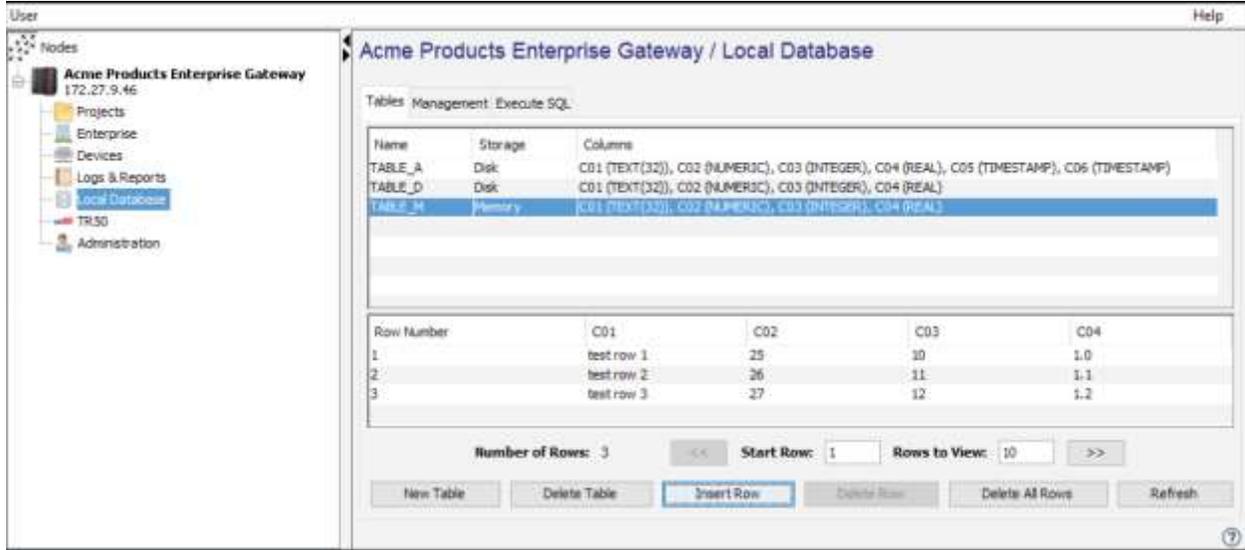
Referencing a local database transport

# IIoTA industrial IoT Platform: Local Database Tables

The **Tables** tab is used to define Local Database tables and manage the tables and their data.

When you select the **Local Database** icon from the Workbench left hand pane, the Local Database window appears in the right-hand pane with individual tabs for different Local Database functions.

When you select the **Tables** tab, the Tables tab is displayed, for example:



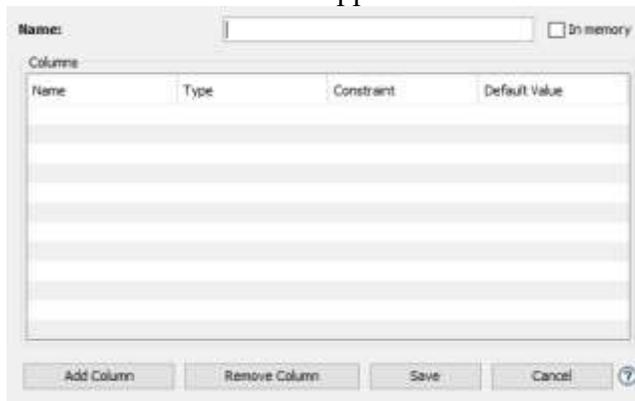
Previously defined tables are displayed with their **Name**, **Storage** and **Columns**.

If you select a previously defined table, its data rows are displayed in the lower portion of the tab.

## Defining a Local Database table

To define a Local Database table, follow these steps:

1. From the Workbench left pane, expand the appropriate node.
2. Select the **Local Database** icon.  
The Local Database window appears.
3. Select the **Tables** tab.
4. From the bottom of the **Tables** tab, select **New Table**.  
The New Table window appears.



5. In the **Name** parameter, type a name for the table. A database table name can be up to 64 characters in length and can include letters, numbers, and the underscore character. You will not be able to type invalid characters. For example, spaces are not allowed.
  
6. By default, the Table is stored on disk. However, the **In memory** checkbox can be selected to create the table purely in memory. Should the node reboot, all data within the table will be lost when the in memory option is chosen.
  
7. The next step is to add a column. You can add as many columns as necessary. From the bottom of the New Table window, select **Add Column**. The New Column window appears.

The New Column window has these parameters:

Parameter	Description
<b>Name</b>	The unique name of the column. A column name can be up to 64 characters in length and can include letters, numbers, and the underscore character. You will not be able to type invalid characters. For example, spaces are not allowed.
<b>Type</b>	<p>The data type for the column. The options are:</p> <p><b>TEXT</b> - Contains character data.</p> <p><b>NUMERIC</b> - Contains numeric data.</p> <p><b>BLOB</b> - Contains a sequence of bytes stored exactly as it was input. The sequence of bytes does not have an associated code page and character set.</p> <p><b>Note:</b></p> <ul style="list-style-type: none"> <li>- The BLOB data type is supported by the Transaction trigger action.</li> <li>- The BLOB data type is not supported by the Local Database trigger actions.</li> </ul> <p><b>INTEGER</b> - Contains a signed integer, stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value. Note that conversions of whole numbers stored in <b>Integer</b> columns to String representations will contain a decimal place.</p> <p><b>REAL</b> - Contains a floating point value, stored as an 8-byte IEEE floating</p>

	<p>point number. For more information on the data type concepts and conversions, see <a href="http://SQLite.org">SQLite.org</a>.</p>
<b>Length</b>	For columns of type <b>TEXT</b> , the maximum length of the text data (up to 65536).
<b>Constraint</b>	<p>The constraint rules that the Local Database enforces. The options are:</p> <p><b>NOT NULL</b> - Do not allow a Null value in the column.</p> <p><b>PRIMARY KEY</b> - In order to use a local database table in a <b>Select with Update</b> or <b>Select with Delete</b> Transaction actions, the table must be created with a primary key.</p> <p>Unlike other databases such as DB2 and Oracle, a local database primary key can have Null and duplicate values. There can be only one primary key on a table. Another constraint with regards to <b>Select with Update</b> transactions is that a primary key column must be selected to be mapped to output map variables or to be updated.</p> <p><b>PRIMARY KEY NOT NULL</b> - By default, primary keys are allowed to be Null. Select this option to prevent Null values from being inserted into primary keys.</p> <p><b>PRIMARY KEY AUTOINCREMENT</b> - This constraint is only used in conjunction with an <b>Integer</b> data type. If you specify another data type for the column, and then specify the primary key autoincrement constraint, you will generate an SQL exception when saving the table.</p> <p>When using the primary key autoincrement constraint, the value in the column is generated by the Local Database and is the largest existing value for that column (plus one). This is true even when the column is not specified with the <b>NOT NULL</b> constraint. Assuming the table is empty, and the first row to be inserted contains a Null in the column with the primary key autoincrement constraint, the actual value inserted into the column is 1 (not Null). If a column is a primary key with autoincrement, trying to insert a row with a value that already exists in the column generates an error and the row is not inserted.</p> <p>If no value is specified in the column with the primary key autoincrement constraint, then the value inserted is the largest for that column (plus one).</p> <p><b>PRIMARY KEY NOT NULL UNIQUE</b> - By default, primary keys are not required to be unique. Select this option to ensure unique values in primary keys.</p> <p><b>UNIQUE</b> - All values in the column must be unique. An attempt to insert a duplicate value will generate an error.</p> <p><b>UNIQUE NOT NULL</b> - By default, a Null is allowed to be inserted into a column even though the column is defined with a unique constraint. Select this option to not allow a Null value inserted into a column with a unique constraint.</p> <p><b>DEFAULT</b> - If the row being inserted does not have a value for this column, the value specified in <b>Default Value</b> is used.</p>

**Default Value**

For columns with a **DEFAULT** constraint, the value to use if the row being inserted does not have a value for this column

8. When you have filled in the values for the column, select **Add**.  
The values for the column are added under the **Columns** section.

**Name:**   In memory

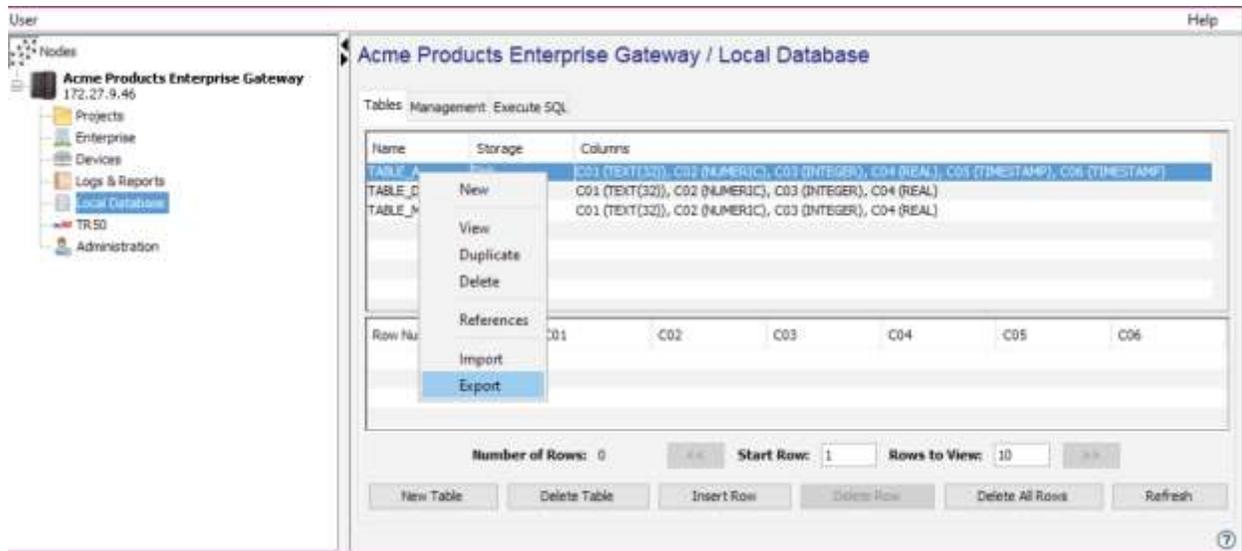
**Columns**

Name	Type	Constraint	Default Value
C01	TEXT(32)		
C02	NUMERIC		
C03	INTEGER		
C04	REAL		
C05	TIMESTAMP		
C06	BLOB		

9. To add another column, repeat steps 7 and 8.
10. When you have completed adding the required columns, select **Save**.  
The name of the table and the columns are added as a row in the **Tables** tab.

## Managing Local Database tables

If you right click in the empty part of the **Tables** tab without a table selected, or on a selected table, a pop-up menu with available options is displayed:



The options are:

Option	Description
<b>New</b>	Define a new Local Database table. Alternatively, you can also select the <b>New Table</b> button at the bottom of the tab.
<b>Duplicate</b>	Make a copy of the table's column definitions. The table's data is not copied.
<b>Delete</b>	Delete a table's definition and its data. Alternatively, you can also select the <b>Delete Table</b> button at the bottom of the tab.
<b>Export</b>	Display an Export window to allow exporting of the table's definition. For more information, see Exporting and Importing Local Database tables.
<b>Import</b>	Display an Import window, allowing the selection of an Export file to import into the node. For more information, see Exporting and Importing Local Database tables.

## IIoTA industrial IoT Platform: Inserting, updating, deleting, and viewing rows in a table

The **Tables** tab provides functions to insert, update, delete, and view rows in a Local Database table.

### Inserting a row

Rows can be inserted in a Local Database table.

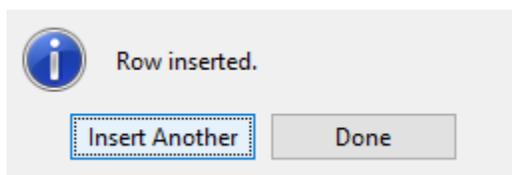
1. From the **Tables** tab, select the table you want to insert a row to, and then select the **Insert Row**.

The Insert Row window appears:

Column	Type	Data
C01	TEXT(32)	
C02	NUMERIC	
C03	INTEGER	
C04	REAL	

Insert Close ?

2. In the Insert Row window, enter a value for each of the table's columns by selecting within the Data parameter for each column.  
Any columns that are defined with a **Default** constraint and a **Default Value**, will be given the default value if data is not entered for the column.
3. When done entering the data for the row, select **Insert**.  
A messages is displayed indicating that the row was inserted and asks if you would like to **Insert another**:



4. Continue to insert rows until you are done.  
The inserted rows will be displayed in the lower portion of the **Tables** tab when the table is selected.

## Updating a row

The data in a table's row can be updated.

1. From the **Tables** tab, select the table which you want to update a row's data.
2. In the lower portion of the **Tables** tab, the rows in the table are displayed.  
You may need to navigate through the rows using the navigation controls to find the one you want to update.
3. Double click in a row and column cell to edit the data. After the data has been updated, press enter or select another portion of the window.  
A confirmation window is displayed asking if the data should be written to the table.
4. Select **Yes** to write the data.  
The column's constraint will be checked and, if valid, the data is updated in the table.

## Deleting a row

Rows can be deleted from a Local Database table.

1. From the **Tables** tab, select the table from which you want to delete a row.
2. In the lower portion of the **Tables** tab, select the row to delete and then select **Delete Row** at the bottom of the tab.  
Alternatively, you can right click on a row to display a pop-up menu and then select its **Delete Row** option.
3. To delete multiple rows, select multiple rows using Ctrl-click, Shift-click or Ctrl-A, and then select **Delete Row**.  
Alternatively, you can use the **Delete All Rows** at the bottom of the tab.

## Viewing rows

When a table is selected in the **Tables** tab, the lower portion of the tab displays the rows in the table. For example:

The screenshot shows the 'Acme Products Enterprise Gateway / Local Database' interface. On the left is a tree view with nodes like 'Acme Products Enterprise Gate', 'Projects', 'Enterprise', 'Devices', 'Logs & Reports', 'Local Database', 'TRSD', and 'Administration'. The main area shows a 'Tables Management' section with a table of database tables:

Name	Storage	Columns
TABLE_D	Disk	C01 (TEXT(32)), C02 (NUMERIC), C03 (INTEGER), C04 (REAL)
TABLE_M	Memory	C01 (TEXT(32)), C02 (NUMERIC), C03 (INTEGER), C04 (REAL)
TABLE_A	Disk	C01 (TEXT(32)), C02 (NUMERIC), C03 (INTEGER), C04 (REAL), C05 (TIMESTAMP), C06 (TIMESTAMP)

Below the table is a data grid with columns 'Row Number', 'C01', 'C02', 'C03', and 'C04'.

Row Number	C01	C02	C03	C04
1	test row 1	25	10	1.0
2	test row 2	26	11	1.1
3	test row 3	27	12	1.2
4	1	1	1	1.0

At the bottom, navigation controls are highlighted with a red box:

Number of Rows: 4    <<    Start Row: 1    Rows to View: 10    >>

Buttons below: New Table, Delete Table, Insert Row, Delete Row, Delete All Rows, Refresh.

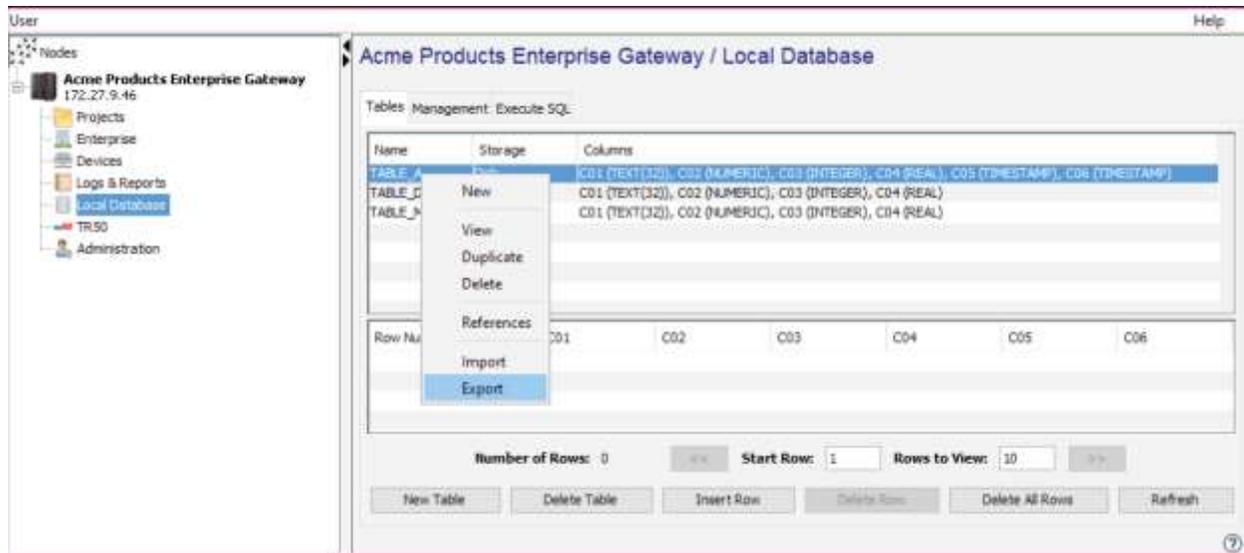
The navigation controls allow you to:

- Page forward
- Page back
- Define the starting row number
- Define the number of rows to display.

The columns as displayed in the lower portion of the tab can be reordered by dragging and dropping to a new position.

## IIoTA industrial IoT Platform: Exporting and Importing Local Database tables

Local Database table definitions can be exported and imported using the **Tables** tab pop-up menu.



## Exporting Local Database tables

The exported information for a table includes its column definitions and any indexes. A table's data is not included in the export from the **Tables** tab.

For information on exporting a Local Database table's data to a comma-separated values (CSV) file, see Local DB Export.

## Importing Local Database tables

The import function will create a new table using the exported table definition, including any indexes. For an existing table, if the import function overwrite confirmation is answered **Yes**, any data in the existing table will be lost.

For information on importing a data into a Local Database table from a comma-separated values (CSV) file, see Local DB Import.

## IIoTA industrial IoT Platform: Altering Existing Tables

Local Database tables allow for a minimal amount of altering of their existing definition.

An additional column can be added to an existing table by right-clicking in the table list, selecting **View**, and adding columns as done during a new table definition. The only constraint available when adding columns to an existing table is DEFAULT. This restriction is defined by SQLite: <http://sqlite.org/faq.html#q11>.

For more advanced table modifications, a new table must be created and the values from the original table inserted into it.

## Duplicating a Table with Data

### Advanced Topic

This is a topic for advanced users and may result in data loss if performed incorrectly. It is recommended you export your table data using the Local DB Export action before performing these actions.

To create a new table containing the data from the original table, follow these steps (using "MyTable" as an example):

1. Go to the **Tables** tab in the **Local Database** panel.
2. View "MyTable" via double-click or by selecting **View** in the context menu.
3. In the name field, enter a new name for the table (e.g. "MyTableNew").
4. Add, delete, or modify the columns, constraints, and defaults as desired.
5. **Save** the "MyTableNew" definition.
6. Go to the **Execute SQL** tab.
7. Execute a command like one of the following. **For tables containing a large amount of data, executing these commands may temporarily impact performance.** You may want to perform these steps when your application is not actively processing data (off shift), if possible.

1. If the columns are identical in name and order, and have compatible constraints:  

```
INSERT INTO MyTableNew SELECT * FROM MyTable
```
2. If the columns are changed or removed, you'll need to specify them explicitly:  

```
INSERT INTO MyTableNew SELECT col1, col2, etc FROM MyTable
```
3. If a column has been added that doesn't have an equivalent in the original table, a null or empty string can be inserted:  

```
INSERT INTO MyTableNew SELECT col1, col2, null FROM MyTable
```
4. Other variations of this command may be necessary depending on differences between the tables.

8. You will get a message stating "Affected Rows: <number>", which is the number of rows inserted into MyTableNew.
  1. If an error is encountered, e.g. "Database constraint violated" it may be necessary to delete MyTableNew and restart at step 1, making sure chosen constraints and types are compatible with the original table.
9. Go to the **Tables** tab.
10. Verify that the data in MyTableNew is correct. It should have the same number of rows as MyTable.
11. Go to the **Execute SQL** tab.
12. Enter the following command:

```
ALTER TABLE MyTable RENAME TO MyTableOld
```
13. You will get a message stating "Affected Rows: 0" which indicates success.
14. Enter the following command:

```
ALTER TABLE MyTableNew RENAME TO MyTable
```
15. You will get a message stating "Affected Rows: 0" which indicates success.
16. Go to the **Tables** tab, select the **Refresh** button.
17. Verify that the data and structure of MyTable is correct.
18. At this point you can delete MyTableOld or keep it until you have verified MyTable is functioning properly.

## IIoTA industrial IoT Platform: Local Database Management

The **Management** tab is used to define and manage Local Database table indexes and to compact the Local Database.

## Managing indexes

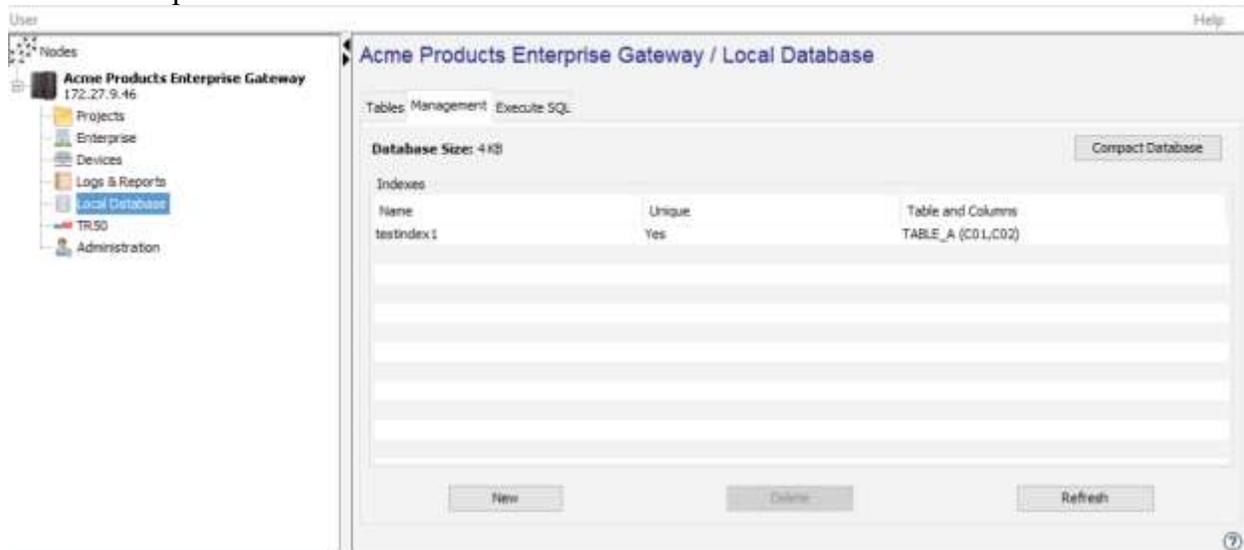
A **database index** is a data structure that improves the speed of data retrieval operations on a database table at the cost of slower writes and the use of more storage space. Indexes can be created using one or more columns of a database table, providing the basis for both rapid random look ups and efficient access of ordered records. A unique index does not allow entry of duplicate values in the columns, thereby guaranteeing that no two rows of a table are the same.

### Defining a Local Database index

To define a Local Database index, follow these steps:

1. From the Workbench left hand pane, expand the appropriate node.
2. Select the **Local Database** icon.
3. Select the **Management** tab.

The **Management** tab appears; any previously defined indexes are displayed. For example:



4. From the bottom of the **Management** tab, select **New**.

The New Index window appears:

**Name:**   Allow Duplicate Values

**Table:**

5. Enter the parameters for the new index as follows:

Parameter	Description
<b>Name</b>	The unique name of the index. An index name can be up to 64 characters in length and can include letters, numbers, and the underscore character. You will not be able to type invalid characters. For example, spaces are not allowed
<b>Table</b>	The list of the Local Database tables on this node. Select the table for the new index. When a table is selected, its columns are displayed in the section below the table name.
<b>Column selection</b>	The list of columns for the selected table. Select one or more columns for the index. Multiple selections can be made using Ctrl-click, Shift-click, and Ctrl-A.
<b>Allow Duplicate Values</b>	An indication of whether the columns in the index will allow duplicate values or not. When this check box is selected, the index will allow entry of duplicate values in the columns. When the check box is cleared, the index will not allow entry of duplicate values in the columns. This will guarantee that no two rows of a table are the same.

6. When done with the parameter entry, select **Create**.

7. If additional indexes need to be defined for the table, repeat steps 4 - 6.

## Deleting a Local Database index

A Local Database table index can be deleted any time it is determined it is not needed.

To delete a Local Database index, follow these steps:

1. From the **Management** tab, select the index to delete.
2. Select **Delete**.  
Alternatively, you can also right click the index to display its pop-up menu and select the **Delete** option.  
A confirmation dialogue will be displayed.
3. Select **Yes** to proceed.

The index will be deleted from the Local Database and removed from the list of indexes on the **Management** tab.

## Compacting the Local Database

The Local Database can be periodically compacted to reduce the size and fragmentation of the data files on the node.

This function should be used during a period of limited or no application execution so that the processing does not impact normal application processing and performance.

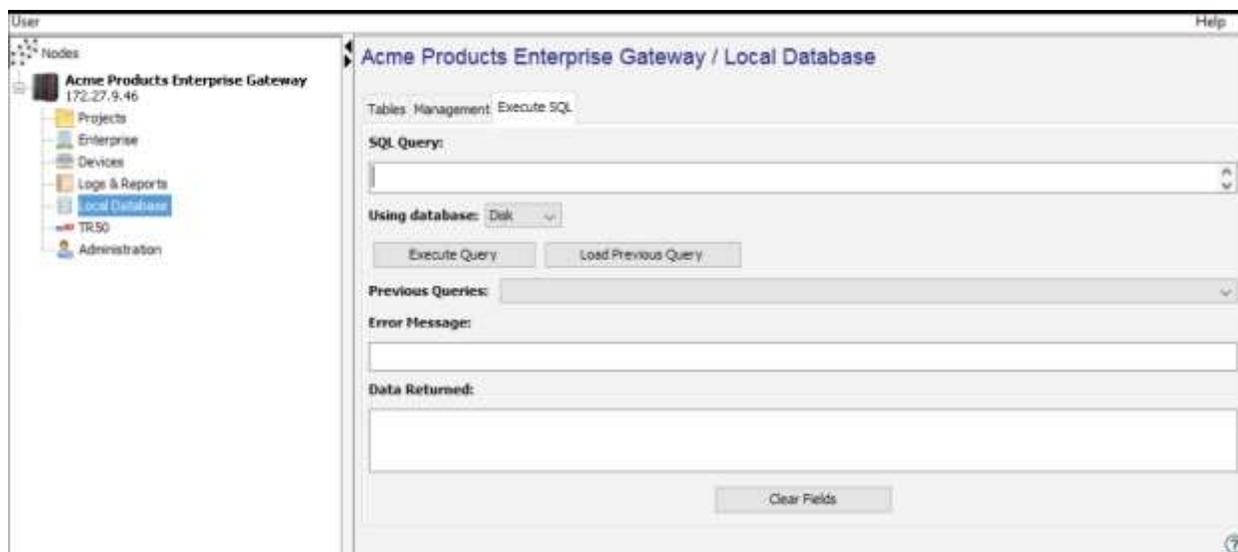
To compact the Local Database, follow these steps:

1. During a period of limited or no application execution, select the **Compact Database** button in the top right of the tab.  
A confirmation dialogue will be displayed.
2. Select **Yes** to proceed.
3. A completion message will be displayed when the processing has completed.

## IIoTA industrial IoT Platform: Local Database Execute SQL

The **Execute SQL** tab is used enter SQL statements that are executed by the Local Database.

The statements can be used to define and manipulate information stored in a database table or index. Local Database table information can be added, viewed, updated, or deleted.



## Using the Execute SQL tab

The Execute SQL tab provides fields for entering the SQL statement, function buttons and display information as follows:

Item	Description
<b>SQL Query</b>	<p>The entry field for the SQL statement.            Most valid SQL statement can be used except CREATE statements such as CREATE TABLE and CREATE INDEX.            The SQL statements are not case sensitive.            For example:</p> <pre>SELECT * FROM [TABLE_NAME] INSERT INTO [TABLE_NAME] VALUES (value1, value2, ...) UPDATE [TABLE_NAME] SET [COLUMN1]=[VALUE2] WHERE [COLUMN1]=[VALUE1] DELETE FROM [TABLE_NAME] WHERE [COLUMN1]=[VALUE1]</pre> <p>After typing the SQL statement, you can either press the enter key or select the <b>Execute Query</b> button.</p>
<b>Using database</b>	<p>Choose the type of Database that the SQL statement should be executed on.            This is determined by the Storage type of your Local Database Table.</p>
<b>Execute Query</b>	<p>Executes the SQL statement currently displayed in the <b>SQL Query</b> field.</p>

<b>Load previous Query</b>	Recalls the previously executed SQL statement into the <b>SQL Query</b> field. Additional selects of this button will recall older SQL statements in a round robin fashion.
<b>Previous Queries</b>	A selection list of the previous queries that be used to recall a SQL statement.
<b>Error Message</b>	Displays any error message encountered when the SQL statement is executed.
<b>Data Returned</b>	Displays the information returned from the SQL statement.