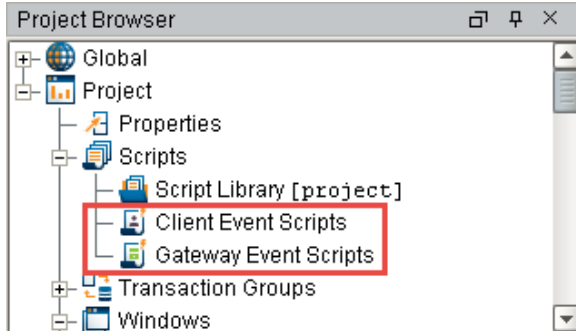


# Client and Gateway Event Scripts

## Client and Gateway Event Scripts

The Client and Gateway Event scripting workspaces are located in the Project menu of the Designer underneath Properties, or in the Project Browser under **Project > Scripts**. Each of the two *Scopes* have their own set of events but there are several repeated events. Even with the same names, different things may trigger the different Event Scripts.

**Note:** These scripts are stored inside a project, and will be copied if a project is copied.



## Client Event Scripts

Client Event Scripts execute in the running Client, which means that they may never execute if no clients are running, or they may execute many times if multiple clients are running. Client Event Scripts will also execute in the Designer, but only in Preview Mode. Because Clients are full-fledged applications, Client Event Scripts run on the computer running the Client, not on the Gateway's host server computer. This means that they don't have access to the Gateway's file system, and so on.

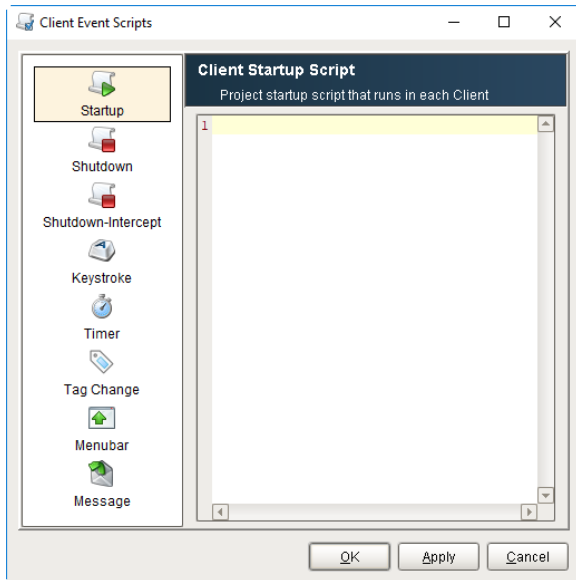
The Client Event Scripts are:

- Startup (client)
- Shutdown (client)
- Shutdown-Intercept (client)
- Keystroke
- Timer
- Tag Change
- Menubar
- Message

## On this page

...

- Client and Gateway Event Scripts
  - Client Event Scripts
  - Gateway Event Scripts
- Startup Script
  - Gateway Startup Behavior
  - Client Startup Behavior
- Shutdown Script
  - Gateway Shutdown Behavior
  - Client Shutdown Behavior
- Shutdown-Intercept Script
  - Client Shutdown-Intercept Behavior (Not Available in Gateway Scripts)
- Keystroke Scripts
  - Client Keystroke Behavior (Not Available in Gateway Scripts)
- Timer Scripts
  - Gateway Timer Behavior
  - Client Timer Behavior
- Tag Change Scripts
  - Gateway Tag Change Behavior
  - Client Tag Change Behavior
- Menubar Scripts
  - Client Menubar Behavior (Not Available in Gateway Scripts)
- Message Scripts
  - Client Message Handlers
  - Gateway Message Handlers
  - Using Message Handlers
- Troubleshooting Gateway and Client Scripts
  - Gateway Scripts
  - Client Scripts



## Gateway vs Client Event Scripts

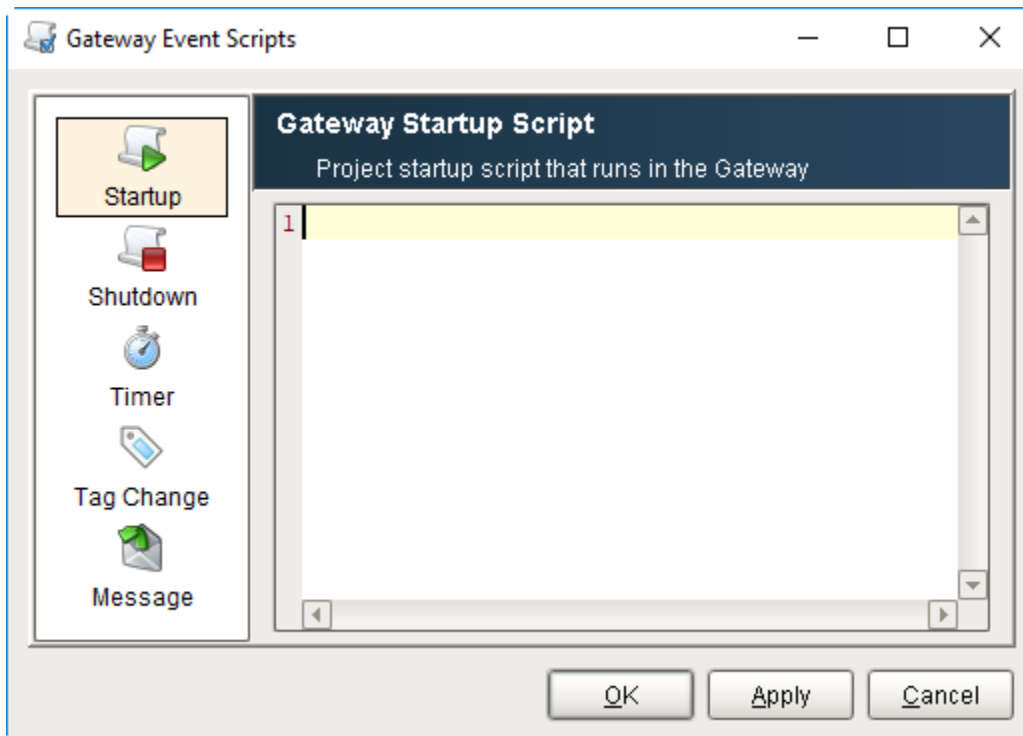
[Watch the Video](#)

## Gateway Event Scripts

Gateway Event Scripts execute on the Ignition Gateway service. This means that they will always execute as long as Ignition is running, even if there are no clients open. If there are multiple clients open, these scripts will still only run once. For example, if you have a script that writes to the Gateway when a Tag changes, it belongs in the Gateway scope so you don't get duplicate records when multiple clients are open.

The Gateway Event Scripts are:

- Startup (Gateway)
- Shutdown (Gateway)
- Timer
- Tag Change
- Message





System functions are available for both Client Event Scripts and Gateway Event Scripts. Some system functions are available for both Client and Gateway Event Scripts, but some system functions are specific to either one or the other. When you're writing event scripts, it's important to remember the scope of where you writing the script: Client or Gateway. You can check [Scripting Functions](#) in the Appendix to see list of all system functions, their descriptions, and what scope they run in.

## Startup Script

The Startup Script event runs at the startup of either the Client or the Gateway. This allows you to run a script at startup, giving you the chance to configure certain things dynamically.

## Gateway Startup Behavior

In the Gateway scripting scope, this means that the script will run when the Gateway starts up and whenever the scripting configuration changes via a Designer save action. This means that while designing, the startup and shutdown events may happen frequently.

There is a specific order to when the various startup scripts are run. When troubleshooting your Gateway startup times, consider the following:

1. **Gateway starts** - The Gateway will start as an OS service, and start the context. No startup scripts can run before this is complete.
2. **Projects are started** - This includes all of the Gateway scoped items in the projects such as Transaction Groups, SFCs, etc. This does not refer to launching clients, and no clients can be automatically launched at this time. All **Gateway Startup Scripts** are run at this time for each project. Note: if you copied a project, always check for Gateway scoped events such as these. You generally don't want a Gateway Startup Script to run twice because it is in two projects.

## Client Startup Behavior

In the Client scripting scope, Client Event Startup scripts run after a user successfully logs in to the client, but before any "Open on Startup" windows are opened. Client Event Scripts can be used to open up a custom set of windows based on who logged in.

## Shutdown Script

The Shutdown Script event runs at the shutdown of either the Client or Gateway. It allows you to run a piece of code as the shutdown is occurring. After the script is run, the shutdown will finish.

Note: if the computer power is lost abruptly (power outage, hard restart, etc) this shutdown script will not run.

## Gateway Shutdown Behavior

The Gateway Shutdown script will run as the Gateway shuts down, such as from the Gateway Control Utility.

## Client Shutdown Behavior

The Client Shutdown script will run as the Client shuts down or logs out, such as with a script, or by clicking the close window "X" at the top of the OS window which is typical in most Operating Systems.

## Shutdown-Intercept Script



### Startup Scripts

[Watch the Video](#)



### Shutdown Scripts

[Watch the Video](#)

## Client Shutdown-Intercept Behavior (Not Available in Gateway Scripts)

The Shutdown-Intercept Script is unique in that it runs when a user attempts to shutdown a client, but before actual shutdown occurs. This means it will actually run before the Shutdown Script, and there is only a client version of it. There is a special event object that you can set a cancel property to prevent shutdown by using the code "event.cancel = 1." Doing this will cancel the shutdown event and leave the user at the spot they were last. This allows you to set special restrictions when the client is actually allowed to shut down, such as having a certain role, as seen in the example below:




```
Python - Cancel Application Exit

# Check to see if the user has a certain role.
if "SuperUser" not in system.security.getRoles():
    # If the role is not present, it will warn the user and cancel the shutdown process.
    system.gui.warningBox("Exit not allowed for non-admin user.")
    event.cancel = 1
```


## Keystroke Scripts

### Client Keystroke Behavior (Not Available in Gateway Scripts)

The Keystroke Scripts let you create different events that will activate on certain key combinations. They only run in the Client, but you may add as many Keystroke Script events as you'd like, as long as each one has a unique key combination.

Since multiple Keystroke Scripts can be added, there are separate buttons that allow you to add (  ) a Keystroke Script, or edit (  ) and delete (  ) the currently selected Keystroke Script.

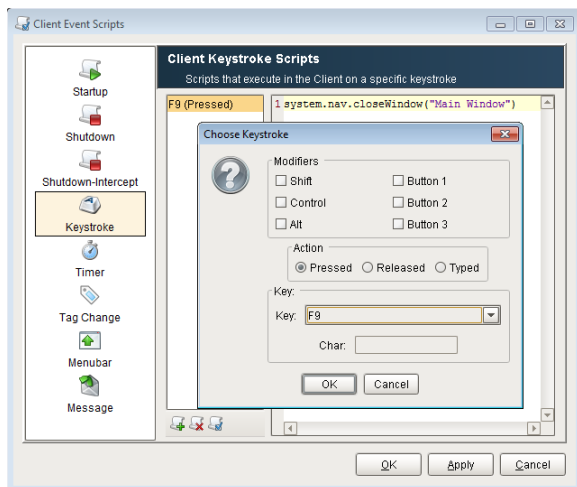
When choosing a keystroke, you may choose any number of modifiers, which are keys or mouse buttons that must be down to activate the keystroke. You can also choose whether or not the keystroke is on the pressed or released event of a keyboard key, or upon the typing of a character. Special keys like the Function keys (F1) or ESC key are only available in the pressed and released actions. The example below shows the F9 key when pressed, closes the window called "Main Window."

 Some Operating Systems reserve certain keys for certain function, and will capture the key press or release before it gets sent to the Client. For example, many Operating Systems use the TAB key to shift focus to the next field.



**Keystroke Scripts**

[Watch the Video](#)






## Timer Scripts



## Timer Scripts

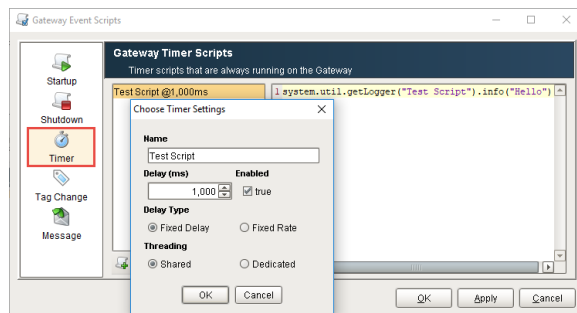
[Watch the Video](#)

The Timer Scripts execute periodically on a timer at a fixed delay or rate. This allows you to set up a sort of heartbeat that can run in the Client or the Gateway Scope. Since multiple Timer Scripts can be added, there are separate buttons that allow you to add (  ) a Timer Script, or edit (  ) and delete (  ) the currently selected Timer Script. The Timer Script has three major settings: Delay in milliseconds, Delay Type, and Threading.

A **Fixed Delay** timer script (the default) waits for the given **Delay** between each script invocation. This means that the script's rate will actually be the delay plus the amount of time it takes to execute the script. This is the safest option since it prevents a script from mistakenly running continuously because it takes longer to execute the script than the delay.

**Fixed Rate** scripts attempt to run the script at a fixed rate relative to the first execution. If the script takes too long, or there is too much background process, this may not be possible. See the documentation for `java.util.Timer.scheduleAtFixedRate()` for more details.

In addition, all timer scripts for a given project that choose to run in a **Shared** thread will all execute in the same thread. This is usually desirable, to prevent creating lots of unnecessary threads. However, if your script takes a long time to run, it will block other timer tasks on the shared thread. In this case, you can have the Timer Script run in a **Dedicated** thread so that it runs independently. The rule of thumb here is that quick-running tasks should run in the shared thread, and long-running tasks should get their own dedicated thread.



## Gateway Timer Behavior




This script will execute in the Gateway, meaning that it will begin running when the Gateway starts, and will continue running at the rate specified until the Gateway is stopped.

## Client Timer Behavior

Since this will execute in a Client, it is important to remember that Client Timer Scripts may never execute (if no clients are open) or may execute many times (once per open client). They start when the client has a successful login.

## Tag Change Scripts

The Tag Change Script event occurs when either the Value, Quality, or Timestamp of a Tag or list of Tags changes. They will also get an initial execution whenever the scripting system starts up. You can set the Tag Change Script to run in either the Client or the Gateway Scope. Multiple Tag

Change Scripts can be created, using the add (  ) button to add a script, and the remove (  ) button to remove the selected script. To specify multiple Tags for a given script, enter them one per line in the Tag paths text area. To quickly add many Tags, you can open the Tag Browser (  ) and drag-and-drop Tags from the window onto this text area. The actual script is then entered in a different tab. When executing, each Tag Change Script runs in a separate thread. This prevents long running scripts from blocking the execution of other Tag Change Script.

These scripts receive three special variables in their namespace when they are run: **event**, **initialChange**, and **newValue**. Both the event and newValue objects refer to the Tag that changed to initiate the event.

- **initialChange** - a variable that is a flag (0 or 1) which indicates whether or not the event is due to the initial subscription or not. This is useful as you can filter out the event that is the initial subscription, preventing a script from running when the values haven't actually changed.

## Tag Change Scripts

[Watch the Video](#)

- **event** - a **TagChangeEvent** object which contains the properties: tag, tagPath, and tagProperty, all of which are also complex objects.
  - **event.tag** - a **Tag** object which contains basic information about the Tag such as name, value, and type that can be accessed via **event.getTag().getName()**.
  - **event.tagPath** - a **TagPath** object which contains information pertaining to the Tag path such as the parent and child paths via **event.getTagPath().getParentPath()**. You can also get the path of the Tag by calling **event.getTagPath().toString()**.
  - **event.tagProperty** - a **TagProperty** object, which can be used to create an enum of a property, which can be used in conjunction with the Tag object to access properties of the Tag such as OPC Item Path or Scale mode via **event.getTag().getAttribute(event.getTagProperty().getProp("OPCItemPath"))**.  
The possible property names are listed here: [Tag Properties](#)
- **newValue** - a **newValue** object which contains a value, quality, and a timestamp of the Tag. They can be called via **newValue.getValue()**.

## Gateway Tag Change Behavior

Having the Tag Change Scripts run in the Gateway Scope means that the scripts will trigger when a Tag in their Tag list changes as long as the Gateway is running. The script will be able to interact with the Gateway and other Tags, but not the project.

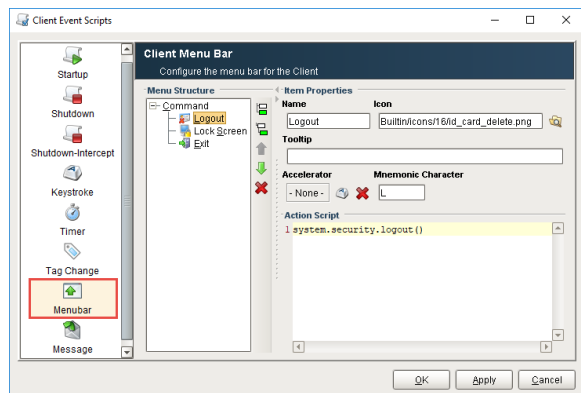
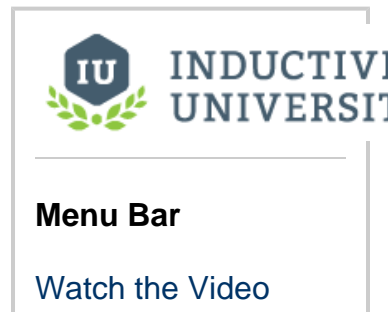
## Client Tag Change Behavior

Having the Tag Change Scripts run in the Client Scope means that the scripts will trigger when a Tag in their Tag list changes, but only if a Client of that project is open. This means that if no Clients are open, the script will not fire, and if many clients are open, the script will fire once for each open client. Additionally, because this is run from the Client Scope, it will have access to Client resources such as Client Tags and can trigger things in the Client such as opening a window.






## Menubar Scripts

### Client Menubar Behavior (Not Available in Gateway Scripts)

The Client Menubar Scripts create and control the options available in the menubar of the Client. As such, these scripts are only available in the Client Scope. By default, a Client will have three menus: Command, Windows, and Help. The Windows and Help Menu are separate, and controlled through the project properties, but the Command menu is actually created in the Client Menubar Scripts. The default menu structure gives you an idea of how to create a menu.



The top level nodes will show up in the menubar. Each node can have an action script unless it has children. In the image above, the Command object will have a grayed out action script field because it has three children. This allows the Client to click into the menu, and look at what is inside. Children of objects can themselves have children, allowing you to create submenus within your menu. The structure is easy

to create, using the buttons to the right to add peer (  ) or child (  ) nodes. You can also move the nodes up (  ) or down (  ) in the hierarchy, as that will affect the order they appear in the Client. You can also delete (  ) nodes that you don't want.

With a node selected you can add to or edit its behavior. Each node can be given a name that it will display in the Client, an icon which will be displayed at the front of the name, and a tooltip that will display when hovering over the menu option. These help to identify what the node is and what it does.

## Accelerators and Mnemonics

Each node can also be given an accelerator and a mnemonic character. The accelerator is a key or key combination that can be pressed at any time in the client to initiate that nodes event. It will display next to the name of the node in the client, and works much like many commonly known accelerators such as Control + S to Save. The mnemonic character is a key that can be pressed when currently in the menu to initiate the node event. If the character chosen is in the name of the node, then that character will be underlined.

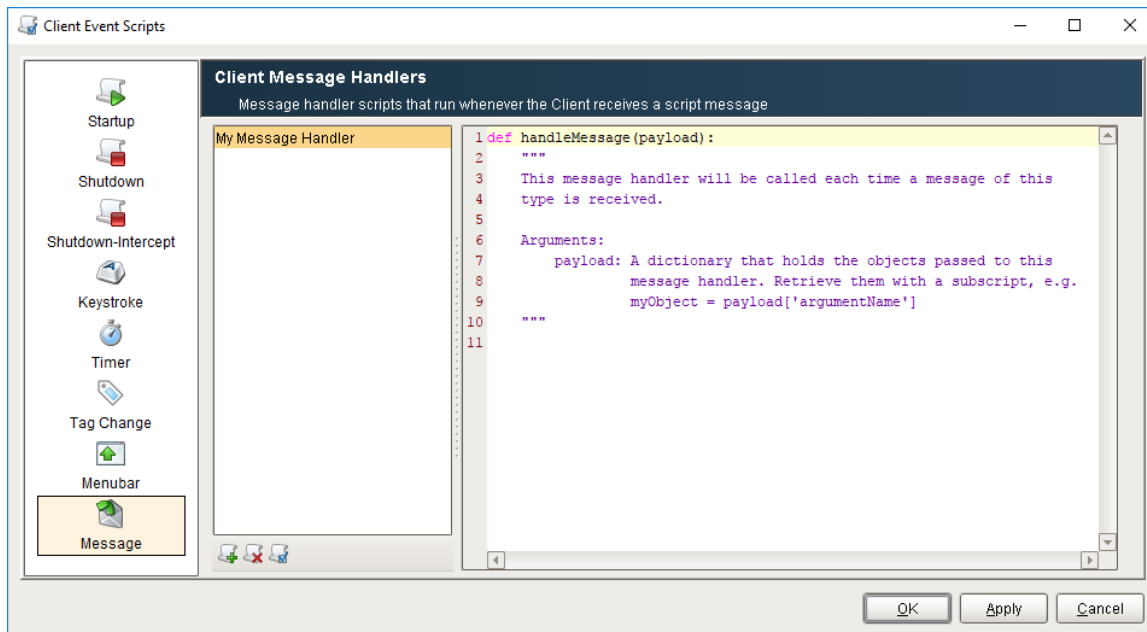
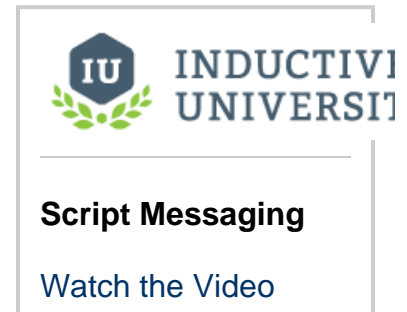
Finally, you can set an action script on the node which will determine what happens when the node is selected. Typically this is used for navigation, such as swapping to a new window, or logging out and exiting, like the default nodes. However, you can enter in any kind of script you want.

## Message Scripts




Message Handlers allow you to write a script that will run in either the Client or Gateway that they are located in, but they can be invoked by making a call from other projects or even other Gateways. They can be called using three different scripting functions: `system.util.sendMessage`, `system.util.sendRequest`, and `system.util.sendRequestAsync`.

## Client Message Handlers

Located in the Message section of Client Event Scripts, client message handlers will execute in the client. This means if you have five clients open for a project with a message handler that gets called, the message handler will run in each client. You can easily create and manage all of your client message handlers in the Client Event Scripts window. Clicking on one of your message handlers will bring up its script on the right.



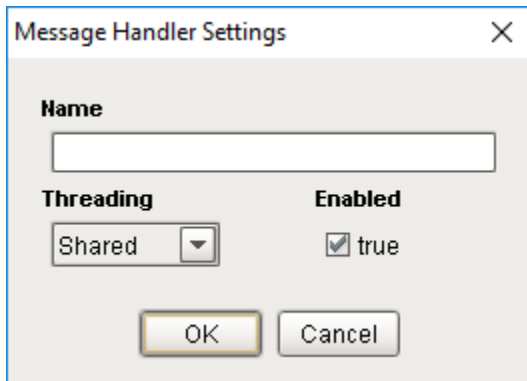
Under the list of handlers, three small buttons allow you to add, remove and manage your handlers.

-  **Add Message Handler** - Will add a message handler.
-  **Remove Message Handler** - Will delete the highlighted message handler.
-  **Modify Message Handler** - Will modify the settings for the highlighted message handler.

When adding or modifying a message handler, a small settings window will popup. In it, you can give your message handler a name or modify its existing name, enable or disable the message handler, and even select how it executes under the **Threading** dropdown. There are three different options that dictate how the message handler will execute:

- **Shared** - The default way of running a message handler. Will execute the handler on a shared pool of threads, in the order that they are invoked. If too many message handlers are called all at once and they take long periods of time to execute, there may be delays before each message handler gets to execute.

- **Dedicated** - The message handler will run on its own dedicated thread. This is useful when a message handler will take a long time to execute, so that it does not hinder the execution of other message handlers.
- **EDT** - This will run the message handler on the Event Dispatch Thread (EDT) which also updates the GUI. If a message handler were to take a long time to execute, it would block the GUI from running which may lock up your client. This is helpful when your message handler will be interacting with the GUI in some way, as the GUI will not be able to update until the message handler finishes.



Inside the message handler is your script. The script will have a single object available to it, the *payload*. The payload is a dictionary containing the objects that were passed into it. Each object in the payload dictionary can be accessed by calling their corresponding key. For example:

**Pseudocode - Payload Values**

```
value1 = payload["MyFirstValue"] # "MyFirstValue" is the key that is associated with a value. We are taking the value associated with MyFirstValue, and assigning it to value1.
value2 = payload["MySecondValue"] # Similarly, we are taking the value associated with MySecondValue and assigning it to value2.
```

## Gateway Message Handlers

Gateway Message Handlers are setup and function similarly to client message handlers. However, there are two major differences:

- They are setup in the Message section of the Gateway Event Scripts. As such, they are executed on the Gateway.

This feature is new in Ignition version **7.9.4**  
[Click here to check out the other new features](#)

- They have an added layer of security. In the Message Handler Settings, it is possible to configure Security Zone/User Role pairs. The user invoking the message handler must match one of the combinations of Security Zone/User Role listed or else the message handler will not execute.



Message Handler Settings
✕

**Name**

**Threading**

Shared ▾

**Enabled**

 true

**Security**

Current zone and role must match one of these entries to execute this message handler. See 'Security' under the Configure tab in the Gateway web page to learn more.

Security Zone	Role
Any	Any

+  
🗑  
🔄

## Using Message Handlers

Once you have your message handlers created, you can then call them from a script using one of three scripting functions: `system.util.sendMessage`, `system.util.sendRequest`, and `system.util.sendRequestAsync`. These functions allow you to call a message handler in any project, even if the project that the message handler resides on is different from the one you are calling it from. The message handler will then execute in the scope in which it was created, and will use any parameters that you pass in through the payload.

**Pseudocode - Calling a Message Handler**

```

project="test"
messageHandler="My Message Handler"
myDict = {'MyFirstValue': "Hello", 'MySecondValue': "World"}
results=system.util.sendMessage(project, messageHandler, myDict)

```

## Troubleshooting Gateway and Client Scripts

For both Gateway and Client scripts, Ignition gives you the tools to quickly check the status, troubleshoot, and diagnose problems with your scripts.

### Gateway Scripts

The Gateway has a special section (**Gateway webpage - Status > Systems > Gateway Scripts**) where you can quickly check to make sure your Gateway scripts are running properly. If any of your scripts have an error, you can find the details of the error to help you troubleshoot what went wrong. You can also find a list of logged errors for all Gateway Event Scripts under Log Activity. To learn more about statusing Gateway scripts and troubleshooting, refer to [Gateway Scripts](#).

### Client Scripts

The Console is very a important tool in Ignition for troubleshooting Client scripts. You can check to see if your script is working directly from the Client window, or the Designer while in Preview Mode. Any client scripting errors along with printouts go to the Console. The Console will identify the script name, error message, what line the script error is in, and a description of the problem.

To access the Console from a Client, go to the menubar and select **Help > Diagnostics > Console**. To access the Console from Preview Mode in the Designer, go to the menubar **Tools > Console**.

#### Related Topics ...

- [Project and Shared Scripts](#)
- [Tag Event Scripts](#)