

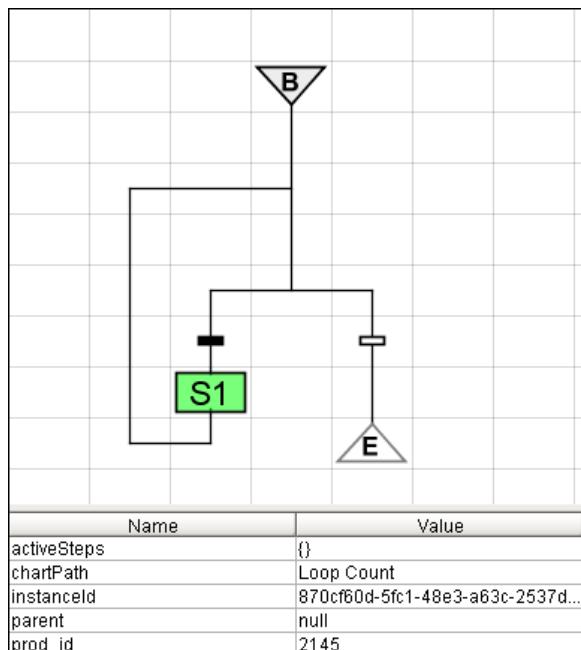
1. Sequential Function Charts	2
1.1 SFC Designer Interface	8
1.2 SFC Basics	15
1.2.1 Chart Flow and Rules	17
1.2.2 Chart Scope and Variables	24
1.2.3 Chart Properties	27
1.3 SFC Elements	30
1.4 SFCs in Action	42
1.4.1 Monitoring and Debugging Charts	46
1.4.2 Pause, Resume, and Cancel	48
1.4.3 Action Step Best Practices	50

Sequential Function Charts

What Are Sequential Function Charts?

A Sequential Function Chart (SFC) is a series of scripts that are defined in a single location, and then called in sequential order. Additional elements in the chart can determine where the flow of the chart will lead. Charts can loop around indefinitely, or execute a set number of times before ending. Sequential Function Charts are based on a graphical programming language in the IEC 61131-1 standard. This language may be familiar to PLC programmers, as it is one of the languages commonly available for programming PLCs.

SFCs are used to execute logic in ways that are more convenient to structure than with Python scripts or PLC programming alone. Because of their inherently visual depiction, they help to illuminate logic to users, and facilitate intuitive development and refinement. Charts can be monitored as they run visually, making troubleshooting easier than with scripting alone.



On this page ...

- [What Are Sequential Function Charts?](#)
- [When Should I Use a Sequential Function Chart?](#)
- [How Do Sequential Function Charts Work?](#)
 - [Simple Visual interface](#)
 - [Chart Elements](#)
 - [Chart Flow](#)
 - [Monitor Chart Activity](#)
 - [SFC Redundancy](#)
- [Sequential Function Chart Architecture Examples](#)
 - [Simple Chart](#)
 - [Incorporate a Handshake](#)
 - [Parallel Processes and Flow Control](#)

When Should I Use a Sequential Function Chart?

SFCs can be [used for many tasks](#), but they shine in the following conditions:

- **Situations where multiple processes need to run in parallel** - The nature of a chart allows for controlled execution. Pauses are handled by the chart, so there is no need to put threads to sleep.
- **When multiple processes must be completed in a specific order** - Charts always execute steps sequentially. A step will never become active out of order.
- **Complicated multi-step processes** - The nature of SFCs allows the user to visually build the work-flow of the chart, so troubleshooting is a breeze.
- **Linked processes** - In cases where several processes should **only** be called together. Scripts in a chart can only be invoked by the chart, so external scripts or resources will not be able to directly call the code from any of the steps.

How Do Sequential Function Charts Work?

SFCs are [built in the Designer](#), and executed on the Gateway, so they run independently of any Clients. They make use of both Python and Ignition's Expression language, so any number of tasks are possible from a single chart. A single SFC in Ignition can be called multiple times. Parameters can also be passed into a chart as it starts, so multiple instances can work on separate tasks individually.

Simple Visual interface

Charts elements are drag-and-drop, and work similarly to the components you are used to using in the rest of Ignition.

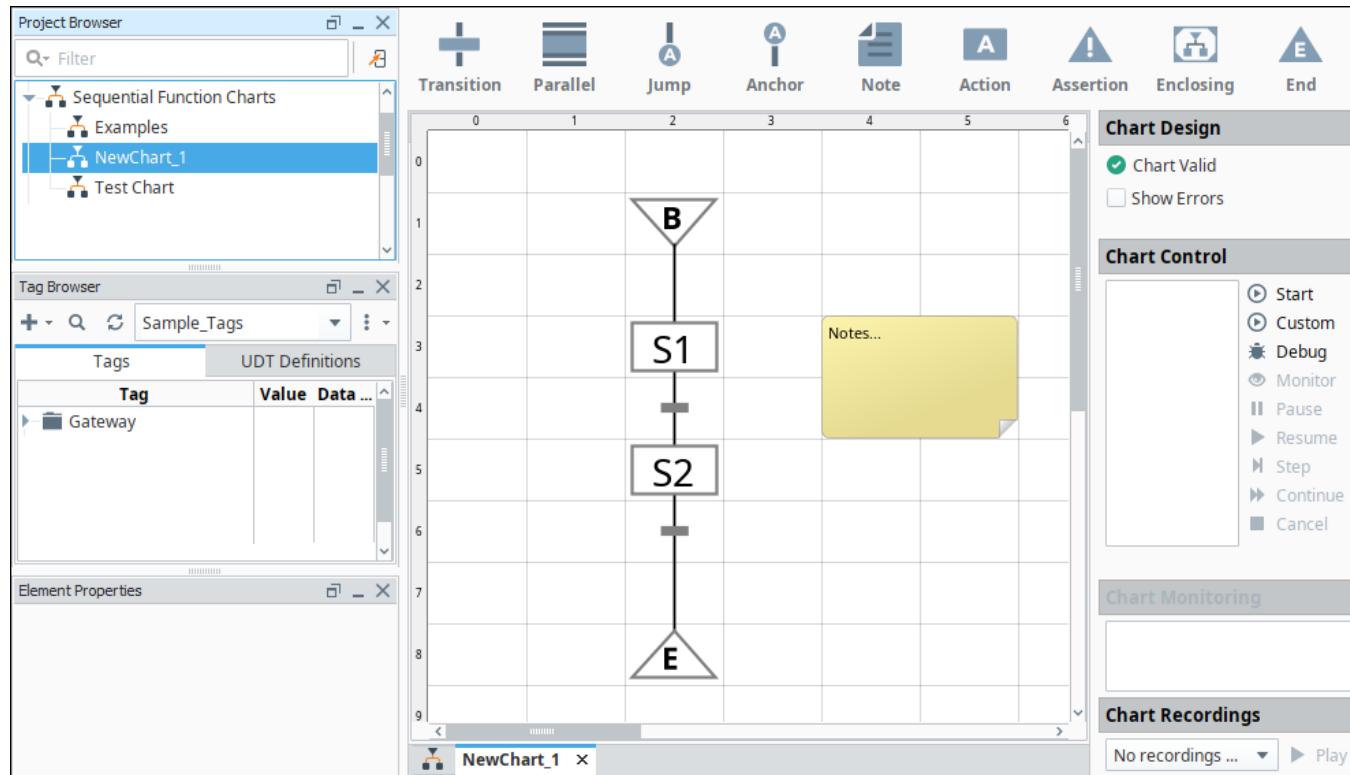


Chart Elements

Charts are comprised of [elements](#), and these elements perform the work in a SFC. Each element does something different, but they generally serve to either control the flow of the chart, or execute one or more Python scripts.

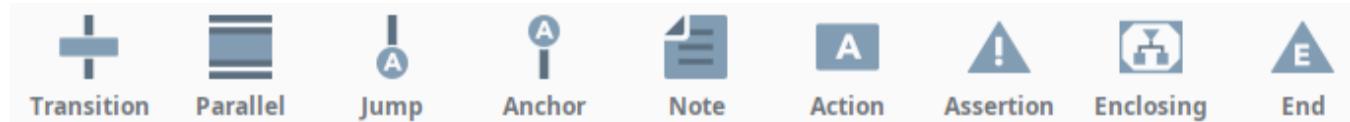
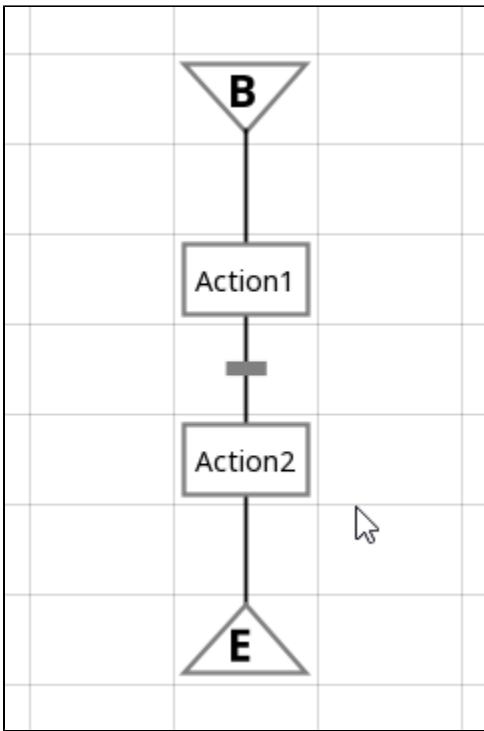


Chart Flow

[Charts always flow in the same way](#). They start at their begin step, and the logic of the chart typically flows from the top to the bottom, however charts are able to loop back to previous steps. Doing so allows for looping logic to be built directly into the chart. Flow of the chart can be halted by a transition element. The state of the transition can update in realtime, so a chart can pause until a user approves the chart to move on.



Monitor Chart Activity

Simple HMI interfaces can be developed to [manage the SFC](#). An SFC can be started with a simple button or it can be managed with the [Vision - SFC Monitor](#) component.

Project Browser

- Windows
 - Main Window
 - Main Window2
 - Main Window3
 - PopupTest
- SFC
 - Root Container
 - SFC Monitor
- Templates
- Named Queries
- Reports

Vision Property Editor

Common	
Name	SFC Monitor
Visible	<input checked="" type="checkbox"/> true
Border	Other Border
Data	
Instance ID	
Scope Dataset	<No Data>
Appearance	

Component Palette

- Admin
 - SFC Monitor

Loop Count
Running 00:03:36
870cf60d-5fc1-48e3-a63c-2537d...

Name	Value
activeSteps	{}
chartPath	Loop Count
instanceId	870cf60d-5fc1-48e3-a63c-2537d...
parent	null
prod_id	2145

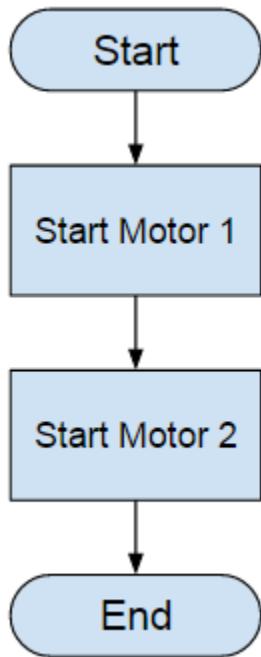
SFC Redundancy

Sequential Function Charts support redundant Gateway clusters and will persist over gateway failovers using the [Redundancy Sync](#) property. A Backup Gateway will now pick up where the Master left off, or the chart can be [canceled, restarted, or even set to run at a different step](#).

Sequential Function Chart Architecture Examples

Simple Chart

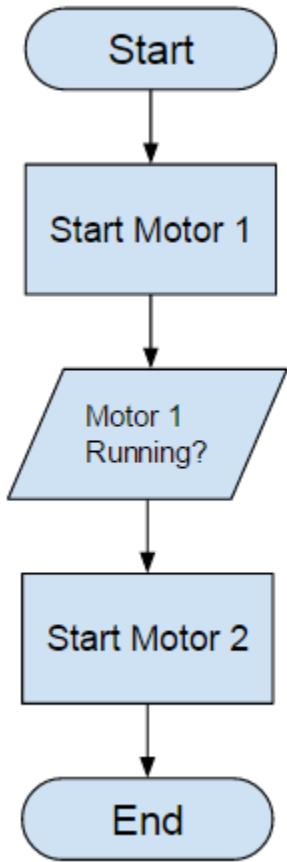
Performing multiple actions with a single call is easy to do with SFCs. Let us assume several motors all need to start from a single call. The work-flow would look like the following:



- The chart would **Start** and then move to the first motor.
- The **Start Motor 1** action would then run a script to start Motor 1.
- Once the script finishes, the chart then flows to **Start Motor 2**, and calls a script that would start Motor 2.
- When the second script finishes, the chart flows to the **End** step, and concludes the chart.

Incorporate a Handshake

In many cases, a chart will need to wait for some other system to finish with a task before moving on. This is similar to receiving a handshake from the PLC before moving on. Charts can freely read and interact with the rest of Ignition, so a step in a chart can read a tag, run a query, make a web services call, read a local file, or do anything that is possible from a Python script. A chart could wait for a specific value on a tag, and then proceed after the value has met some set-point.

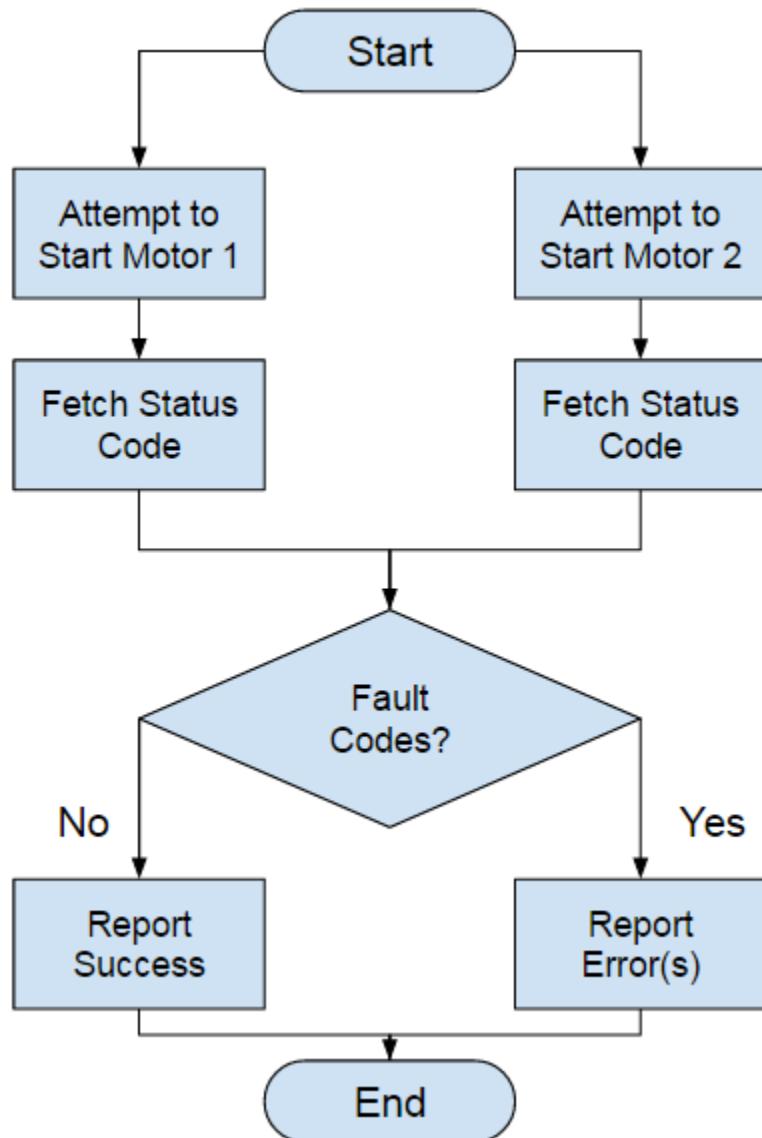


- **Start** the chart
- Run the script on **Start Motor 1**
- **Wait** until the PLC sets the value of a specific tag to a "Running" status code. All other scripts are on-hold while waiting.
- Run the script on **Start Motor 2**
- **End** the chart

Parallel Processes and Flow Control

SFCs work great when multiple processes must run simultaneously. Transitioning from one step to another only occurs when the active step finish executing. This means multiple steps can execute in parallel, and later steps will not begin until all of the currently active steps have finished. This type of control is normally very difficult to accomplish with just Timer or Tag Change scripts because each script needs to be able to notify the other script once complete. SFCs allow the chart to monitor each step, and determine when it is time to move forward.

Charts can also make use of local parameters. After reading values from outside the chart, these values can be stored in a parameter on the chart. The value of these parameters can then be referenced by other elements, and the chart can decide where the flow should move towards.



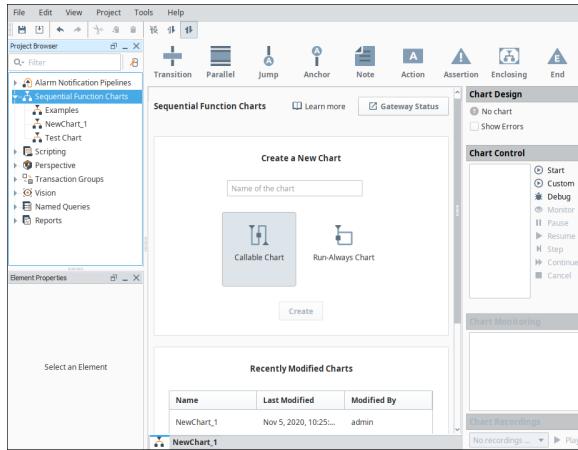
- Start the chart.
- Attempt to start both **Motor 1** and **Motor 2**.
- Check the **Status Code** from both motors. Pause all additional activity until a response code is retrieved from each motor.
- If **Fault Codes** were returned, then **Report the Errors**, otherwise **Report the Success**.
- End the chart.

[In This Section ...](#)

SFC Designer Interface

The SFC module features a unique user interface within the Designer. For starters, the SFC Interface has a Welcome tab that allows you to create two types of charts: Callable and Run-Always. Each chart type is basically a template to help you get started creating your system function chart. Once you select a chart type, enter a name, and press 'create', and the the specific chart template will open. You will have some of the necessary elements displayed in the chart workspace to begin designing your chart. The SFC Welcome tab will show you any recently modified charts along with the date it was modified and who modified it. You can even double click on a recently modified chart and open it.

The SFC Welcome tab provides a quick way to create new charts and update existing charts.

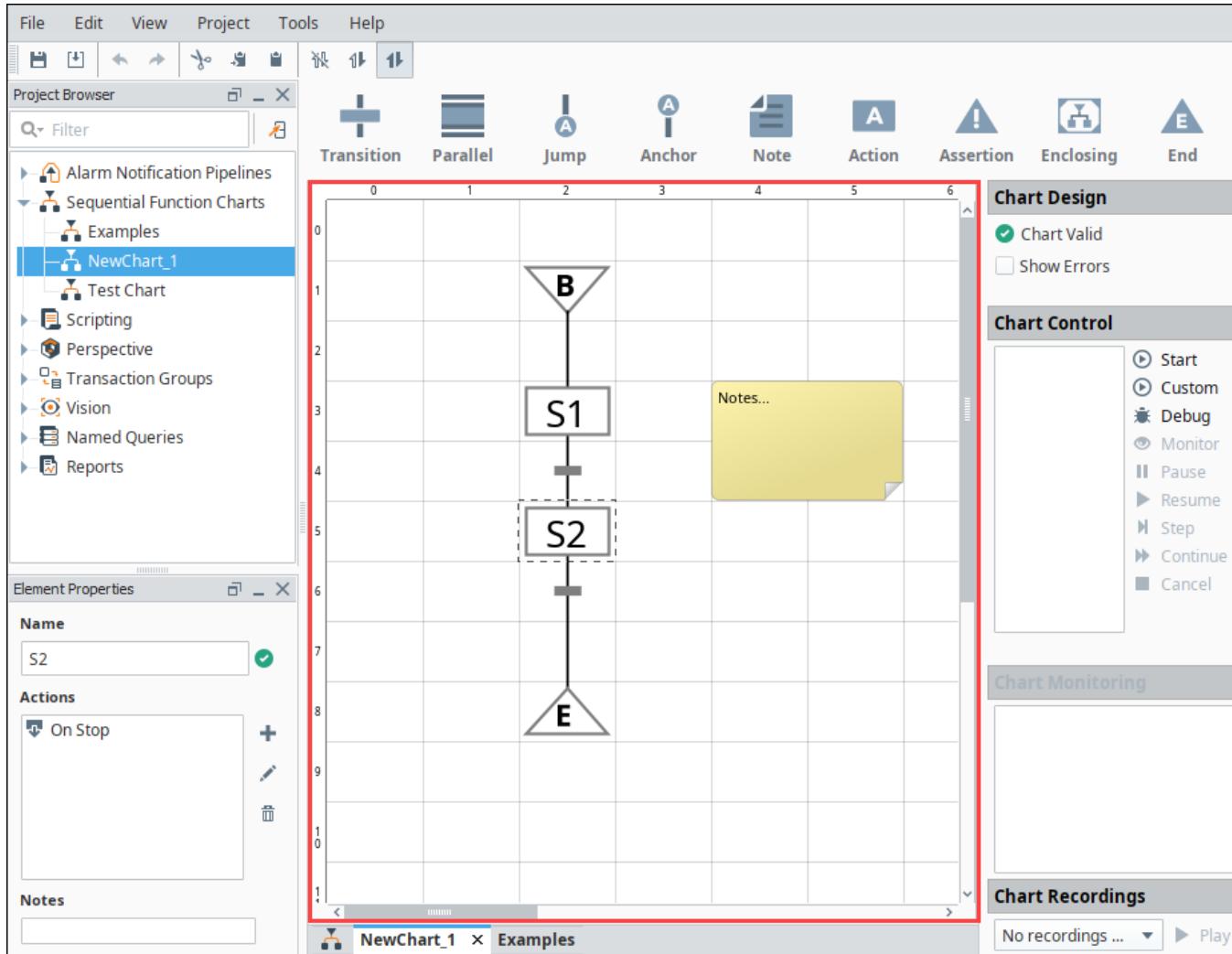


On this page ...

- [Chart Workspace](#)
 - [Design View and Monitor View](#)
 - [Right-Click Menu](#)
- [Element Blocks](#)
- [Element Properties](#)
- [Chart Design](#)
- [Chart Control](#)
- [Chart Monitoring](#)
- [Chart Recording](#)
- [Gateway Settings](#)

Chart Workspace

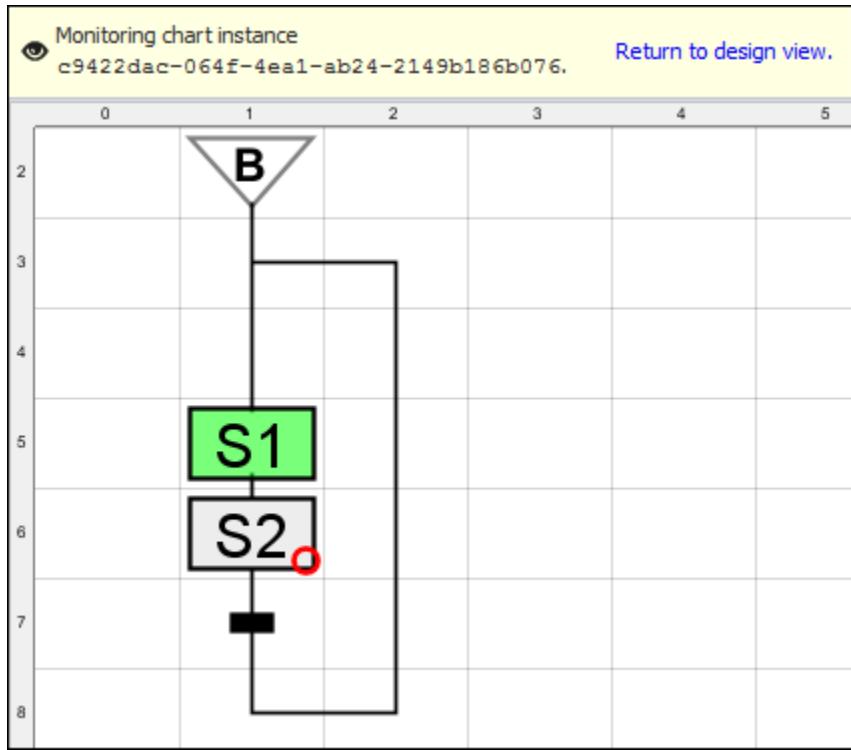
The main workspace of a chart features a grid, and is where your elements will be placed and selected.



Design View and Monitor View

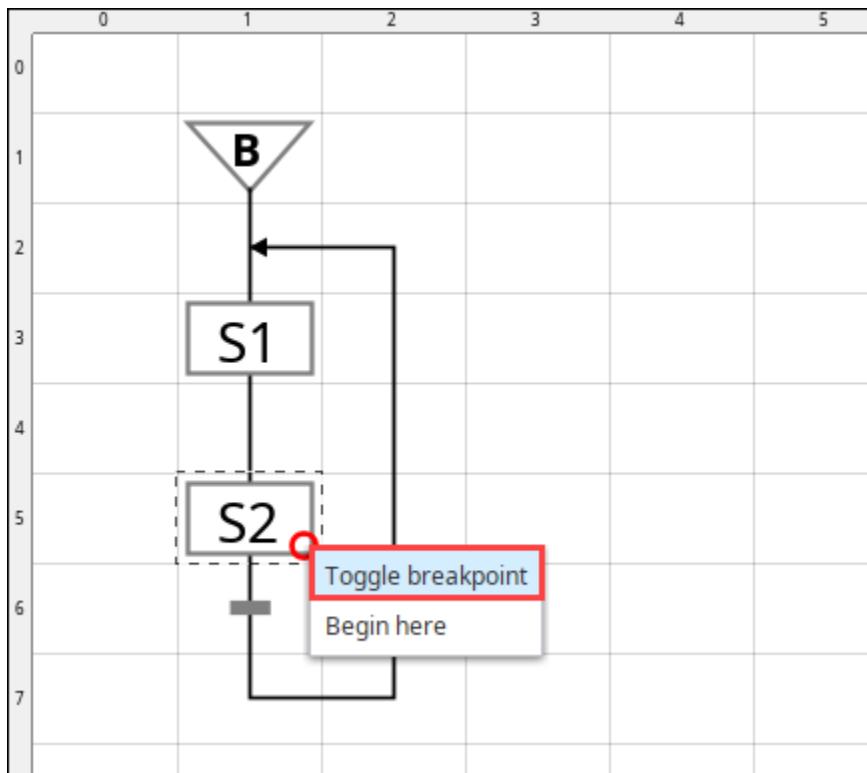
Initially the SFC interface is in **Design View**, which means new elements can be added, repositioned, or edited. This is similar to the Design Mode in the Vision module.

When monitoring charts, the interface switches to **Monitor View**, allowing the Designer to view the current state of any running charts. You may switch into Monitor View by double-clicking a chart in the Chart Control list, or by selecting a chart and clicking the **Monitor** icon. Once ready to make changes, the "Return to design view" link will revert the workspace back to Design View.



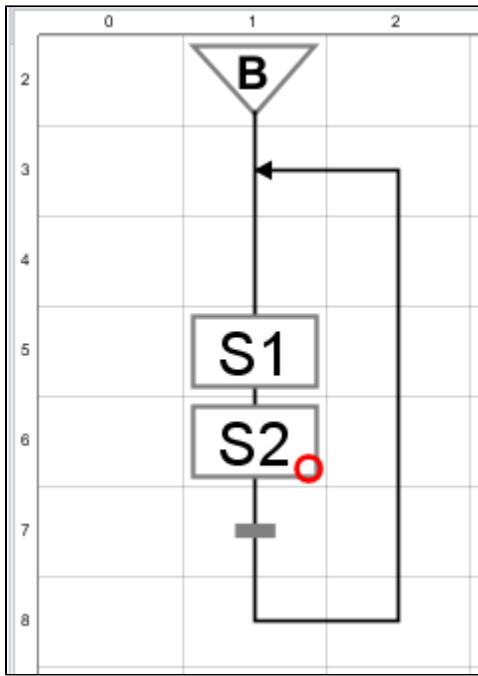
Right-Click Menu

When right-clicking on an Action, Assertion, Begin, End, or Enclosing block, a popup menu will appear with the following options



Toggle Breakpoint

Applies a breakpoint to a block, which will pause the chart when flow reaches the block in **Debug Mode**. Below we see that the S2 element has a breakpoint applied

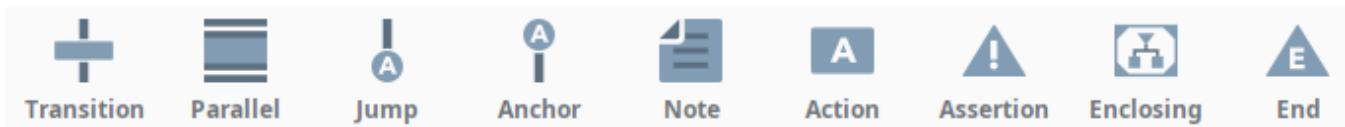


Begin Here

Starts a new instance of the chart at the specified element. This is similar to using the **Custom** icon under Chart Control, in that you can potentially set the value for any parameters.

Element Blocks

Provides a single location to drag and drop elements into the chart. The various blocks are listed under the [SFC Elements](#) page.



Element Properties

This panel provides a way to interact with selected elements. The contents of the panel change based on which element is selected. See [SFC Elements](#) for more details.

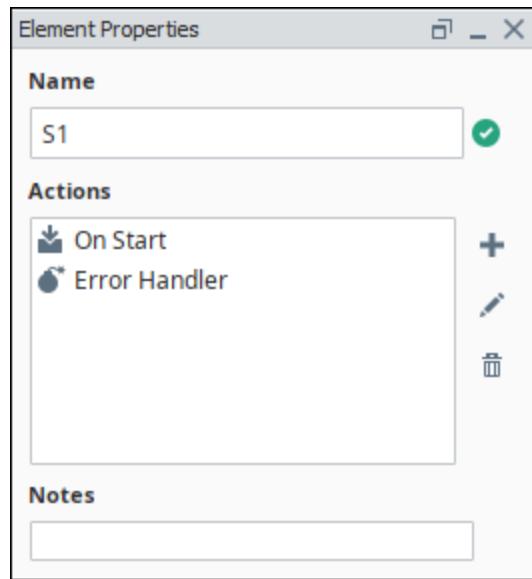
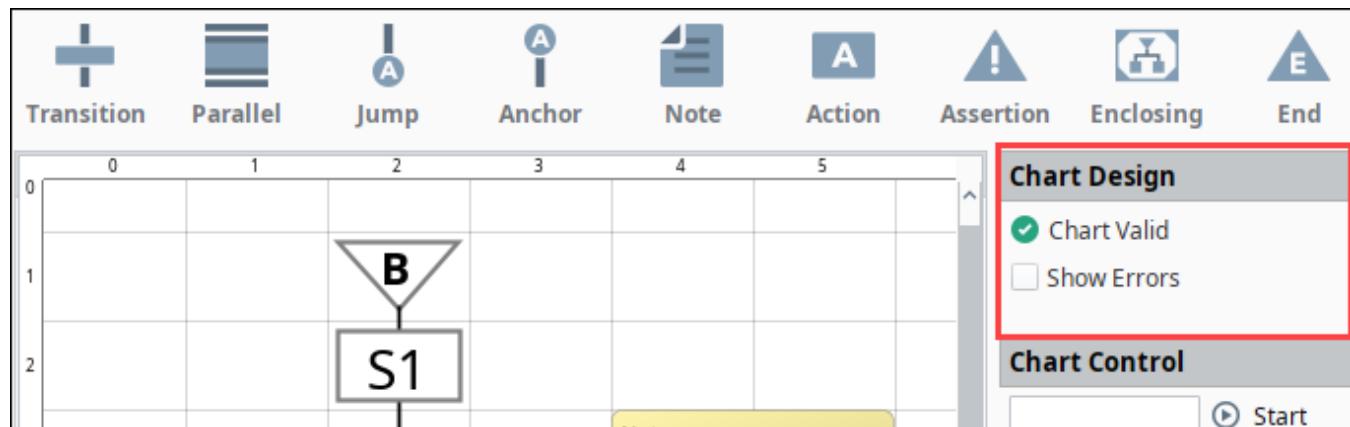


Chart Design

This section of the interface provides helpful diagnostic information. The section will state if a chart is valid.



In the following example, the link after S1 is broken. You'll notice a **Chart Invalid** icon appears. And if you click the **Show Errors** checkbox, the error is annotated with a red triangle in the chart.

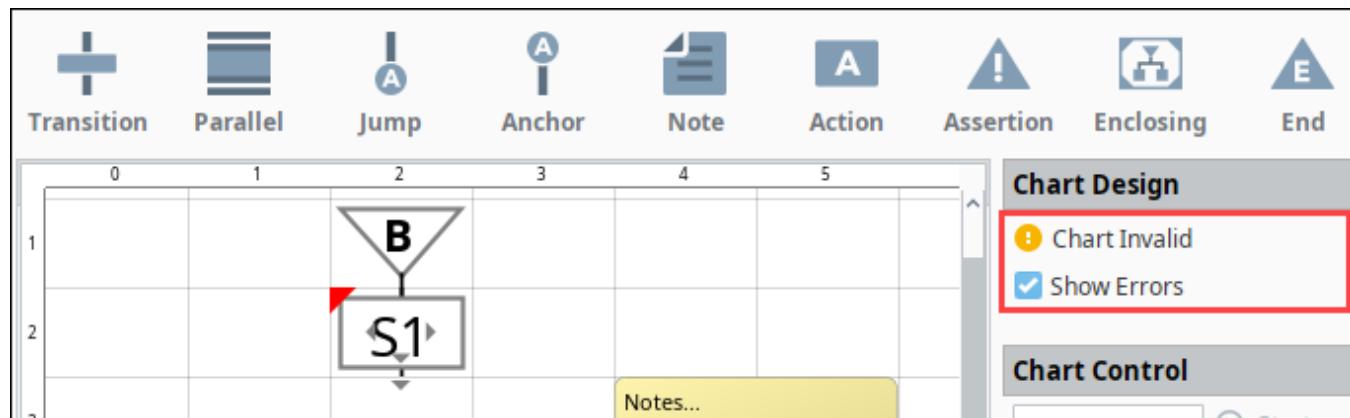
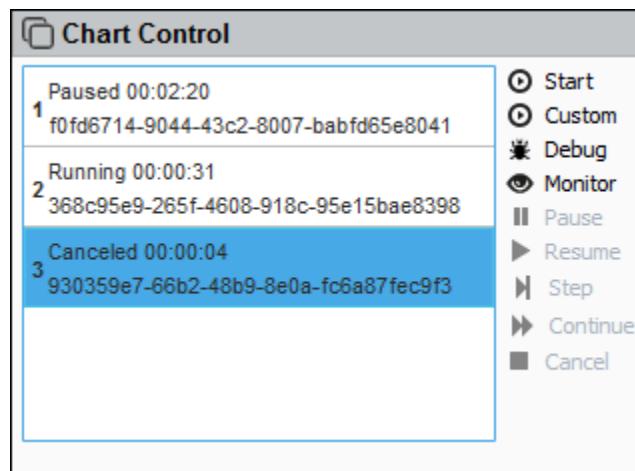


Chart Control

The Chart Control section allows you to Start or Stop charts, as well as view any running charts.



1	Paused 00:02:20 f0fd6714-9044-43c2-8007-babfd65e8041
2	Running 00:00:31 368c95e9-265f-4608-918c-95e15bae8398
3	Canceled 00:00:04 930359e7-66b2-48b9-8e0a-fc6a87fec9f3

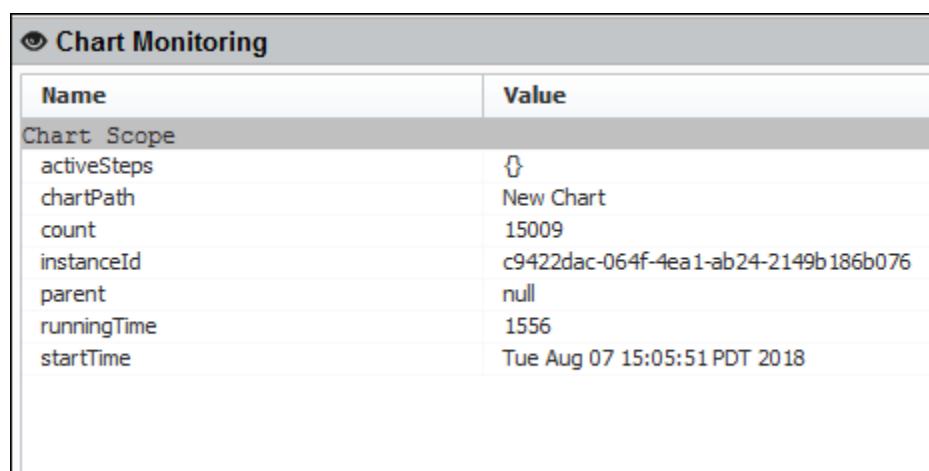
- Start
- Custom
- Debug
- Monitor
- Pause
- Resume
- Step
- Continue
- Cancel

Chart Control has the added feature of displaying information about any running charts, including the current state, duration, and instance id. Double clicking on a chart in the list will cause the interface to monitor the chart, similar to clicking on the **Monitor** link.

Icon	Description
Start	Start the chart that is currently viewable in the main workspace (selected tab at the bottom of the workspace).
Custom	Starts the current chart, but allows you to manually pass values into any chart parameters.
Debug	Starts the current chart, but stops at the first breakpoint. Allowing you to examine properties on the chart.
Monitor	Switches from Design View to Monitor View, allowing you to monitor a running chart.
Pause	Pause a running chart, allowing you to later resume the chart.
Resume	Resumes a previously paused chart.
Step	While halted at a breakpoint, moves the chart towards the next step. This icon is only available in Debug Mode.
Continue	Continue and break at the next defined breakpoint. This icon is only available in Debug Mode.
Cancel	Cancels the selected chart.

Chart Monitoring

This section is inactive in Design View, but becomes active while in Monitor View. Displays any properties associated with the running chart.



Name	Value
Chart Scope	
activeSteps	0
chartPath	New Chart
count	15009
instanceId	c9422dac-064f-4ea1-ab24-2149b186b076
parent	null
runningTime	1556
startTime	Tue Aug 07 15:05:51 PDT 2018

Chart Recording

This section populates with past chart executions, and allows you to replay them from the Monitor View. Only recordings that have finished running may be examined. By default, chart recordings are disabled. You can enable recordings in the Gateway settings.

Gateway Settings

There are several settings on the Gateway that impact chart recording for SFCs. By default, chart recordings are disabled. You can enable recordings in the Gateway **Config** section under **Sequential Function Charts > Settings**.

Chart Recording	
Chart Recording Enabled	When enabled, every detail of chart execution is recorded on disk, to aid in later analysis and debugging. Default is false.
Recordings Per Chart	The maximum number of recordings stored for each chart. Default is 5.
Prune Age	The maximum age of recordings stored on disk. Recordings will be deleted after this point.
Prune Age Units	Unit of time for the prune age. Options are: Milliseconds, Seconds, Minutes, Hours, Days, Weeks, Months, and Years. Default is Day.

Config > SFC > Settings

Chart Recording

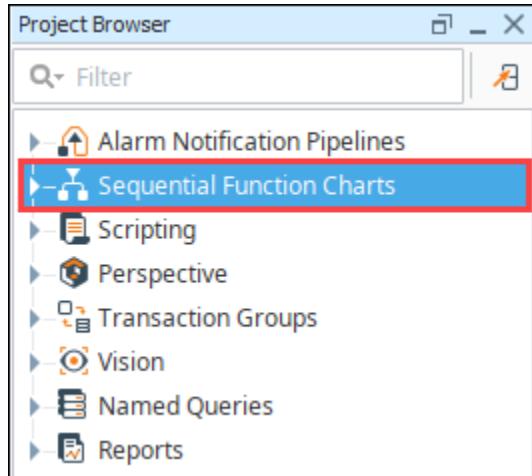
Chart Recording Enabled	<input type="checkbox"/> When enabled, every detail of chart execution is recorded on disk, to aid in later analysis and debugging. (default: false)
Recordings Per Chart	5 The maximum number of recordings stored for each chart. (default: 5)
Prune Age	7 The maximum age of recordings stored on disk. Recordings will be deleted after this point. (default: 7)
Prune Age Units	Days (default: DAY)

Save Changes

SFC Basics

Architecture

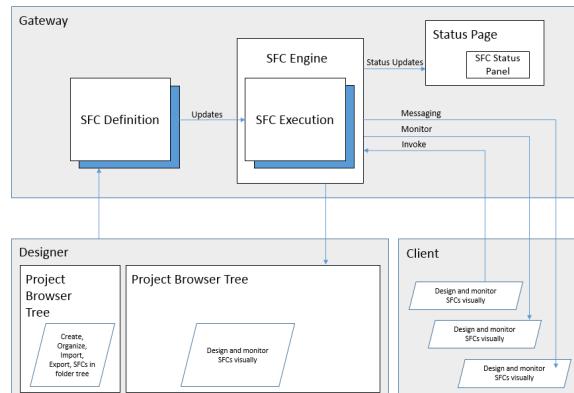
Sequential Function Charts (SFC) are designed through drag-and-drop manipulation in the Designer. The charts are located in a folder in the Designer's Project Browser. Charts are not part of any specific project; they are shared by all projects.



SFCs are executed in the Gateway. While any scope (Client, Designer, or Gateway) can start a new chart instance, the chart instance always executes in the Gateway.

SFC instances can be monitored in either the Client or the Designer. To monitor a running chart instance in the Designer, you can open that chart in the SFC workspace and double-click on the running instance. To monitor a running chart in the Client, a Vision project must be designed that uses the SFC module's monitoring panel component.

SFC Execution in Gateway



Run Modes

Each sequential function chart can be configured to use one of the following **Execution Modes**:

- **Callable**
This chart can be started via scripting or another chart's enclosing step on-demand. Any number of instances of this chart can be simultaneously running.
- **RunAlways**
This chart can be started by the Gateway upon startup. It can not be executed in any other way. It is probable that this chart be designed to never end, the idea being that there will always be exactly one instance of this chart running.
- **Disabled**
This chart is not available for execution.

On this page ...

- Architecture
 - SFC Execution in Gateway
 - Run Modes
 - Steps
 - Designing Charts
 - Chart Instance
 - Scripting Reference



Introduction to Sequential Function Charts

[Watch the Video](#)

Steps

Steps are the parts of the chart that do useful work. Steps are represented as a rectangle that occupies a one-cell region of the chart, except for the begin and end steps, which are triangles. Steps might run scripts, or execute other charts depending on how you set them up.

There are four types of steps:

- Begin and End Steps
- Action Step
- Assertion Step
- Enclosing Step

In addition to these, there are multiple [SFC elements](#) that allow for more complex logic in your SFC.

Designing Charts

Before a chart is executed, it must have a correct structure. When designing a chart, the Designer will constantly let you know whether or not your chart is valid. If your chart is not valid, you may choose to show the errors. The error shows up as red triangles in the corner of any element which has a problem. Hover your mouse over these elements to discover what is wrong with them.

Here are some rules about structure to keep in mind:

- Everything must be fully connected (except for Notes).
- Flow typically moves from top to bottom. All elements must be entered from the top and exit from the bottom (not the sides).
- There can be any number of end steps. If flow reaches an end step, the chart is stopped. If there are no end steps, it means that your chart must loop back upon itself to satisfy the connected rule.
- End steps are not allowed inside parallel sections.



INDUCTIVE
UNIVERSITY

Designing Charts

[Watch the Video](#)

Chart Instance

Each chart you define in the Designer may be invoked multiple times, and each invocation will start a new instance of that chart. The instances may be started with different starting parameters which affect how the chart works. Each instance runs completely independently of the other instances. The ability to have multiple instances of a chart is one important feature that makes SFCs within Ignition different than SFCs inside of PLCs.

Scripting Reference

For information about the scripting API methods available for sequential function charts, see [System Functions](#).

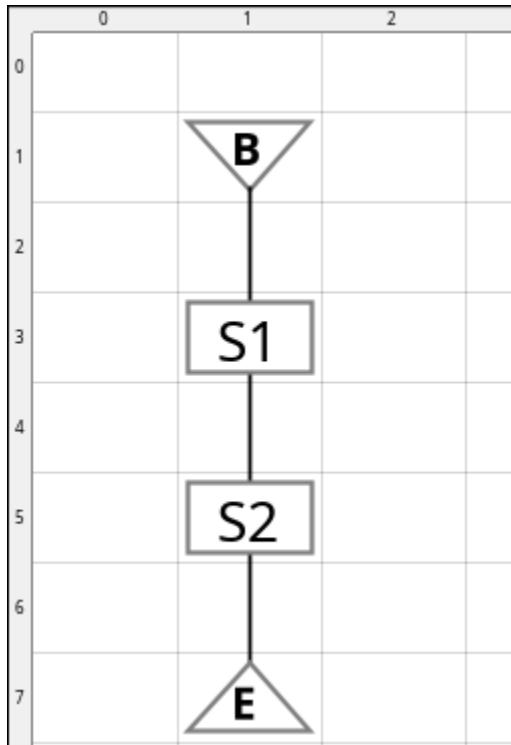
[In This Section ...](#)

Chart Flow and Rules

Understanding Flow

Similar to how water flows down a river, execution of an SFC chart flows along one or more paths. When a chart is running, flow typically moves from top-to-bottom. To reinforce this concept, begin steps only flow down, and end steps only accept flow from the top.

In the following chart, execution begins at the Begin Step (**B**), flows down into the first Action Step (**S1**), flows into the next Action Step (**S2**), and then into the End Step (**E**).



Vertical Movement

While charts generally flow from top-to-bottom, they can advance bottom-to-top. This allows for looping logic to be built directly into the chart: the chart will loop around until something either redirects the flow, or cancels the chart.

In the image below, the chart will flow out from the bottom of **S2**, travel up, and loop back around to the top of **S1**. The chart will continue looping around until canceled.

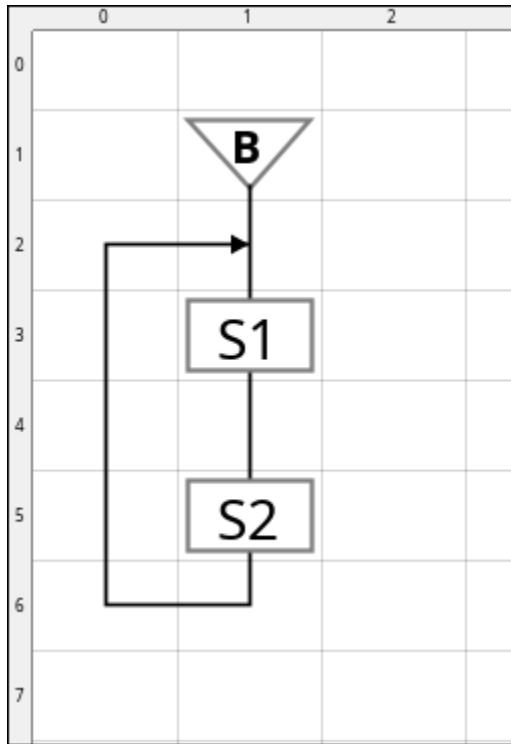
On this page ...

- Understanding Flow
 - Vertical Movement
 - Determining when to Proceed
 - Directing Flow
- Chart Rules
 - Flow Must Always Lead into the Top of an Element
 - All Elements Must Be Connected to the Chart
 - All Links Must Lead to an Element
 - Flow May Not Lead into Multiple Actions
 - Charts Do Not Need an End Step
- Chart Lifecycle
 - Active States
 - Terminal and Intermediary States



Chart Flow Steps and Transitions

[Watch the Video](#)



Determining when to Proceed

Before flow may exit from an Action, the following requirements must be met:

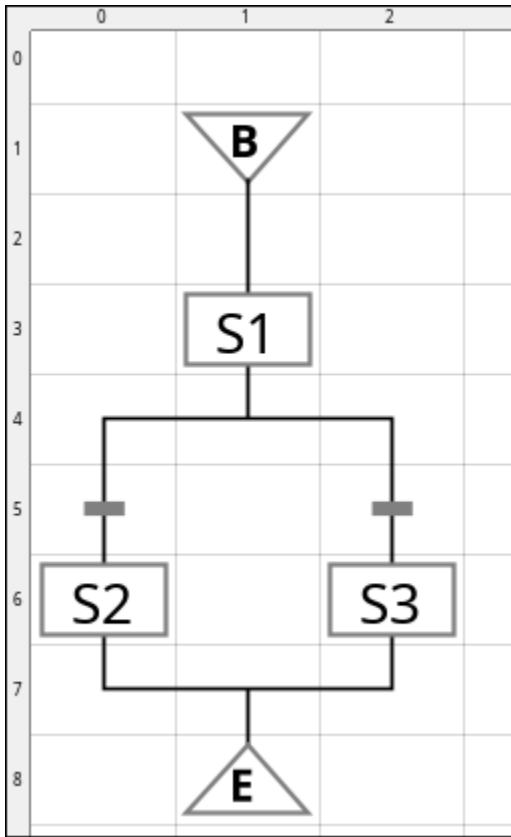
- **A sequential step must be available.** Flow of the chart must be able to reach a step, such as another Action Step, or an End Step. Transitions are generally used to control the availability of steps.
- **The active step must be ready to finish.** Action steps may have a script running when the next Transition opens or returns "true". As soon as an available path is ready, the step will finish any running scripts, and then flow will move on. In situations where a script may take a long time to finish, this waiting prevents partial execution, and guarantees that any attached scripts will be allowed to finish.

Once both condition are true, flow continues.

Directing Flow

After exiting an element, flow of the chart may branch off into multiple potential paths. When multiple paths are present, the chart will choose a single element. Transitions are used to help coerce flow of the chart when multiple paths are available. When the expression on a transition resolves as true, flow may proceed through. When a transition resolves as false, flow is halted.

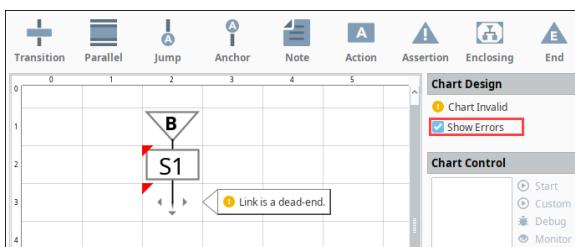
In cases where multiple open transitions are present, flow is biased towards the left-most path. In the image below, flow exits from **S1**, and has two possible paths. When both transitions are true, flow will always take the left path, so **S2** would execute next. In this scenario, **S3** would be ignored.



In most cases, transitions will not statically be set to a true value. By implementing more meaningful expressions on the transitions, the chart can determine which path to take while running. Furthermore, as long as both transitions are false, flow of the chart will be blocked, waiting for one of the transitions to become true. This structure is very similar to a logical OR: if either Transition is true, flow will continue.

Chart Rules

When developing a chart, there are several rules to keep in mind. Violating some of these rules will result in compilation errors on the chart. The Designer's interface will report any chart errors in the Chart Design section. When enabled, the **Show Errors** checkbox will highlight problem areas on the chart, as well as show the error when the mouse cursor is moved on top of the problematic area.



**INDUCTIVE
UNIVERSIT**

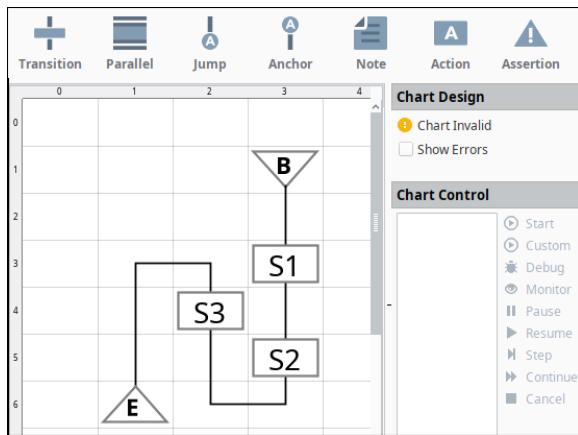
Chart Rules

[Watch the Video](#)

Flow Must Always Lead into the Top of an Element

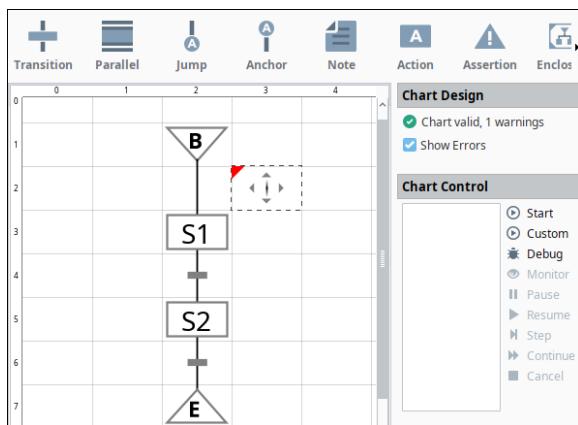
While moving bottom-to-top is legal, flow may never enter the bottom of an element. If the flow leads to the bottom of an element, it is considered a dead-end, and the chart will fail to execute.

The image below demonstrates an **illegal** chart. This chart will not execute because the flow from **S2** is attempting to lead into the bottom of **S3**.



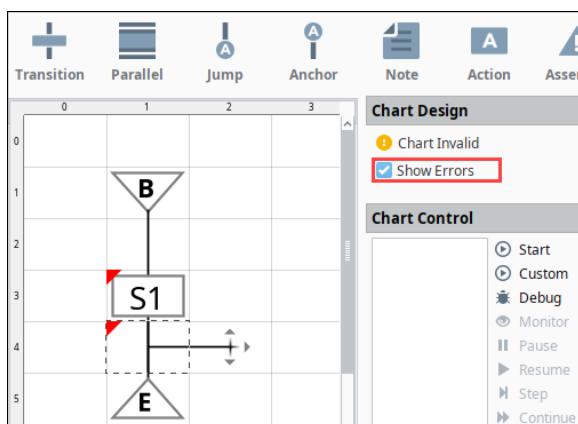
All Elements Must Be Connected to the Chart

Every element present should be linked to other elements in the chart. Rogue elements should be removed before the chart runs.

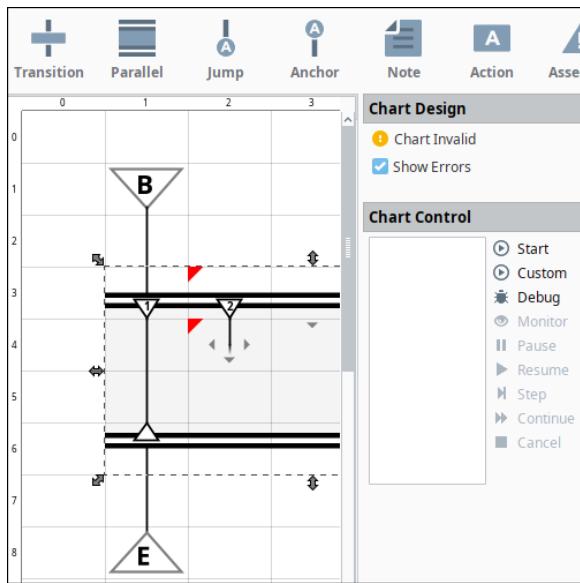


All Links Must Lead to an Element

Links must connect to an element. Any link that is not connected is considered a dead-end, and must be removed.

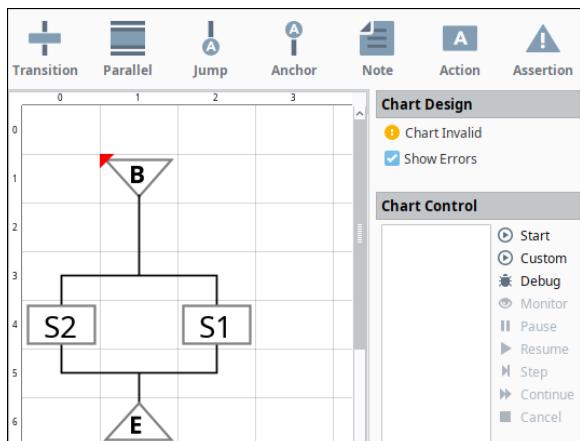


This rule extends to links created inside of a Parallel element.



Flow May Not Lead into Multiple Actions

Outside of a Parallel element, only a single action should run at any time. Flow that leads into multiple actions will cause a compilation error because the chart does not know which action to run. In cases where the chart flows into multiple Actions, transitions should be used to guide the flow. If multiple actions must execute simultaneously, a parallel element should be used.



Charts Do Not Need an End Step

Flow of the chart must always lead somewhere, but an End Step does not have to be used. Charts can freely loop around indefinitely. If flow is blocked by a Transition, then the chart is effectively awaiting input to execute.

Chart Lifecycle

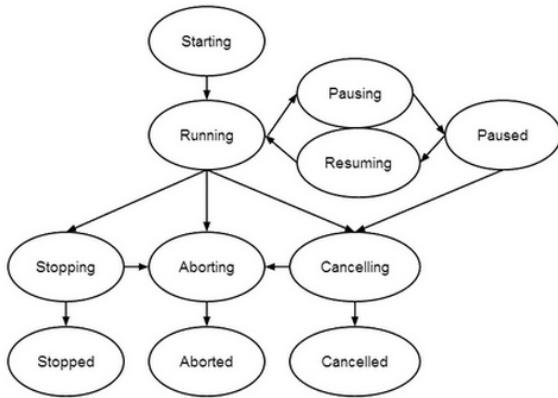
Each running chart is effectively a finite-state machine: the chart can be in one of many different states, but it is only ever in a single state at any given moment. The current state of the chart carries significant meaning.

The figure below shows each potential state a chart can be in, and outlines which states are accessible from each other state.



Chart Lifecycle

[Watch the Video](#)



Active States

The following states are common to the lifecycle of a running chart.

State	Description
Starting	The chart begins. The On Start Event Script triggers at this point.
Running	The chart flows through its structure. Running charts generally spend most of their time in this state.
Pausing	When the chart is requested to pause, it moves into this transitional state. The chart will attempt to finish execution on all active scripts, and will not start any additional scripts. Once all scripts have successfully finished, the chart will transition to the Paused state. Charts may be paused from either the chart Control section of the Designer, or by calling system.sfc.pauseChart .
Paused	The chart will sit here indefinitely until requested to resume, or is canceled. This is an idle state where the chart does not take any actions unless requested to do so.
Resuming	Once requested, the chart will start up and briefly enter this state. Resuming a chart can be done from the Chart Control section of the Designer, or by calling system.sfc.resumeChart .

Terminal and Intermediary States

Charts have three different terminal states, or end states. **Terminal states** denote that the chart has ended due to some reason. There are also three intermediary states that lead to the terminal states. When a chart ends in some way, the chart transitions to an **intermediary state** before moving to the associated terminal state. This allows the chart a chance to do some closing work before ending. The intermediary states are as follows:

State	Description
Stopping	The chart has reached an End Step, and will stop soon. In most cases, this is the preferred terminal state as the chart successfully reached its end. The On Stop Event Script will trigger at this state. Once finished, the chart will transition to the Stopped state.
Aborting	The chart has encountered some sort of error, and must end abnormally. The On Abort Event Script will trigger. Once On Abort ends, the chart will transition to an Aborted state.
Cancelling	Something requested the chart to cancel, or end before reaching the End Step. The On Cancel Event Script will trigger, and the chart will transition to a Cancelled state. Cancelling can occur from the Chart Control section of the Designer, or by calling system.sfc.cancelChart .

All three terminal states essentially mean the same thing for the chart: the instance will no longer run. The differences is how the chart ended. When troubleshooting, it is helpful to know why a chart ended; did it finish successfully, was there an error that prematurely ended the chart, or did something request that the chart stop?

Understanding how the chart ends can greatly aid in tracking down any problems. To that end, it is highly recommended to place scripts on the **On Abort** event. This will clearly log when a chart failed.

Chart Scope and Variables

Chart-Scope Variables

Charts can have variables created within them. The term “scope” means a collection of named variables that are accessible to the elements of a chart. Each chart instance gets its own, private scope. Scopes are basically free-form name-value maps, whose values may be any Python object, including scalar and multivariate types.

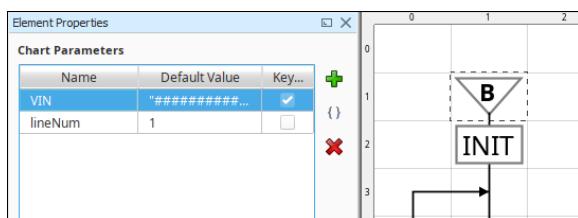
Each chart gets a scope object that can be accessed from all steps and transitions within that chart. When starting a chart (for example, from a script), you’ll be able to pass variables to the chart: those variables will appear in the chart’s scope.

By defining a variable as chart-scoped, all of the scripts and expressions in the chart may access the variable. Variables that will be referenced by multiple elements should be made chart-scoped.

Chart Parameters

The Begin Step of a chart allows chart parameters to be defined. Chart Parameters are chart-scoped variables that the chart expects to be initialized with. Values may be defined for each parameter, but can be overridden when a chart is called, or starts. If the parameter values are override, then the initial values will be ignored.

One of the chart parameters defined on the begin step may be marked as the **Key Param**. This means that this parameter may be used as an identifier for the chart. For example, suppose your chart defined the automation process for a car through an assembly line. You might define the car’s VIN as the key param. This means that instances of this chart may be identified by the VIN they were started with.



Note: SFC parameters are unable to store entire QualifiedValue objects, such as those returned by the `system.tag.readBlocking` function. However fields of a QualifiedValue, such as the value (`.value`), can still be stored in parameter.

Defining Variables in Actions

Chart-scoped variables may also be defined on Action Steps in a chart. To define a chart-scoped variable, “`chart.`” should be prepended before the variable name. Creating a variable called “`partNumber`” on an Action Step’s **On Start** would use the following syntax:

```
chart.partNumber = 111
```

As long as the name of the variable matches a pre-existing variable, then the new value will be assigned. If a script on an Action attempts to reference a variable that has not yet been defined, the script will create it.

Built-in Variables

There are a number of built-in variables maintained by the SFC engine that can be read through the chart scope.

SFC Built-in Variable	Description
<code>chart.instanceId</code>	The string UUID of the running chart instance.

On this page ...

- Chart-Scope Variables
 - Chart Parameters
 - Defining Variables in Actions
 - Built-in Variables
 - Reserved Words
 - Chart-Scope Variables in Transitions
 - Chart Monitoring



INDUCTIVE
UNIVERSITY

Chart Scope

[Watch the Video](#)

chart.startTime	A <code>java.util.Date</code> object that indicates when the chart instance started running.																																
chart.runningTime	An integer representing the number of seconds the chart has been running for.																																
chart.parent	The chart scope of the enclosing chart (if any). Value is null if this chart was not executed as part of an enclosing step.																																
chart.running	Returns true if the chart is in the running state.																																
chart.state	<p>This feature was changed in Ignition version 8.1.34:</p> <p>The following integer representations were changed in 8.1.34 to accommodate the addition of <code>InitPaused</code>, <code>Suspended</code>, and <code>RedundantInactive</code> states.</p> <table border="1"> <thead> <tr> <th>Value</th><th>State</th></tr> </thead> <tbody> <tr><td>0</td><td>Aborted</td></tr> <tr><td>1</td><td>Aborting</td></tr> <tr><td>2</td><td>Canceled</td></tr> <tr><td>3</td><td>Canceling</td></tr> <tr><td>4</td><td>Initial</td></tr> <tr><td>5</td><td>InitPaused</td></tr> <tr><td>6</td><td>Paused</td></tr> <tr><td>7</td><td>Pausing</td></tr> <tr><td>8</td><td>Resuming</td></tr> <tr><td>9</td><td>Running</td></tr> <tr><td>10</td><td>Starting</td></tr> <tr><td>11</td><td>Stopped</td></tr> <tr><td>12</td><td>Stopping</td></tr> <tr><td>13</td><td>Suspended</td></tr> <tr><td>14</td><td>RedundantInactive</td></tr> </tbody> </table>	Value	State	0	Aborted	1	Aborting	2	Canceled	3	Canceling	4	Initial	5	InitPaused	6	Paused	7	Pausing	8	Resuming	9	Running	10	Starting	11	Stopped	12	Stopping	13	Suspended	14	RedundantInactive
Value	State																																
0	Aborted																																
1	Aborting																																
2	Canceled																																
3	Canceling																																
4	Initial																																
5	InitPaused																																
6	Paused																																
7	Pausing																																
8	Resuming																																
9	Running																																
10	Starting																																
11	Stopped																																
12	Stopping																																
13	Suspended																																
14	RedundantInactive																																
chart.abortCause	Should the chart abort, returns the exception. Only available on the chart's On Abort event script.																																

Reserved Words

Certain chart-scoped variables may interfere with the internal functions of the chart. For example, creating a variable like `chart.values` will conflict with a Python dictionary's `values()` method and therefore the chart will show an error. Since SFCs use Python dictionaries to manage chart-scoped variables the methods associated with Python dictionary's act like reserved words.

In addition to the built-in variables above, the following names should be avoided when declaring chart or step variables:

clear	copy	fromkeys	get	has_key
items	keys	setdefault	update	values

Chart-Scoped Variables in Transitions

Because transitions use Ignition's Expression Language, referenced chart-scoped variables use different syntax. Chart-scoped variables can be denoted by typing the name of the variable between the "{" and "}" characters. The "partNumber" variable from above would look like the following:

```
{partNumber}
```

Referencing chart-scoped variables in Transitions allows you to easily control the flow of a chart. This is commonly used to create a **{counter}** that blocks flow of the chart until a certain condition has been met.

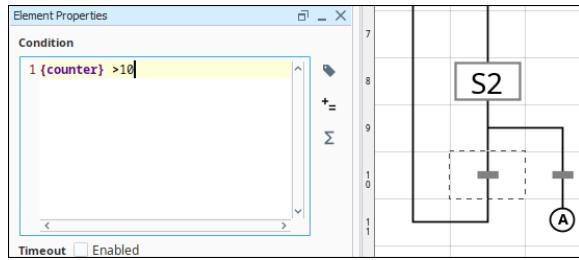
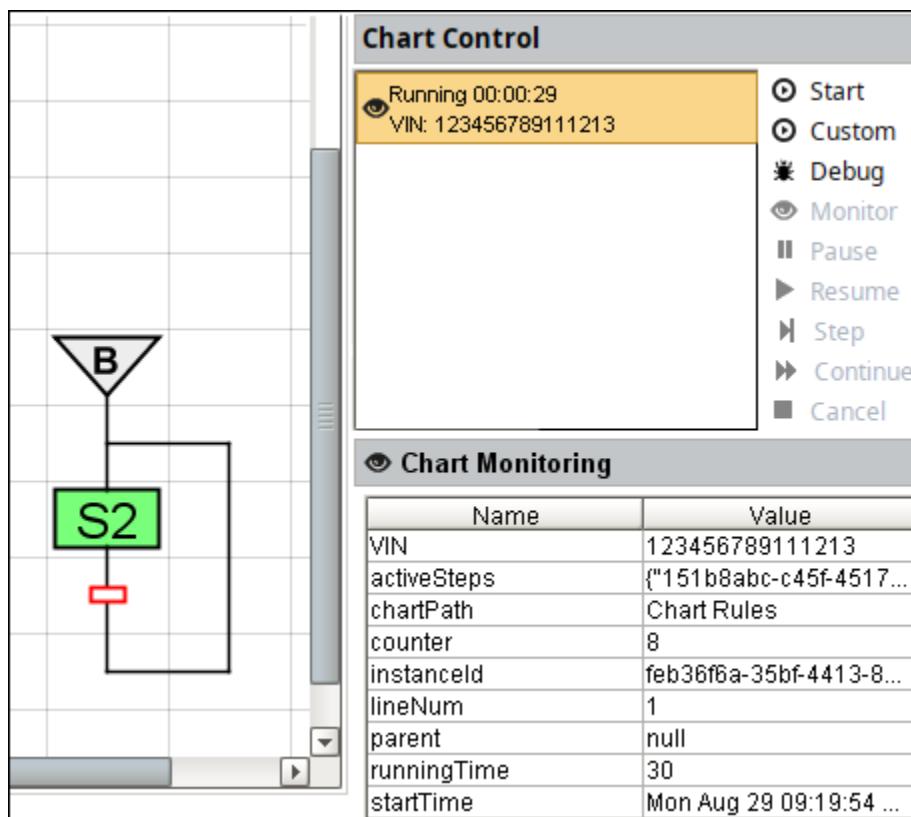


Chart Monitoring

Once chart-scoped variables have been defined, their values can be viewed while the chart is running in the Chart Monitoring section. This allows for easy troubleshooting from the Designer. More details can be found on the [Monitoring and Debugging Charts](#) page.



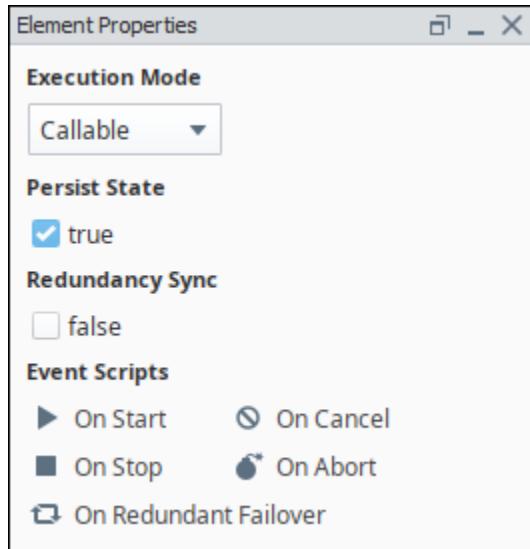
Related Topics ...

- [Chart Properties](#)

Chart Properties

Properties on a Chart

Much like properties on a [Vision](#) window, charts have properties that drastically modify their behavior. These properties will appear in the Element Properties panel when clicking on the background of a chart.



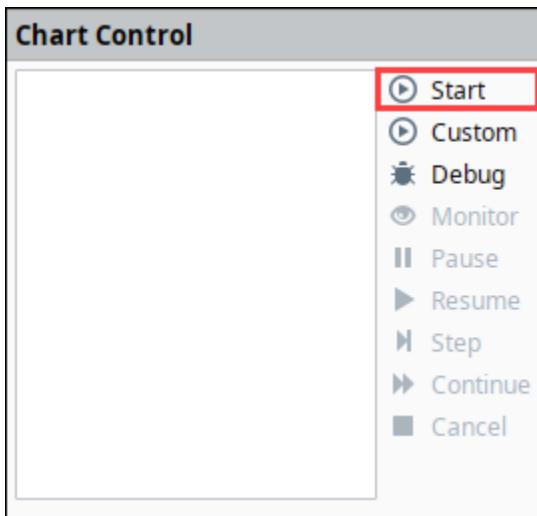
Execution Mode

This property determines how and when the chart should run.

Callable

Callable charts must be called before running. Any number of instances may run simultaneously. There are several ways to invoke a Callable chart:

- **Scripting** - `system.sfc.startChart` will invoke an instance of the chart.
- **Enclosing Step** - Charts using an Enclosing Step may call another chart.
- **Designer** - While viewing the chart in the Designer, chart instances may be called from the Chart Control panel by clicking the **Start** link.



RunAlways

On this page ...

- [Properties on a Chart](#)
- [Execution Mode](#)
 - Callable
 - RunAlways
 - Disabled
- [Persist State](#)
- [Redundancy Sync](#)
- [Event Scripts](#)



INDUCTIVE
UNIVERSITY

Chart Properties

[Watch the Video](#)

The chart will be initiated by the Gateway upon startup. It can not be executed in any other way. It is probable that this chart will be designed to never end, the idea being that there will always be exactly one instance of this chart running.

If the chart is aborted, canceled, or flow leads into an End Step, a new instance of the chart will not be called. Because of this, it is recommended to test your charts with the Callable execution mode, and design the Chart with a looping structure in mind.

Disabled

Disables the chart. This mode is great when a chart should be "shut off", such as when maintenance is working on a machine. Instances of the chart can not be called while set to **Disabled**.

Persist State

If enabled, each running chart will save its state when the Gateway is shut down. Prior to shutting down, the Gateway will record the state of each persistent chart to a file in Ignition's installation directory. Once the Gateway comes back online, the Gateway will read this file, and the chart will resume from where it left off. Values of each chart-scoped variable will be maintained between restarts.

Because the state is recorded on shutdown, all of the following conditions must be true for the state to be preserved:

- The Gateway has time to record the state before the operating system terminates the process.
- The chart is between script executions.

In the case of an expected shutdown (the operating system was requested to restart), all running scripts need to stop in a timely manner for the record to be taken, otherwise the operating system may force the Gateway to stop before the record is taken. It is recommended to design your chart in a manner that can easily be paused. This is usually accomplished by breaking up tasks into multiple smaller scripts. More details can be found on the [Action Step Best Practices](#) page.

In the event of unexpected shutdown, such as power to the server was cut-out, then the state will not be properly recorded. However, [Ignition Redundancy](#) can be used to protect against unexpected shutdowns.

Redundancy Sync

When enabled, the chart state and parameters will be synchronized across a redundant cluster, allowing a backup node to continue after chart execution. This can continue where the chart left off in the master, or the On Redundant Failover function can be used to restart, modify, or cancel the charts execution within the backup.

Synchronization only occurs when starting a new step. This means that any long running steps that updates variables multiple times throughout its execution will not be properly synced, and the variables will be out of date when starting at that step. The `system.sfc.redundantCheckpoint` function can create a "checkpoint" in the step execution, allowing you to manually sync the steps variables to the same step on the redundant node. The function can be used multiple times throughout a step, depending how long the step takes to execute, and how often values are updated.

Event Scripts

Similar in concept to Event Handlers on Vision components, Event Scripts trigger when certain actions occur in a running chart. The following events are available:

Event Name	Description
On Start	This will run once when the chart is started.
On Stop	This will run once when the chart is stopped normally. Specifically, when flow leads into an End Step.
On Cancel	This will run once if the chart is cancelled.
On Abort	This will run once if the chart is aborted. This occurs when an error in a script causes the whole chart to fail. If the script on this event fails, a log message will appear in the Gateway console. The exception that caused the abort is available via <code>chart.abortCause</code>

Example - `chart.abortCause`

```
#Create a logger. Use the path of the chart so each chart will use separate loggers
logger = system.util.getLogger(chart.chartPath + " Logger")

#Invoke the logger. Use chart.abortCause to report the issue
logger.error("Chart Aborted. Reason: %s" % (chart.abortCause))
```

On Redundant Failover	<p>This will run once if the chart is activated due to a redundancy failover. Has two special arguments:</p> <ul style="list-style-type: none">• activeSteps - A reference to the steps in the chart that are about to become active• restartAction - A dictionary containing settings that allow the chart to be cancelled, restarted from the beginning, or set to a specified step when this script fires. You cannot specify a parallel block as a GOTO step Examples:<ul style="list-style-type: none">◦ restartAction.cancel=True◦ restartAction.goto=S2
-----------------------	--

Related Topics ...

- [SFC Elements](#)

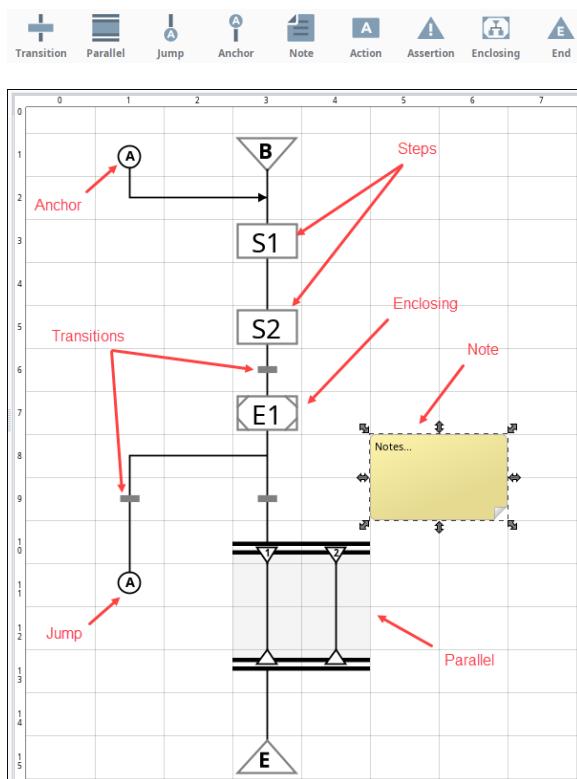
SFC Elements

SFC Elements are the driving force of a chart. Each element has a purpose or dedicated functionality, and can be combined with other elements to perform complex tasks.. In this section you will see the basic elements that make up every chart. For more information about putting them together, see [SFCs in Action](#) .

The chart has some configuration that can determine how and when the chart is started up, as well as opportunities to respond to chart lifecycle events with scripting, such as onStart, onStop, onCancel, and onAbort, see [Chart Lifecycle](#) for details.

Chart Elements

There are many elements available for charts, similar to Alarm Pipeline Blocks or other Ignition Components. These elements can be combined in various ways to create a flow chart with logic.



On this page ...

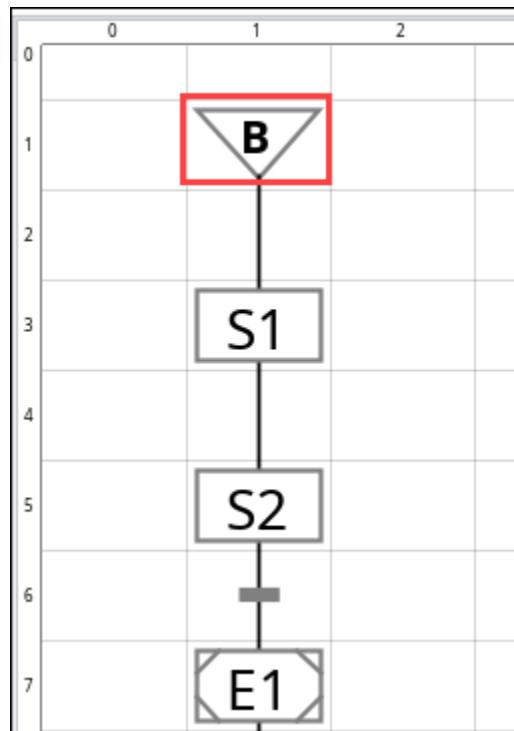
- Chart Elements
- Begin and End Steps
 - Begin Step
 - End Step
- Action Step
- Action Options
 - On Start
 - On Stop
 - Timer
 - Error Handler
- Assertion Step
- Transition
- Parallel Element
- Jump and Anchor
- Enclosing Step
 - Parameter Passing
- Links
- Notes



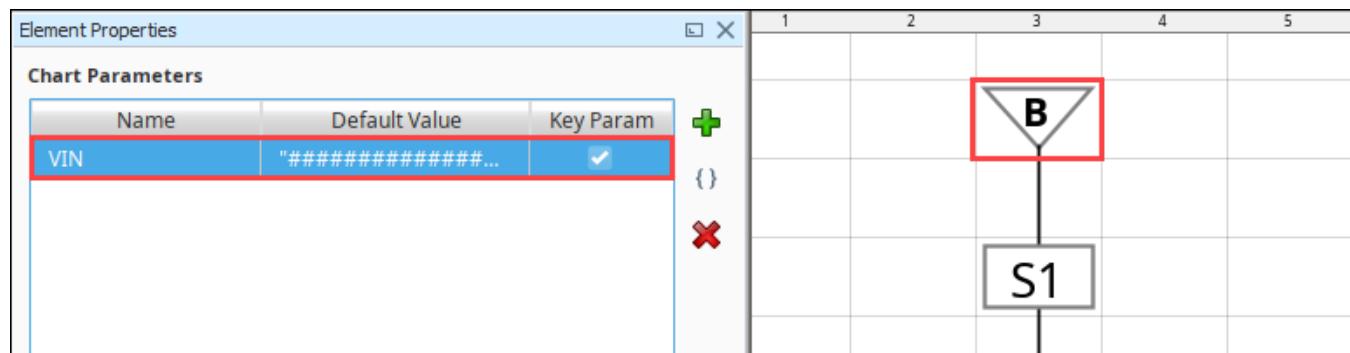
Chart Elements

[Watch the Video](#)

The **Begin Step** is where all charts start, and cannot be removed, cut, or copied. The begin step is where you can define initial values for your chart's scope. These initial values are also hints as to what parameters your chart expects. If the chart receives any of these parameters as starting parameters, the initial values are ignored.

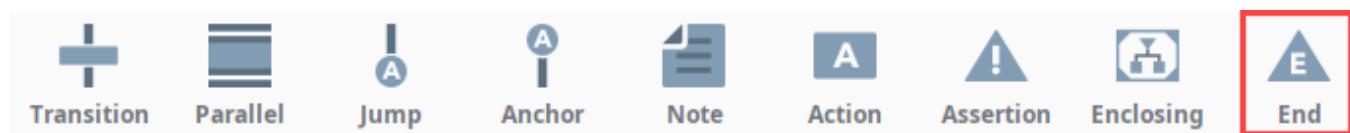


One of the chart parameters defined on the Begin step can be marked as the **Key Param**. This means that this parameter is used as an identifier for the chart. For example, suppose your chart defined the automation process for a car through an assembly line. For example, you can define the car's VIN as the key param, which means that instances of this chart are identified by the VIN they were started with.

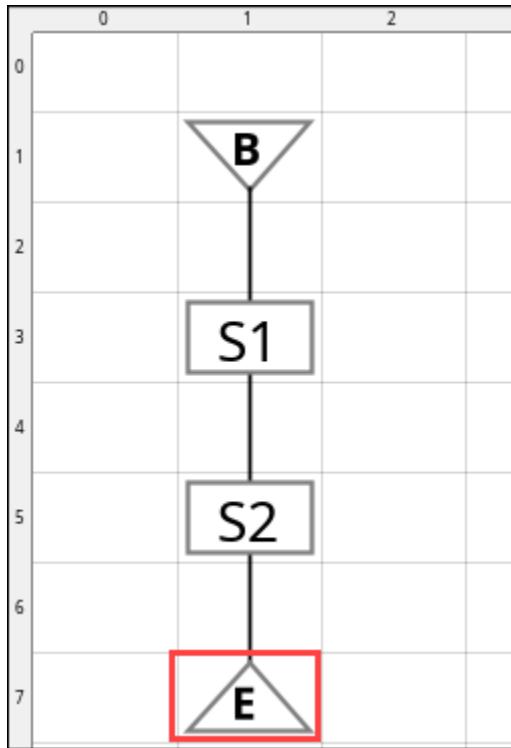


End Step

The **End Step** of a chart has no configuration. This is used to mark the termination of the chart. When a chart reaches this step, it stops executing.



There can be many end steps in a chart, although only one is ever reached for a given chart instance. End steps are not allowed inside parallel sections.

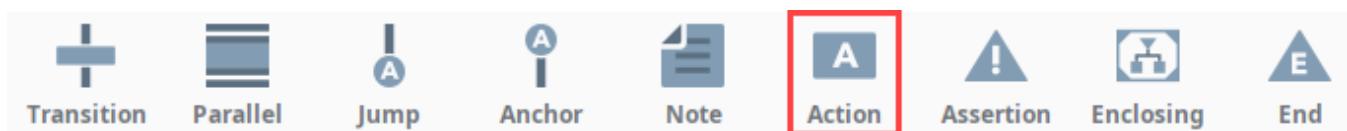


Action Step

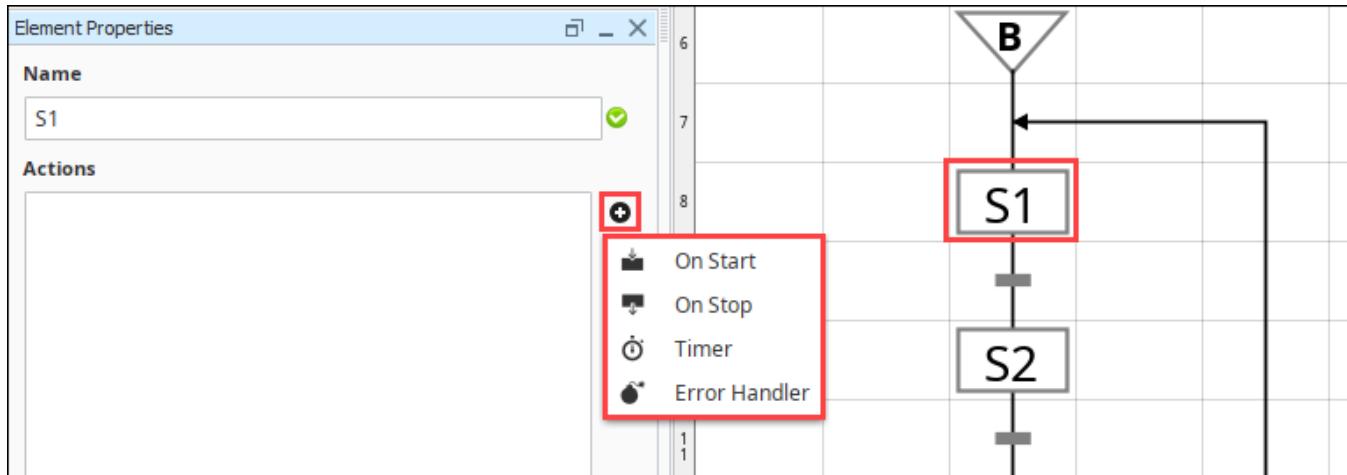
Action steps do the bulk of the work in an SFC.

The action step can have any number of scripts associated with it. Each of these scripts is called an action. There are various kinds of actions. The different kinds of actions execute at different times in the step's lifecycle. During the lifecycle of the step, many actions can run, but only one action is ever allowed to run at a time.

The scripts configured in the action step have access to the chart's scope, as well as a special scope that is private to the step. This scope, called "step scope" is initialized when the step starts and destroyed when the step stops, but retained while the step is running. This is a good place for timer actions to store intermediate results.



Action Options



On Start

This action runs when the step is started. These actions always run to completion before flow can move beyond the step, and before any other scripts on the action step will run. This action will always run at least once.

On Stop

When flow is ready to move beyond the step, it is told to stop. The step will then wait for any currently executing action, for example, **On Start** or **Timer** actions, to finish. Then it will execute its **On Stop** action, if configured. Only after these actions complete will the chart be allowed to move on. This action will always run at least once.

Timer

Timer actions run every so often while the action step is running. The timer actions only start running after the **On Start** action finishes (if there is one). In addition to the timer action's script, it must have a rate, specified in milliseconds. This is the amount of time to wait between running the timer action's script. The clock starts after the **On Start** action finishes.

It is important to realize that, unlike **On Start** and **On Stop** scripts, a timer action can not run at all for an action step. If the step is ready to stop before the timer is ready, it will never run.

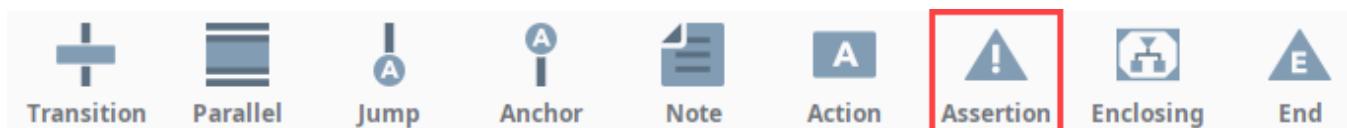
Error Handler

This action will run only if any of the other actions throw an unexpected error. This provides a chance for chart designers to do their own error handling, and gracefully recover from an error. If this script throws an error, the chart will abort, see [Chart Concepts](#) for more information.

Assertion Step

Check some conditions before moving on.

The Assertion Step makes one or more conditional assertions and will abort the chart or set a flag based on the result. Each assertion looks at a chart parameter and checks its value. If the value meets the condition, then the assertion passes, and the chart moves on.



Multiple assertions can be added to each Assertion Step, with each requiring a parameter name, an operator, and a value.

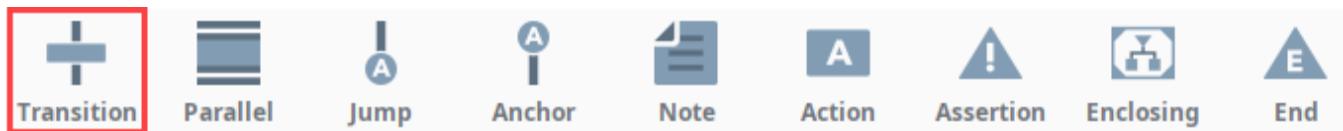
Element Properties

Name <input type="text" value="S2"/> 											
Assertions <table border="1"> <thead> <tr> <th>Parameter</th> <th>Operator</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>new_param</td> <td>=</td> <td>5</td> </tr> <tr> <td colspan="3">   </td> </tr> </tbody> </table>			Parameter	Operator	Value	new_param	=	5	 		
Parameter	Operator	Value									
new_param	=	5									
 											
Upon failure: <input checked="" type="radio"/> Abort <input type="radio"/> Set Flag: <input type="text"/>											

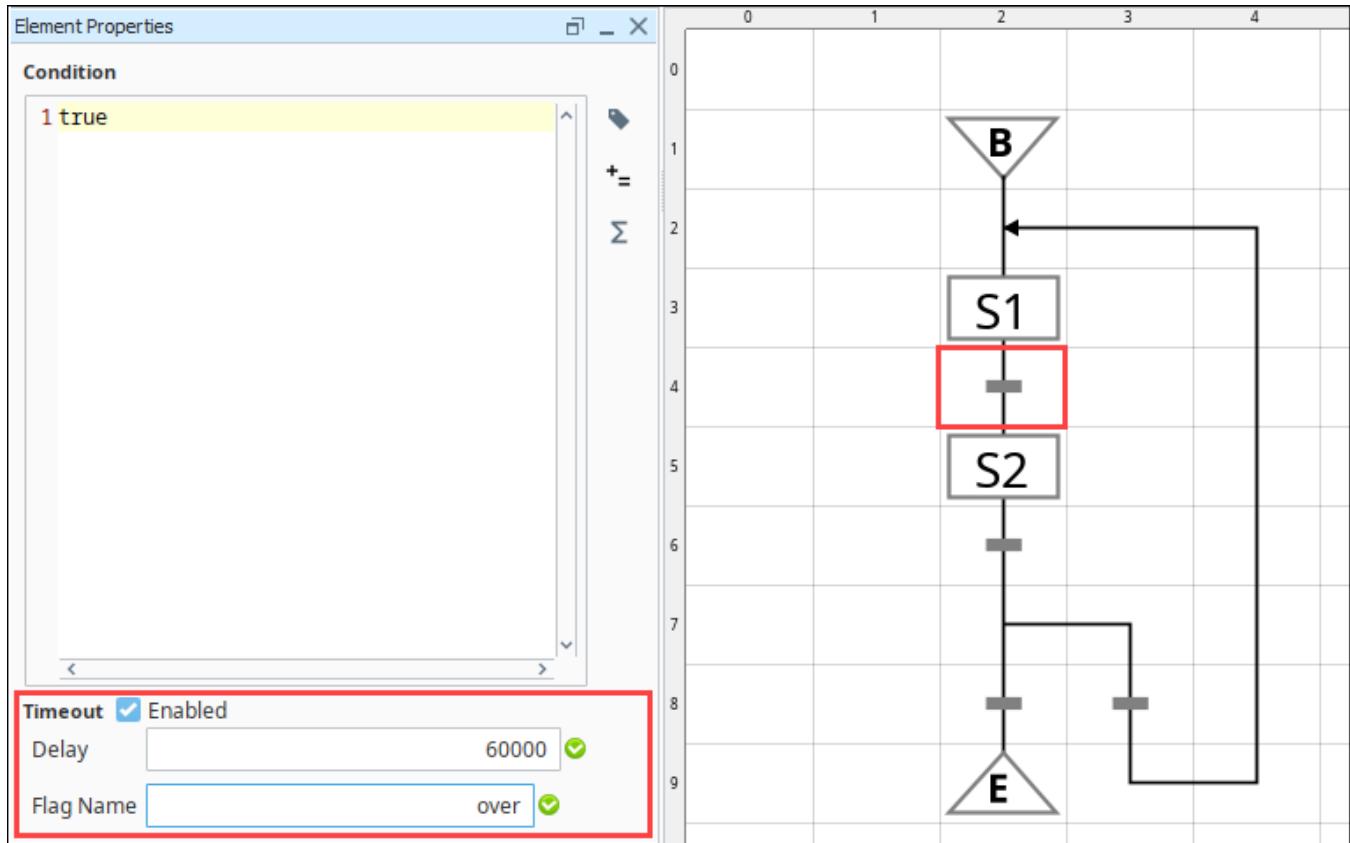
Transition

Transition elements control the flow of the chart.

A transition serves to either block or allow flow, depending on its value. All transitions have a value: true or false, which is determined by an [expression](#). Transitions occupy a one-cell region of the chart, and are represented by a short horizontal bar in the middle of the cell. Read more about how transitions control chart flow in the [Chart Flow and Rules](#) section.



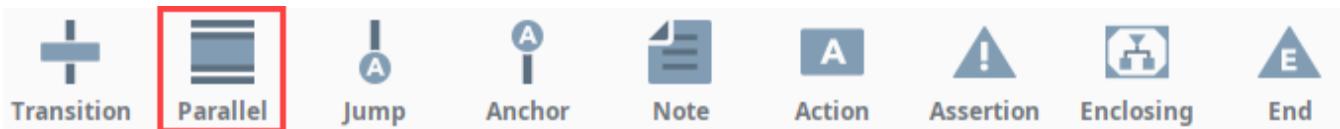
In addition to their expression, a transition can also specify a timeout. If enabled, the transition will set a flag after a certain amount of time has passed. This flag is a boolean variable set in chart scope, and can be used to make the expression or another expression close.



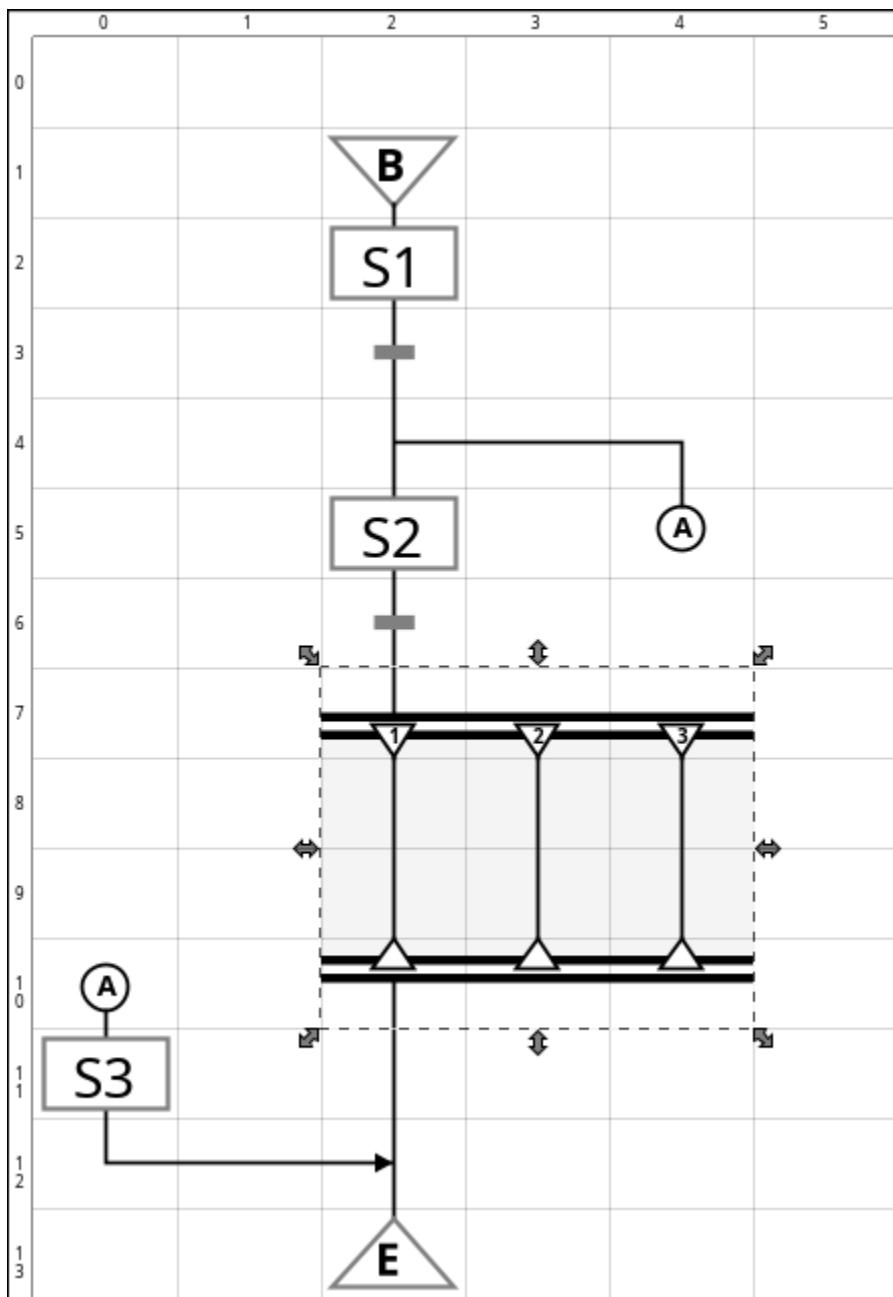
Parallel Element

A Parallel element contains other elements and executes the logic in parallel. This element is ideal in cases where multiple actions need to execute simultaneously, and you want the chart to pause until all of those actions have completed.

A parallel section is a rectangular section of the chart that can contain other chart elements inside it. The top and bottom of the parallel section is demarcated by two thick, parallel lines. Parallel sections allow for concurrent execution of multiple steps.



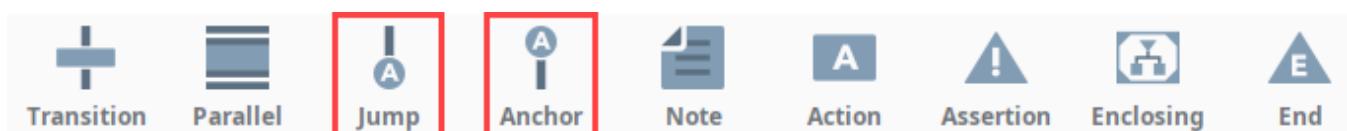
The top lines are called the "parallel branch" and the bottom are the "parallel sync". These sections are used to execute multiple branches of the chart at the same time.

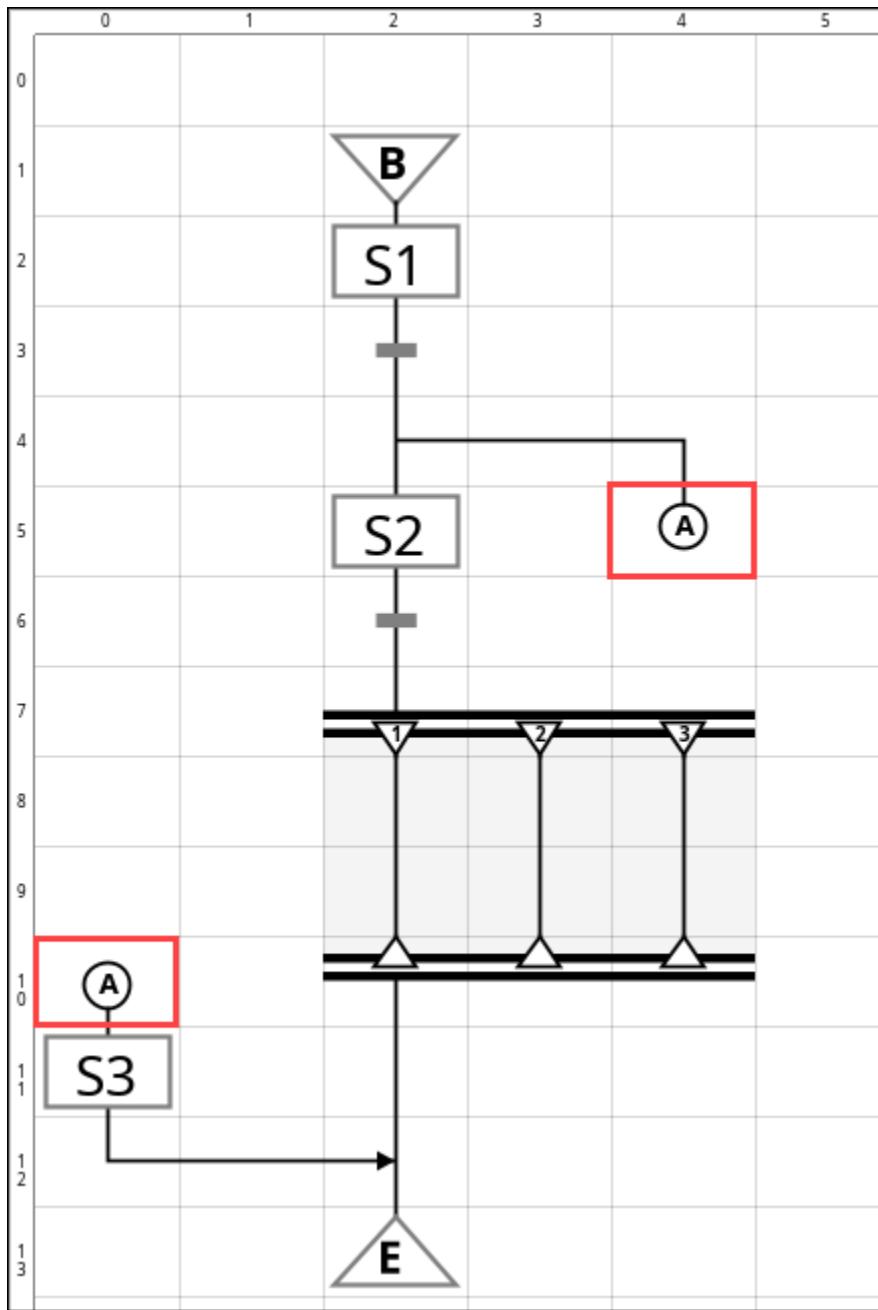


Jump and Anchor

A **Jump** element moves the logic of a chart from the jump to a matching anchor element.

A jump is an element that moves the flow to its matching anchor. This is a convenience for when a link would be unsightly or impossible due to crossing other links. Each jump and anchor element is identified by a single character; jumps identified by **X** will jump to the anchor also identified by **X**. There can be many jumps on a chart that all jump to the same anchor.

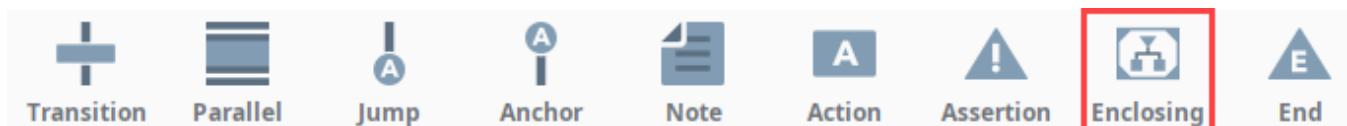




Enclosing Step

Run another chart inside this one, allowing complex tasks to be spread across several different charts.

The **Enclosing Step** references another SFC defined on the same Gateway. This is an important tool for SFC design, because it lets the chart designer create reusable blocks of logic encapsulated into charts, which can make chart design more modular.



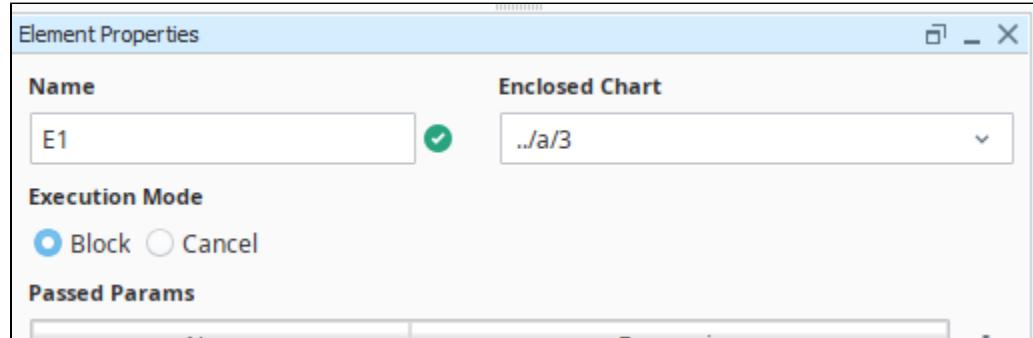
When talking about enclosing steps, the chart that the enclosing step references is called its **enclosed chart**, or **subchart**. The chart that the enclosing step is in is called the **parent chart**.

The following feature is new in Ignition version **8.1.2**
[Click here](#) to check out the other new features

As of release 8.1.2 you can include a relative path in the Enclosed Chart field.

```
# A '..' is used to reference a hierarchy object in the same folder  
./myChart  
  
# '...' is used to reference the parent folder of the current object  
.../myChart
```

In addition, the Enclosed Chart dropdown will now automatically use relative path notation.



When flow reaches an enclosing step, it starts its enclosed chart. Using the enclosing step's Execution Mode property, the step can be configured to work in one of two very different ways:

- **Execution Mode = Block**

Let the enclosed chart run to completion. This means that the enclosed chart should have an **End Step** in it, and that flow will not be able to move beyond the enclosing step until the enclosed chart stops by reaching its end step.

- **Execution Mode = Cancel**

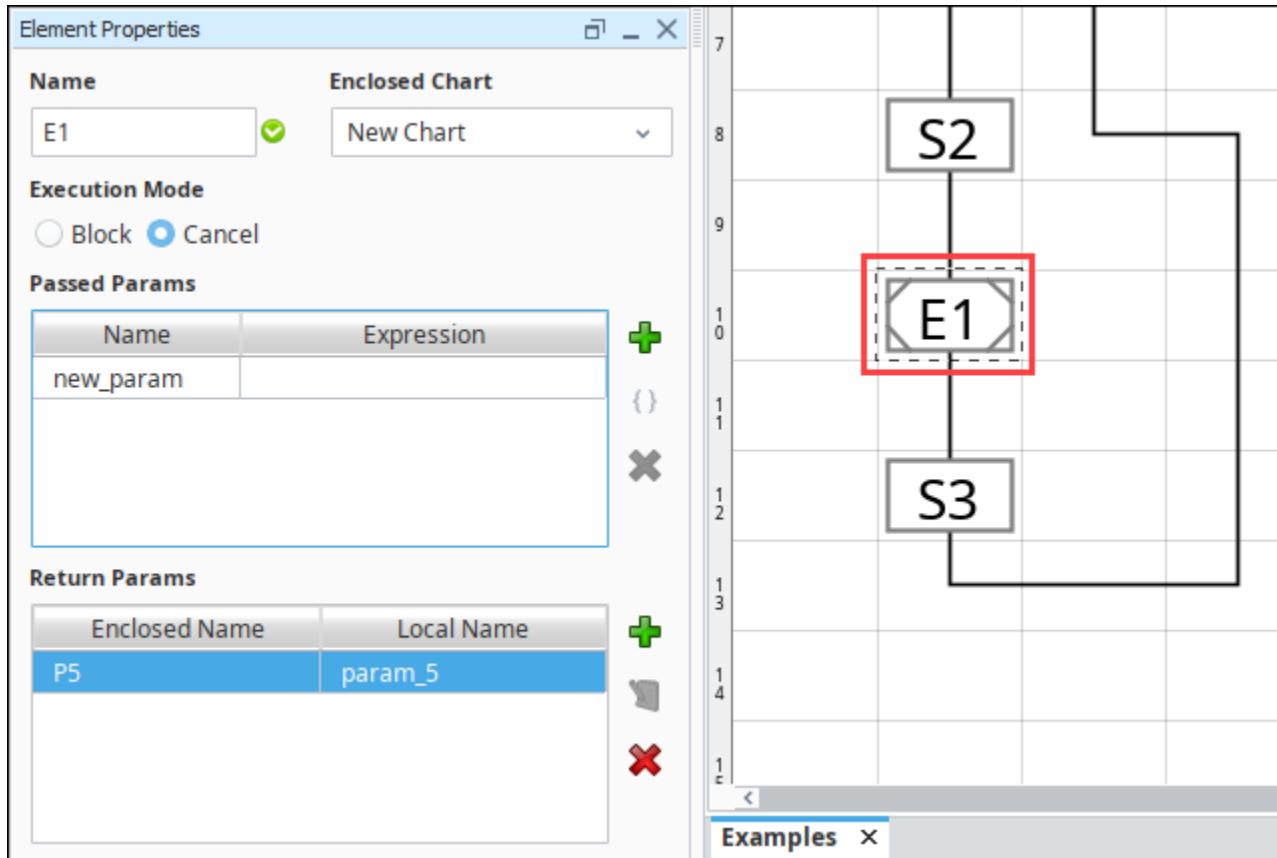
Cancel the subchart when the enclosing step is ready to stop. This means that the subchart is canceled when flow is ready to move beyond the enclosing step. Any running steps in the enclosed chart are told to stop, and flow ceases in the enclosed chart.

Parameter Passing

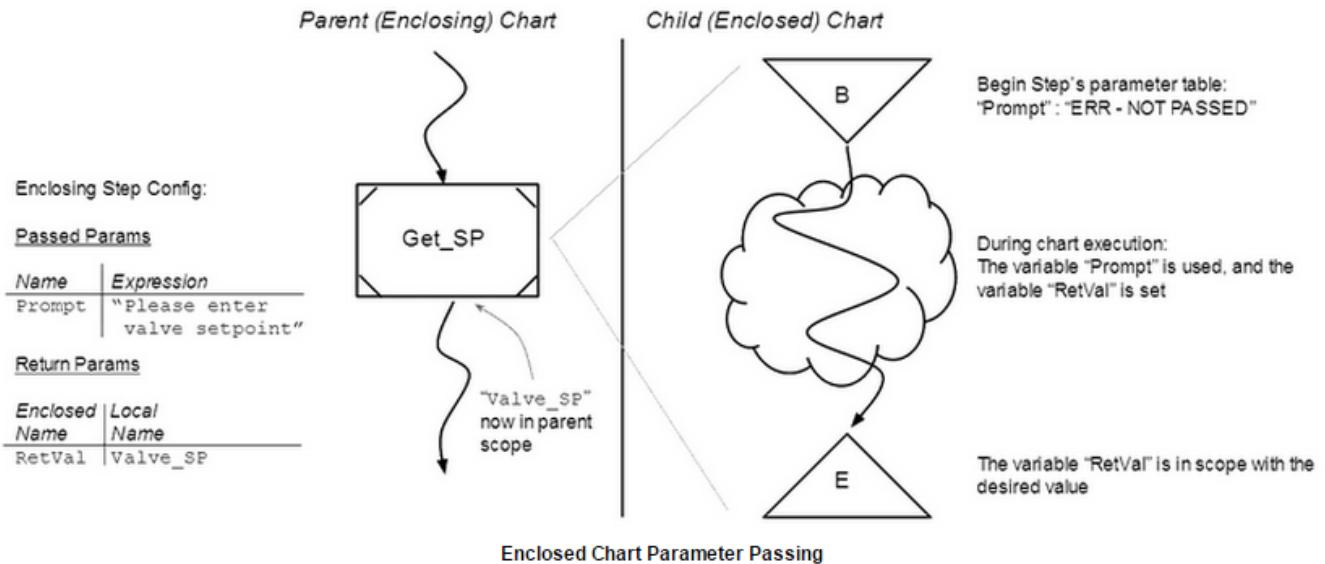
When invoking a subchart via an enclosing step, you have the opportunity to define how variables are passed and returned between the parent and child chart's scopes.

The enclosing step can define a list of parameters to be passed into the enclosed chart's scope. The values for the parameters will be expressions, thus they can be literal values or they can be references to variables in the enclosing chart's scope.

The enclosing step can also define a list of **return values** to receive from the enclosed chart. This is a mapping of variable names from the enclosed chart's scope to variable names in the parent chart's scope.

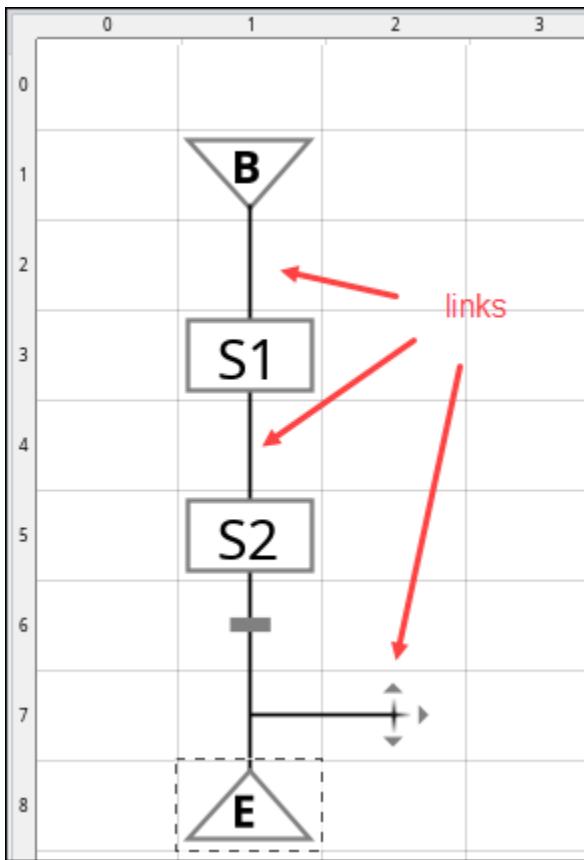


Return values can now be mapped to Chart Scoped Variables by using the same syntax that they would in the passed parameters. See [Chart Scope and Variables](#).



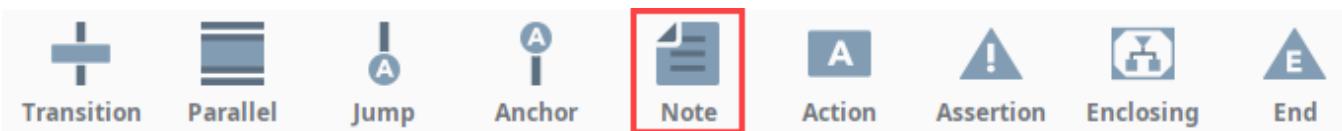
Links

A link is simply a line that connects other elements. Links are created in the Designer by dragging the arrows that appear next to unconnected elements. Links cannot cross above or below other elements or links. Links only travel in a single direction. This direction is determined by what the link is connecting to. Most elements such as steps and transitions only accept incoming links from above and outgoing links from below.

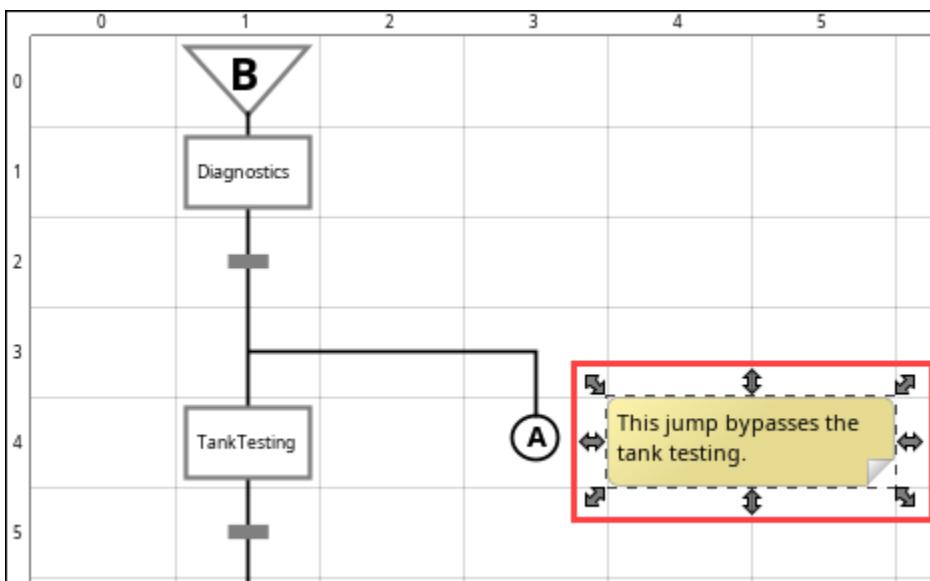


Notes

Notes are elements which have no import or function, but serve as documentation.



Note elements can be placed anywhere except that they can not overlap other elements. Notes are used to annotate chart logic but have no effect on the chart itself.



SFCs in Action

Chart Flow

All charts have the same basic flow to them. Some have loops, jumps, or enclosing steps that include whole other charts, but the flow is always the same:

- All charts start at their begin step. The begin step can define initial values for variables in the chart's scope. These initial values are defined as expressions.
- Flow always moves downward out of chart elements, except for links, which can move flow in any direction. When a transition splits into two or more, they are evaluated left-to-right.
- When flow hits a step, that step is started. The step continues to execute until the transition beneath it becomes true. If there is no transition beneath a step, the step starts and is told to stop as soon as possible. In practice, this means that an action step's `onStart` and `onStop` scripts will be run, but no timer scripts.
- When any End step is activated, the chart stops.

On this page ...

- [Chart Flow](#)
- [Starting a Chart](#)
- [Interaction and Monitoring](#)
- [Examples](#)
 - [Basic Transition](#)
 - [Branching Transition](#)
 - [Loop](#)
 - [Parallel Execution](#)
- [Interacting with a Client](#)

Starting a Chart

A chart can be started in one of four ways:

1. From Scripting

Using the `system.sfc.startChart` method of the scripting API, a chart can be started from anywhere. The chart must be in **Callable** execution mode.

2. From an Enclosing Step

A chart can spawn an instance of another chart using an [Enclosing Step](#).

3. Automatically

A chart whose execution mode is **RunAlways** is automatically started when the Gateway starts up. If the chart stops, the Gateway does not re-execute it. If you want a chart that runs all the time, it should be designed to never stop, for example, by looping back upon itself continuously.

4. From the Designer

While designing a chart, you can start instances of it from the **Chart Control** panel in the Designer using the **Start** link.

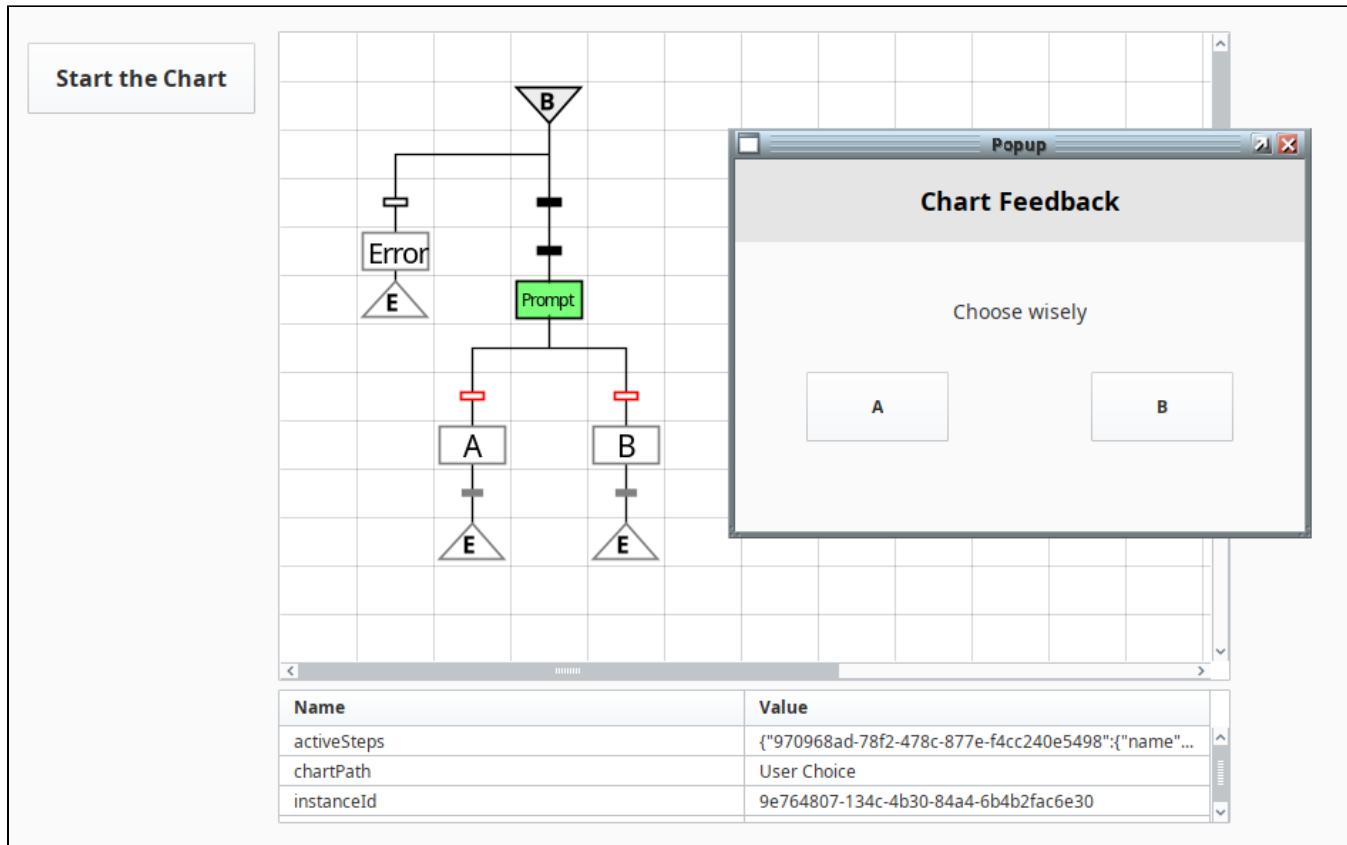
Interaction and Monitoring

While SFCs are run in the Gateway, Ignition has tools to help you interact with and monitor charts in the client. There is a [chart monitor](#) component that you can use to see the status of your SFCs in the client, there are scripting tools to [start, stop, pause, and resume charts](#) from the client, and you can send operator input to a chart with scripting functions.



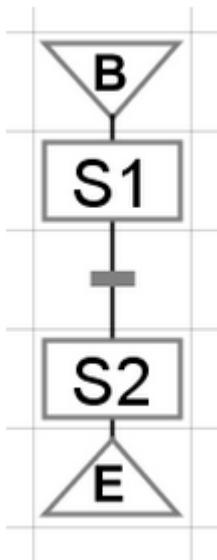
Starting a Chart

[Watch the Video](#)



Examples

Here are some examples of common paths or loops to get you started thinking about your process. You can combine these steps in any way, and create charts large or small.



Basic Transition

In this example, step S1 executes as soon as the chart starts, and continues executing until the transition beneath it becomes true. Once that transition becomes true, Step S1 is told to stop, which means it finishes executing any scripts that are currently running, and then it executes its onStop action (if any).

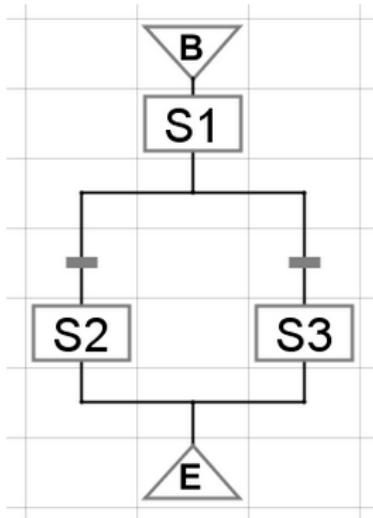
After S1 has stopped, step S2 starts. It is immediately told to stop, which means that if it has any timer actions, they will not run, but the start and stop actions will run.

After S2 is finished, the chart stops.

Branching Transition

In this example, step S1 executes as above, except that it has two transitions beneath it. This is how you do conditional logic in a chart. S1 runs until either of these transitions becomes true. When one transition becomes true, flow will follow that branch of the chart. If both transitions are true, the transition on the left is chosen. Position is meaningful for charts - transition precedence goes from left to right.

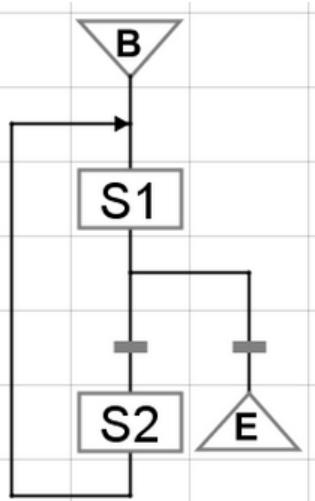
Only one of S2 or S3 will run, but never both.



Loop

In this example, S1 executes as above, looping until one of the transitions becomes true. If the branch to S2 becomes active, S2 runs once and then S1 starts looping again immediately. This way the chart can execute multiple times.

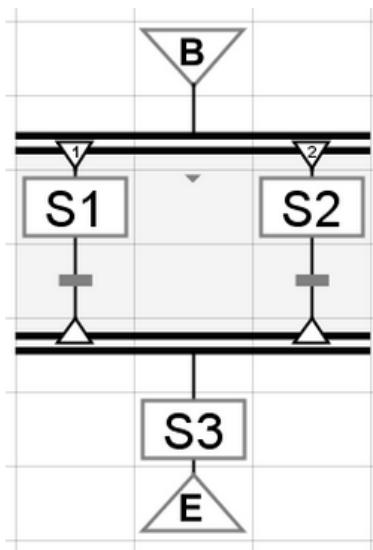
This is how you configure repeating logic in a chart. The two transitions determine whether this chart continues running (possibly indefinitely) or stops.



Parallel Execution

In this example, steps S1 and S2 execute simultaneously. They both continue to run until the transitions beneath them become true.

Flow only moves past the parallel sync (the bottom of the parallel section) once both transitions become true. Step S3 then runs, and then the chart stops.



Interacting with a Client

Chart instances are executed in the Gateway scope, which means they can't interact with a client in the typical way. Instead, they need to use message handlers to send information to the client. From a chart, we can use `system.util.sendMessage` to call a client message handler, which can then interact with the client in some way. This may range from altering something on the window to requesting user input.

The client can then call `system.sfc.setVariable` to write back to the chart if necessary, allowing the chart to continue if it was waiting for the input.



Interacting with a Client

[Watch the Video](#)

[In This Section ...](#)

Monitoring and Debugging Charts

While a chart is running, it can be monitored visually in the Designer or via a Vision Client.

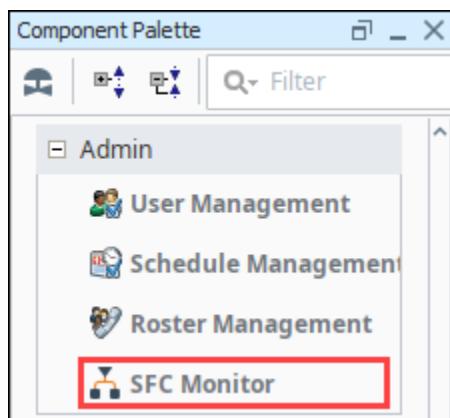
In the Designer

Open the chart you wish to monitor, and any running instances of that chart will appear in the list to the right of the design space with the heading **Chart Control**.

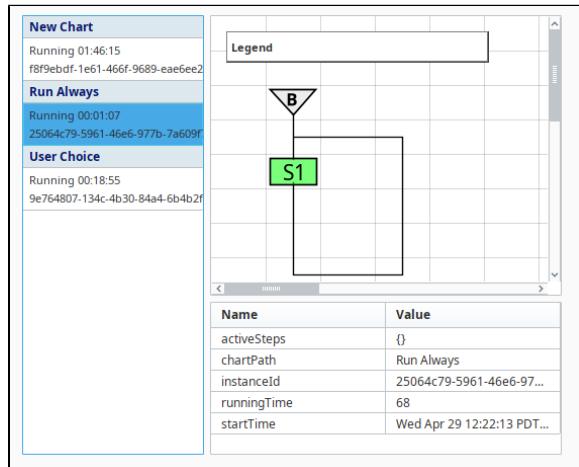
Double-click on an instance to enter monitoring mode. While in monitoring mode, you'll view the current state of the chart elements. There is a banner at the top of the Designer which will bring you back to design mode.

In a Vision Client

The SFC module adds a component to the Vision module under the **Admin** category called the **SFC Monitor**. Add this component to a window to be able to monitor SFC instances from your project.



This component can either display a pick-list on its left side to pick which instance to monitor, or you can give it the ID of a specific chart to monitor and hide the pick list.



Element Legend

On this page ...

- [In the Designer](#)
- [In a Vision Client](#)
- [Element Legend](#)



Monitoring Charts

[Watch the Video](#)

Steps	Transitions
	Unvisited
	ActiveFalse
	ActiveTrue
	InactiveFalse
	InactiveTrue

Related Topics ...

- [Pause, Resume, and Cancel](#)

Pause, Resume, and Cancel

This page details manual state changes that can be enacted on a chart after it has started. These state changes can be triggered via scripting (using some of the [system.sfc](#) functions listed on this page), or via the [SFC Monitor](#) component.

Security Considerations

Charts can run silently on the Gateway, but being able to interact with the chart from a Client (i.e., pausing a chart) can be useful. Since a chart can be executing some critical process, we highly recommend utilizing [security settings](#) to restrict access to components that are able to modify the state of a chart.

State Changes - Scripting

Pausing and Resuming

Any running chart can be paused by using the [system.sfc.pauseChart](#) function. When pausing a chart, any currently executing [action steps](#) must finish before the chart will transition to a paused state. As a result, it may take some time before the chart fully transitions into the Paused state.

Once the chart is paused, it can then later be resumed. This can be done using the **Resume** button in the chart control when testing the SFC Monitor Component, or using the [system.sfc.resumeChart](#) function.

Canceling

Canceling a chart works similarly to pausing a chart in that it must first wait for any currently running [action steps](#) to finish execution before the chart will cancel. A chart can be canceled from a script by using [system.sfc.cancelChart](#). Normal [rules for canceling](#) apply, thus the chart's **On Cancel** event will trigger, but not the **On Stop** event.

State Changes - SFC Monitor Component

Changing the state of a chart can easily be accomplished without scripting from the [SFC Monitor](#) component. From the Client you can right-click on a running chart, and click the state you wish the chart to transition to.

Note: The [Chart Lifecycle](#) is still in effect, so only charts in a Paused state may be resumed.

On this page ...

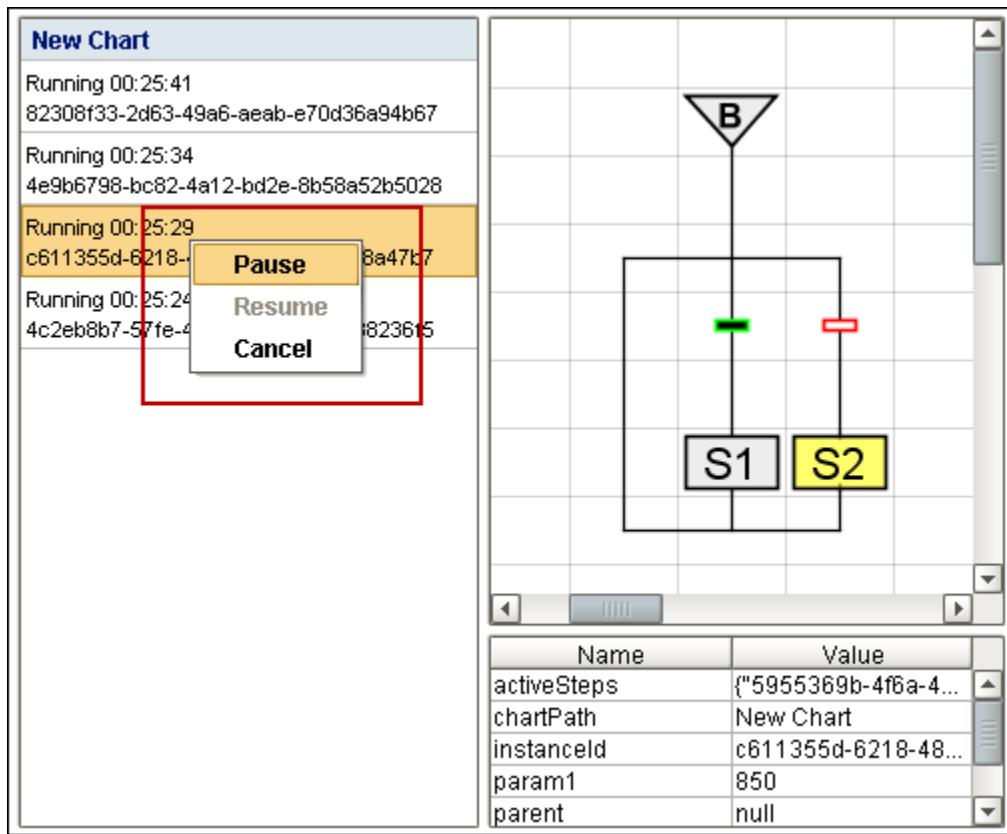
- [Security Considerations](#)
- [State Changes - Scripting](#)
- [State Changes - SFC Monitor Component](#)



INDUCTIVE
UNIVERSITY

Pause, Resume, and Cancel

[Watch the Video](#)



Related Topics ...

- [Action Step Best Practices](#)
- [SFC Monitor Component](#)

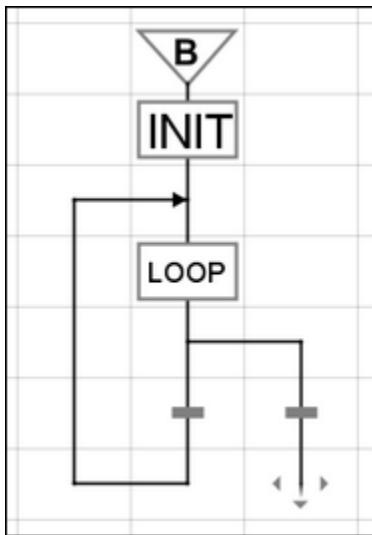
Action Step Best Practices

When designing SFCs, the Action step will be the primary workhorse of your charts. There are a few best practices to keep in mind while writing your scripts for the action step.

Use Transactions for Pausing or Waiting

This is the primary rule for SFCs. You want your scripts to run as quickly as possible. This means that you don't want to use any sort of sleep() or wait() call. Pausing or waiting is the job of transactions in an SFC.

Some sorts of blocking is, of course, unavoidable. For example, running a SQL query that takes some time to execute, or writing to a device over a slow radio connection.



INDUCTIVE
UNIVERSIT

Action Step Best Practices

[Watch the Video](#)

Use Refractor Loops

This is really just an extension of the don't block rule. Imagine that you have 100 widgets that need processing. Each widget takes some non-trivial amount of time, let's say, 20seconds, to process. The most obvious way to handle this would be with an **On Start** script that had a `while` loop from 1 to 100, processing each widget. This script would take about 33 minutes to run.

Instead of processing the 100 items in a `while` loop, you can solve this problem in two different ways with SFCs:

- **Timer Action** - In this option, you have a single action step, but instead of having your loop from 1 to 100 in a `while` loop, you initialize a counter to 1 in the **On Start** action. Then you write a timer action with a rate of 0ms. The timer action processes one widget, and then increments the counter. You place a transition beneath the step whose expression is: `{counter} >= 100`
- **Chart Loop** - Similar to option 1, you can design the loop in the chart itself. Have one action step do your initialization work, in our example: `chart.counter = 0`. Have another step do the processing work for one item in its **On Start** script. Use two transitions, the one on the left set to `{counter} < 100` and the one on the right set to `true`. The loop action will run 100 times, and then the flow will continue down the other path.

Keep Script Duration to a Minimum

By now you should understand that you want to keep your individual script duration to a minimum. You may be wondering why this is so important. After all, in the example above, it still takes 33 minutes to complete the given work after refactoring the loop as shown.

The reason this is important is to support pausing, persistence, and redundancy. Charts can only be paused after any running script finishes. Similarly, a chart's state only gets synchronized with the redundancy system between script executions. If you had a script that took half an hour to run, you couldn't pause that chart until the script ended, and the redundancy system would be very out of date and not able to recover as well if the Gateway went offline.

As a bonus, breaking your work up into smaller units helps make the chart easier to debug and more visible for monitoring.