**User Manual**

**ILS SFC Extensions**

**Document Version: 0.55**

**November 30, 2015**

Table of Contents

[1. Introduction 4](#_Toc436670047)

[2. Recipe Data 5](#_Toc436670048)

[2.1 Recipe Data Cross-Reference 5](#_Toc436670049)

[2.2 Last Value 5](#_Toc436670050)

[2.3 Unit Support 5](#_Toc436670051)

[2.3.1 Time Scaling 5](#_Toc436670052)

[2.3.2 Time versus Timestamp 5](#_Toc436670053)

[3. Debugging 7](#_Toc436670054)

[4. Isolation Mode 8](#_Toc436670055)

[5. Custom Steps 9](#_Toc436670056)

[5.1 Foundation Steps 9](#_Toc436670057)

[6. Scripting / Action Steps 10](#_Toc436670058)

[6.1 Types of Actions 10](#_Toc436670059)

[6.2 Python location 10](#_Toc436670060)

[6.3 Logging Support 11](#_Toc436670061)

[6.4 Exception Handling in Action Blocks 11](#_Toc436670062)

[6.5 Scripting API 11](#_Toc436670063)

[6.5.1 Recipe Data Access Examples 15](#_Toc436670064)

[7. Extensions to Transition Expressions 17](#_Toc436670065)

[8. Find/Replace 18](#_Toc436670066)

[9. Substitution Expressions in Messages 19](#_Toc436670067)

[10. Units 20](#_Toc436670068)

[11. Conversion From G2 to Ignition Design Patterns 21](#_Toc436670069)

[11.1 Conditional Transitions with a Callback 21](#_Toc436670070)

[11.2 Calculating time deltas 23](#_Toc436670071)

**Change History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Changes** |
| 0.1 | Feb 23, 2015 | R. Forbes (ILS) | Initial Notes |
| 0.2 | Mar 16, 2015 | R. Forbes (ILS) | Add Isolation Mode, changes for Foundation steps |
| 0.3 | May 1, 2015 | C. Coughlin (ILS) | Reference partial project with essentials |
| 0.4 | May 26, 2015 | R. Forbes (ILS) | Remove reference to recipeData prefix, add notes on scripting and units |
| 0.5 | Sep 21, 2015 | R. Forbes (ILS) | Add section on message substitution |
| 0.51 | Oct 1, 2015 | R. Forbes (ILS) | Add section on error handling in Action steps |
| 0.52 | Oct 24, 2015 | P. Hassler (ILS) | Added design pattern for converting a transition with a callback from G2 to Ignition. Added examples of accessing recipe data in action steps. |
| 0.53 | Nov 12, 2015 | R. Forbes (ILS) | Add section on transition expressions |
| 0.54 | Nov 25, 2015 | P. Hassler | Updated logger examples |
| 0.55 | Nov 30, 2015 | P.Hassler | Added note about recipe data support unit conversion and distinction between TIME and TIMESTAMP recipe data |
| .56 | Dec 29, 2015 | R. Forbes(ILS) | Add Architecture/Persistent Window section and add description of client tags to Isolation Mode section |

# Introduction

This document explains how to create and run SFC Charts in Ignition using the ILS custom step types. It assumes familiarity with Ignition development.

**Initial Configuration**

* The ILS-SFC module must be loaded
* Isolation Mode settings must be specified in the Ignition Designer via the View/External Interface Configuration. See Isolation Mode below.
* The SFC-specific tables must be present in the database. They can be create by running the “createSfcTables.sql” script in the deliverable.
* If units other than time units are desired, the desired units must be loaded into the database (see section on Units below).
* External Python modules must be in place. This is accomplished as part of a normal installation as described in Ignition Applications Installation Manual.
* UDTs for recipe data must be present. These should be imported from the file recipeDataUDTs.xml into the “production” tag provider’s Data Types folder.
* ILS Ignition project components must be present (e.g. message handlers, windows, etc). This can be most easily done by importing the partial project ils\_sfc\_essentials\_partial.proj that is supplied with the release distribution. The all-encompassing XOM.proj and SFC\_Demo.proj projects already include these resources.

# Recipe Data

Recipe Data are specified by using the Recipe Data editor pane. If you do not see this panel, check View/Panels in the Designer and see that Recipe Data is selected.

## Recipe Data Cross-Reference

The following table is a cross-reference between the old class of recipe data and the new class.

|  |  |
| --- | --- |
| G2 Recipe Data Class | Ignition Class/Type |
| EM-RECIPE-DATA | Recipe |
| S88-RECIPE-DATA-GROUP | Folder |
| S88-RECIPE-INPUT-DATA | Input |
| S88-RECIPE-MATRIX-DATA | Matrix |
| S88-RECIPE-OUTPUT-DATA | Output |
| S88-RECIPE-QUANTITY-ARRAY-DATA | List |
| S88-RECIPE-SEQUENCE-DATA | List |
| S88-RECIPE-SQC-DATA | SQC |
| S88-RECIPE-SQC-TXT | ? |
| S88-RECIPE-STRUCTURE-DATA | Structure |
| S88-RECIPE-TEXT-LIST-DATA | List |
| S88-RECIPE-VALUE-ARRAY-DATA | List |
| S88-RECIPE-VALUE-DATA (FLOAT) | Value |
| S88-RECIPE-VALUE-DATA (SEQUENCE) | List |
| S88-RECIPE-VALUE-DATA (TEXT) | Value |

## Last Value

Recipe data is implemented as tags, therefore the recipe data values are available for inspection as the SFC runs via the tag browser in any designer.

## Unit Support

Automatic unit conversion is implicit in recipe data and is based on the generic unit definition module. Most types of recipe data support the designation of a unit.

### Time Scaling

Recipe data that is of type “TIME” is automatically subject to time scaling when run in isolation. When configuring the isolation execution environment, one of the settings is a time acceleration factor. When setting or getting “TIME” recipe data this acceleration is automatically applied to scale the time.

### Time versus Timestamp

There is an import distinction between recipe data of type “TIME” and recipe data of type “TIMESTAMP”. “TIME” refers to a period of time and is subject to scaling when executing in isolation as mentioned above. “TIMESTAMP” refers to a moment in time, such as when the drier regen SFC started running. “TIMESTAMP” recipe data is not subject to time scaling.

# Debugging

The previous platform used a “debug” flag to enable debugging at run time. The flag could be set on individual blocks and procedures providing a high degree of granularity. Debug mode was either on or off, there was not the notion of “debug levels”. Messages were written to the message board.

The new platform does not allow users, either operators or engineers, to conveniently being able to conveniently set properties of a running chart in the gateway. However, the platform does provide a powerful logging capability based on log4j. The new platform will utilize log4j as follows:

* Use chart scope logging
* Every chart will have a logger, determined at run time, based on the call hierarchy, that can be turned on and off from the gateway at runtime.
* There is an API that action steps can use to get the appropriate logger:

Log=getChartLogger(chart)

Log.trace(“In myBlock: %f” % (F101.SP))

* Eliminate the debug property of blocks
* Replace “post” and “inform the operator” messages inside an “If Debug-Mode” block with log.trace().
* Replace “post” and “inform the operator” messages that are not inside an “If Debug-Mode” block with log.info().
* All custom blocks (not just callback blocks) will use this same system

# Isolation Mode

There are two modes to run an SFC in, Production and Isolation. This is specified by a Boolean client tag [Client]IsolationMode. A check box or radio buttons in the ILS UI used to run the chart may be supplied to set this tag.

These modes use different database and tag providers as specified in the View/External Interface Configuration dialog. This allows use of an isolated test mode that does not affect actual production equipment.

In order to support isolation mode in a way that is consistent with Ignition’s datasource binding functionality, there are two Client tags, Database and Provider, that are set automatically with a change script on the IsolationMode tag. The default datasource or provider should NEVER be used in client scripting; bind to those tags instead.

# Custom Steps

This section describes the custom steps.

## Foundation Steps

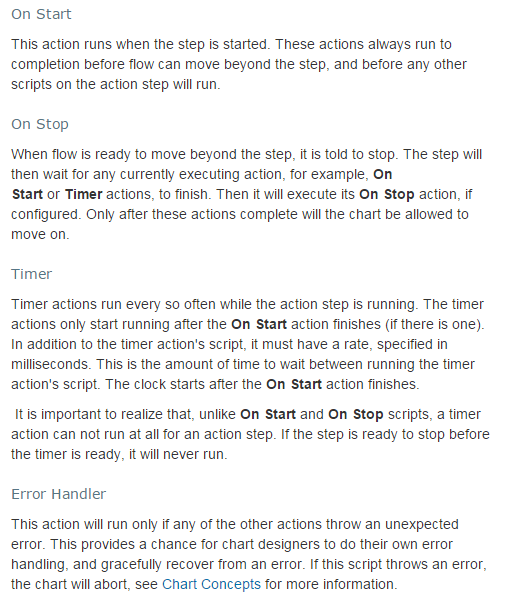
Unit Procedure (global), Operation, and Phase steps are created by dragging them from the Foundation menu.

# Scripting / Action Steps

Action steps are a standard feature of IA’s SFC module. Action steps perform actions specified by a Python script.

## Types of Actions

Action steps allow Python to run in four different ways as described in this excerpt from the IA user manual:

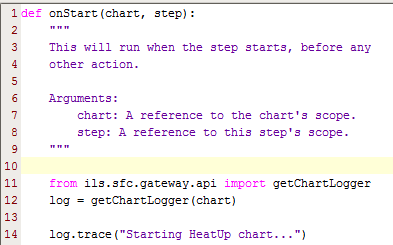


## Python location

The preferred location for the Python is to embed it directly in the step. Alternatively, the it can be located externally in the ils or xom scope or in the global or shared Ignition scope. Because SFCs are global, an action step cannot reference project scope Python. An example of calling global or shared scope is: “shared.test.getDelay”.

## Logging Support

It is important to consider logging, introduced above in section 3, when implementing an action step. The ILS SFC framework will automatically create a logger for each chart that can be set to info or trace mode (debug mode is also a level but is generally not used in favor of trace). There are four logger levels that are typically used: log.info, log.warn, log.error, and log.trace. The logger level is set in the gateway at run time from the gateway web page. An example of getting and writing to the logger is shown below.



## Exception Handling in Action Blocks

It is important to realize that if an error occurs in the Python of an Action Block, the default Ignition action is to abort the chart AT THAT LEVEL ONLY (i.e. an enclosing chart will continue to run)! The top-down cancellation and abort behavior in the ILS steps depends on calling particular scripts. This means that if the desired top-down behavior is to be maintained in Action blocks, a call to the proper error handling method must be called explicitly by either a) placing a try/except block around all the Python in the onStart and/or onStop methods or b) placing the call to the error handler in the Error Handler method of the Action step.

## Scripting API

The external Python module ils.sfc.gateway.api contains methods intended for use from ActionSteps. The Pydoc documentation for these is shown below. In most cases, the chart scope that is passed to the onStart method (e.g.) is also passed in to the api method. The naming is a bit inconsistent; chartScope and chartProperties are synonymous.

It is a recommended practice to use definitions from ils.sfc.common.constants wherever string constants are required (e.g. recipe data scope).

addControlPanelMessage(chartProperties, message, ackRequired)

display a message on the control panel

cancelChart(chartProperties)

cancel the entire chart hierarchy

convertToTagType(fullTagPath, value)

if necessary, convert a string value to match the tag type

convertUnits(chartProperties, value, fromUnitName, toUnitName)

Convert a value from one unit to another

getAssociatedUnitName(chartProperties, stepProperties, valuePath, location)

given a value key, get associated unit name, or None if there is none.

getChartLogger(chartScope)

Get the logger associated with this chart

getCurrentMessageQueue(chartProperties)

Get the currently used message queue

getDatabaseName(chartProperties)

Get the name of the database this chart is using, taking isolation mode into account

getIsolationMode(chartProperties)

Returns true if the chart is running in isolation mode

getProject(chartProperties)

Get the project associated with the client side of this SFC (not the global project!)

getProvider(chartProperties)

Like getProviderName(), but puts brackets around the provider name

getProviderName(chartProperties)

Get the name of the tag provider for this chart, taking isolation mode into account

getTagType(tagPath)

Get the value type of a tag; returns one of INT, FLOAT, BOOLEAN, STRING from ils.sfc.common.constants

getTimeFactor(chartProperties)

Get the factor by which all times should be multiplied (typically used to speed up tests)

getTopChartStartTime(chartProperties)

Get timespamp for chart start

getUnitByName(chartProperties, unitName)

Given the name of a unit, get the Unit object for it.

getUnitsPath(valuePath)

Get the key for the units associated with a recipe data value; None if not found

isValueKey(key)

return true if the given key references the "value" attribute

parseValue(strValue, tagType)

parse a value of the given type from a string

pauseChart(chartProperties)

pause the entire chart hierarchy

postToQueue(chartScope, status, message, queueKey=None)

Post a message to a queue from an SFC.

If the queueKey is left blank then the current default queue for the unit procedure is used.

Expected status are Info, Warning, or Error

readTag(chartScope, tagPath)

Read an ordinary tag (ie not recipe data), substituting provider

according to isolation mode setting

resumeChart(chartProperties)

resume the entire chart hierarchy

s88DataExists(chartProperties, stepProperties, valuePath, location)

Returns true if the the specified recipe data exists

s88Get(chartProperties, stepProperties, valuePath, location)

Get the given recipe data's value

s88GetFullTagPath(chartProperties, stepProperties, valuePath, location)

Get the full path to the recipe data tag, taking isolation mode into account

s88GetType(chartProperties, stepProperties, valuePath, location)

Get the underlying recipe data type; return one of STRING, INT, FLOAT, or BOOLEAN from ils.sfc.common.constants

s88GetWithUnits(chartProperties, stepProperties, valuePath, location, returnUnitsName)

Like s88Get, but adds a conversion to the given units

s88Set(chartProperties, stepProperties, valuePath, value, location)

Set the given recipe data's value

s88SetWithUnits(chartProperties, stepProperties, valuePath, value, location, valueUnitsName)

Like s88Set, but adds a conversion from the given units

scaleTimeForIsolationMode(chartProperties, value, unit)

If the supplied unit is a time unit and we are in isolation mode,

scale the value appropriately--otherwise, just return the value

sendMessageToClient(project, handler, payload, clientSessionId=None)

Send a message to the client(s) of this chart

sendOCAlert(chartProperties, stepProperties, post, topMessage, bottomMessage, buttonLabel, callback=None, callbackPayloadDictionary=None, timeoutEnabled=False, timeoutSeconds=0)

Send an OC alert

setCurrentMessageQueue(chartProperties, queue)

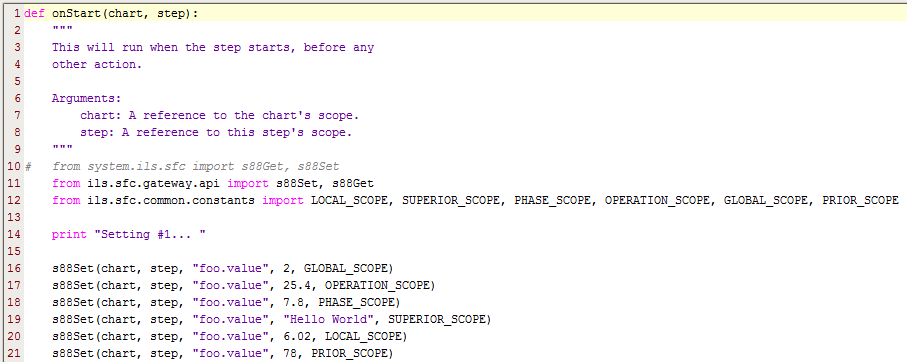
Set the currently used message queue

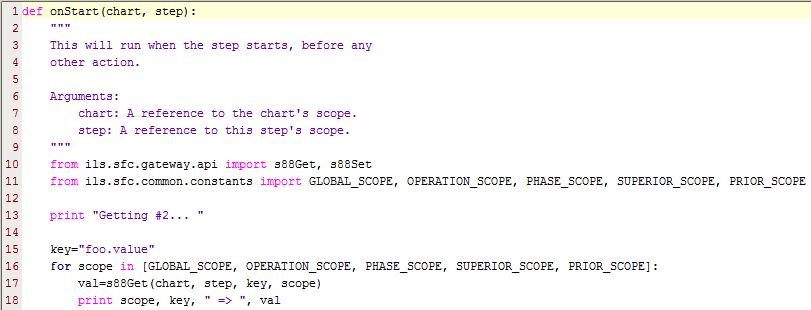
writeLoggerMessage(chartScope, block, unit, message)

Write a message to the system log file from an SFC.

### Recipe Data Access Examples

Typical examples of setting and getting recipe data are shown below





# Extensions to Transition Expressions

Recipe data may be accessed in transitions using the following syntax: {<scope>.key} where scope is one of the standard recipe data scopes. For instance, {global.mydata.value}. Although Ignition already supports incorporating tag values in expressions, that has no awareness of our ILS isolation mode. In order to support isolation mode settings, tag values may be accessed by the syntax: {tag.<tag path>}, e.g. {tag.mytag.value}. Do NOT include the tag provider in this expression; the provider will be determined by the isolation mode setting. This mode is currently limited to the “value” attribute of the tag, but could be extended to other attributes (e.g. AlarmActiveAckCount).

# Find/Replace

The standard Ignition find/replace capabilities have been extended to include the Sequential Function Charts. There are three elements of this feature: finding, replacing and locating.

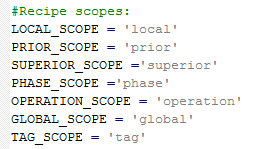
The search may be restricted to any or all of the categories: chart names, step names, step scope names, recipe keys, step scope values and recipe data values.

The current implementation does not allow editing via the find/replace mechanism. Any edits must be made through the normal SFC Designer interfaces.

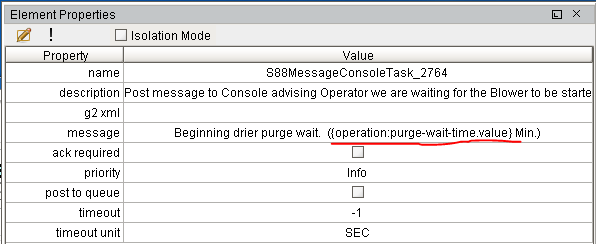
Double-clicking on the icon associated with a discovery will display the chart containing that element. For this display to appear, the SFC workspace must be already selected in the Designer.

# Substitution Expressions in Messages

In certain step types, an expression in curly braces in a message will be substituted with the value of run-time chart data. The following steps support this: Queue Insert, Control Panel Msg, Simple Query. The form of the expression is a scope and a key separated by a colon, e.g. “{local:rdata.value}”. Allowable scopes are tag, chart, step, and any recipe data scope (from ils.sfc.common.constants.py). These constants should also be used when used in a script; however when used in-line in a property of a step, the legal values are:



An example that substitutes the value attribute from the recipe data with key purge-wait-time form operation scope for a Control Panel step is:



# Units

Units are defined in the database. If the units table is empty, unit definitions may be loaded from a file in the XOM unit format. At time of writing, no standard UI for loading units exists, though the SFC Demo project does have a Unit Conversion window that can be used. Units can be loaded into the database with Python scripting, e.g.

import ils.common.units

file = system.gui.inputBox("Enter unit file", "")

if file != None:

newUnits = ils.common.units.parseUnitFile(file)

ils.common.units.Unit.addUnits(newUnits)

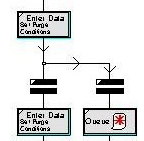
# Conversion From G2 to Ignition Design Patterns

This section discusses design patterns for specific configurations that cannot be directly translated via the automated translator.

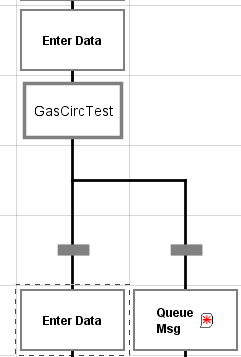
## Conditional Transitions with a Callback

Callbacks for conditional transitions is a feature of the old system which is not directly supported by SFCs. Transitions are one aspect of the Ignition SFC module which are not extendable.

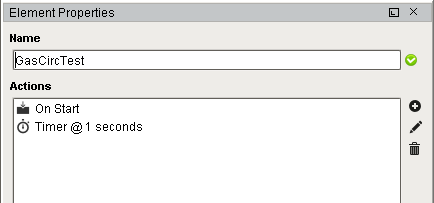
G2 Implementation:



Ignition Implementation:

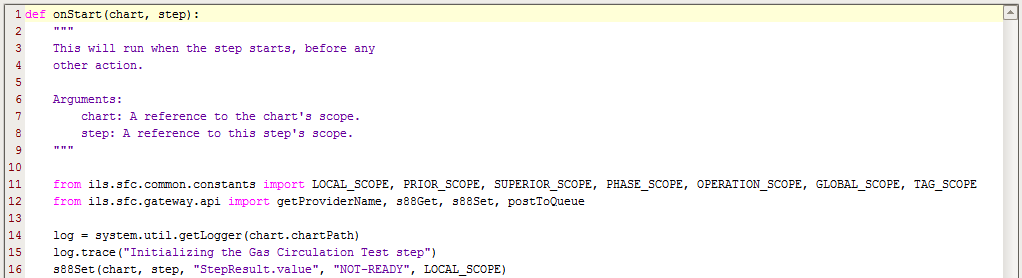


The callback procedure from the G2 implementation has been moved into the action step that precedes the pair of transitions. The action step has tws

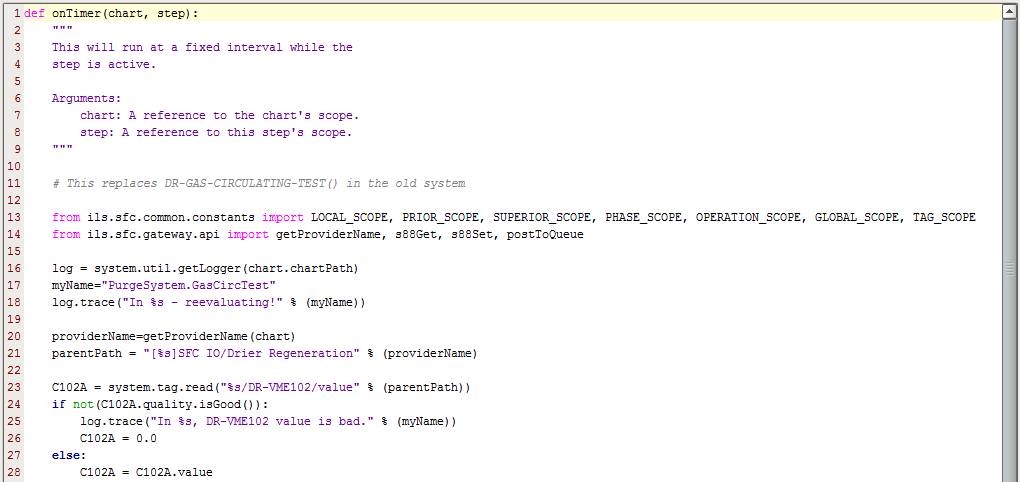


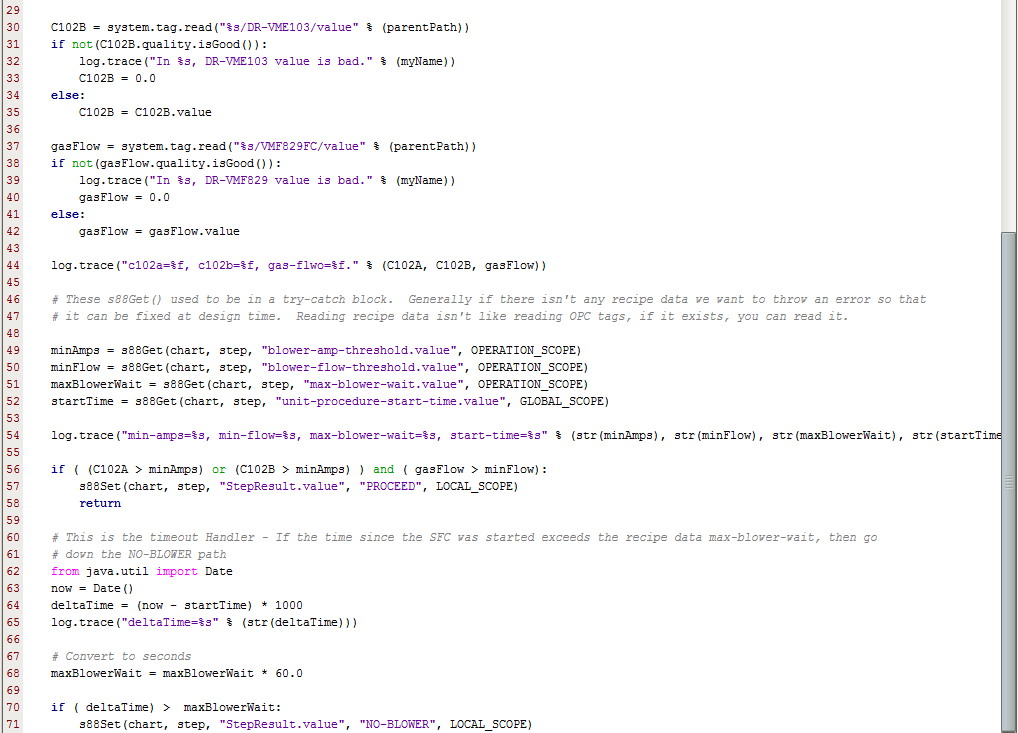
The On Start procedure performs initialization that will force both transitions to be false.

The timer script is called repeatedly, every second, until one of the transitions is true. The body of the transition callback is embodied in the timer script. Somewhere in the timer script, the recipe data or tags that the transitions monitor should be set. The body of the On Start is shown below:



The body of the timer script is shown below:





## Calculating time deltas

A common technique is to detect an event, such as a PV crossing a SP, recording the time, and then waiting a fixed period of time. The following code snippet demonstrates one way of implementing this.

The time that the event is occurs is recorded in recipe data as shown below:

from java.util import Date

from ils.sfc.gateway.api import s88Set

from ils.sfc.common.constants import GLOBAL\_SCOPE

def onStart(chart,block):

startTime=Date()

s88Set(chart,block,"startTime.value",startTime,GLOBAL\_SCOPE)

The start time can be comared to the current time as shown below:time that the event is occurs is recorded in recipe data as shown below:

from java.util import Date

from ils.sfc.gateway.api import s88Set

from ils.sfc.common.constants import GLOBAL\_SCOPE

def onStart(chart,block):

startTime=Date()

s88Set(chart,block,"startTime.value",startTime,GLOBAL\_SCOPE)

# Architecture

## Introduction

This section presents information intended for engineers that will be involved with enhancement of the software.

## Persistent Windows

A basic requirement of the application is to be able to recover the windows that were present if the client is closed either deliberately or as a result of a client failure. Also, if a control panel for a currently running chart is opened on a second client, all windows visible to the operator will be visible on this second client as well.

This functionality is supported by storing all information for currently open windows in the database. Each open window has a record in the SfcWindow table, as well as a record in one or more window-specific tables. For instance, each input window has one record in SfcInput, and each Select Input window has several rows in SfcInputChoices in addition.

When implementing a new persistent window, a particular pattern should be followed. Use the SelectInput Vision window, gateway python ils.sfc.gateway.steps.commonInput.py, and ddl for SfcInput and SfcInputChoices in svn/EMChemicals/Database/createSfcTables.sql as a template. The general sequence in gateway-scope step code is to 1) create window records 2) message clients that the window has been created 3) wait for a response from the client (if appropriate) 4) delete window records.

In gateway-side step scripting it is essential that the window records be removed even if the chart is canceled or an error occurs. Arrange exception and pause handlers so this is the case, and do not use “return” statements that will bypass the record removal.

Each window has custom properties of windowId (String) and data (dataset). The data property is bound to a sql query of that window’s table, e.g. "select \* FROM SfcInputs WHERE windowId = '{Root Container.windowId}'". All bindings using column values can be accessed from expressions referencing this dataset using the dataset access syntax. Such expressions should protect against a 0-length dataset. For example, the Yes/No window text property is bound to the expression “if(len({Root Container.data}) > 0, {Root Container.data}[0,'prompt'], '')”

Never use the default datasource for SQL bindings! As noted above under Isolation Mode, the datasource should always be bound to the [Client]Database tag.

If a window contains tabular data, this data will be held in a second database table. The table or combo box component can probably be bound directly to a SQL Query.

By default, SQL bindings have a polling interval. This is unnecessary overhead for most windows with static content, so the polling mode should be set to “Off” unless there is dynamic content (e.g. in Download GUI).

Avoid referencing the “data” custom property in visionWindowOpened scripts; it may not have been read from the database yet.

Timeouts should function correctly even if no client windows are open—this means that timeout logic should be written in the gateway step code. When scripting client-side messages associated with time delays or timeouts, do NOT use the time the window opened as a reference, as the window may have opened in the middle of an operation.

A step-specific default button label should be set in the gateway step code.