|  |
| --- |
|  |
| S88 Sequential Control Toolkit |
| Ignition Module Use Cases |
|  |
| **ILS Automation**  **Version 1.3** |
| **01/18/2013** |

|  |
| --- |
|  |

**TABLE OF CONTENTS**

1. Introduction 1

1.1. Background 1

1.2. Actors 1

1.3. Reference 2

1.4. Legend 2

2. Configuration 2

2.1. Gateway Configuration 2

3. Use Cases 2

3.1. Basic Module 2

3.1.1. Installation 2

3.2. Project Tree 3

3.3. Palette 4

3.4. Basic Control Block 5

3.4.1. Callback Block 5

3.5. Basic Execution Behavior 6

3.5.1. Conditional Transition Task 7

3.5.2. Custom GUI 8

3.5.3. Parallel Task Execution 11

3.5.4. Encapsulation 13

3.6. Execution Engine 13

3.7. Error Handling 16

4. Traceability Matrix 17

# Introduction

This document contains use case descriptions of the S88 Sequential Control Toolkit (SCTk) from ILS Automation. The toolkit is an module written in Java that runs on the Ignition™ platform from Inductive Automation. The use cases document behavior from the user’s point of view. They also used as testable requirements.

SCTk is a module based on the Ignition™ platform from Inductive Automation. It is written in Java. Automated tests are written in Python.

## Background

The SCTk is a graphical toolkit designed for implementation of sophisticated control and optimization strategies. SCT applications consist of diagrams containing control blocks, each block executing in turm to afect a part of the overall recipe.

An important feature of the SCTk toolkit is the concept of encapsulation where a higher-level block is itself described by an embedded block diagram.

A S88 Sequential Control recipe consists of blocks. The root block of a recipe is a Unit Procedure block. A unit procedure consists of a number of operations, only one of which may execute at a time. An operation encapsulates phases that in turn encapsulate tasks. Phases are defined by encapsulated tasks that perform specific functions. The block hierarchy is described in more detail in section 2.1 of the S88 User Manual. A recipe will consist of a unit procedure and tasks where the tasks are placed on the subworkspace of the unit procedure.

## Actors

The actors that are involved with the proof-of-concept are defined as follows:

Operator the operator is the end user of the system. In Ignition terms, the operator uses the “Client” scope. The operator uses recipes. They interact with recipes via a control panel and will interact with various screens that are posted per blocks in the recipe. The operator does not typically watch the recipe schematic as it is executing.

Engineer the engineer is a domain specialist who specifies application requirements and who is likely to be intimately involved in testing and configuring the application. In Ignition terms, the engineer uses the “Designer” scope. The engineer designs and implements the recipe. The engineer must be able to test individual blocks in the “Designer” scope by selecting S88 commands (Start, Abort, etc) from a popup menu on any block.

Developer the developer is responsible for the module code.

## Reference

The applicable standard is:

[ANSI/ISA-88.00.02-2001 Batch Control Part 2: Data structures and guidelines for languages](http://www.isa.org/Template.cfm?Section=Find_Standards&template=/Ecommerce/ProductDisplay.cfm&ProductID=2623)

## Legend

Sections that are highlighted in red are not implemented in the current release

# Configuration

This section describes any special configuration to the server, Ignition Gateway, or ancillary systems

## Gateway Configuration

The SCT module requires the following modules:

* Vision - from Inductive Automation
* ILS-Core - from ILS automation.

# Use Cases

This section describes the uses cases that constitute requirements for the SCTk Ignition module.

## Basic Module

Core features of S88 applications are supported by a custom Ignition module. This module must be loaded as part of any Sequential Control Toolkit application.

The module that supports Sequential Control and any dependent modules shall be available to an Engineer. The module itself is named Sequential Control Toolkit.

### Installation

The module may be loaded into any Ignition installation given proper licensing.

**Requirement**

An Engineer shall be able to load the SCT Module into a newly installed Ignition system.

**Steps to Reproduce**

1. Obtain signed Vision, SCT and ILS-Core module files. (The standard Ignition distribution includes the Vision module).
2. Load the ILS-Core and Vision modules into the Ignition gateway.
3. Load the SCT module into the Ignition gateway.

**Expected Result**

The SCT module should be visible in the gateway on the installed modules page. It should have a “Running” state. If the Vision module is not loaded, the SCT module will show a "Faulted" state.

## Project Tree

Once the SCT module has been loaded, a workspace node appears in the project tree in the designer scope. Workspaces are the windows/canvases upon which S88 diagrams can be constructed.

This functionality is supported from the ILS Core modiule.

**Requirement**

In Designer scope, the project browser will show a “SCT Workspace” container. This container shall support behavior similar to the Windows container supplied by IA.

1. A right-click on the SCT container will launch a popup menu with the following options:
   * New Workspace
   * Rename
2. Once a workspace has been created, the navigation tree will display a +/- box to expand or contract the tree.
3. A right-click on a workspace window launches a popup-menu with the following options:
   * Rename
   * Delete
4. Once a sub-workspace has been created, the navigation tree will display a +/- box to expand or contract the workspace tree. It is not possible to create a sub-workspace from the project tree.
5. Sub-workspaces are shown in a separate tab of the same window that shows the root workspace.

These features are provided for an Engineer.

**Steps to Reproduce**

Load the SCT and dependent modules. Open a Designer project. Create a new workspace from the project tree.

**Expected Result**

View the new workspace root node in the project tree. Observe that mouse actions perform as described above.

## Palette

When the SCT module is loaded, a new palette appears in Designer projects. The palette contains blocks from which S88 diagrams can be constructed.

**Requirement**

Once the SCT and ILS-Core modules have been loaded into the Gateway in the Designer scope, a new S88 tab shall appear in the palette area. The palette shall support behavior similar other palettes supplied by IA.

1. The palette contains icons for the following execution block types:
   1. Begin
   2. Callback
   3. Delay
   4. Encapsulation
   5. End
   6. Join
   7. Parallel
2. A drag-and-drop action may be used drag an icon from the palette and create a block instance on the workspace.

This feature is provided for an Engineer.

**Steps to Reproduce**

1. Load the SCT and dependent modules. Open a Designer project. Select the S88 palette.
2. Create a new workspace for the diagram/recipe.
3. Drag an icon of each type onto the new diagram.

**Expected Result**

Custom blocks of each type should be viewable on the workspace. It should be possible to change the position of each block by dragging. The block positions should persist when the window is opened and closed, when the project is saved and restored.

## Basic Control Block

The ILS-Core module contains an abstract base class from which all S88 blocks are derived. The behaviors encoded in this class are available to and can be tested on every control block, no matter what its type.

**Requirement**

Each control block will exhibit the following behaviors:

1. When the block is selected, the following properties are viewable and editable in the properties edit frame of the designer:
   1. Name – the name entered must be alphanumeric and unique
2. When the block is selected, the following properties are viewable and NOT editable in the properties edit frame of the designer:
   1. State – the S88 execution state of the block
3. Excluding previously mentioned exceptions, the block components shall behave just as other Ignition components dropped from palette to workspace.

**Steps to Reproduce**

Drag and drop a sample of each block type (Begin, Callback, Delay, etc) onto an ILS workspace. For each block test the features required above.

**Expected Result**

Observe the correct behavior for each of the requirements tested above. Verify that any deviations from the generic behavior are described in the use case section for that block type.

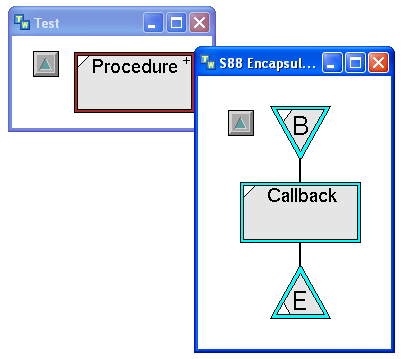
### Callback Block

One of the most powerful aspects of the SCTk is the callback block which executes a user defined Python script.

**Requirement**

Create a callback class which has an attribute that specifies the name of a Python script that will be called when the block executes. The script must not be embedded in the block so that the same script could be called from multiple blocks.

If there were a graphical component it might look like:



**Steps to Reproduce**

Write a Python script that may print a message to the log file or write a random value to some tag.

Execute the unit procedure.

**Expected Result**

Inspect the log file or tag and observe the effect of executing the block.

**Key Concept to be Implemented by Inductive Automation**

Capability of calling a Python script which will be named by an attribute of the block and will be stored in the Script Module library from the Java engine.

## Basic Execution Behavior

Control blocks dragged from the palette onto a workspace form a recipe. Block attributes (in the future connections) are used to determine a block order.

**Requirement**

Create and execute a simple recipe that utilizes at least one instance of each of the original block types (Callback, Delay, End, Join. Parallel, Start).

**Steps to Reproduce**

From the S88 palette, drag at least one block of every type listed above onto a new workspace. Edit attribtutes to determine control flow.

**Expected Result**

The recipe will execute.

### Conditional Transition Task

Another important capability of the SCTk is the conditional transition block. This block is analogous to an IF statement in a traditional programming language. The conditional transition can be used to choose a path of execution among many paths.

**Requirement**

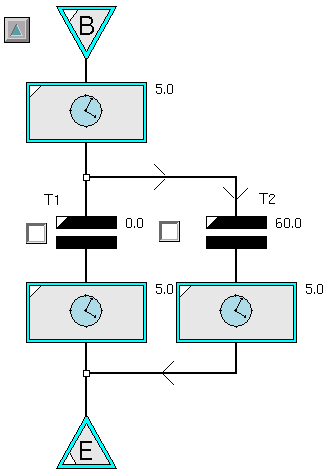
Create a conditional transition class which has as an attribute an embedded Boolean expression tag, or a similar equivalent. The engineer can specify any legal Boolean expression that gets evaluated when the block is evaluated. If true, then execution passes to the block connected to the output. For the purpose of the proof-of-concept, the conditional transition is only evaluated once.

If there were a graphical component it might look like:



**Steps to Reproduce**

Create the following schematic:



Execute the unit procedure.

**Expected Result**

Inspect the log file or tag and observe the effect of executing the block.

**Key Concept to be Implemented by Inductive Automation**

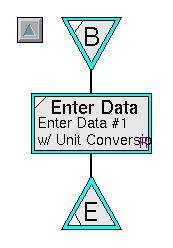
None, ILS will implement this functionality.

### Custom GUI

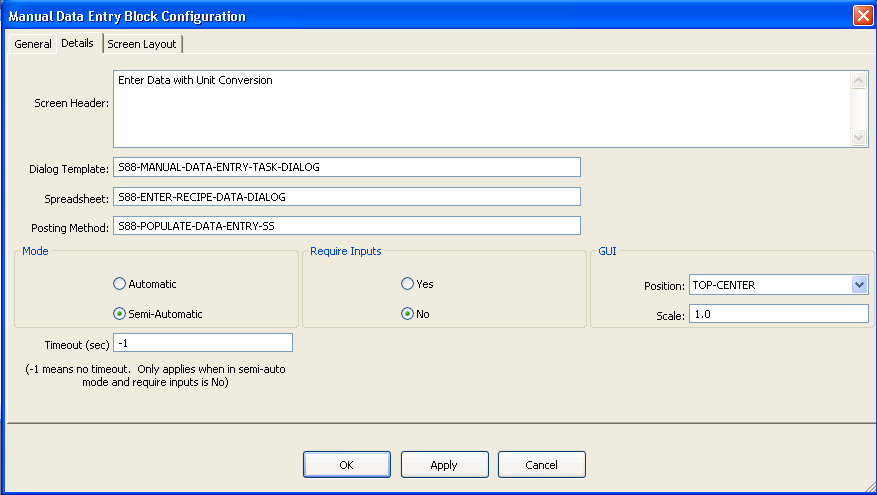
The operator needs to interact with the recipe via a variety of GUI blocks. The GUI blocks allow the operator to enter or review things such as run rates, setpoints, alarm limits, etc. There are a number of standard GUI blocks and GUI screens that will be part of the toolkit, but the user shall also be able to design their own Vision window and specify it in an attribute of the block.

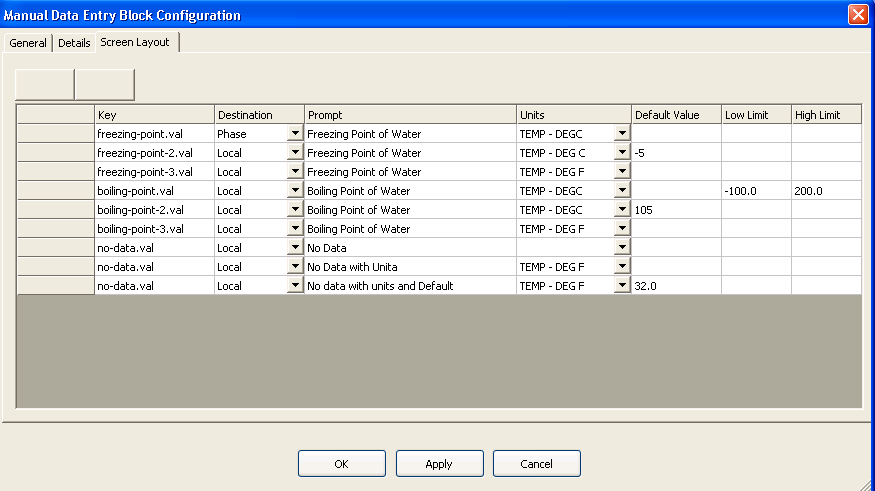
**Requirement**

If there were a graphical component it might look like:



Most of the GUI blocks have a configuration screen for specifying what will be displayed to the operator when the block executes. A typical screen is shown below:





Define a custom GUI block task, a UI for the designer scope for configuring the block, and a Vision window that will be displayed in the client scope when the block executes and uses the configuration in the block.

**Steps to Reproduce**

Construct a diagram with the block shown above and run the recipe.

**Expected Result**

A Vision window is displayed on the client window executing the block.

**Key Concept to be Implemented by Inductive Automation**

Ability for the Java engine to post a vision window, specified by the GUI task, on every client window logged on as a certain mode.

### Parallel Task Execution

Another powerful aspect of the SCTk is the Parallel Transition block. This block is used to start the parallel execution of blocks and also to synchronize the completion of parallel execution of blocks. Refer to section 7.1 of the S88 User Manual for a more complete example.

**Requirement**

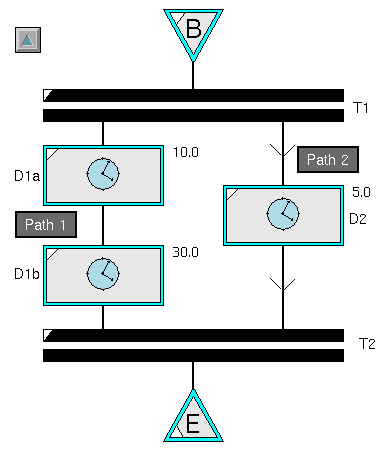
Create a parallel transition class which can have any number of connections coming into the block and any number of connections coming out of the block. For simplification for the proof-of-concept, a list of input blocks and a list of output blocks is sufficient. The evaluation method for a parallel transition block will run until all of the blocks in the input list are complete. Once the input blocks are complete, then all of the blocks in the output block list will start simultaneously. Once all of the output blocks are started, the block is complete.

If there were a graphical component it might look like:



**Steps to Reproduce**

Construct a diagram as shown below



Execute the unit procedure.

**Expected Result**

Inspect the log file or tag and observe the effect of executing the block.

Begin Complete

T1 begin

D1a begin

D2 begin

T1 complete

D2 Complete

T2 begin

D1a complete

D1b begin

D1b complete

T2 complete

End Complete

**Key Concept to be Implemented by Inductive Automation**

Demonstrate how to start multiple threads and to synchronize the completion of the threads.

### Encapsulation

Every programing language supports the notion of encapsulation. Encapsulation is used to hide implementation details and to make a large recipe easier to maintain.

**Requirement**

Implement an encapsulation task which encapsulates a set of tasks. The encapsulated tasks must contain a single begin task and a single end task. Encapsulation tasks can be nested to an infinite depth.

Modify the recipe from section 3.1 by adding an encapsulation task after the delay task. The encapsulation task should contain a Begin task, a 10 second delay task, and an End task.

## Execution Engine

The Sequential Control Toolkit consists of an “engine” that executes the blocks. The engine responds to a well-defined set of commands and a corresponding set of states. The full set of states and commands are described in sections 2.4.1 and 2.4.2 of the S88 User Manual. The blocks that make up the S-88 “language” are inherently object-oriented. Every block should implement an *s88-eval()* method. The engine should support various levels of logging including a level that logs the start and completion of a blocks execution. Marker blocks such as the begin and end block should log a single entry.

**Requirement**

Implement the following commands:

|  |  |
| --- | --- |
| Command | Description |
| Start | Can be sent to a block that is idle to start the blocks evaluation method |
| Abort | Can be sent to a block that is running to abort the current evaluation immediately |
| Reset | Can be sent to a block that is complete |

Implement the following states:

| State | Description |
| --- | --- |
| Running | The state of a block while it is currently executing. A block that encapsulates other blocks is “running” until all encapsulated blocks are no longer running. |
| Complete | The state of a block that completed successfully. |
| Idle | The state of a block that has been reset and is ready to run. |

**Steps to Reproduce**

Issue a start command to the recipe,

**Expected Result**

A new thread should run as long as the recipe is running. The thread should be visible somewhere.

The log file should show the following execution flow.

Unit Procedure begin

Begin Complete

Delay begin

Delay end

End Complete

Unit Procedure end

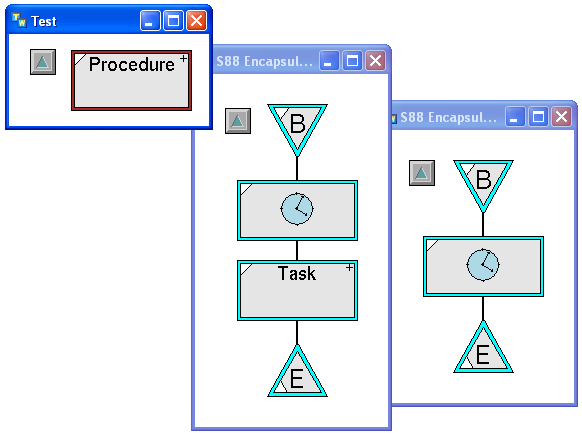
**Key Concept to be Implemented by Inductive Automation**

Discuss the pros and cons of implementing a block’s evaluation logic in Python vs Java.

Should we use the “Custom Palette” feature to define all of our blocks, or just use that feature for customers to add custom blocks?

If we were to implement it in Python, could we add a new event type, i.e., execute, and then the user would write a custom handler for that event, and the Java engine would execute the block by sending an execute signal to the block.

If there were a graphical component the recipe might look like this:



**Steps to Reproduce**

Turn on logging and run the recipe.

**Expected Result**

The log file should be

Unit Procedure begin

Begin Complete

Delay begin

Delay end

Encapsulation begin

Begin Complete

Delay Begin

Delay End

End Complete

Encapsulation end

End Complete

Unit Procedure end

**Key Concept to be Implemented by Inductive Automation**

Implement encapsulation.

## Error Handling

Most modern programing language supports the notion of error handling where an error handler can be defined at the appropriate scope where it makes the most sense to handle the error. The notion of error handling in the SCTk has been expanded to handle timeouts, stop commands, and abort commands. An example of the usefulness of an abort handler would be to consider a chemical plant running a startup operation. The operation may contain hundreds of blocks with many layers of encapsulation. As the operator monitors the startup, he may determine that the startup needs to be stopped immediately. Based on how far the startup has progressed would determine special actions that should be taken to abort the startup. Refer to section 2.7.6 for a full discussion of SCTk handlers.

**Requirement**

Handlers are optionally places on the subworkspace of an end block. If there were a graphical component it might look like:



Define three additional classes, Stop Handler Begin, Abort Handler Begin, and Timeout Handler Begin.

The path defined by the regular Begin block will be executed if the superior schematic completes normally. The Stop handler will be executed if the stop command is issued while any block on the schematic is running. The Abort handler will be executed if the abort command is issued while any block on the schematic is running. Finally, the Time-Out handler will execute if the end block does not execute and there are no blocks running on the diagram. (This can happen if a set of conditional transitions do not cover the complete problem space).

**Steps to Reproduce**

Construct a diagram with the handlers shown above. Issue Stop and Abort commands and observe which blocks ran. Also construct a diagram that will result in a timeout and execute the diagram.

**Expected Result**

Inspect the log file or tag and observe the effect of executing the block.

**Key Concept to be Implemented by Inductive Automation**

Ability for the Java engine to catch an error thrown by a blocks evaluation method, including an error thrown by the Python script called by a callback block, and to then start the correct handler at the current scope or the nearest calling scope.

# Traceability Matrix

Each of the above requirements must be traceable to either a test case or completed/validated development task. ILS-Automation has selected TargetProcess for project tracking (see <http://ils.tpondemand.com>). The task and test numbers in the chart below reference entities in TargetProcess.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| UseCase | Description | Task | Test | Date |
| 3.1.1 | Installation |  | Manual | 2013-10-10 |
| 3.2 | Project Tree | 237 |  |  |
| 3.3 | Palette | 249,265 |  | 2013-10-10 |
| 3.4 | Basic Block | 272 |  | 2013-10-10 |

Table 4.1 - Requirements Traceability