**Ignition 8 Site Migration Guide**

**Version 1.0**

**March 9, 2023**

**Prepared by:**

**ILS Automation Inc.**

**Table Of Contents**

1 Introduction 3

2 Database 3

2.1 Overview 3

2.2 Recipe Download 3

2.2.1 Timeout 3

2.2.2 Feedback 3

2.2.2.1 Success 3

3 Symbolic Ai 4

3.1 Creating and Deleting Tags 4

3.1.1 OPC Tags 4

3.1.2 Local Memory Tags 4

3.1.3 Recipe Detail User Defined Type 4

4 Sequential Control 5

4.1 Foo 5

5 Python 6

5.1 Download Trigger 6

5.2 Recipe Details 6

# Introduction

The purpose of this document is to provide a roadmap for migrating a site project using Ignition 7.x and Artie 2.2r4 to Ignition 8 and Artie 3.x

Some of the Baton Rouge sites are using a .

# Database

This section provides a description of the operator console and user interface.

## Overview

The general steps to using the recipe system are as follows.

The screens involved in this workflow are shown below. The sequence begins by pressing the “Recipe” button on the operator console.

## Recipe Download

The following sections describe what happens when the user presses the “Download Values” button. The details about how the recipe download is processed in discussed in section 3.

### Timeout

Each tag write has its own timeout that is individually determined based on the class of the tag and should consider the communication scan rate. There is also an overall timeout that serves as a safeguard for the overall download. Because the download are processed in parallel, the timeout is independent of the number of tags to be written. The time is site configurable via the tag named *downloadTime* in the *Recipe/Constants* folder. An initial setting of two minutes is reasonable.

### Feedback

Once a download starts the operator will receive feedback as described in the following sections.

#### Success

As each value is successfully downloaded from Ignition to the DCS tag or Ignition tag, the STOR and COMP values will update to be the PEND value. The background of the COMP field will change from pink to light blue if the download is successfully confirmed.

If all PEND values downloaded were successfully confirmed, the background of the recipe spreadsheet will turn green.

# Symbolic Ai

This section outlines the tasks that are performed by the recipe ‘engine’ automatically to download, monitor and audit tag values.

## Creating and Deleting Tags

OPC tags are created automatically by the recipe system as they are needed. Shared tags that are used by the recipe, and other toolkits, are implemented via memory tags and are created programmatically on startup.

### OPC Tags

OPC tags are created whenever a new recipe is selected and fetched from the database. Tags are defined in the *RtValueDefinition* table where the *WriteLocation* names an alias in the *RtWriteLocation* table. OPC tags are implemented using the UDTs shown below. Refer to the **I/O Facilities Design Specification** for details.

Figure 7 - Basic I/O User Defined Types Used by Recipe Toolkit

### Local Memory Tags

The Recipe toolkit writes values to OPC tags and to memory tags. By definition, memory tags are local to the Ignition system and are only used by other applications in the Ignition platform. These tags are specified by the “Write Location” value: *Local*. These tags are expected to exist if they are specified in the recipe. The recipe toolkit will not create or delete these tags.

These tags may be created manually or programmatically in a site specific script. For Vistalon, this script is in the external Python library named: *xom.vistalon.startup.createTags()* which is called by *xom.vistalon.startup.gateway()* which is called on startup.

### Recipe Detail User Defined Type

Recipe Detail UDTs may be created automatically whenever a new recipe is selected and fetched from the database. A recipe detail object is created, if one does not already exist, whenever a tag with certain suffixes is encountered. The suffixes and recipe detail type are described below.

|  |  |
| --- | --- |
| Suffixes | Recipe Detail Type |
| SP, SPCH, SPCL, SPHILM, SPLOLM | Recipe SP Details |
| PV, PVHILM, PVLOLM | Recipe PV Details |
| OP, OPCH, OPCL, OPHILM, OPLOLM | Recipe OP Details |

The purpose of these objects is to define the relationship between otherwise independent tags. The relationship is used to coordinate writes to the same controller where we may write a SP and limits around that SP. Although the tags may actually be part of a single controller in the DCS, in Ignition they have been defined as stand-alone tags. It is important that the proper order is used so that a transient alert is not triggered. The coordination is described in section 3.2 It is possible that a PV, SP, or OP tag is encountered without any corresponding limit object. In that case, the detail tag is not really needed, but it will be created anyway.

The UDT for implementing recipe details is:

Figure 8 - Recipe Detail User Defined Type

# Sequential Control

When it is determined that there are clamps for a download, there are as many as three tags whose writes need to be correctly ordered: a high limit, the value, and a low limit. The order must be determined such that value never violates the limits. For example, if the current value of the high limit, value, and low limit are 60, 50, and 40 respectively, and the new values are 90, 80, and 70. Then the order of writes must be high limit, value, and then low limit. Conversely, if the current values are 90, 80, and 70 and the new values are 60, 50, and 40; then the download order must be low limit, value, and then high limit. Whereas all simple write proceed in parallel, these write must occur serially. This is implemented in *ils.io.recipedetail.py* and is accessed by the API *writeRecipeDetail* in *ils.io.api*.

## Foo

bar

# Python

This section describes changes in the API provided by ILS in Python or in some cases by the Symbolic Ai or SFC extension module.

|  |  |  |
| --- | --- | --- |
| **API** | **Old** | **New** |
| system.ils.blt.diagram | This is no longer directly callable. There is now a well defined API to access internal data structures in Symbolic Ai. | ils.blt.api.py |
| ils.diagToolkit.api. getLabValueName() |  |  |
|  | (blockName, blockUUID, diagramUUID) | (diagramName, blockName) |
|  |  |  |
|  |  |  |

## Download Trigger

Refer to section 3.5.1 describes a UDT for implementing automatic recipe downloads.

## Recipe Details

This UDT is used to coordinate writes to a controller in the DCS when there are high and/or low limits and an associated setpoint where the writes need to be coordinated in order to not violate the limits. The recipe toolkit creates instances of the UDT as needed based on the configuration in the recipe database.

The UDT definition is shown below: