

ExxonMobil Chemical Company

Lab Data

User's Guide

Version 1.5

July 28, 2022

**Prepared by:
ILS Automation Inc.**

TABLE OF CONTENTS

1	<i>Introduction</i>	5
2	<i>Architecture</i>	5
2.1	Context Diagram	5
2.2	Data Flow Diagram	6
2.3	Ignition Resources	7
2.3.1	Database Tables	7
2.3.2	UDTs	8
2.3.3	Python	9
2.3.4	Windows and Templates	9
3	<i>Installation, Configuration, and Initialization</i>	9
3.1	Installation	9
3.2	HDA Server	10
3.3	Configuration Preferences	11
3.4	Initialization	13
4	<i>Lab Values</i>	13
4.1	Lab Value UDT	13
4.2	Custom Validation	14
5	<i>Lab Limits</i>	16
5.1	Lab Limit Database Design	16
5.2	SQC Limits	17
5.2.1	SQC Lab Limit Configuration	17
5.3	Validity Limits	18
5.4	Release Limits	21
5.5	Updating Lab Limits	21
5.5.1	Recipe Source	22
5.5.2	DCS Source	22
5.5.3	Constant Source	22
6	<i>Lab Value Selectors</i>	22
6.1	Basic Selector Design	22
6.2	Selector Configuration API	24
6.3	Automatic Selector Configuration	25
6.4	Storing Selector Value History	25
7	<i>Local Lab Data</i>	26

8	<i>Derived Value Processing</i>	26
9	<i>Lab Value Processing</i>	27
10	<i>Lab Data User Interface</i>	28
10.1	Operator User Interface	28
10.1.1	Lab Data Table Window	28
10.1.2	Local Lab Data	29
10.1.3	SQC Plot Screen	32
10.2	Administrator User Interface	34
10.2.1	Configure Lab Data	34
10.2.2	Configure Derived Lab Data	35
10.2.3	Configuring Lab Limits	38
10.2.4	Configuring Display Tables	39
10.2.5	Configure Selector Lab Data	41
10.2.6	Configuring Unit Parameters	42
10.2.7	Raw Value Viewers	42
10.2.8	Load SQC Limits from Recipe	42
11	<i>Unit Parameter Classes</i>	43
11.1	Repeat Sample Value Handling	44
11.2	Data Consistency Due to Grade Change or State Change	45
11.3	Client User Interface	46
12	<i>Lab Data Troubleshooting</i>	47
12.1	OPCHDA Status	47
12.2	Message Queue	47
12.3	Loggers	47
13	<i>Lab Feedback Control – Bias Calculations</i>	47
13.1	Theory	47
13.1.1	Exponential Bias	48
13.1.2	PID Bias	48
13.2	UDTs	48
13.3	External Python	50
13.4	Initialization	51
13.5	Inhibiting I/O	51
13.6	User Interface	53
13.6.1	Bias Summary User Interface	53
13.6.2	Bias Configuration User Interface	54
13.7	Troubleshooting	54

1 Introduction

This document describes the design and use of the Lab Data system. The system manages lab data from three sources:

- PHD – data that is entered into a laboratory management system and accessed using Historical Data Access (HDA). Performing the laboratory analysis takes time, therefore the measured value is reported considerably after the sample was taken.
- DCS – data is available in real time via an OPC tag. The source of the measurement may be an on-line instrument or it may be a laboratory measurement that is reported via the DCS and communicated via OPC.
- Local - This data is entered manually using a window.

Laboratory data differs from most data used in Ignition in that there is a significant delay between the time that the sample is taken and the results are reported. Laboratory data may be used in many ways, but it typically is used by SQC applications that monitor and control product quality. The system provides a robust scheme for validating the value and for handling it if the value does not appear valid. It also calculates the lab value bias and provides feedback to the laboratory measurement system.

2 Architecture

This section describes the Lab Data architecture

2.1 Context Diagram

This diagram shows the systems that are involved in getting measurement data from the Laboratory system. Over time, the interfaces have become more reliable between the systems, but the diagram points out that there are numerous points of failure.

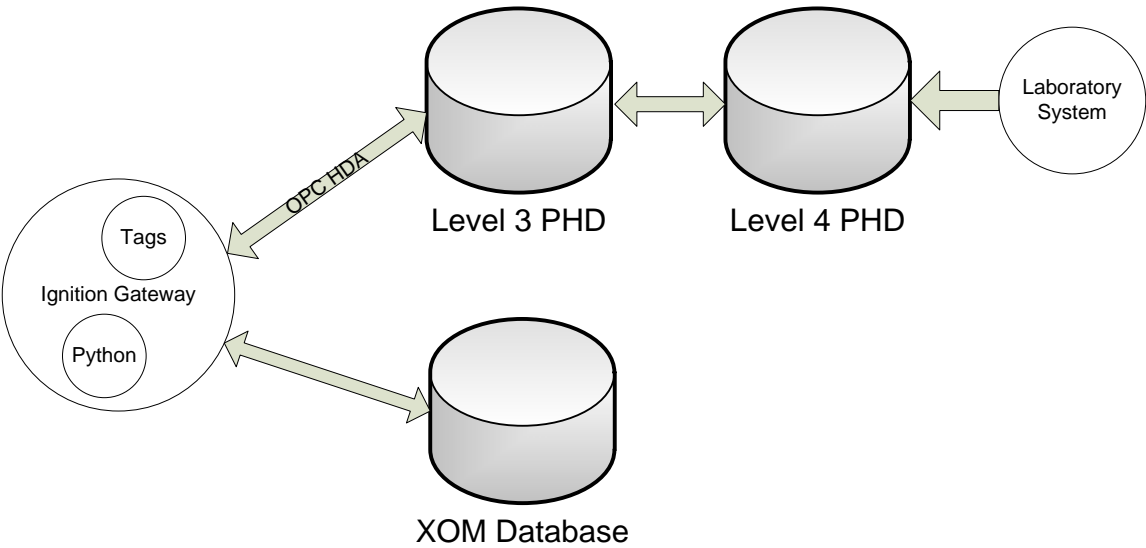


Figure 1 - Lab Data Context Diagram

2.2 Data Flow Diagram

This diagram shows the data stores, processes, and interfaces that are involved in the lab data system. Understanding the data flow is crucial to troubleshooting the system.

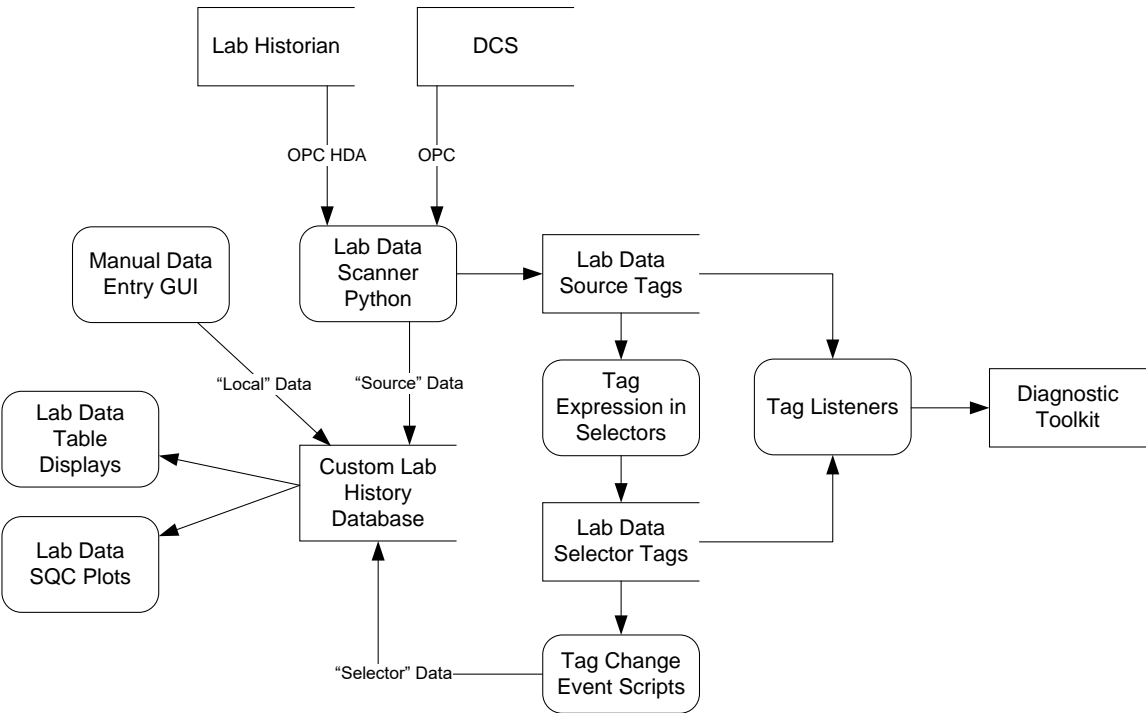


Figure 2 - Lab Data Context Diagram

The toolkit consists of User Defined Templates (UDTs), windows, and Python that executes both in gateway and client scope. The Python is contained in an external package *ils.labdata* in the *user-lib/pylib* folder under the Ignition folder in *Program Files*.

2.3.1 Database Tables

The diagram illustrates a database schema for the 'LTP' system, featuring 15 tables and their relationships. Each table is represented as a box containing its fields, primary keys (PK), and foreign keys (FK). Arrows indicate the relationships between these tables.

Tables and their fields:

- LtDisplayTable**: PK DisplayTableId, FK1 PostId, DisplayTableTitle, DisplayPage, DisplayOrder, DisplayFlag, OldTableName.
- TkPost**: PK PostId, U1 MessageQueueId, LogbookId, DownloadActive.
- LtDerivedValue**: PK DerivedValueId, FK2 ValueId, FK4 TriggerValueId, Callback, SampleTimeTolerance, NewSampleWaitTime, ResultItemId, ResultsInterfaceId, FK1.
- LtHDAInterface**: PK InterfaceId, InterfaceName.
- LtPHDValue**: PK PHDValueId, FK2 ValueId, ItemId, FK1 InterfaceId, AllowManualEntry.
- LtLocalValue**: PK LocalValueId, FK1 ValueId, ItemId, InterfaceId.
- LtLimit**: PK LimitId, FK1 ValueId, FK2 LimitTypeId, FK3 LimitSourceId, RecipeParameterName, UpperValidityLimit, LowerValidityLimit, UpperSQLLimit, LowerSQLLimit, UpperReleaseLimit, LowerReleaseLimit, Target, StandardDeviation, OPCUpperItemid, OPLowerItemid, OPCInterfaceId, FK4.
- LtValue**: PK ValueId, U1 ValueName, Description, DisplayDecimals, UnitId, FK3 ValidationProcedure, LastHistoryId.
- LtRelatedData**: PK RelatedDataId, FK1 DerivedValueId, FK2 RelatedValueId.
- LtDisplayTableDetails**: PK Id, FK1 DisplayTableId, FK2 ValueId, DisplayOrder.
- LtValueViewed**: PK Id, FK1 ValueId, UserName, ViewTime.
- LtSelector**: PK SelectorId, FK1 ValueId, hasSQLLimit, hasValidityLimit, hasReleaseLimit, sourceValueId.
- LtHistory**: PK HistoryId, FK1 ValueId, RawValue, SampleTime, ReportTime, Grade.
- LtDCSValue**: PK DCSValueId, FK1 ValueId, ItemId, FK2 InterfaceId, MinimumSampleIntervalSeconds, AllowManualEntry.
- LtOPCInterface**: PK InterfaceId, InterfaceName.
- Lookup**: PK LookupId, U1 LookupTypeCode, U1 LookupName, LookupDescription, Active.

Relationships (Arrows):

- LtDisplayTable (FK1 PostId) → TkPost (PK PostId).
- TkPost (U1 MessageQueueId) → TkUnit (PK UnitId).
- TkUnit (U1 UnitName) → LtValue (U1 ValueName).
- LtDisplayTable (FK2 ValueId) → LtValue (PK ValueId).
- LtDisplayTable (FK1 PostId) → LtDisplayTableDetails (FK1 DisplayTableId).
- LtDisplayTableDetails (FK2 ValueId) → LtValue (PK ValueId).
- LtValueViewed (FK1 ValueId) → LtValue (PK ValueId).
- LtValueViewed (FK1 ValueId) → LtSelector (FK1 ValueId).
- LtValueViewed (FK1 ValueId) → LtHistory (FK1 ValueId).
- LtValueViewed (FK1 ValueId) → LtDCSValue (FK1 ValueId).
- LtValueViewed (FK1 ValueId) → LtOPCInterface (PK InterfaceId).
- LtValueViewed (FK1 ValueId) → LtLimit (FK4).
- LtValueViewed (FK1 ValueId) → LtLocalValue (FK1 ValueId).
- LtValueViewed (FK1 ValueId) → LtPHDValue (FK1 InterfaceId).
- LtValueViewed (FK1 ValueId) → LtHDAInterface (PK InterfaceId).
- LtValueViewed (FK1 ValueId) → LtDerivedValue (FK1).
- LtValueViewed (FK1 ValueId) → LtLimit (FK1 ValueId).
- LtValueViewed (FK1 ValueId) → LtLimit (FK2 LimitTypeId).
- LtValueViewed (FK1 ValueId) → LtLimit (FK3 LimitSourceId).
- LtValueViewed (FK1 ValueId) → LtLimit (FK4).
- LtValueViewed (FK1 ValueId) → LtLimit (FK5).
- LtValueViewed (FK1 ValueId) → LtLimit (FK6).
- LtValueViewed (FK1 ValueId) → LtLimit (FK7).
- LtValueViewed (FK1 ValueId) → LtLimit (FK8).
- LtValueViewed (FK1 ValueId) → LtLimit (FK9).
- LtValueViewed (FK1 ValueId) → LtLimit (FK10).
- LtValueViewed (FK1 ValueId) → LtLimit (FK11).
- LtValueViewed (FK1 ValueId) → LtLimit (FK12).
- LtValueViewed (FK1 ValueId) → LtLimit (FK13).
- LtValueViewed (FK1 ValueId) → LtLimit (FK14).
- LtValueViewed (FK1 ValueId) → LtLimit (FK15).
- LtValueViewed (FK1 ValueId) → LtLimit (FK16).
- LtValueViewed (FK1 ValueId) → LtLimit (FK17).
- LtValueViewed (FK1 ValueId) → LtLimit (FK18).
- LtValueViewed (FK1 ValueId) → LtLimit (FK19).
- LtValueViewed (FK1 ValueId) → LtLimit (FK20).
- LtValueViewed (FK1 ValueId) → LtLimit (FK21).
- LtValueViewed (FK1 ValueId) → LtLimit (FK22).
- LtValueViewed (FK1 ValueId) → LtLimit (FK23).
- LtValueViewed (FK1 ValueId) → LtLimit (FK24).
- LtValueViewed (FK1 ValueId) → LtLimit (FK25).
- LtValueViewed (FK1 ValueId) → LtLimit (FK26).
- LtValueViewed (FK1 ValueId) → LtLimit (FK27).
- LtValueViewed (FK1 ValueId) → LtLimit (FK28).
- LtValueViewed (FK1 ValueId) → LtLimit (FK29).
- LtValueViewed (FK1 ValueId) → LtLimit (FK30).
- LtValueViewed (FK1 ValueId) → LtLimit (FK31).
- LtValueViewed (FK1 ValueId) → LtLimit (FK32).
- LtValueViewed (FK1 ValueId) → LtLimit (FK33).
- LtValueViewed (FK1 ValueId) → LtLimit (FK34).
- LtValueViewed (FK1 ValueId) → LtLimit (FK35).
- LtValueViewed (FK1 ValueId) → LtLimit (FK36).
- LtValueViewed (FK1 ValueId) → LtLimit (FK37).
- LtValueViewed (FK1 ValueId) → LtLimit (FK38).
- LtValueViewed (FK1 ValueId) → LtLimit (FK39).
- LtValueViewed (FK1 ValueId) → LtLimit (FK40).
- LtValueViewed (FK1 ValueId) → LtLimit (FK41).
- LtValueViewed (FK1 ValueId) → LtLimit (FK42).
- LtValueViewed (FK1 ValueId) → LtLimit (FK43).
- LtValueViewed (FK1 ValueId) → LtLimit (FK44).
- LtValueViewed (FK1 ValueId) → LtLimit (FK45).
- LtValueViewed (FK1 ValueId) → LtLimit (FK46).
- LtValueViewed (FK1 ValueId) → LtLimit (FK47).
- LtValueViewed (FK1 ValueId) → LtLimit (FK48).
- LtValueViewed (FK1 ValueId) → LtLimit (FK49).
- LtValueViewed (FK1 ValueId) → LtLimit (FK50).
- LtValueViewed (FK1 ValueId) → LtLimit (FK51).
- LtValueViewed (FK1 ValueId) → LtLimit (FK52).
- LtValueViewed (FK1 ValueId) → LtLimit (FK53).
- LtValueViewed (FK1 ValueId) → LtLimit (FK54).
- LtValueViewed (FK1 ValueId) → LtLimit (FK55).
- LtValueViewed (FK1 ValueId) → LtLimit (FK56).
- LtValueViewed (FK1 ValueId) → LtLimit (FK57).
- LtValueViewed (FK1 ValueId) → LtLimit (FK58).
- LtValueViewed (FK1 ValueId) → LtLimit (FK59).
- LtValueViewed (FK1 ValueId) → LtLimit (FK60).
- LtValueViewed (FK1 ValueId) → LtLimit (FK61).
- LtValueViewed (FK1 ValueId) → LtLimit (FK62).
- LtValueViewed (FK1 ValueId) → LtLimit (FK63).
- LtValueViewed (FK1 ValueId) → LtLimit (FK64).
- LtValueViewed (FK1 ValueId) → LtLimit (FK65).
- LtValueViewed (FK1 ValueId) → LtLimit (FK66).
- LtValueViewed (FK1 ValueId) → LtLimit (FK67).
- LtValueViewed (FK1 ValueId) → LtLimit (FK68).
- LtValueViewed (FK1 ValueId) → LtLimit (FK69).
- LtValueViewed (FK1 ValueId) → LtLimit (FK70).
- LtValueViewed (FK1 ValueId) → LtLimit (FK71).
- LtValueViewed (FK1 ValueId) → LtLimit (FK72).
- LtValueViewed (FK1 ValueId) → LtLimit (FK73).
- LtValueViewed (FK1 ValueId) → LtLimit (FK74).
- LtValueViewed (FK1 ValueId) → LtLimit (FK75).
- LtValueViewed (FK1 ValueId) → LtLimit (FK76).
- LtValueViewed (FK1 ValueId) → LtLimit (FK77).
- LtValueViewed (FK1 ValueId) → LtLimit (FK78).
- LtValueViewed (FK1 ValueId) → LtLimit (FK79).
- LtValueViewed (FK1 ValueId) → LtLimit (FK80).
- LtValueViewed (FK1 ValueId) → LtLimit (FK81).
- LtValueViewed (FK1 ValueId) → LtLimit (FK82).
- LtValueViewed (FK1 ValueId) → LtLimit (FK83).
- LtValueViewed (FK1 ValueId) → LtLimit (FK84).
- LtValueViewed (FK1 ValueId) → LtLimit (FK85).
- LtValueViewed (FK1 ValueId) → LtLimit (FK86).
- LtValueViewed (FK1 ValueId) → LtLimit (FK87).
- LtValueViewed (FK1 ValueId) → LtLimit (FK88).
- LtValueViewed (FK1 ValueId) → LtLimit (FK89).
- LtValueViewed (FK1 ValueId) → LtLimit (FK90).
- LtValueViewed (FK1 ValueId) → LtLimit (FK91).
- LtValueViewed (FK1 ValueId) → LtLimit (FK92).
- LtValueViewed (FK1 ValueId) → LtLimit (FK93).
- LtValueViewed (FK1 ValueId) → LtLimit (FK94).
- LtValueViewed (FK1 ValueId) → LtLimit (FK95).
- LtValueViewed (FK1 ValueId) → LtLimit (FK96).
- LtValueViewed (FK1 ValueId) → LtLimit (FK97).
- LtValueViewed (FK1 ValueId) → LtLimit (FK98).
- LtValueViewed (FK1 ValueId) → LtLimit (FK99).
- LtValueViewed (FK1 ValueId) → LtLimit (FK100).
- LtValueViewed (FK1 ValueId) → LtLimit (FK101).
- LtValueViewed (FK1 ValueId) → LtLimit (FK102).
- LtValueViewed (FK1 ValueId) → LtLimit (FK103).
- LtValueViewed (FK1 ValueId) → LtLimit (FK104).
- LtValueViewed (FK1 ValueId) → LtLimit (FK105).
- LtValueViewed (FK1 ValueId) → LtLimit (FK106).
- LtValueViewed (FK1 ValueId) → LtLimit (FK107).
- LtValueViewed (FK1 ValueId) → LtLimit (FK108).
- LtValueViewed (FK1 ValueId) → LtLimit (FK109).
- LtValueViewed (FK1 ValueId) → LtLimit (FK110).
- LtValueViewed (FK1 ValueId) → LtLimit (FK111).
- LtValueViewed (FK1 ValueId) → LtLimit (FK112).
- LtValueViewed (FK1 ValueId) → LtLimit (FK113).
- LtValueViewed (FK1 ValueId) → LtLimit (FK114).
- LtValueViewed (FK1 ValueId) → LtLimit (FK115).
- LtValueViewed (FK1 ValueId) → LtLimit (FK116).
- LtValueViewed (FK1 ValueId) → LtLimit (FK117).
- LtValueViewed (FK1 ValueId) → LtLimit (FK118).
- LtValueViewed (FK1 ValueId) → LtLimit (FK119

Page 7

2.3.2 UDTs

The following UDTs are defined expressly for Lab Data. The purpose of these UDTs is to expose the Lab Data values processed by the Lab Data toolkit for use by other toolkits. Even though the tags are memory tags, they should not be edited in order to coerce a new lab data value into the system.

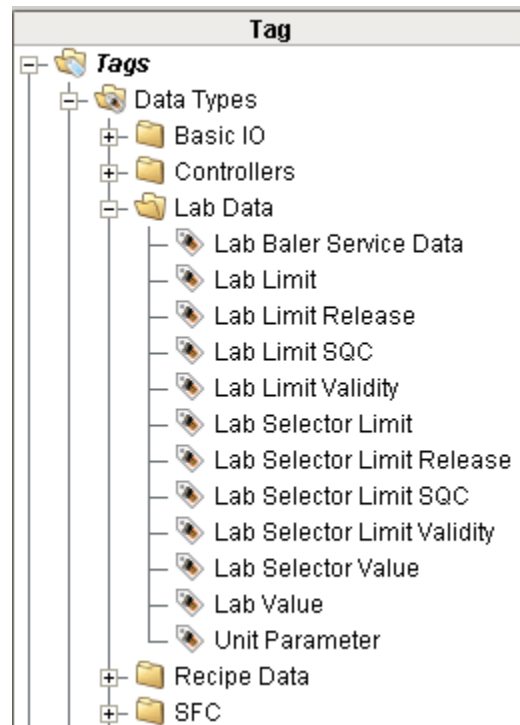


Figure 4 - Lab Data User Defined Types (UDTs)

2.3.3 Python

The python used to implement Lab Data is contained in external Python in the ils.labData package

2.3.4 Windows and Templates

The windows and templates are contained in the Lab Data folder.

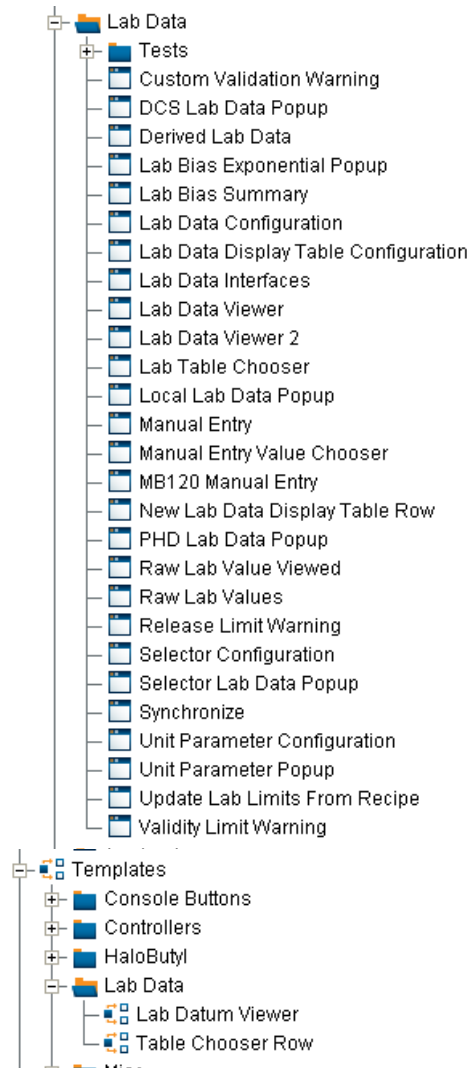


Figure 5 - Ignition Windows for Lab Data

3 Installation, Configuration, and Initialization

This section provides a description of the required software for installation and the start-up configuration options for the Lab Data toolkit.

3.1 Installation

There are no special installation steps required for this toolkit; all screens, UDT's, and Python scripts are delivered as part of the normal installation.

3.2 HDA Server

The lab data module uses the OPC Historical Data Access protocol to access lab data in the PHD historian. In Ignition, from the gateway web page, a special historical tag provider needs to be configured as shown below.

Home

Status

Configure

Launch Designer

System

Status

Licensing

Backup/Restore

Console

User Manual

Configuration

Projects

Modules

Gateway Settings

Redundancy

Security

Users, Roles

Auditing

Databases

Connections

Drivers

Store and Forward

Alarming

General

Journal

Schedules

Tags

Realtime

History

Database Tag History Providers

Data Connection	Enabled	Status	
BatchExpert	true	Running	edit
GSKBEDB	true	Running	edit
Lafarge	true	Running	edit
Vistalon_Recipe	true	Running	edit
Vistalon_SQC	true	Running	edit
Vistalon_Tower_Recipe	true	Running	edit
XOM	false	Running	edit
XOMMigration	true	Running	edit
XOMhistory	true	Running	edit

Other Historical Tag Providers

Provider Name	Enabled	Description	Status	
PHD-HDA	true	Kepware's OPC HDA server which emulates PHD.	Connecting	edit delete

Create new Historical Tag Provider...

Figure 6 - HDA Server Configuration

3.3 Configuration Preferences

The Lab Data module uses a set of memory tags to customize the behaviour of the module. The following characteristics of the Lab Data Toolkit can be customized using the Ignition Designer.

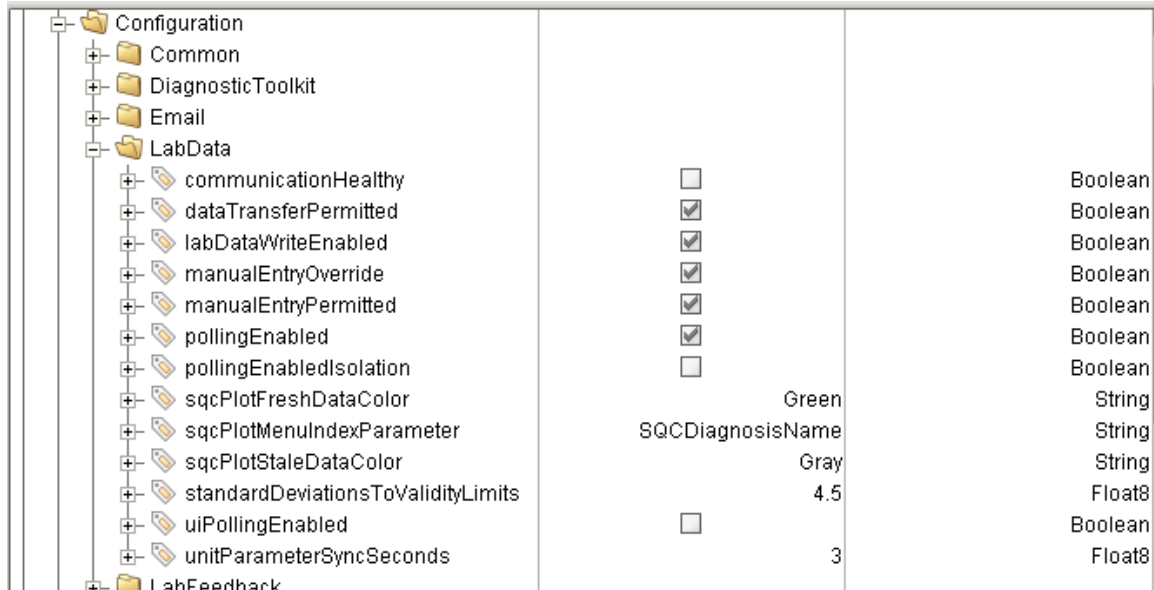


Figure 7 - Configuration Tags used for Lab Data

Tag	Data Type	Description
Communication Healthy	Boolean	This is automatically set by the Lab Data / HDA watchdog logic. It is considered by the logic that sets the <i>manualEntryPermitted</i> tag.
dataTransfer Permitted	Boolean	This is used by the data transfer UDTs. It is checked by the permissive expression inside each UDT. This is set manually, it is not managed by any of the watchdogs. It should be added to the Communication Status window.
labDataWrite Enabled	Boolean	Used to enable / disable writing values to external systems. This does NOT inhibit reading lab data from various sources nor from writing the processed values to local memory tags. It is used when writing derived lab values out to PHD.

Tag	Data Type	Description
manualEntry Override	Boolean	This can be manually set to override the manual lab data entry policy of only allowing local data to be entered manually. It may also be used when communication to the lab data system is down, but the watchdogs have not yet detected it.
manualEntry Permitted	Boolean	This is automatically set by the HDA watchdogs and should not be set manually. It is the primary tag considered by the Manual Data Entry system to determine which Lab Data values can be entered manually.
pollingEnabled	Boolean	This must be True for the module to automatically retrieve values from the OPC-HDA server
pollingEnabled Isolation	Boolean	This must be True for the module to automatically retrieve values from the OPC-HDA server for ISOLATION lad data tables and tags
sqcPlotFresh DataColor	String	Used to configure the chart pen for the fresh data.
sqcPlotMenu IndexParameter	String	Specifies the property that will be used as the button labels in the SQC Plot chooser (section 10.1.3). The options are: "SQCDiagnosisName", "LabValueName", or "LabValueDescription".
sqcPlotStale DataColor	String	Used to configure the chart pen for the stale data.
standardDeviations ToValidityLimits	Float	The number of standard deviations from the target value to the upper and lower SQC limits defined in the RtSQLLimit tables. This is used to calculate the target and standard deviation.
uiPolling Enabled		Specifies if the Lab Data Table chooser and the SQC plot chooser table update automatically via a timer to reflect change in status.
unitParameter SyncSeconds	Float	Amount of time to wait, when processing unit parameters, to synchronize the lab value and the lab sample time.

3.4 Initialization

The Lab Data module has its own initialization logic in *ils.labData.startup.gateway*. Its only purpose is to create required configuration tags and assign default values if they do not already exist. It is generally a requirement to restore lab data history on startup to recover data that has arrived while the system was down or if this is an initial installation to restore data for the recent past. However, it is generally necessary to perform site specific selector configuration first. Therefore, there is no built-in history restoration as part of the lab data module. The module provides a function *ils.labData.startup.restoreHistory(tagProvider, daysToRestore)*. The *daysToRestore* argument is an optional argument with a default value of 7.

4 Lab Values

As mentioned in the introduction, the system manages lab data from three sources:

- PHD / HDA-OPC – data that is entered into a laboratory management system and accessed using Historical Data Access (HAD). Performing the laboratory analysis takes time, therefore the measured value is reported considerably after the sample was taken.
- DCS / OPC – data is available in real time via an OPC tag. The source of the measurement may be an on-line instrument or it may be a laboratory measurement that is reported via the DCS and communicated via OPC.
- Local / Manual GUI - also known as Local Lab Data. This data is entered manually using a window. Refer to section 7 for details.

The most common source of lab data is from PHD. The normal use case is that a technician takes a product sample, takes it to the lab, the lab analyses it and enters the results into a Lab Information Management System (LIMS). The LIMS transfers the results to the PHD historian where it can be accessed by Ignition using a HDA-OPC communication. Using HDA-OPC preserves the original sample time in addition to being able to record the time that the results were reported.

Regardless of the source, lab values are processed and stored the same.

Refer to section 10 for details on viewing lab data. Refer to section 10.2 for details on configuring lab data.

4.1 Lab Value UDT

Regardless of the source of the lab data, the same “Lab Value” UDT is used. The UDT defines the public interface to lab data values. Any of the tags that are available in the UDT are available to other toolkits. The two tags of particular interest are *value* and *sampleTime*. The last update time of the value tag is the time that the value was reported as opposed to the time that the sample was taken, which is what the toolkits are generally interested in. The *badValue*, *rawValue*, and *status* tags are present only

for troubleshooting. It is unclear why any toolkit would ever want to use the *rawValue* since it is not validated.

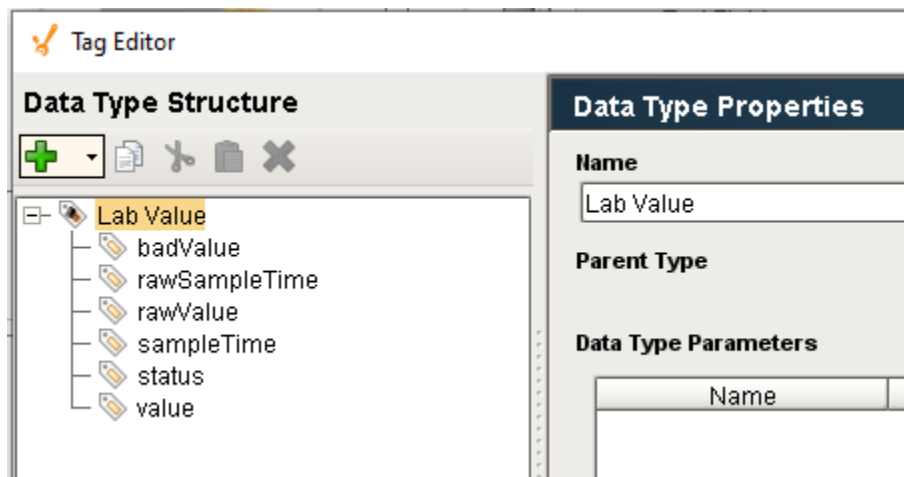


Figure 8 - Lab Value UDT

4.2 Custom Validation

In addition to built-in value validation provided by validation and release limits, discussed in sections 5.3 and 5.4, the system allows a totally custom validation strategy to be implemented by writing a Python script. The Python can be located in external Python, if using an external IDE such as Eclipse, or in the global shared scope. If using external Python then it should be placed in the module *xom.site.labData.validationCallbacks.py* where site is your site name such as Vistalon.

If in shared scope it should be placed in *shared.site.labData.validationCallbacks*. The custom validation script takes seven arguments as shown below:

```
7 import system
8 from ils.labData.common import postMessage
9 log = system.util.getLogger("com.ils.labData.customValidation")
10
11 |
12 # This callback expects a single value in the data dictionary
13 # This callback is called by several DCS lab datums
14 def labBalerInService(valueId, valueName, rawValue, sampleTime, unitName, tagProvider, database):
15     log.trace("In LabBalerInService with %s - %s" % (valueName, str(rawValue)))
```

It returns three values as shown below:

Variable	Type	Description
isValid	Boolean	Specifies if the lab data value is valid. If it is valid then the lab value handling framework will promote the lab value to the lab data UDT and to the lab value history database. If the value is not valid then the handling depends on the notifyConsole setting below.
notifyConsole	Boolean	This value is only relevant if isValid = False. If notifyConsole is True then the framework will attempt to notify the appropriate console to allow the operator to accept the value in spite of it failing validation. If no console is available, then the value will be accepted. The notification is similar to the Validity Limit notification described above. If notifyConsole is False then the value will be rejected without notifying the operator.
statusMessage	String	If the value is not valid, then this will be written to the status tag of the lab value UDT.

If the custom validation chooses to implement some totally custom notification / override then it is up to the custom method to save the values to the Lab Data tags and Lab Data history tables. The custom validation procedure runs in the gateway and should not have wait states. Any user interaction runs in a client so communication between the validation procedure in the gateway and the client notification will require a message.

The custom validation callback is specified in the Lab Data Configuration screen described in section 10.2.1. It is stored in the database in table *LtValue* in attribute *ValidationProcedure*.

5 Lab Limits

There are three types of limits that can apply to any lab datum. Each limit type has an upper and lower limit and are generally symmetrical about the target. The three limit types are:

- SQC – the tightest limits, these generally define the process limits. Generally +/- 3 standard deviations from the target value. When SQC limits are defined, validity limits are automatically calculated from the SQC limits.
- Validity - the tightest limits, these generally define the process limits. Generally +/- 4.5 standard deviations from the target value.
- Release – the loosest limits, these define the product specification limits. Any material outside of these limits will need to be scrapped or reworked.

5.1 Lab Limit Database Design

The limits are modelled in the database using the LtLimit table which is related to the LtValue table via the ValueId. Whenever a new lab value is entered, from PHD, OPC or local, the LtLimit table is searched for a limit record. The table would allow all three sets of limits to be defined for the same lab datum, although this is not generally the practice. Although

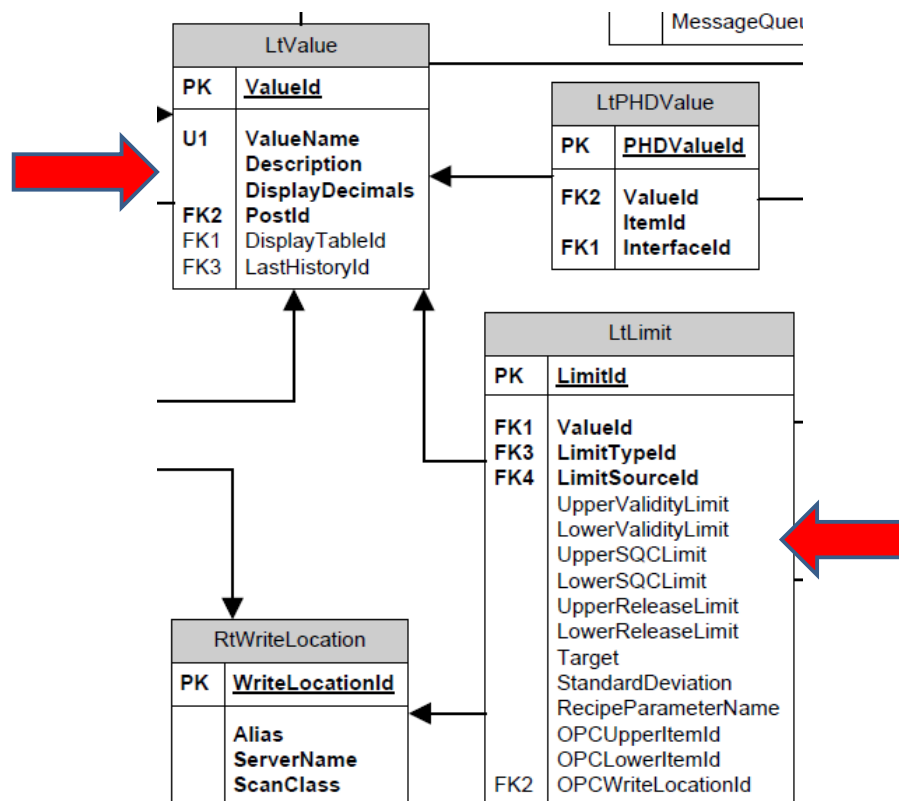


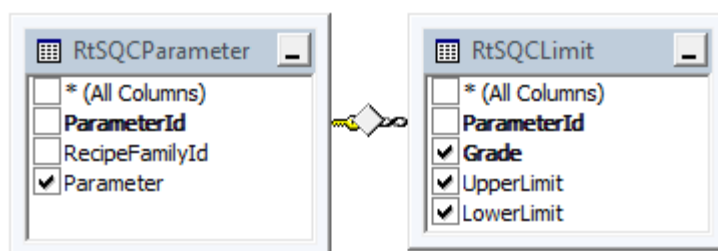
Figure 9 - Lab Limit Database Tables

5.2 SQC Limits

SQC limits are the tightest limits and generally define the process limits. They are assumed to be ± 3 standard deviations from the target value although there is a configuration tag that defines the number of standard deviations between the upper and lower limits. Whenever SQC limits are defined, validity limits are automatically calculated from the SQC limits assuming ± 4.5 standard deviations although this can also be changed via a configuration tag (see section 3.3). The main purpose of SQC limits is to feed the SQC diagrams in *Symbolic Ai* and to define the target and limits for the SQC plotting facilities. Lab values are not validated against the SQC limits like they are validated against validity and release limits.

5.2.1 SQC Lab Limit Configuration

SQC lab limits are generally loaded from the recipe system automatically when a grade change is detected. In order to avoid inconsistent data, only the SQC limits are stored in the recipe database. The target, standard deviation, and validity limits are calculated from the SQC limits when the recipe limits are loaded into lab data. The recipe system tables that model SQC limits and some typical data are shown below. If either the upper or lower limit is NULL then the LtLimit will be updated with NULL and NAN will be written to tags. Generally, the limits are double-sided which allows a target value, standard deviation, and validity limits to be calculated.



Parameter	Grade	LowerLimit	UpperLimit
SPEC_MLR	404	65	155
SPEC_MLR	706	NULL	NULL
SPEC_MLR	707	25	95
SPEC_MLR	8731	70	270
SPEC_MLR	878	NULL	NULL
SPEC_MLR	919	NULL	NULL
SPEC_C2	1696	59	65
SPEC_C2	1705	75.3	78.3
SPEC_C2	2504	54.5	60.5
SPEC_C2	2727	54.5	58.5
SPEC_C2	3666	61	65

Figure 10 - Recipe Toolkit Database Tables and Data

Data from the recipe tables shown above are loaded into the active lab data tables by the Python script *ils.labData.limits.updateSQLLimitsFromRecipe(grade)* which should be called by the grade change logic if lab data and SQC limits are being used. The lab data limits must be carefully configured so that the *LtLimit.RecipeParameterName = RtSQCParemeter.Parameter*. A portion of the *LtLimit* data for Vistalon is shown below.

ValueName	RecipeParameterN...	UpperValidityLimit	UpperSQLLimit	Target	StandardDeviat...	LowerSQLLimit	LowerValidityLim
R1-C9-LAB-DATA	RX_C9	200	200	NULL	NULL	-200	-200
R1-C2-LAB-DATA	RX_C2	61.5	61.5	NULL	NULL	59.5	59.5
R1-ML-LAB-DATA	NULL	57.8	57.8	NULL	NULL	48.2	48.2
R2-C9-LAB-DATA	RX_C9	200	200	NULL	NULL	-200	-200
R2-C2-LAB-DATA	RX_C2	61.5	61.5	NULL	NULL	59.5	59.5
FD-C9-LAB-DATA	RX_C9	200	200	NULL	NULL	-200	-200
FD-C2-LAB-DATA	RX_C2	61.5	61.5	NULL	NULL	59.5	59.5
PROD-ML-LAB-DATA	PROD_ML	150	150	NULL	NULL	-50	-50
CA-LAB-DATA	PROD_CA	1500	1500	NULL	NULL	-500	-500
STAB-LAB-DATA	PROD_IRG	0.14	0.14	NULL	NULL	0.06	0.06

Figure 11 - Typical Lab Data Limit Data

SQC limits can also be loaded manually from a client with access to the admin menu. This can be used if the limits in recipe were updated after the grade change has processed or if the grade change process failed for some reason. This will update all of the SQC limits for the current grade. Individual limits can also be edited using the configuration screens described in section 10.2

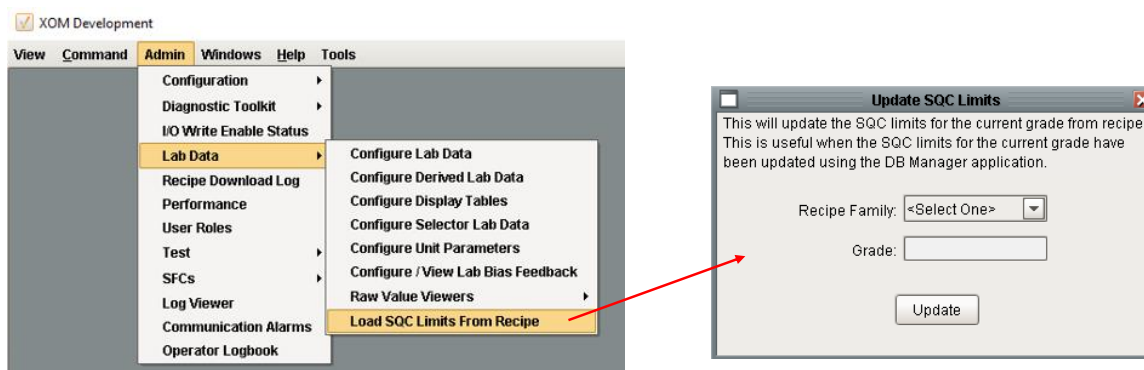


Figure 12 - Manually Loading SQC Limits From Recipe

5.3 Validity Limits

Validity limits have an upper and lower limit. They may be manually configured but are generally calculated from the SQC limits which are loaded automatically on grade change. The toolkit provides automatic validation of any **lab-data** for which a validity limit has been defined. The validation compares the raw value with a high limit and a low limit. Values that are outside of the acceptable limits are a serious problem and it is critical that the operator be notified immediately. If the raw value is outside the configured limits, then the responsible console/post will be notified of

the lab data error by sending an “OC Alert” message to the appropriate post. The display shown below is large and the background toggles between purple and red.

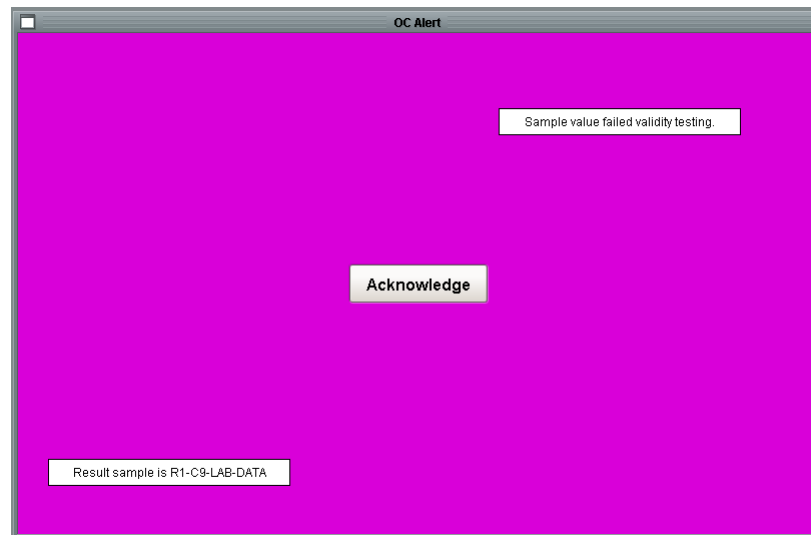


Figure 13 - OC Alert Screen for Validity Limit Violation

When the user presses the ‘Acknowledge’ button, then the screen shown below is displayed.

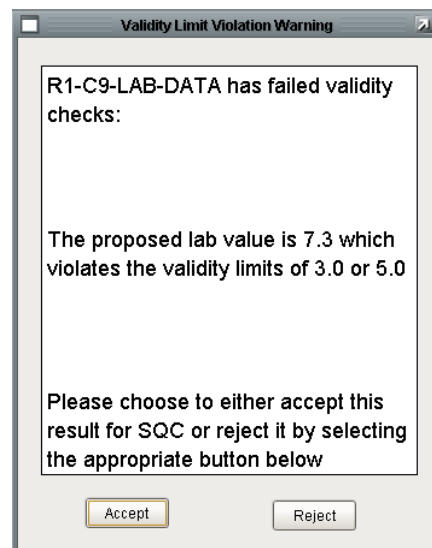


Figure 14 - Validity Limit Violation Warning Screen

The operator needs to determine if the raw value is believable and if the product attribute being measured is really out-of-spec. If the operator determines that the measurement is not believable due to an operator error, a measurement error, or a data communication error, then the operator can press “Reject” to reject the measurement. If the operator believes the measurement then the operator should press “Accept”.

If the operator presses "Reject" then no further action is taken and the lab value is effectively discarded. If the operator presses "Accept" or if the timeout period elapses, then the value and sample time are written to the UDT and the value is stored in history.

The action taken by the operator is recorded to the *LABDATA* message queue. The message contains details about the lab data parameter including parameter name, value and limits.

Additional Considerations:

- If the console is not connected when the message is sent, then the value is automatically accepted – the presumption is that the measurement is valid.
- There is not any logic for a new value arriving while processing the previous value. Depending on how long the timeout is for the operator response to a bad value; a new value could arrive while the operator is pondering whether to accept or reject the current value.
- There is not any logic to protect against the client window becoming disconnected while the warning window is displayed. The timeout runs in the client. If the client is lost then the value will not be accepted.
- There is a timeout on the violation warning screen but not on the OC notice screen – so that screen could flash for hours and the lab value would never be accepted.

5.4 Release Limits

Release limits implement product performance limits that are based on product specifications rather than on process capability. Violation of release limits mean that the product is probably off-spec and may require special treatment in order to be released. Similar to validity limits, the toolkit provides automatic validation of any **lab-data** for which release limits have been defined. The validation uses the flashing notification screen to the operator just the validity limit processing. Once the notification is acknowledged, the following dialog is posted. It provides a convenient mechanism for automatically generating a UIR.

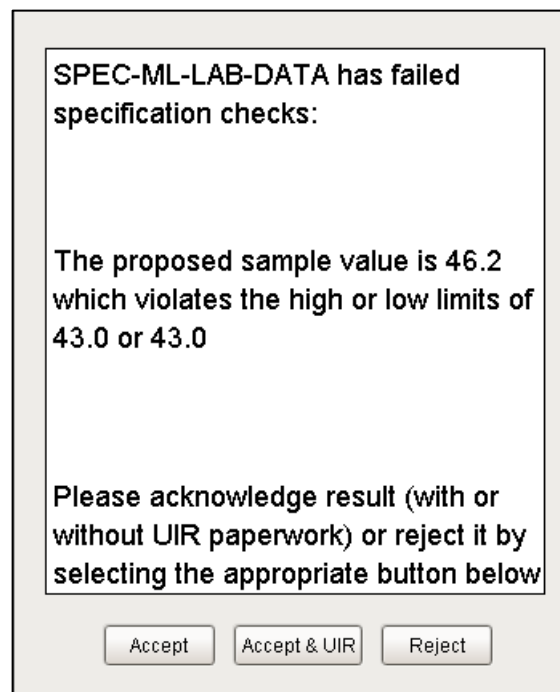


Figure 15 - Release Limit Notification Dialog

5.5 Updating Lab Limits

The lab data database tables define current limits that are active at the current moment in time. The Lab Data UDTs display the current limits but they do not define the limits, editing lab data limit UDTs have no effect on the limits used during value processing. The system does not contain any provision for defining a profile of limits to be used based on a criteria such as grade, reactor configuration, or rate. The current lab limits can always be changed in the configuration user interface, see section 10.2.3 for details. Additionally, there are a couple of mechanisms for updating the lab limits depending on the source type.

5.5.1 Recipe Source

Lab limits whose source is “recipe” can be updated from the *RtSQCLimitView* database view. Limit data whose source is “recipe” define a recipe key which is combined with the current grade to update the limits from the recipe database. This can be done automatically on grade or reactor configuration change via a custom script. It can also be done using the “*Load SQC Limits From Recipe*” window, see section 10.2.8 for details.

5.5.2 DCS Source

Lab limits whose source is “DCS” are continuously being read from the source to see if they have been updated. If the remote system implements a limit profile, then the limits will be updated in Lab Data whenever they are updated in the remote system.

5.5.3 Constant Source

Lab limits whose source is “constant” can be changed in the configuration user interface, see section 10.2.3 for details. Constant limits can also be updated by a custom Python script that directly updates the database using site specific logic.

6 Lab Value Selectors

Lab Value Selectors provide a mechanism whereby a generic piece of lab data may come from different tags or sources based on the configuration of the unit. This allows other toolkits within the application framework to reference the generic piece of lab data and does not need to be concerned with the original source or the management of the source.

6.1 Basic Selector Design








Lab Value tags/UDTs are configured for each of the possible data sources and then the appropriate value is promoted to generic selector object which may then be propagated to the appropriate unit parameter. In a scenario where a lab value could come from three different sources, there would be three tags and one selector. The selector would be configured to reference one of the three different sources depending on some business rule.

A typical implementation is shown below. The lab datum objects, R1-C9-LAB-DATA, R2-C9-LAB-DATA, and FD-C9-LAB-DATA are configured and update automatically from PHD. Based on the configuration of the unit, one of the lab datum objects are designated to promote their values to the lab selector C9-LAB-DATA. The purpose of using a lab selector is that regardless of the unit configuration, the C9 lab results will










be available in C9-LAB-DATA. This simplifies the design of windows and diagnostic diagrams that are interested in C9 regardless of where the sample is taken.

 LabData	
 C9-LAB-DATA	Lab Data/Lab Selector Value
 FD-C9-LAB-DATA	Lab Data/Lab Value PHD
 R1-C9-LAB-DATA	Lab Data/Lab Value PHD
 R2-C9-LAB-DATA	Lab Data/Lab Value PHD

The Lab Value UDTs have a set of memory tags that are explicitly updated from the Lab Data module.

 R1-C9-LAB-DATA		Lab Data/Lab Value
 badValue	<input checked="" type="checkbox"/>	Boolean
 rawSampleTime	2018-08-06 2:36:37 PM	DateTime
 rawValue	9.1	Float8
 sampleTime	2018-08-06 2:36:37 PM	DateTime
 status	Demo Expired	String
 value	9.3	Float8

The selector UDT has a set of expression tags that refer to the appropriate Lab Value UDT instance.

 C9-LAB-DATA		Lab Data/Lab Selector Val...
 badValue	<input checked="" type="checkbox"/>	Boolean
 processingEnabled	<input checked="" type="checkbox"/>	Boolean
 rawSampleTime	2018-08-06 2:36:37 PM	DateTime
 rawValue	9.1	Float8
 sampleTime	2018-08-06 2:36:37 PM	DateTime
 status	Demo Expired	String
 trigger	<input type="checkbox"/>	Boolean
 value	9.3	Float8

The expression tags are configured from properties on the UDT. The figure shows that the source of the selector at this moment is R1-C9-LAB-DATA:

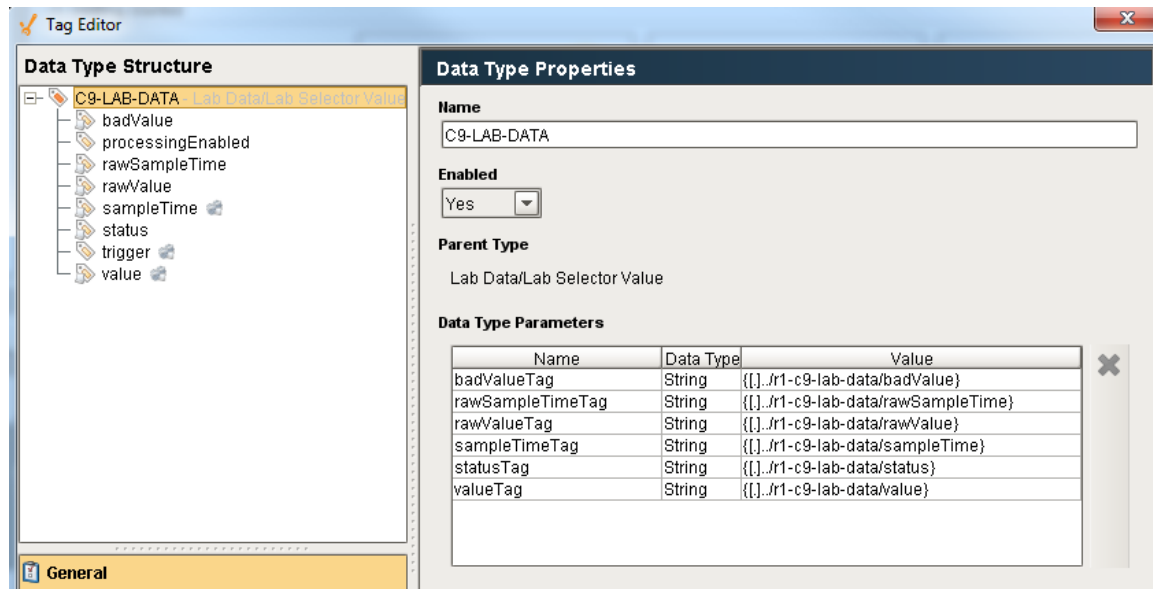


Figure 16 - Selector Instance Configuration

The *processingEnabled* Boolean memory tag is worth discussing. As will be discussed below in section 6.4, storing selector values in history is event based whenever the selector receives a new value rather than the mechanism implemented for reading values from PHD. The *processingEnabled* tag is used to prevent unwanted values from propagating into lab value history. Without the tag, whenever a selector is configured to point to a new source the current value of that source would be treated as a new value even though it is not new. The tag is managed by the selector configuration API discussed in section 6.2 and should not need to be manually set.

6.2 Selector Configuration API

Selectors such as the one shown in the previous section are generally configured via a Python scripting interface based on unit configure, grade, or whatever logic is appropriate. For example, the Vistalon selector configuration considers the unit configuration and the flash drum configuration. A portion of a script demonstrating the API is shown below:

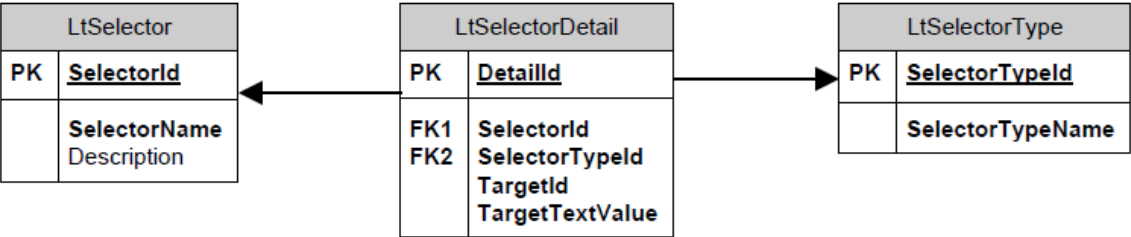
```
if not(seriesRxGrade):
    from ils.labData.selector import configureSelector
    configureSelector("mooney-lab-data", "r1-ml-lab-data")
    configureSelector("mlr-lab-data", "r1-mlrq-lab-data")
    configureSelector("mlr-raw-lab-data", "r1-raw-mlrq-lab-data")
```

Figure 17 - Selector Caption API Example

6.3 Automatic Selector Configuration

This is just an idea with the goal being a data driven way of configuring selectors. I don't think it would work for Vistalon, but maybe it would work for other sites. At this time it will not be implemented, but it remains in the specification for future consideration.

The idea behind automatic configuration is that an event would be defined and then a set of detail configurations would be defined for that event. The processing of the details would be provided by the Lab Data module. Detection of the event is left to the site-specific logic. An API would be provided to implement the details for the event in a single call. The table structure to support this is shown below.



6.4 Storing Selector Value History

The history for selector lab data is stored in an entirely different manner than normal lab value handling. Lab Data history is stored in the *LtHistory* database table. Normal lab data inserts data into this table by *labData.scanner.py* in the same thread as the data is read from PHD or the DCS. As described above, lab data selectors are implemented entirely by tags. The Lab Selector Value UDT shown below defines value changed handlers that insert the lab values into the *LtHistory* table. The UDT contains six expression tags that generally all update at the same time. The *value* and *sampleTime* need to be stored in the *LtHistory* table. In order to synchronize the insert for near simultaneous update the UDT contains a trigger tag. The “Value Changed” handler for the *value* and *sampleTime* is identical and shown below:

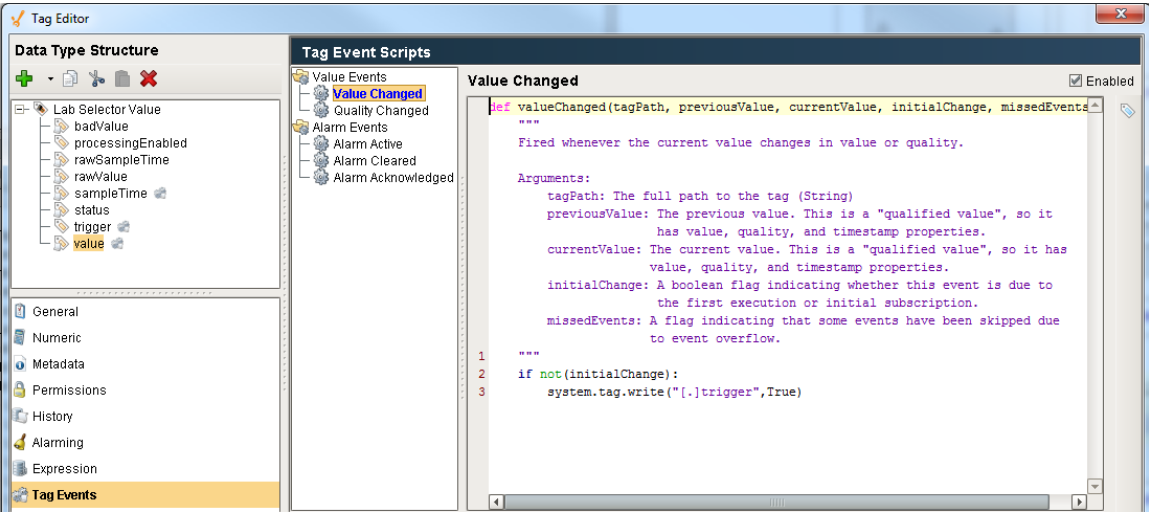


Figure 18 - Selector Value Change Script

The trigger tag change scripts is shown below. It calls a Python script which inserts the data from the *value* and *sampleTime* tags and inserts it into *LtHistory*.

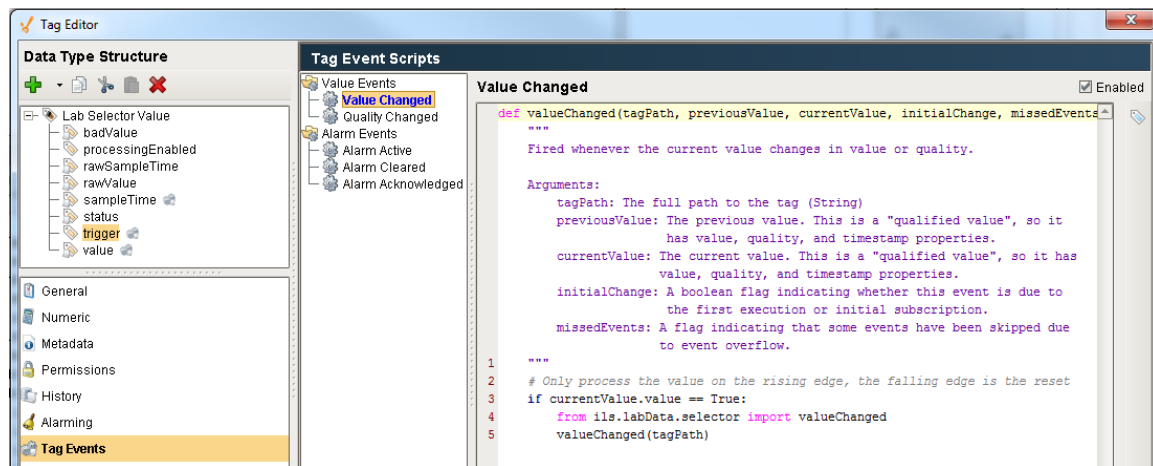


Figure 19 - Trigger Value Change Script

7 Local Lab Data

The majority of lab data is expected to be entered automatically via OPC or OPCHDA communication to external lab measurement systems. However, lab data can also be defined as “Local” meaning that it does not come from an external source. Refer to section 10.1.2 for details on entering *local* lab data.

8 Derived Value Processing

Derived values are values that are derived from one or more lab values. The derived value is calculated in a Python script using custom logic. The value that triggers the calculation is known as the trigger value. The logic generally involves one or more additional lab values, also known as related data. The toolkit ensures that all of the lab measurements are consistent by checking that the sample time of all the measurements are within a user specified time window, known as the *Time Tolerance*, **before** the user-written script is called. If they are not immediately consistent, the module will continue to check for a user specified amount of time, known as the *Wait Time*. Once all of the lab values are consistent, then the Python calculation method will be called. The previous toolkit required that this bookkeeping be done in the user written script.

The values, including the trigger value, are passed to Python in a dictionary of dictionaries.

```
{
  'ENB-IN-CRUMB': {'valueId': 3086, 'valueName': 'ENB-IN-CRUMB', 'rawValue': 31.788999557495117},
  'C9-IN-CRUMB': {'valueId': 3092, 'valueName': 'C9-IN-CRUMB', 'rawValue': 18.5}
}
```

The following examples show actual Vistalon callback methods that illustrate how to access the data dictionary. The first example, `c9InCrumb`, iterates over all of the related data and simply sums its values. Because the toolkit validated the data before

calling this method, the callback is quite straight forward. Similarly, the callback just needs to return the value, the toolkit will write the value to the UDT tag, the local history, and to OPC if the data is configured with a write target.

```
# The derived value is simply the sum of the raw values in the data dictionary.
# The order is not important!
def c9InCrumb(dataDictionary):
    log.trace("In c9InCrumb - the data dictionary is: %s" % (str(dataDictionary)))

    derivedVal = 0.0
    for d in dataDictionary.values():
        valueName=d.get("valueName", "")
        rawValue=d.get("rawValue", 0.0)
        log.trace("Adding: %s raw value: %s" % (valueName, str(rawValue)))
        derivedVal = derivedVal + rawValue

    return derivedVal
```

The second callback is written in a generic fashion but is dependent on there being exactly two values in the dictionary. The logic is to calculate the derived value as the trigger value minus the other value. This called from two different trigger values each with a different related data.

```
# This can be used in conjunction with a single related value and calculates the difference
# between the trigger and the related value (trigger - related)
def rxPropertyDiff(dataDictionary):
    if len(dataDictionary) != 2:
        print "ERROR"

    for d in dataDictionary.values():
        trigger=d.get("trigger", False)
        if trigger:
            triggerValue = d.get("rawValue", 0.0)
        else:
            relatedValue = d.get("rawValue", 0.0)

    derivedVal = triggerValue - relatedValue

    return derivedVal
```

The related data is generally always lab data. If real-time data is needed in a calculation callback then it can be acquired directly via a tag read. Related data is generally only used when consistency is a concern.

The trigger value for derived data is generally a PHD lab value, but it could also be a DCS or local value. The derived value is configured in the LtDerivedValue table. The data that is related to the trigger value is defined in the LtRelatedData table.

9 Lab Value Processing

Processing new lab values is entirely automatic. Lab data is polled on a regular interval for new lab data. The value that is read is compared to the last value processed. If they are the same then there is no further processing.

The flowchart below shows the general processing that occurs when a new lab value is received.

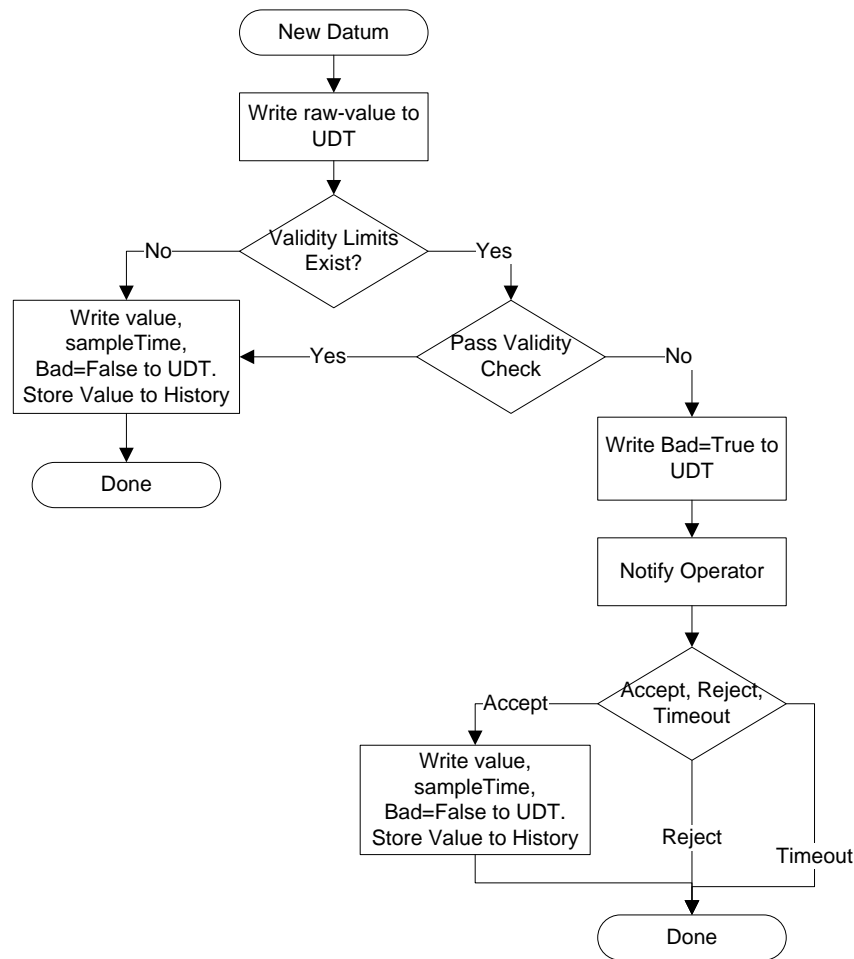


Figure 20 - Lab Value Processing Flowchart

10 Lab Data User Interface

The Lab Data system provides a user interface for viewing and configuring lab data.

10.1 Operator User Interface

There are two main windows used by the operator related to lab data, the Lab Data Table screen and the SQC plot screen.

10.1.1 Lab Data Table Window

An application can contain many measurements; therefore the measurements are organized into logical groups or pages for viewing purposes. The user interface is accessed via the lab data pushbutton on the common console, shown below. Pressing

the button opens the “Lab Data Table Chooser” window. The window provides a menu of the lab data tables available for the operator console that pressed the button.

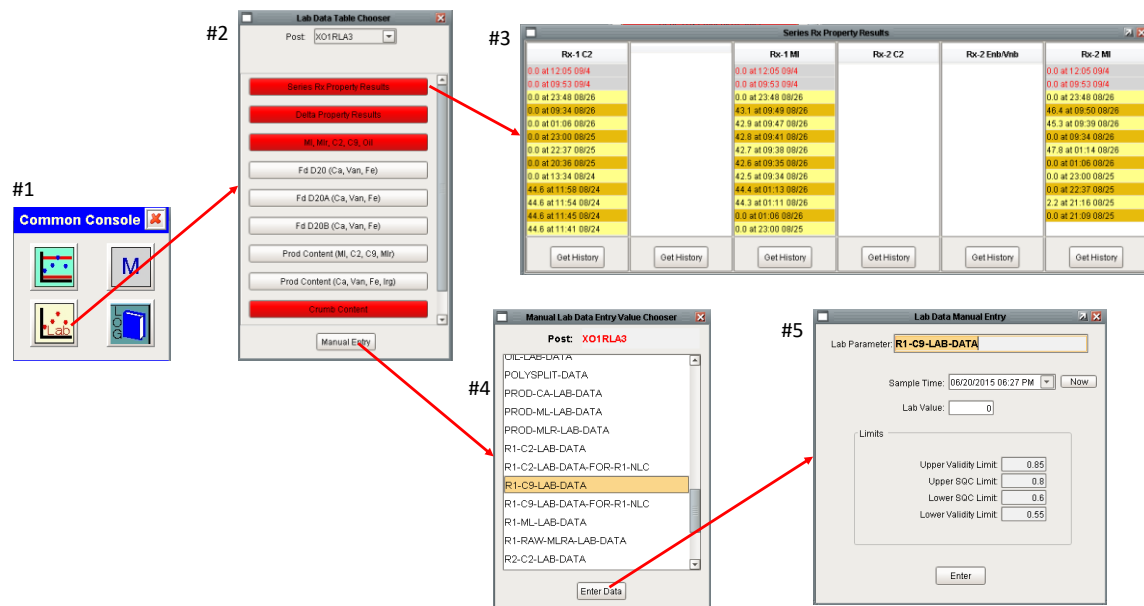


Figure 21 - Main User interface Context

Selecting any of the table names opens the screen in the top right that contains a row of tables each of which displays the most recent thirteen measurements for a lab value. The table updates automatically every 15 seconds. The previous application updated every ten minutes.

The “Get History” button is used to restore measurements that may have arrived when the OPC HDA interface was down. This should only be necessary if more than one measurement arrived while the interface was down as the regular 15 second poll will get the last measurement. When pressed, the last fourteen days of data will be fetched. Any data that is restored is not subject to validation, nor will it trigger a derived variable that may be dependent on the new data. Any new values that encountered will not be written to the lab data tags and therefore will not propagate to SQC diagnostic diagrams.

10.1.2 Local Lab Data

Local lab data is manually entered with the user-interface shown below, which is launched from the lab data pushbutton on the common console. Data may be entered manually for lab data that meets the following criteria:

- All local lab data. These are defined in *LtLocalValue*.
- Communication to the lab system is temporarily not available OR the lab data configuration override tag named “*Configuration/LabData/manuallyEntryOverride*” is True OR the user is an engineer AND the lab datum is configured to allow manual entry by the

ManualEntryPermitted attribute in the LtValue table AND the lab data is not a derived value.

Data is manually entered with the user-interface shown below, which is launched from the “Lab Data” pushbutton on the common console.

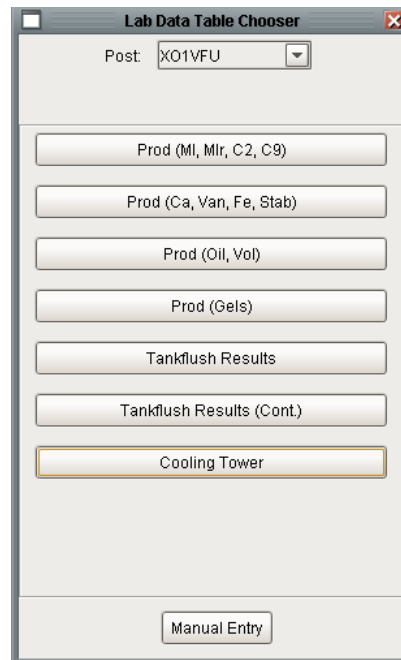
The image shows a software dialog box titled "Lab Data Table Chooser". At the top, there is a "Post:" label followed by a dropdown menu showing "X01VFU". Below this, there is a vertical stack of seven buttons: "Prod (MI, Mlr, C2, C9)", "Prod (Ca, Van, Fe, Stab)", "Prod (Oil, Vol)", "Prod (Gels)", "Tankflush Results", "Tankflush Results (Cont.)", and "Cooling Tower". The "Cooling Tower" button is highlighted with a yellow border. At the bottom of the dialog, there is a "Manual Entry" button.

Figure 22 - Lab Data Table Chooser Screen

Pressing the “Manual Entry” button displays the dialog shown below which displays a choice of all of lab measurements for the post that are currently available for manual data entry. The screen on the left shows the choices for an operator when communication is healthy and the manual override is not set. The screen on the right shows the choices when the user is an operator and communication is down or the manual override is set.

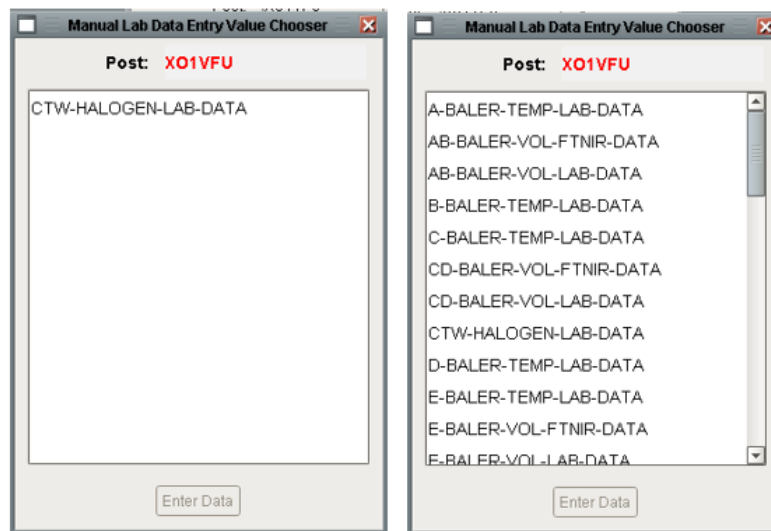


Figure 23 - Manual Lab Data Entry Value Chooser

Selecting a measurement and pressing “Enter Data” displays the screen shown below:

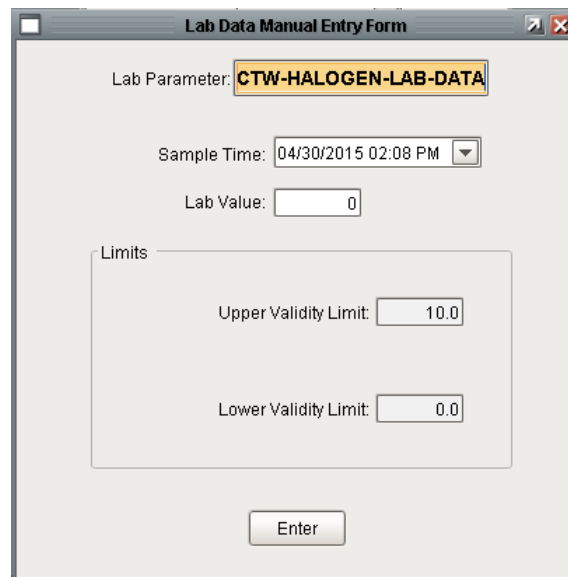


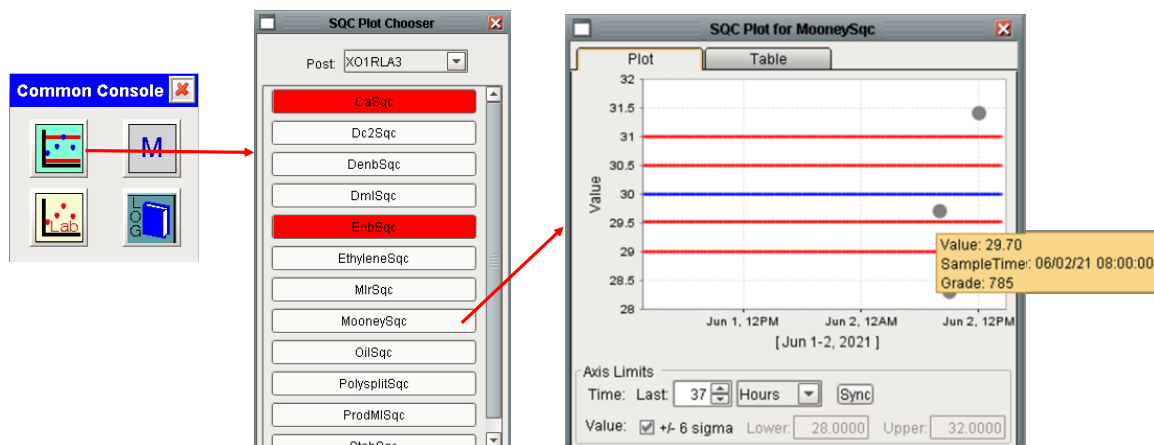
Figure 24 - Lab Data Manual Entry Screen

The sample time widget has a pulldown button that displays a nice calendar for specifying and validating the date and time. The limits container displays limits that are defined for the selected measurement. When the user presses “Enter” the value is validated using the normal validation logic. If the value fails validation the user will be notified via a pop up error message, but not the loud workspace which is used for automated lab data. Once the value passes validation it is stored to the local lab data history system. If the lab data is a “local” value then it is written to PHD via a normal OPC tag write.

10.1.3 SQC Plot Screen

The SQC plot screen combines lab data with diagnostic diagrams. It shows an SQC plot of lab data that is used by SQC blocks in a diagnostic diagram. Not all lab data are used in a diagnostic diagram involving SQC so not all lab data is viewable in this manner.

The following figure shows the screens involved in the SQC plot user interface.



The SQC plot user interface is accessed from the upper left button on the common console. It launches the SQC Plot Chooser screen. The chooser shows all of the SQC diagnosis for the user's post. SQC diagnosis for other posts are available by choosing a different post from the post dropdown.

The list of SQC diagnosis used to populate the list comes from the DtSQCDiagnosis table. The plot chooser looks a lot like the lab data table chooser described in section 10.1.1 but they are totally independent. This table is populated automatically when a SQC diagnosis block is saved in Designer. The list is animated to show SQC diagnosis that are currently active in red. Selecting a SQC plot will open the SQC plot window. The labels of the buttons in the list are configurable to show the name of the SQC Diagnosis, the name of the lab value that feeds the SQC diagnosis, or the description of the lab value that feeds the SQC diagnosis. This is configured as described in section 3.3

The SQC plot window is passed the name of the SQC diagnosis and the internal id of the SQC diagnosis in the block language toolkit. Using the block id, the utility queries the block language toolkit to determine the target, standard deviation, SQC limits, violated rules, and source of the lab data. Specifically, the utility collects all of the SQC Observation blocks on the diagram and extracts the target, standard deviation, limit type (high or low), the number of standard deviations from the target, and the state. All of the observation blocks on the diagram will have the same target and standard deviation. The target is shown as the thick blue line in the plot. The limits are reflected in the red lines. Limits that are less than 0.5 standard deviations from the target are not shown. At most two upper and two lower limits will be shown on the plot – this is the typical configuration. The blocks that are currently active are listed in the violated rules list in the upper right. Neither the SQC diagnosis nor the SQC

limit observation blocks contain the name of the lab data measurement used in the diagram. The only block on the diagram that contains the name of the lab data measurement used in the diagram is the Lab Data Entry block. Every diagram that contains a SQC diagnosis must also contain exactly one Lab Data Entry block upstream of the diagnosis.

The main component on the window is the EZ Chart. An EZ Chart was used rather than a simple chart because of the ease of drawing the blue and red lines using the calculated pens feature. The data for the chart are fetched from the database view named LtValueView. By querying the database for the values, the plots are not subject to losing data when the system is restarted. The chart is a real-time plot meaning that it updates periodically (every minute). The amount of time shown on the time axis is configurable from the widgets at the bottom of the chart. A popup tooltip is provided when you hover the mouse over a data point. The tooltip shows the time, value, and grade of the point.

10.2 Administrator User Interface

This section describes the user interface for configuring lab data instances. There are a number of screens for creating and configuring lab data which are accessed from the main Admin menu shown below:

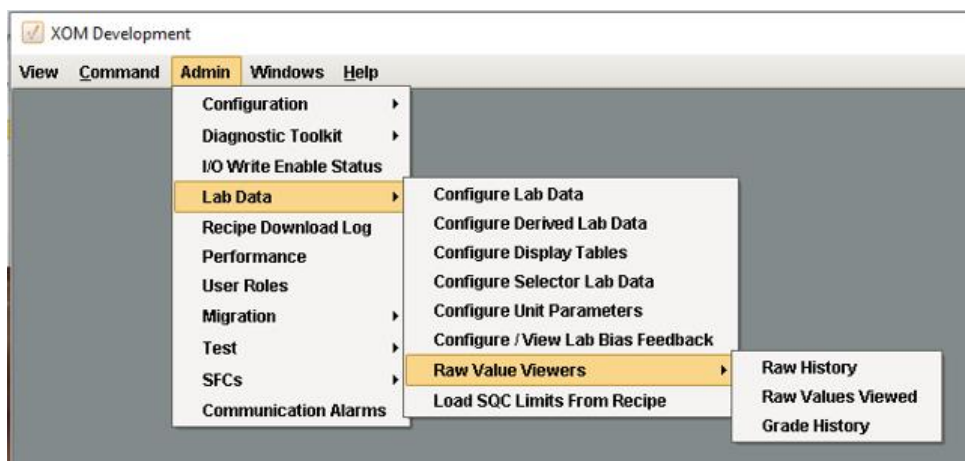


Figure 25 - Lab Data Administration Configuration

10.2.1 Configure Lab Data

This window is used to configure lab data that comes from PHD, the DCS, or locally entered. The window shows lab data by unit and contains tabs for each different source. Lab data can be added (+ button), deleted (x button), validated (check button), and values viewed (magnifying glass button). The table can also be edited to alter the configuration.

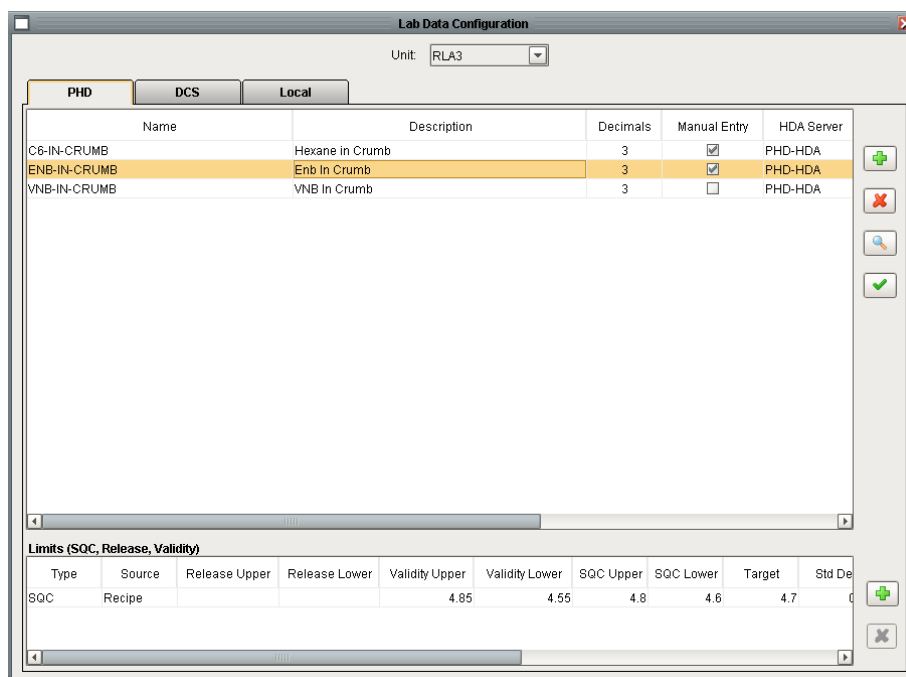


Figure 26 - Lab Data Configuration User Interface

When a row is selected, the appropriate limits are displayed in the table at the bottom. Limits can also be added, deleted or edited.

To operate correctly, a lab datum requires data in the database and tags in Ignition. Everything that is required is created automatically from this window. If a tag should inadvertently be deleted manually in the Designer, the validate button will recreate the tags.

Pressing the “+” button at the top will open the appropriate Lab Data popup window based on the tab that is selected.

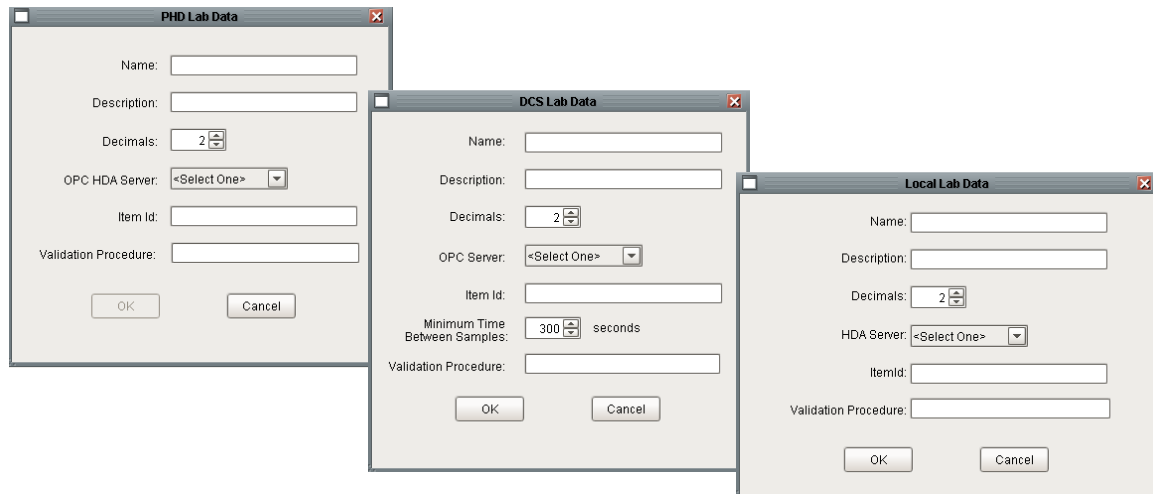


Figure 27 - Lab Data Popup Window

Pressing OK will perform all of the configuration that is needed to fully implement the new lab data item. The necessary database records and UDT instances will be created. Acquisition of the new lab data will begin immediately, a project save or reset is not necessary. If the new DCS lab datum is created, in addition to the Lab Data UDT, a plain OPC tag will be created in the Lab Data/*unitName*/DCS-Lab-Values folder. The tag is for debug purposes only. The OPC tag is not used during processing. The three popups are only available when creating a new lab datum. Changing an existing lab datum can be done by editing the cells in the table. The database and UDTs will be updated appropriately.

10.2.2 Configure Derived Lab Data

This window is used to configure derived lab data (discussed in detail in section 8) including the trigger value, related data that is used in the calculation, the destination and the OPC tag where the result will be written. Derived lab data can be added (+

button), deleted (x button), validated (check button), and values viewed (magnifying glass button). The table can also be edited to alter the configuration

Value Name	Description	Trigger Name	Decimals	Call
GLUCOSE_AVG	Glucose average	GLUCOSE_LAB	4	ils.test.labData.glucos

Value Name	Description
GLUCOSE_DCS	An online glucose measurement
GLUCOSE_LOCAL	A local glucose value

Type	Source	Release Upper	Release Lower	Validity Upper	Validity Lower	SQC Upper	SQC Lower	Target	Std Dev	Recipe Key
SQC	Constant			14	0	10	0	6.25	2.01	

Figure 28 - Derived Lab Data Configuration

When a row is selected, the appropriate limits are displayed in the table at the bottom. Limits can also be added, deleted or edited.

To operate correctly, a lab datum requires data in the database and tags in Ignition. Everything that is required is created automatically from this window. If a tag should inadvertently be deleted manually in the Designer, the validate button will recreate the tags.

Derived Lab Data Popup

Name:

Description:

Trigger Name:

Decimals:

Callback:

Time Tolerance: (min)

Wait Time: (min)

OPC HDA Server:

Item Id:

OK Cancel

The meaning of the individual fields is described in section 8. Pressing OK will perform all of the configuration that is needed to fully implement the new derived lab data item. The necessary database records and UDT instances will be created. Acquisition of the new lab data will begin immediately, a project save or reset is not necessary.

Related Data:

Value Name	Description
GLUCOSE_DCS	An online glucose measurement
GLUCOSE_LOCAL	A local glucose value

Related Data:

Value Name	Description
GLUCOSE_DCS	An online glucose measurement
GLUCOSE_LOCAL	A local glucose value
GLUCOSE_DCS	
GLUCOSE_AVG	
GLUCOSE_DCS	
GLUCOSE_LAB	
GLUCOSE_LOCAL	

10.2.3 Configuring Lab Limits

Limits may be added to regular and derived lab data. A new limit can be created by selecting a lab value in the top table and pressing the “+” button adjacent to the “Limits” table. The Lab Limit popup window will be displayed:

The screenshot shows a window titled "New Lab Limit Popup". At the top, there is a "Lab Value" field containing "GLUCOSE_LAB". Below it are two dropdown menus: "Type" and "Source", both currently showing "<Select One>". The window is divided into two main sections for limits: "Upper Limits" and "Lower Limits". Each section contains three input fields: "Release", "Validity", and "SQC". In the "Upper Limits" section, the "SQC" field is also labeled "Target". Below these sections are two buttons: "Create New Limit" and "Cancel".

Figure 30 - New Lab Limit Popup

The name of the lab datum selected in the upper table will be displayed in the “Lab Value” field and cannot be changed. The first step is to select the limit type from the combo box choices: SQC, Validity, and Release. See section 5 for a complete discussion about the different limit types. The limit fields will become enabled based on the selected limit type. Recall that SQC limits always include validity limits, a target value and a standard deviation. If a SQC limit type is selected, the following fields will be enabled:

This screenshot shows the same "New Lab Limit Popup" window, but with the "Type" dropdown menu set to "SQC". The "Lab Value" remains "GLUCOSE_LAB" and "Source" remains "<Select One>". The "Upper Limits" section still has "Release", "Validity", and "SQC" fields. The "Lower Limits" section now has "SQC", "Validity", and "Release" fields. The "Create New Limit" and "Cancel" buttons are still at the bottom.

Figure 31 - Lab Limit Popup for an SQC Limit

The next step is to select a source from the combo box choices: Recipe, DCS, and Constant. Depending on the choice, additional widgets will become visible:

The image displays three overlapping 'New Lab Limit Popup' windows, each showing a different source configuration for a lab limit. All windows have a 'Lab Value' field set to 'GLUCOSE_LAB' and a 'Type' dropdown set to 'SQC'.

- Top Left Window (Source: Recipe):** Shows a 'Recipe Key' dropdown set to '<Select One>' and a 'Read limits from Recipe' button. The 'Upper Limits' section includes Release, Validity, and SQC fields, all set to 0. The 'Lower Limits' section includes Target, Standard Deviation, SQC, Validity, and Release fields, all set to 0.
- Top Right Window (Source: Constant):** Shows a 'Calculate' button and a text instruction: 'Enter the SQC limits and then press Calculate to calculate the validity limits, target, and standard deviation.' The 'Upper Limits' and 'Lower Limits' sections are identical to the Recipe source window.
- Bottom Window (Source: DCS):** Shows an 'OPC Server' dropdown set to '<Select One>', 'Upper Limit Item Id' and 'Lower Limit Item Id' text fields, and a 'Read limits from DCS' button. The 'Upper Limits' and 'Lower Limits' sections are identical to the other two windows.

Figure 32 - Lab Limit Popup for all Three Sources

The limit fields on the right can be edited for any of the limit types, but depending on the source, they may also be loaded from the source. If the source is “Constant” and the type is “SQC” then the validity limits, standard deviation and target value can be calculated after the SQC limits are entered by pressing the “Calculate” button. The calculation assumes 6 standard deviations between the SQC limits and 9 standard deviations between the validity limits. The number of standard deviations can be customized as described in section 3.3.

Pressing “Create New Limit” will create the required UDT and configure the required database tables.

10.2.4 Configuring Display Tables

This window is used to configure the operator user interface for displaying lab data (see the screens labelled #2 and #3 Figure 21). Window #2 is the Lab Data Table Chooser. Window #3 is a Lab Data Table. The purpose of the display tables is to display lab data in logical groups and to only expose data that is useful to the operator.

The window shown below is used to define the data tables or logical groups and to specify the measurements in the table. The top table displays all of the Lab Data tables for the post selected in the dropdown. There is no limit to the number of Lab Data tables that can be defined. New windows are created by pressing the green plus button to the right of the top table. The order that window names will be displayed on the user's menu can be controlled by selecting a window and using the up or down

arrows. A screen is deleted using the red “X” button. The “Page” column in the top table can be used to add tabs to the Lab Table chooser. This can be used to provide another level of organization and a way to create a chooser where the list does not have scroll bars.

The bottom table shows the lab values that will be displayed in the Lab Data table selected in the top table. Only existing lab data may be added to a window, new lab data cannot be created from this window. Lab data from any source (PHD, DCS, LOCAL, derived, and lab selectors) can be added to a lab table.

Lab Data Display Table Configuration

Post:

X01TEST

Lab Data Table Title	Page	Display
Foo	1	<input checked="" type="checkbox"/>
Glucose	1	<input checked="" type="checkbox"/>
Sugar	1	<input checked="" type="checkbox"/>
Fructose	1	<input checked="" type="checkbox"/>
Cake	2	<input checked="" type="checkbox"/>
Frosting	2	<input checked="" type="checkbox"/>

Lab Datum Name	Description
GLUCOSE_AVG	Glucose average
GLUCOSE_LAB	Glucose lab value
GLUCOSE_LOCAL	A local glucose value
GLUCOSE_DCS	An online glucose measurement

Figure 33 - Lab Data Display Table Configuration

10.2.5 Configure Selector Lab Data

This window is used to configure selectors. Refer to section 6 for details about Lab Selectors.

Unit: VFU

ValueName	Description	Decimals	Validity Limit	SQC Limit	Release Limit
SPEC-C2-LAB-DATA	Rx-1 C2	2	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
SPEC-C9-LAB-DATA	Rx-1 Entb/Vnb	2	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

OK Apply Cancel

Figure 34 - Lab Data Selector Configuration

A new lab selector can be created by pressing the “+” button which posts the popup shown below:

Name: C2-LAB-DATA

Description: C2 Analyzer Results

Decimals: 2

☐ Validity Limit

☒ SQC Limit

☐ Release Limit

OK Cancel

Figure 35 - New Lab Selector Popup

Both of these windows update the database and the lab data UDTs. UDTs will be created or deleted for the selector and limits. As discussed in section 6, the logic that dynamically configures the selector references is not configured through the GUI.

10.2.6 Configuring Unit Parameters

This window is used to configure unit parameters.

The screenshot shows a window titled "Unit Parameter Summary". It contains a table with the following columns: Unit Parameter, Number of Points, Ignore Sample Time, Value Source, and Sample Time Source. The table lists four parameters related to BALER-FILTERED-VALUE, BALER-VOL-FTNIR-FILTERED, and BALER-VOL-LAB-FILTERED. Below the main table is a summary table with columns: Index, Value, Sample Time, and Receipt Time. The summary table shows two rows of data.

Unit Parameter	Number of Points	Ignore Sample Time	Value Source	Sample Time Source
[XOM]LabData\FU\A-BALER-FILTERED-VALUE	2	<input type="checkbox"/>	([] ./A-BALER-TEMP-LAB-DATA/value)	([] ./A-BALER-TEMP-LAB-DATA/sampleTime)
[XOM]LabData\FU\AB-BALER-VOL-FTNIR-FILTERED	2	<input type="checkbox"/>	([] ./AB-BALER-VOL-FTNIR-DATA/value)	([] ./AB-BALER-VOL-FTNIR-DATA/sampleTime)
[XOM]LabData\FU\AB-BALER-VOL-LAB-FILTERED	2	<input type="checkbox"/>	([] ./AB-BALER-VOL-LAB-DATA/value)	([] ./AB-BALER-VOL-LAB-DATA/sampleTime)
[XOM]LabData\FU\CD-BALER-VOL-LAB-FILTERED	2	<input type="checkbox"/>	([] ./CD-BALER-VOL-LAB-DATA/value)	([] ./CD-BALER-VOL-LAB-DATA/sampleTime)

Index	Value	Sample Time	Receipt Time
0	0.0000	12/29/19 10:52:00 AM	12/29/19 10:53:17 AM
1	0.0000	12/16/19 11:15:00 AM	12/17/19 2:36:53 PM

Figure 36 - Unit Parameter Configuration

10.2.7 Raw Value Viewers

There are three windows provided to view raw values. The purpose of these windows is to help troubleshoot the system. They provide a convenient way to view the data in the data base. Windows are provided to view:

- Raw lab values by time.
- Viewed value history.
- Grade history by unit.

10.2.8 Load SQC Limits from Recipe

This window is provided to recover from the situation where SQC limits did not get downloaded from recipe for whatever reason. Normally SQC limits will be loaded automatically but in the event where they didn't then this will allow the system to recover. This can also be used if a correction was made to the SQC limits in DB Manager after the grade change and the updated limits need to be loaded. This will

update the current limits in the LtLimit table, which are read by the Lab Data scanner every cycle. It will also update the lab data UDTs.

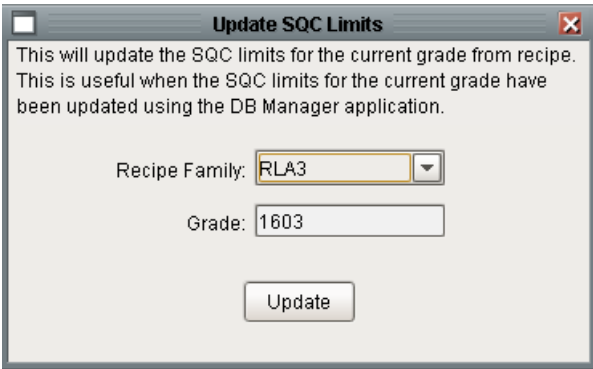


Figure 37 - Update SQC Limits Window

11 Unit Parameter Classes

Unit parameters are a general-purpose class / UDT but since a typical lab data value may propagate to a unit parameter it is implemented as part of the lab data module. Unit parameters are also referred to as a filtered parameter. The unit parameter is configured with a sample size, n . Its value is calculated by averaging the last n values.

Unit Parameters are implemented in the UDT shown below. The UDT contains the buffer of values used to calculate the filtered value. Unit parameters may be created manually in the designer or by using the client discussed in section 11.3.

As of Ignition 8.x, there is nothing in the database related to Unit Parameters.

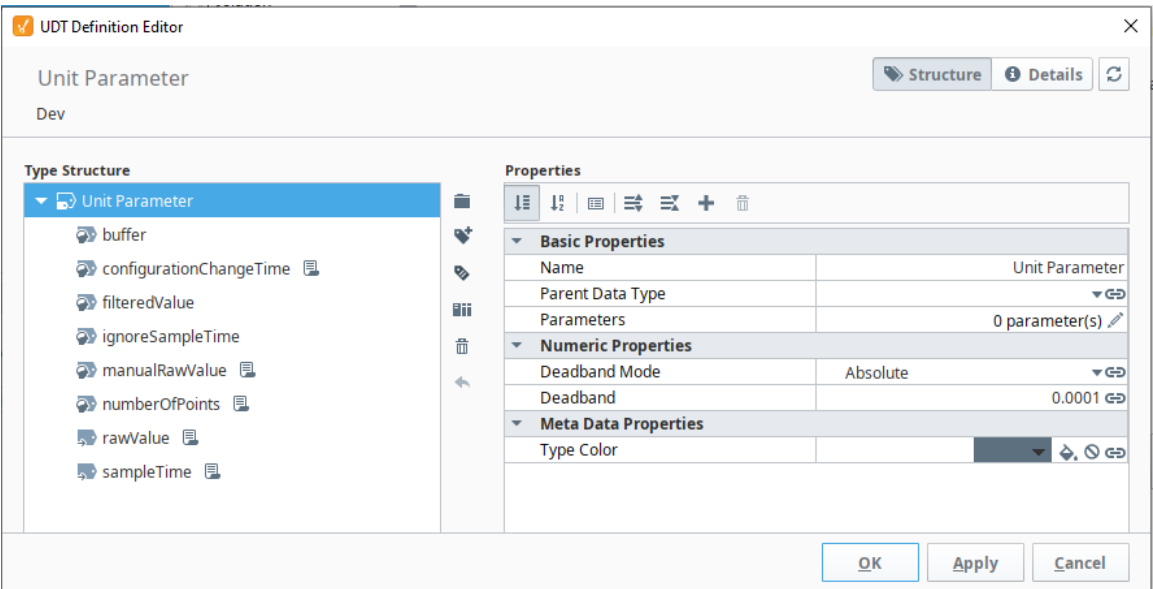


Figure 38 - Unit Parameter UDT

The *numberOfPoints* tag defines the size of the buffer used to calculate the mean. The buffer is maintained in the database. Changes to the *numberOfPoints* value causes the buffer to automatically resize.

The *buffer* tag is a dataset that contains the most recent number of points. The most recent value is always the first element in the dataset.

The *configurationChangeTime* tag is a simple memory tag that records the time of the most recent unit configuration change. Some outside logic must update this tag in every Unit Parameter that is impacted by the configuration change. There is a change script that fires when the value changes.

The *filteredValue* tag is a simple memory tag that reflects the average of all of the values in the buffer.

The *manualRawValue* tag is a simple memory tag that reflects the average of all of the values in the buffer. This can be used from Designer or from a custom window or logic. It is appropriate for values that do not originate from lab data and where a sample time other than the current time is not required.

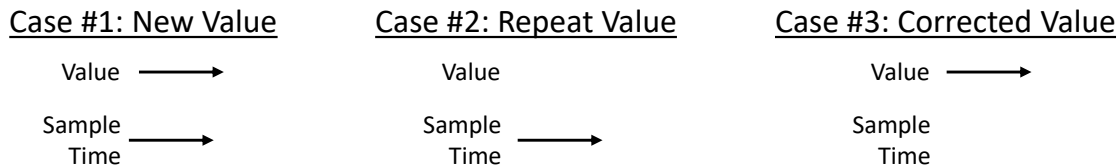
The *rawValue* and *sampleTime* tags are reference tags that typically refer to a lab data UDT. There is a change script on both of these tags that initiate processing and ultimately place a new value into the buffer and recalculate the filtered value.

A Unit parameter has two modes of operation, automatic or manual. In automatic mode, the *rawValue* reference tag typically references a tag such as a lab datum or OPC tag. In manual mode, something writes a value into the *manualRawValue* tag which is a simple memory tag. Both the *rawValue* and the *manualRawValue* tags trigger the same calculation and the filtered value is written to the *filteredValue* tag. Typically, the source of the unit parameter is either the *rawValue* or the *manualRawValue*, but not both, although nothing prevents this.

11.1 Repeat Sample Value Handling

When a Unit Parameter is used with a Lab Data value input, special consideration must be given to handle repeat sample values. When used with Lab Data, the *sampleTime* expression tag should be configured to reference the *sampleTime* of the lab data UDT. The UDT has a value changed script on both the raw value and on the

sample time. When configured in this manner there are three possible operating scenarios as shown below:



The first case is when a new different value with a new sample time is received. The second case is when two identical values are received, in this case, only the sample time is updated. The third case is when a lab value is updated, in this scenario, only the value is updated and the sample time remains unchanged.

In the figure above, each arrow represents a processing thread. In the first case there are two threads and the algorithm needs to be careful to prevent the value being placed into the value buffer twice. It is assumed that the *rawValue* and *sampleTime* are received at nearly the same time. The algorithm checks that the value and *sampleTime* were received at nearly the same time. If they were not, then it will wait a configurable dwell time. If only one of the two was updated then we are operating in case 2 or 3. Each thread reads both the raw value and the sample time and the time that those values were received as reflected in the *LastChange* attribute of these tags. If both threads detect that both the *rawValue* and the *sampleTime* have been updated, then the *sampleTime* thread exists, thereby preventing duplicating the value in the buffer.

Note: A unit procedure can operate in automatic mode with just the *rawValue* expression tag configured. The *sampleTime* tag is optional although it is necessary if consecutive identical lab datums are expected.

11.2 Data Consistency Due to Grade Change or State Change

Unit Parameters are often used with Lab Data and Lab Data may have a significant delay between the time the lab sample is collected and when the lab sample is reported. It is important that data collected before a state change but reported after the state change should be ignored. There are two aspects to this: clearing the buffer of data that has already been collected and preventing data yet to be reported but collected before the state change from being added to the data buffer. This is automatically accomplished by updating the *configurationChangeTime* tag of the Unit

Parameter UDT. This is implemented in the *configurationChanged()* function in *ils.labData.unitParameter.py*

11.3 Client User Interface

A client user interface is available from the menu shown below:

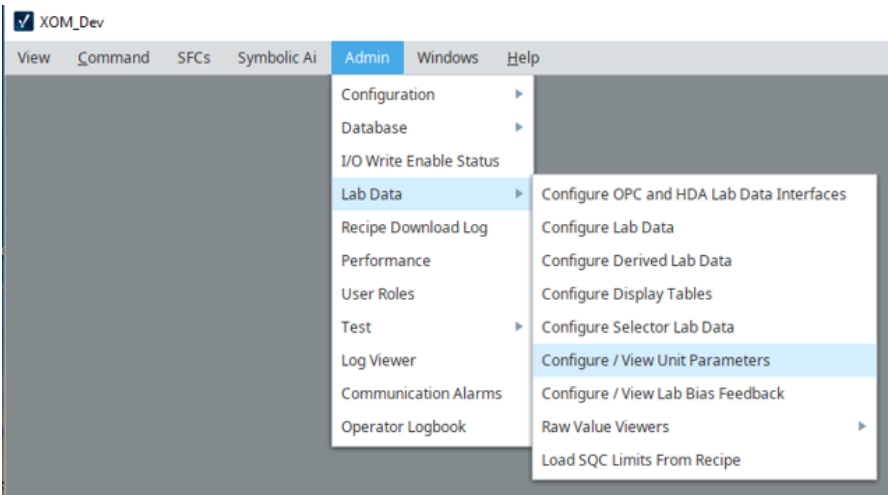


Figure 39 - Menu for Accessing the Client User Interface

Selecting “Configure / View Unit Parameters” opens the window shown below:

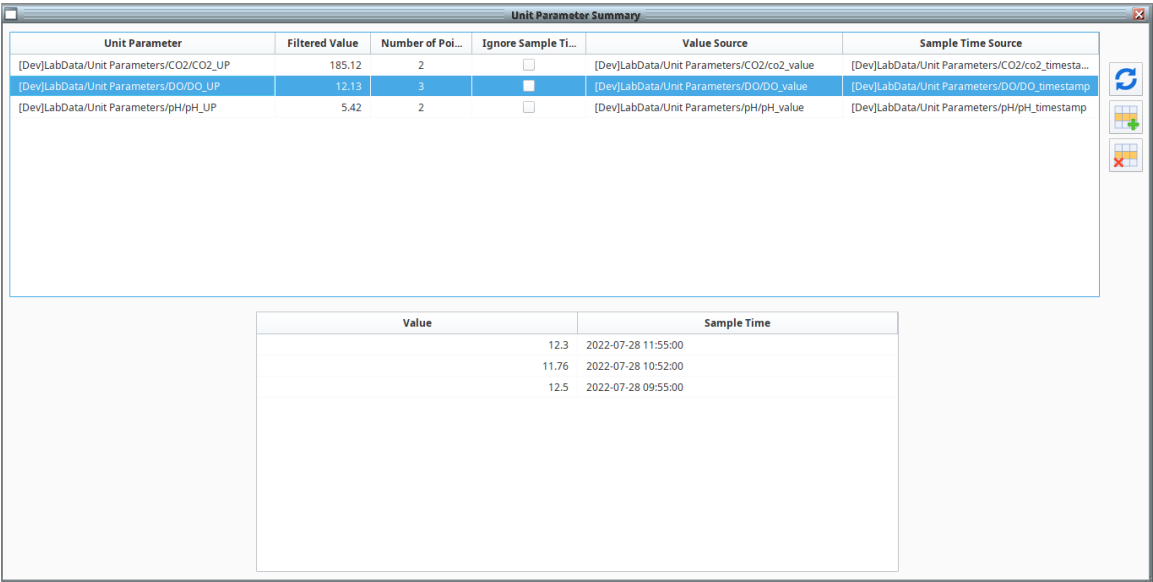


Figure 40 - Unit Parameter Client User Interface

12 Lab Data Troubleshooting

The new toolkit is significantly different than the old toolkit and the troubleshooting strategy is also quite different.

12.1 OPCHDA Status

The status of the OPC-HAD interface can be monitored from the gateway web page using the screen shown in Figure 6.

12.2 Message Queue

There is a message queue with key "LABDATA", which is titled "Lab Data Messages" dedicated to Lab Data messages. Messages are inserted when lab data that fails validity or release limits checks and the disposition of the check, either accepted or rejected.

12.3 Loggers

Loggers are used to provide detailed logging information. Loggers are most useful for activity that executes in the gateway. Therefore the logger levels need to be set on the gateway and viewed either in the wrapper logfile or on the gateway console page. The following loggers are provided:

- com.ils.labData
- com.ils.labData.selector
- com.ils.labData.SQL

13 Lab Feedback Control – Bias Calculations

13.1 Theory

Bias calculations compares two values to create a bias to adjust a model based upon on a value that is assumed measured accurately. For example, a model calculates a value that is infrequently measured. A bias adjusts the calculated model value based upon the measured value. A simple bias is given by:

$$\text{Bias} = \text{Lab Value} - \text{Model Value}$$

The calculated value then becomes:

$$\text{Calculated Lab Value} = \text{Model Value} + \text{Bias}$$

There are different approaches to updating the bias. The above model is simply based upon the last measured lab value. This approach may cause issues if lab value changes dramatically from the previous update. In addition, statistically this may not be the proper approach.

13.1.1 Exponential Bias

Another approach is to apply an exponential filter to the bias calculation. The exponential bias calculation is:

$$\text{Raw Bias} = \text{Lab Value} - \text{Model Value}$$

$$\text{Bias} = \text{filter constant} * \text{Raw Bias} + (1 - \text{filter constant}) * \text{Last Bias}$$

13.1.2 PID Bias

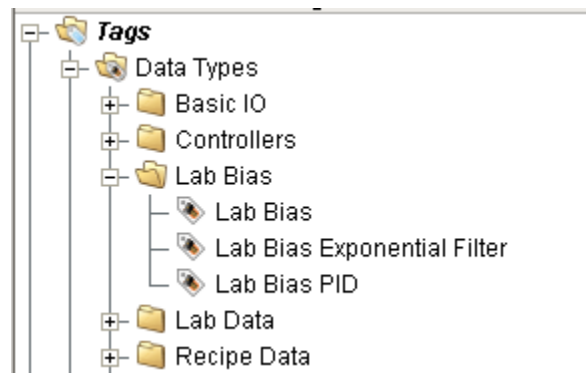
A second approach is to apply a PID filter to the bias calculation. The PID bias calculation is:

$$\text{Raw Bias} = \text{Lab Value} - \text{Model Value}$$

$$\begin{aligned} \text{Bias} = & \text{proportional gain} * (\text{new lab value} - \text{previous lab value}) + \\ & \text{Integral gain} * \text{sample time} * \text{new lab value} + \text{Last Bias} \end{aligned}$$

13.2 UDTs

The lab bias is implemented entirely in UDT tag instances, there is nothing in the database related to lab bias feedback. The calculation is entirely event based, there are no timer scripts that periodically update the bias. There are three UDTs that implement the bias calculations which are shown below.



Lab Bias is the base definition that should not be instantiated. It defines properties that are common to both bias calculation strategies. Its definition is shown below:

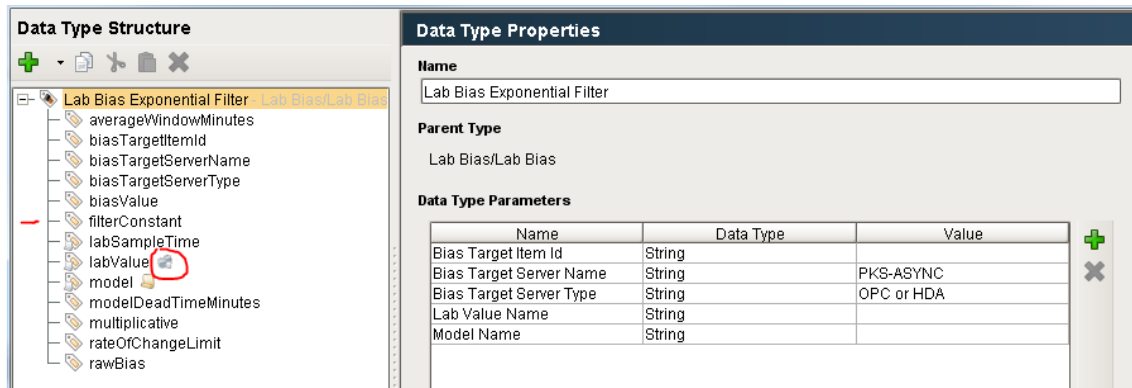
The screenshot shows two windows from a software interface. The 'Data Type Structure' window on the left displays a tree view with 'Lab Bias' selected, showing its members: averageWindowMinutes, biasTargetItemId, biasTargetServerName, biasTargetServerType, biasValue, labSampleTime, labValue, model, modelDeadTimeMinutes, multiplicative, rateOfChangeLimit, and rawBias. The 'Data Type Properties' window on the right shows the configuration for 'Lab Bias'. It has a 'Name' field set to 'Lab Bias' and a 'Parent Type' field. Below these is a 'Data Type Parameters' table with columns for Name, Data Type, and Value.

Name	Data Type	Value
Bias Target Item Id	String	
Bias Target Server Name	String	PKS-ASync
Bias Target Server Type	String	OPC or HDA
Lab Value Name	String	
Model Name	String	

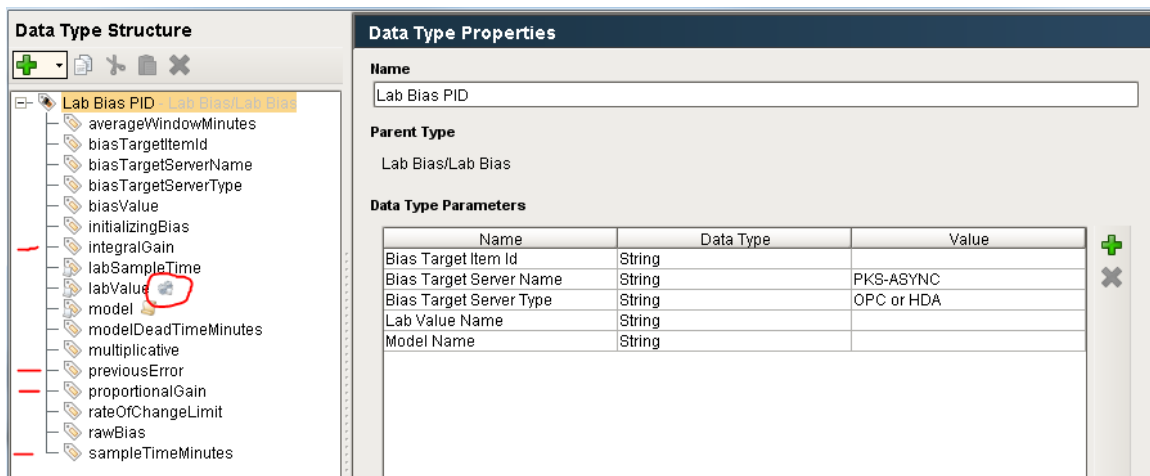
The base UDT defines the following properties which are used in the implementation of the bias.

Parameter	Description
Bias Target Item Id	This specifies the OPC item-id where the calculated bias will be written. This is referenced by the member tag biasTargetItemId.
Bias Target Server Name	This specifies the OPC server where the calculated bias will be written. This is referenced by the member tag biasTargetServerName.
Bias Target Server Type	This specifies the type of the OPC server where the calculated bias will be written. The legal values are OPC or HDA. This is referenced by the member tag biasTargetServerType.
Lab Value Name	This specifies the lab data value that is the driving source of the bias calculation. This is linked to the labValue and labSampleTime tags of the UDT.
Model Name	This specifies the tag that provides the model value that will be used in the bias calculation.

The exponential filter bias adds a filter constant tag; a script on the lab value tag, but no additional properties:



The PID bias UDT adds tags *integralGain*, *proportionalGain*, and *sampleTimeMinutes*. These are constants. It also adds a script on the *labValue* tag.



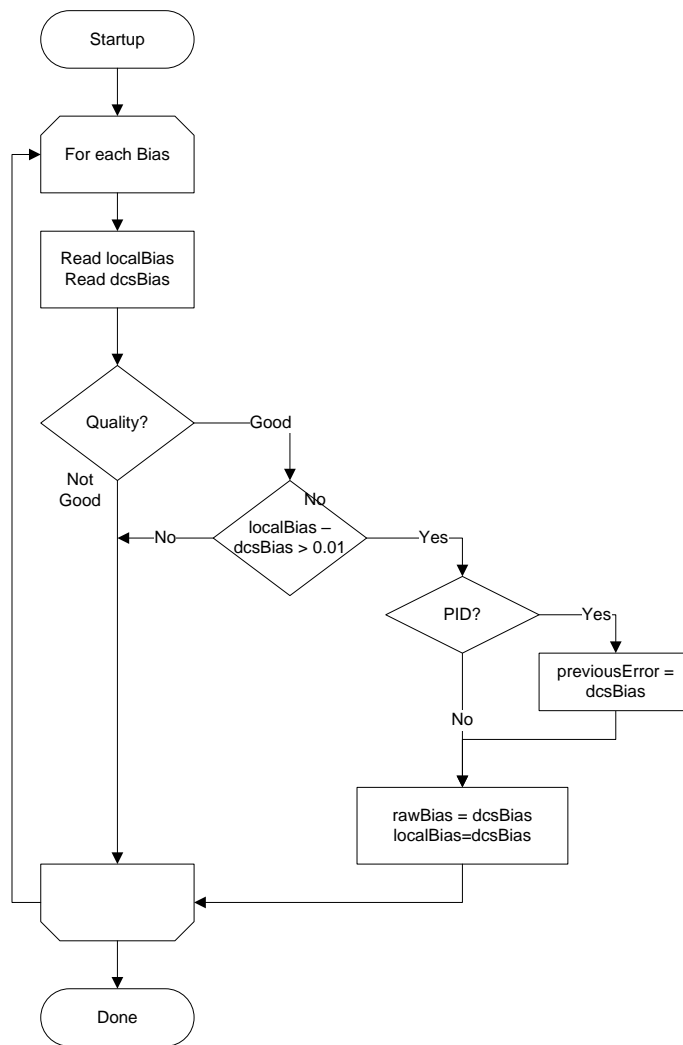
Note: The reason that there are memory tags that shadow the UDT properties is that properties are not accessible from scripting; therefore a memory tag is created that references the property. The reason that some configuration is done through the properties and others directly through the tags is that numeric tags cannot reference UDT properties.

13.3 External Python

The Python that implements the Lab Feedback handling is in the *ils.labFeedback* package. The logic is initiated when the *labValue* tag receives a new value. The logic executes in the gateway.

13.4 Initialization

The initialization of lab bias is specified in the flowchart below. It is implemented in *ils.labFeedback.startup.gateway* which is called by the site specific gateway start-up script.

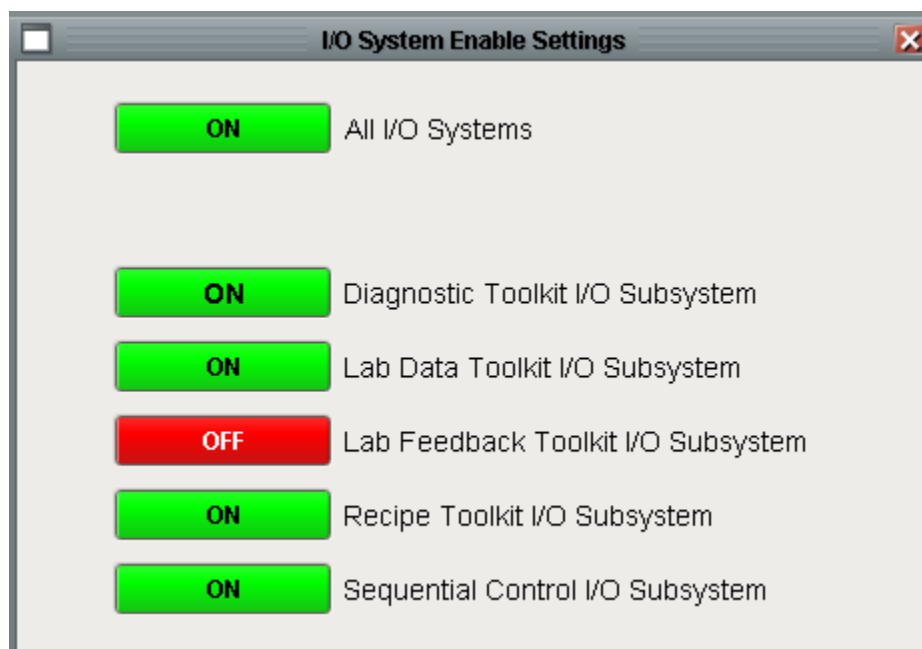


13.5 Inhibiting I/O

The lab bias calculations run automatically once the UDTs are configured. The calculation is triggered when a new lab value is received. If a new bias value is calculated it is immediately written to the external system as long as writing I/O has not been inhibited. I/O must be enabled globally and for the Lab Feedback toolkit. This is implemented using configuration memory tags as shown below:

Tag	Value	Data Type
<div> <div>Tags</div> <div> <div>Data Types</div> <div>Configuration</div> <div>Common</div> <div> <div>automatedGradeChangeHandlingEnabled</div> <div>historyTagProvider</div> <div>writeEnabled</div> </div> <div>Diagnostic Toolkit</div> <div>Email</div> <div>LabData</div> <div>LabFeedback</div> <div>labFeedbackWriteEnabled</div> <div>Recipe Toolkit</div> <div>SFC</div> <div>UIR</div> <div>Diagnostic Toolkit</div> </div> </div>	<div> <input checked="" type="checkbox"/> </div> <div> XOMHistory </div> <div> <input checked="" type="checkbox"/> </div> <div> </div> <div> <input type="checkbox"/> </div>	<div>Boolean</div> <div>String</div> <div>Boolean</div> <div></div> <div></div> <div>Boolean</div>

The I/O can also be inhibited in a client from the Admin menu as shown below:

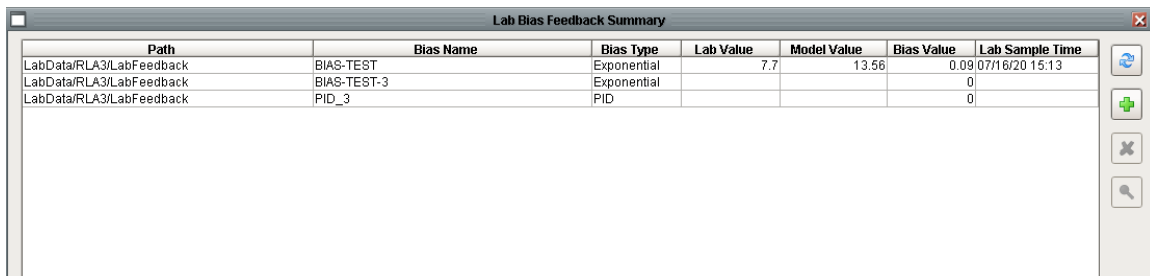


13.6 User Interface

The user interface consists of two windows.

13.6.1 Bias Summary User Interface

Configuring a lab bias object is done through an Ignition client using the **Admin -> Lab Data -> Configure/View Lab Bias Feedback** window.



Path	Bias Name	Bias Type	Lab Value	Model Value	Bias Value	Lab Sample Time
LabData/RLA3/LabFeedback	BIAS-TEST	Exponential	7.7	13.56	0.09	07/16/20 15:13
LabData/RLA3/LabFeedback	BIAS-TEST-3	Exponential			0	
LabData/RLA3/LabFeedback	PID_3	PID			0	

This window is used to view the lab bias summary. The window shows every lab bias UDT. This window is built dynamically by browsing for lab bias UDTs. The tag browse system call is slow when searching over all tags; therefore it is constrained to only look for these UDTs in the LabData folder. It displays the latest values and the time it was last updated. The window also contains a series of buttons along the top. In order from top to bottom, the buttons are:

- Refresh – Refresh the window.
- Add – Create a new lab bias, opens the configuration window with empty fields.
- Delete – Delete the selected bias.
- View Details / Edit – Edit the selected bias using the configuration window.

13.6.2 Bias Configuration User Interface

A lab bias can be created and configured using the user interface described here or it can be done entirely using the Designer and directly creating and editing the UDTs described above. The User Interface does contain some validation to ensure that the configuration is valid. The bias configuration window is shown below. The same window is used for both Exponential and PID bias types. For an existing bias, the bias type cannot be changed, nor can the unit. The “Unit” and “Bias Type” combo boxes are enabled when creating a new bias.

Pressing OK will update an existing UDT or create a new UDT. There is nothing in the database related to the lab bias feedback tags.

13.7 Troubleshooting

The first step to troubleshooting is to observe the behaviour of the UDT in the Designer. All of the processing occurs in gateway scope. Detailed diagnostics can be obtained by putting the *com.ils.labFeedback* logger into trace mode. This will turn on detailed logging information that is written to the *wrapper.log* file. The bias calculation is initiated from the *ValueChanged* tag event script on the *labValue* tag.