

Endeavor Engineering Inc.
19360 NW Melrose Drive
Portland, OR 97229
503.706.6913
www.endeavoreng.com
twitter: @endeavoreng



National Instruments IoT Technologies

Examples Documentation

09.26.2017

Contact: James Tillett
Mobile: +1 503.706.6913
Email: jtillett@endeavoreng.com
LinkedIn: www.linkedin.com/in/jetillett



CONTENTS

Overview.....	3
SystemLink.....	3
LabVIEW Cloud Toolkit for AWS.....	4
LabVIEW WebVI.....	4
This Demonstration Example	4
How It Works.....	5
Example Architecture Diagram.....	6
Larger Application Example.....	6
Larger Example Architecture Diagram.....	7
The example Use Case	7
Working With It.....	8
Why all the Orange Controls?	8
The LabVIEW Project.....	8
Reusable Architecture.....	9
The LabVIEW SystemLink API Class Wrapper.....	10
IoT Demo User Interface (UI).....	11
Simulated Vibration Signal Function	11
Process Conveyor Stage Function.....	12
Signal Capture Function.....	12
Notification Function	13
RIO Board & Software Applications.....	13
Interactions Between RIO & UI.....	14
Amazon Web Services (AWS).....	15
Example Setup & Configuration.....	16
Setup SystemLink Server.....	17
Setup RIO Target	17
Add RIO Target to SystemLink.....	17
Setup AWS Services	18
Setup LabVIEW Development Environment.....	18
Application Startup/Operation.....	18
Where to from Here?.....	18
Conclusion	20



OVERVIEW

SYSTEMLINK

[SystemLink](#) from National Instruments (NI) is an exciting set of software components to enable remote management of and interaction with NI devices. Follow this link for the SystemLink home page.

<http://www.ni.com/en-us/landing/systems-management-software.html>

Devices or “Things” are evolving to be smaller, widely distributed, more intelligent, less expensive, and connected to each other and the rest of the world. The world of “Connected Intelligence” continues to enable new applications that improve efficiencies and create new capabilities. This has become so prevalent that it is currently known as the Internet of Things (IoT), or the Industrial Internet of Things (IIoT) to denote a more “industrial” focus.

Generally, our connected world consists of cloud services, typically in a virtual space, powered by deep computing and data manipulation capabilities. This cloud then connects and interacts with Things at the edge of the network, typically for interaction between the cyber and real worlds. The benefit derived from the cloud, Things, and their interconnected capabilities are only limited by our imagination.

The intelligent nodes in an IoT solution are based on a computing engine that requires networks, interaction with other systems, maintenance, updates/upgrades, and more.

NI has built SystemLink to address these needs for their diverse “Things” and the many connected solution variations that their customers build with them. SystemLink is built with common and open technologies such as Apache, Mongo, RabbitMQ, and others, with special sauce built for the unique needs of IoT and NI products.

At a high level SystemLink is a service, typically installed on a Windows based computer, but due to the foundational technologies chosen is easily translatable to other operating environments such as Linux. Typical devices are built with LabVIEW and are deployed to PXI, cRIO, and sbRIO based targets. SystemLink provides the following capabilities.

- ⚙ Device management, like MAX but always connected to deliver operating status and configuration
- ⚙ Software packaging and managed deployment to devices
 - Repository with “feeds” to organize and manage software packages
 - Manage software up/down grades and clones
 - Similar to using application builder for exe/rtexe files but then also delivered and managed
- ⚙ Device data interactions
 - Tags: for most recent (tag) values – read/write
 - These behave a lot like shared variables with better reliability & performance
 - Files: to move files to/from the server
 - Messages: Generic message passing



- ⚙️ Open API for extensibility
 - Example: use C# to move tag data from SystemLink to long term storage

LABVIEW CLOUD TOOLKIT FOR AWS

The LabVIEW Cloud Toolkit for AWS provides a LabVIEW API to access services on [Amazon Web Services \[AWS\]](#). The supported services are

- ⚙️ [S3](#): Simple Storage Service
 - Store data sets in “buckets”
- ⚙️ [SNS](#): Simple Notification Service
 - Capture and forward notification messages to different recipient types such as SMS and email
- ⚙️ [SQS](#): Simple Queue Service
 - Put data sets in a queue for pickup by other online services
- ⚙️ [IoT](#): Internet of Things
 - Manage, respond, and coordinate messages from “Things”

More can be found on www.ni.com on this toolkit [here](#).

LABVIEW WEBVI

With the release of [LabVIEW NXG](#) NI has created the first release of a new LabVIEW built for the connected world. When this document was created [LabVIEW NXG](#) was officially released as version 1.0 with a [LabVIEW NXG version 2.0 beta](#) available at their Technology Preview page. NXG version 2.0 introduces the LabVIEW WebVI available [on this page](#), which shows examples of WebVI. For a comparison of established LabVIEW and NXG go [here](#).

[WebVI](#) enables building simple LabVIEW code to natively run in a browser and interact with services. It is powered by an engine built on Javascript and associated modern and commonly available web technologies that run natively inside the browser. The LabVIEW programmer then uses a browser resident editor to build custom LabVIEW WebVI browser code with a focus on simple data manipulation and visualization in the browser window.

While this is just the beginning this powerful concept now enables the supported LabVIEW API to operate in any Javascript based environment, especially modern browsers. The result is that you can program a similar rich LabVIEW UI to operate in any modern browser without the complications of past solutions that forced different technologies together with mixed results.

THIS DEMONSTRATION EXAMPLE

This demonstration was built to show an example of the technologies presented in the [Overview](#) section. The driving considerations were to build an approachable example based on a real-world configuration and using available NI devices.

I started with the [LabVIEW RIO Evaluation Kit](#), which is built on an [sbRIO-9637](#) board with a specialized demonstration carrier board that brings out the I/O in interactive forms such as buttons, knobs, terminal blocks, etc., a built-in waveform generator, and a simple LCD display. A picture of the RIO board can be found in the [RIO Board & Software Applications](#) section.



The RIO Eval Kit comes with software, including a LabVIEW application example that simulates a manufacturing conveyor with triggers for signal capture and an onboard LCD status display. It also has a Windows UI program which interacts with the sbRIO target through shared variables.

This demonstration example started with the RIO Eval Kit code, enhanced it to take advantage of the IoT capabilities described in this document, and architected it to align more with a real-world production example.

The current example code was built on the following software.

- ⚙ [LabVIEW 2017](#)
 - [Real-Time module](#)
 - [FPGA module](#)
 - Application Builder
- ⚙ [SystemLink EAR release](#)
 - Includes required LabVIEW libraries
- ⚙ [LabVIEW Cloud Toolkit for AWS](#)
 - v1.0.0.3 – loaded from VI Package Manager (VIPM) through tools network
 - The basic VIPM free version installs with LabVIEW but you can find the full version from [JKI here](#).
- ⚙ LabVIEW NXG v2.0 beta
 - Through [NI Software Technology Preview](#)
 - *DevNote:* WebVI is not currently included in this release but coming soon

HOW IT WORKS

I built this example using Amazon Web Services as the hosting environment for all aspects. This includes a Windows instance to host SystemLink as well as a service called Seeq, which is not featured in this application example.

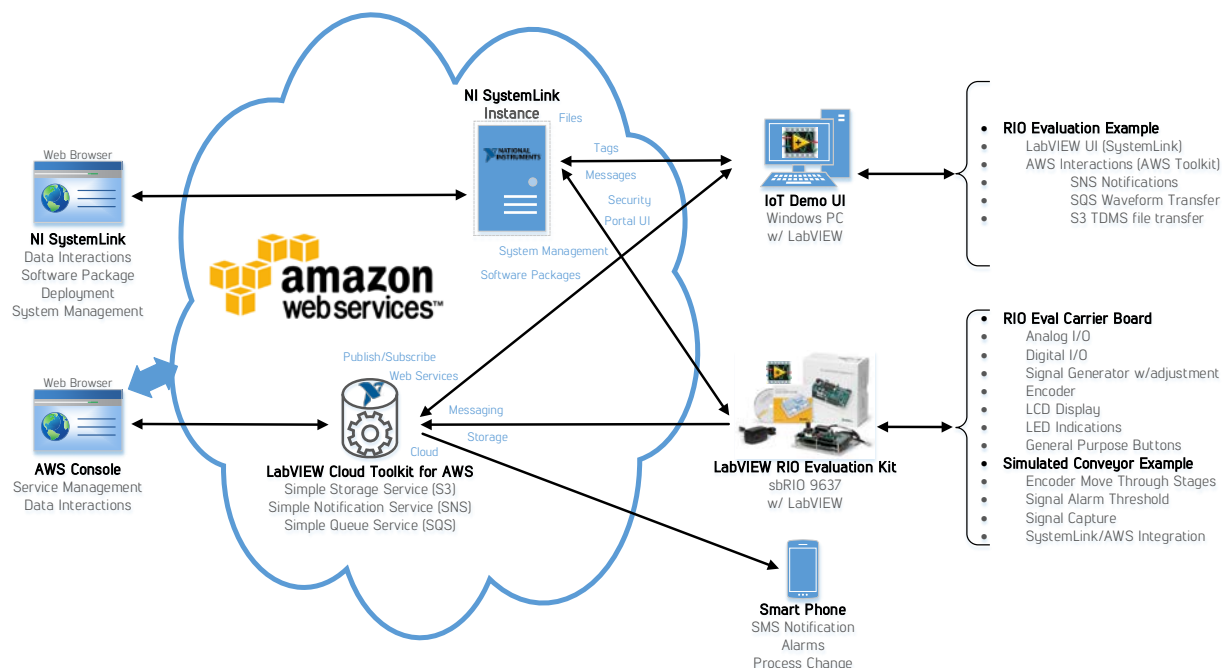
Seeq is an innovative time-series data analytics and asset management platform. It takes your time-series data, along with relevant relational data, and allows interactive analytics and collaboration with other team members. I really like it and if interested in more detail go to seeq.com.

I also enable the AWS services in this example with the [LabVIEW Cloud Toolkit for AWS](#).



EXAMPLE ARCHITECTURE DIAGRAM

Below is a diagram describing this example as I set it up with AWS. You can host your SystemLink instance on AWS or any recent Windows computer. I have tested it on Windows 7 and Windows 10.



LARGER APPLICATION EXAMPLE

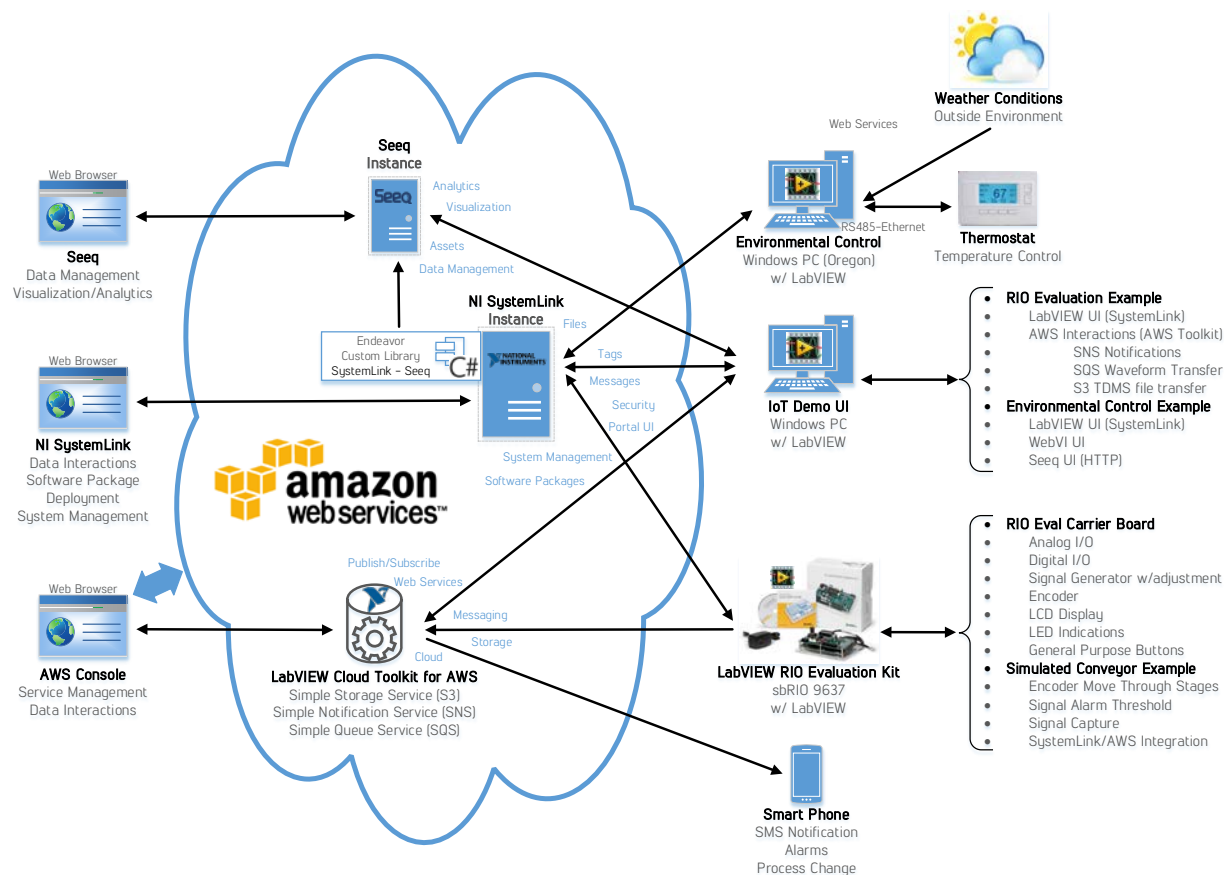
This application example is part of a larger example built to use more IoT elements. This larger example includes these additional elements.

- ⚙ Live thermostat control with analytics
- ⚙ [Seeq](#) data management/visualization/analytics and asset management platform
- ⚙ C# based connector to move tag data into Seeq data store



LARGER EXAMPLE ARCHITECTURE DIAGRAM

To understand how this example can be expanded to a larger solution, below is a diagram describing the larger example. Maybe this will give you more ideas on application potential and options for the tools described in this document.



THE EXAMPLE USE CASE

This example is designed to simulate a simple real-world use case of an industrial process controlling a conveyor that moves through process stages 1, 2, & 3. Once it moves through all 3 stages it will then shutdown gracefully by first going through a shutdown state and then end with an ESTOP state.

The RIO carrier board generates a signal waveform which is then fed into an analog input to simulate conveyor vibration. The RMS waveform signal value represents the vibration level and if the level exceeds the set alarm threshold then an alarm is raised and the system enters a shutdown.

For more detail on the RIO and software please see the [RIO Board & Software Applications](#) section.

The Windows program provides the main User Interface (UI). This simulates the HMI used with the industrial process and provides operating status as well as various settings to manage the systems operating configuration. By using the [LabVIEW Cloud Toolkit for AWS](#) along with [SystemLink](#) additional options for notification and data transfer are enabled.

For more detail on the UI please see the [IoT Demo User Interface \(UI\)](#) section.



Note: AWS interactions require setting up your own account on AWS and configuring them. While this may sound somewhat complicated, you will likely be surprised by the simplicity and easy functionality with both AWS and the [LabVIEW Cloud Toolkit for AWS](#)

For more detail on AWS please see the [Amazon Web Services \(AWS\)](#) section.

This is all enabled using LabVIEW, and various add-ons, setup in a LabVIEW project. For more detail on the LabVIEW project please see [The LabVIEW Project](#) section.

WORKING WITH IT

This section describes the objectives of the application, how it operates when using it, and basic information on programming it in LabVIEW.

WHY ALL THE ORANGE CONTROLS?

You will see a lot of orange custom controls in the source code. An example follows.





I get this question a lot so here is the answer for clarification. It can be confusing when a cluster is a typedef and I think good to be clearly identified. It seems that in my programming almost all clusters end up being typedefs to maintain consistency across Vis and over time tend to get replaced with LabVIEW classes but that is a different story. Making incorrect assumptions on whether a cluster is a typedef or not can result in unexpected behavior and debugging.

To address this, I have adopted the practice of making typedefs orange to clearly indicate that attribute. NI has partially addressed this on the wiring diagram with a mark but I think a clearer indication that could be turned on/off in options would be a great add to the control panel objects as well. *Are you listening NI?*

THE LABVIEW PROJECT

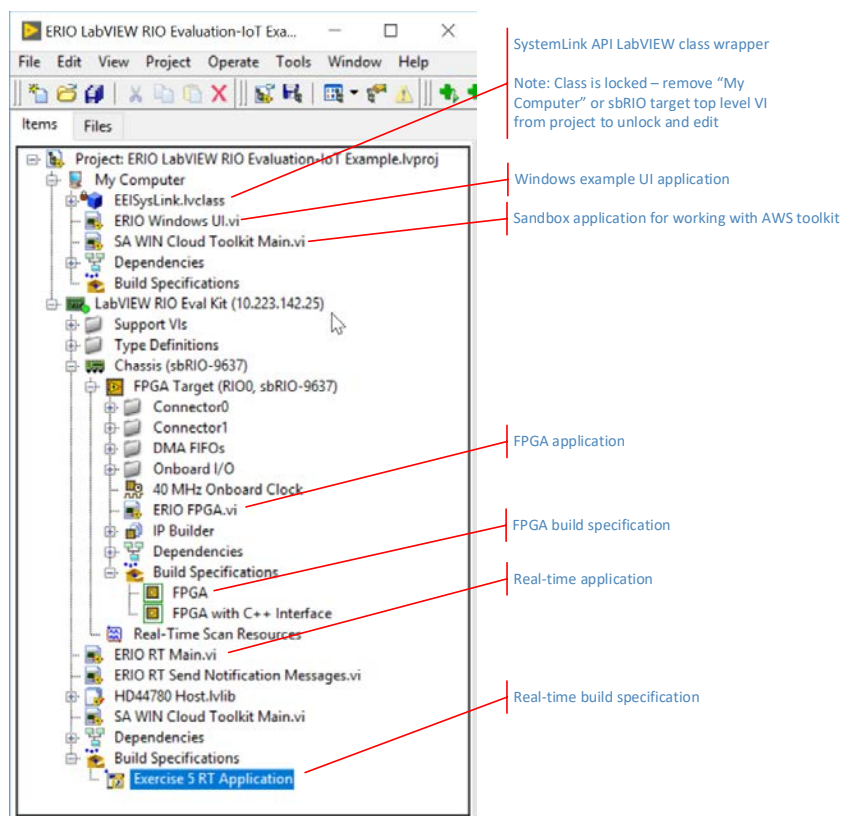
This application was built using LabVIEW 2017 as described in the [This Demonstration Example](#) section.

If you have this file open then likely you have downloaded all the project files from a source such as GitHub. Note that within the project files directory there are a couple code folders as follows.

-  ***sbRIO Eval IoT Example Code*** – Source code for this example
-  ***sbRIO Eval ORIGINAL Example Code*** – Original RIO Eval Kit shipping example source code this example was built from (exercise 5)

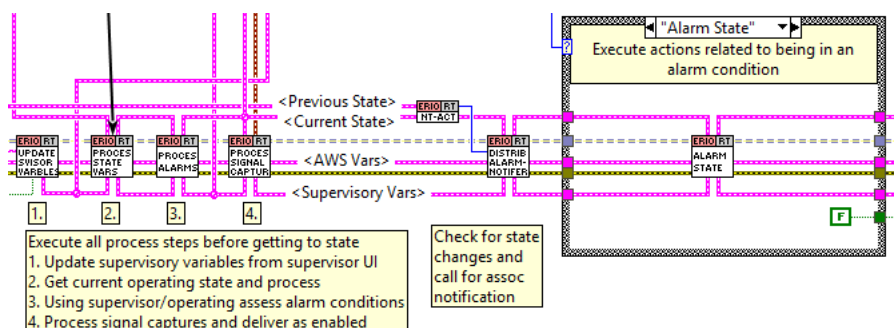


To get started working with this example open the file named *ERIO LabVIEW RIO Evaluation-IoT Example.lvproj*. Once opened you will see the following project. Below I have marked some areas of interest in the project.



REUSABLE ARCHITECTURE

This example was built to provide an architecture example for a basic controller application. The top-level architecture can be seen in the *ERIO RT Main.vi* with the main steps shown below.



The main steps here are as follows.

- ⚙ Update all supervisory variable from the controlling process
 - In this case it comes from the Windows UI via SystemLink tags
- ⚙ Input and process local controller process variables
 - Update SystemLink tags for controlling supervisory process



- ⚙ Assess alarm conditions
- ⚙ Execute any specialized processes
 - In this case signal capture
- ⚙ Execute appropriate process state
 - Based on all previous steps
- ⚙ Can put this in producer/consumer architecture and build consumer loops for more specific processing

This is just one architecture response to the solution objectives and there are many ways to approach a problem with tradeoffs. The main architecture objective is to actively plan an architectural approach to the solution before diving in.

DevNote: if you are a LabVIEW veteran then this is probably old-hat to you. If not then feel free to experiment with this architecture and modify it for your needs.

THE LABVIEW SYSTEMLINK API CLASS WRAPPER

A LabVIEW API installs with SystemLink to provide the essential calls for programmatic interaction between a LabVIEW based application and SystemLink. This includes the ability to interact with tags, messages, files, and configuration settings.

Note that in the SystemLink EAR release that this example was built on the LabVIEW SystemLink palette is under *Data Communication* and is currently named *Skyline*. Skyline was the name of the product before it reached its current release state.

In looking at the code you will see that the SystemLink API calls are inside a LabVIEW class library named *EEISystLink.lvclass*. Given that a primary focus of this example is SystemLink interaction I think it's useful to get more context around this design decision.

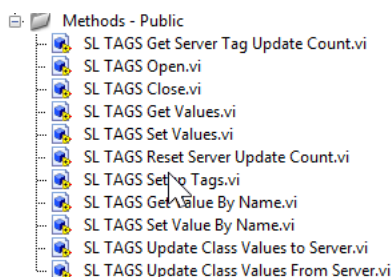
In general, using classes in your software design can take a while to get comfortable with, but they represent a very powerful tool in the LabVIEW toolbox. Classes allow you to package data and associated logic inside the class and expose an API designed for your specific purposes.

Specific purposes could be to simplify underlying complexity, abstract diverse considerations into a common interaction, and many other reasons. In the case of this SystemLink based class, it provides the following benefits.

- ⚙ SystemLink is early-stage and evolving – the class better enables an ability to manage an API common to the application
- ⚙ Introduce concept of local data and server-side data by keeping a local copy of the tag data in the class
 - This allows working on data locally when an expensive server round-trip is not needed
 - Ability for read/write locally or with server
- ⚙ Create application specific data types in native LabVIEW format that are translated to a supported SystemLink data type such as double, integer, or string (JSON, XML, CSV, etc.)
- ⚙ Add internal class business logic applied to the data
 - Example is caching tag updates when target is offline
- ⚙ Enable data search capabilities such as “get tag value by name”

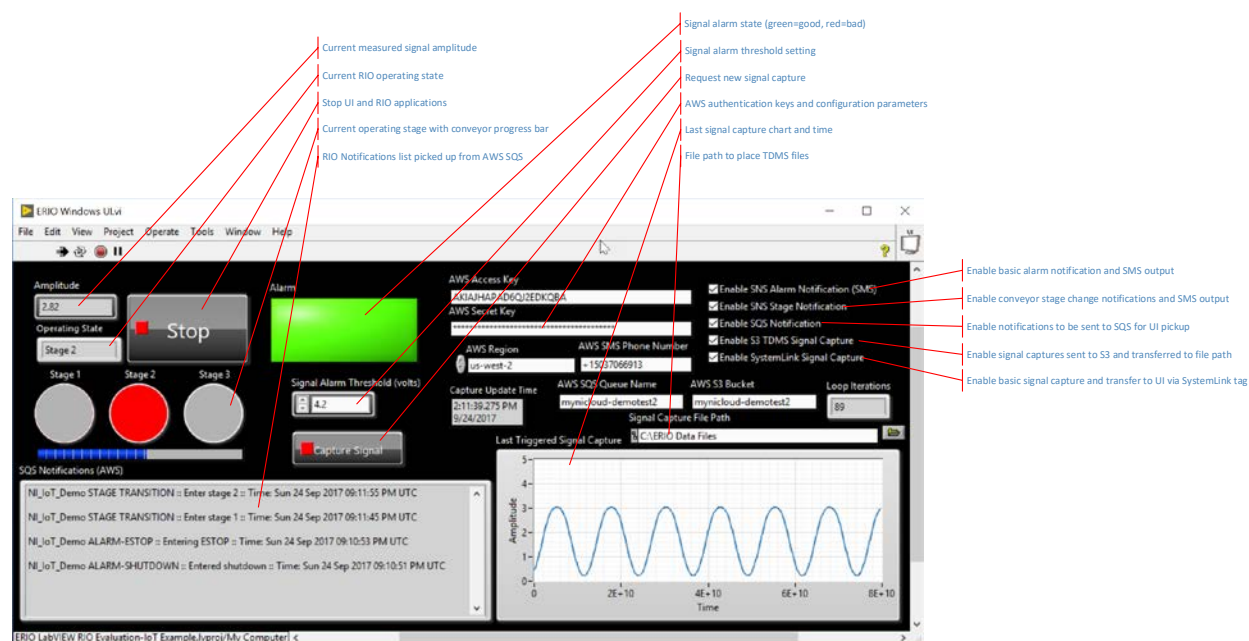


The public methods currently supported by this class are shown below.



IOT DEMO USER INTERFACE (UI)

This example user interface is a Windows application written with LabVIEW. It interacts with the RIO resident process through SystemLink and AWS services. Following is a visual overview diagram of the UI and then followed by more in-depth description of key functions.



Simulated Vibration Signal Function

A waveform generator built into the RIO Eval Kit carrier board is used to generate a simulated vibration signal. The generator can adjust shape (sine, square, sawtooth), frequency (range + variable) and amplitude.

This generated waveform output is wired into one of the RIO Eval Kit analog inputs where it is sampled at a rate set in the FPGA program. This sampled waveform is moved to the real-time application where an RMS value is calculated and returned as the amplitude, and when requested signal samples are processed as described in the [Signal Capture Function](#) section.

In the UI the calculated RMS amplitude is displayed and the user has the option to set a threshold level. If the amplitude exceeds the threshold on the RIO then an alarm is generated and the RIO program enters a shutdown state. If enabled, the resulting alarm is indicated on the UI and alarm notifications are sent to various destinations using the AWS toolkit.



Alarm destinations include SMS messages to the specified phone number and the UI, through an SQS queue. If enabled the alarm and other notifications are picked up from SQS and displayed in the *SQS Notifications (AWS)* text box.

Process Conveyor Stage Function

Like the software example that ships with the RIO Evaluation Kit, this example simulates a conveyor based process. As you turn the encoder knob clockwise the encoder counts up to simulate movement along a conveyor. If you turn it backwards it will count down but process logic is not setup to handle that well.

DevNote: there are many variations you could add to this basic conveyor/stage simulation to explore your specific application.

The RIO application starting state is a *Reset*, but as the encoder counts up it moves progressively through stages 1, 2, & 3. When it passes out of stage 3 the program enters a shutdown state, which then goes to an ESTOP and ends the RIO application.

The UI, as well as the RIO LCD, provides indication of the current operating stage in increments. The UI progress bar and the first 3 RIO LEDs show the process start to finish as counts.

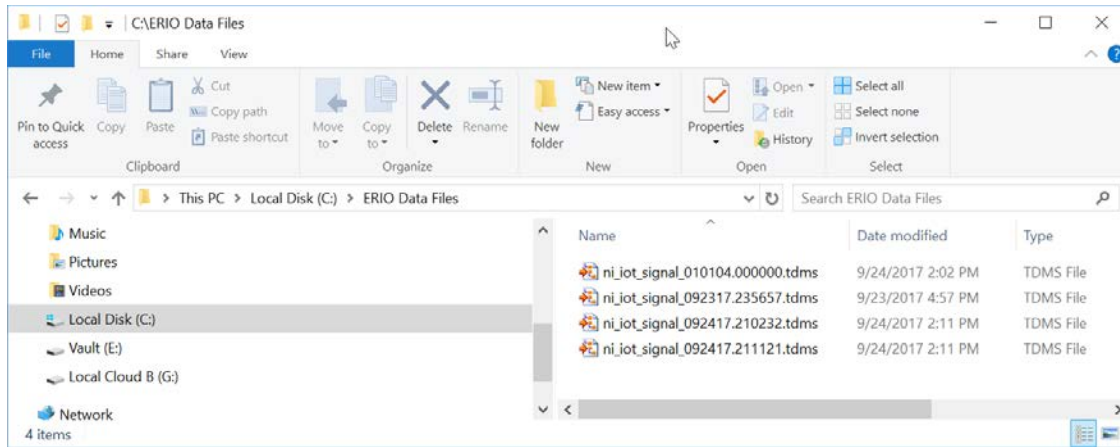
In the progress bar the counts are normalized to the progress bar range. For the LEDs the count is translated to PWM outputs to each of the 3 LEDs to adjust their brightness. As the count goes up the LEDs lite up from 1-3.

Signal Capture Function

When the signal capture button is pressed, this tells the RIO application to take 800 samples of its most recent signal waveform and “capture them” in a LabVIEW waveform object. This waveform capture is then distributed as directed in the UI settings. It is distributed to the following destinations.

- ⚙ To the UI via a SystemLink tag
 - The handshaking is a bit messy with SystemLink tags
 - *DevNote:* this application may benefit to move handshaking from tags to SystemLink messages
 - The waveform is converted within the LabVIEW class wrapper to a JSON format and transferred as a string tag
 - See [The LabVIEW SystemLink API Class Wrapper](#) section for more
- ⚙ To the UI computer as a TDMS file via S3
 - If enabled the RIO converts waveform to local TDMS file and sends to S3 bucket
 - The UI monitors the bucket and moves found files to a local folder
 - It then deletes the files from the S3 bucket
 - *DevNote:* an SQS queue could also do this but has size limitations.
 - *DevNote:* this could also be done with SystemLink files.

Once captured through a SystemLink tag, the UI then displays the waveform. In addition, if S3/TDMS is enabled then a new TDMS file will show up in the specified location as shown below.



Notification Function

This application uses AWS services to distribute notifications. The UI has checkboxes to enable the desired notifications with the following options.

- ⚙ SMS alarm notifications that the currently set threshold has been exceeded
- ⚙ SMS stage notifications that a stage state has changed
- ⚙ Echo the notifications to an SQS queue
 - They are then picked up by the UI, displayed, and deleted from SQS

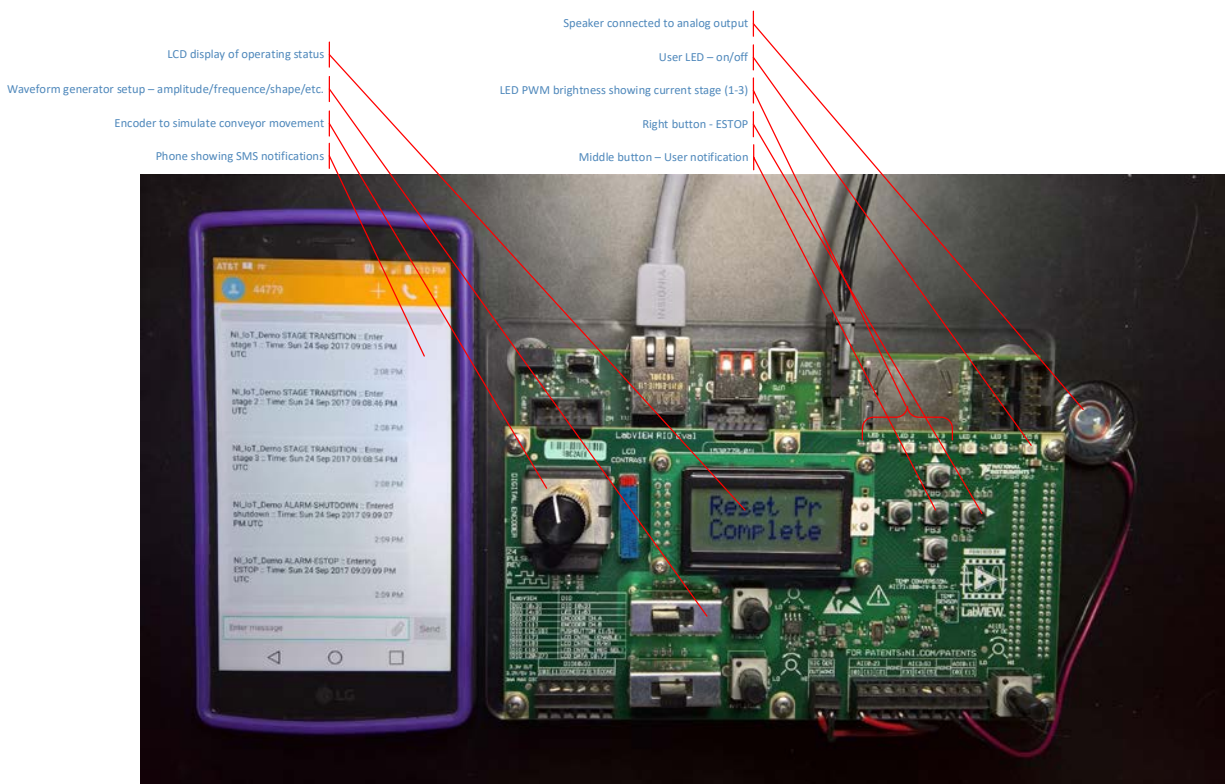
Notifications can be seen in the above UI screen capture and below on the phone displayed in the RIO board image.

RIO BOARD & SOFTWARE APPLICATIONS

The RIO board used comes directly from the [LabVIEW RIO Evaluation Kit](#) without modification and is the only specialty hardware needed for this example. As seen in the below image the kit consists of an sb-9637 board with a specialized carrier board to provide user interactions with all the standard I/O.

Other sections describe the high-level behavior of this example but there are a few board specific elements.

- ⚙ Right button will put the RIO application into an ESTOP state
- ⚙ Middle button is tracked in the RIO application as a “user notification”
- ⚙ Speaker comes with the RIO Eval Kit
- ⚙ User LED is made available in the RIO application to turn LED on/off



INTERACTIONS BETWEEN RIO & UI

All interactions between the Windows based UI and RIO applications are enabled with a combination of NI SystemLink and AWS services enabled by the [LabVIEW Cloud Toolkit for AWS](#). The following list describes functions and their enabling technology.

- ⚙ SystemLink
 - Tags used to transfer configuration and operating status
- ⚙ AWS
 - SNS for all SMS based notifications
 - SQS for notifications picked up by the UI
 - S3 for TDMS signal capture file transfer

Key SystemLink technologies are not currently explored in this example but it could be extended to include them. Technologies of note include the following.

- ⚙ Software package & deployment
 - You can use the LabVIEW application builder to build software into a “package”
 - The package is then uploaded to a SystemLink “feed”
 - Targets such as this RIO board can then subscribe to the feed
 - The targets can then remotely be up/down-graded to the desired release package version
 - This is my favorite new feature supporting remote system management
 - *DevNote:* try building the RIO application into a package, move to a SystemLink feed, and deploy to the target. For more go [here](#).
- ⚙ System management



- You can view target operating and configuration parameters like a simplified MAX
- This ability is expected to expand as SystemLink evolves
- ⚙ Web UI
 - Out of the box SystemLink has a web based UI
- ⚙ WebVI
 - Discussed in the [LabVIEW WebVI](#) section
 - Coming is the ability to connect WebVI to SystemLink tags
 - *DevNote*: download the LabVIEW NXG 2.0 beta and build a WebVI interface to the solution
- ⚙ Extensibility
 - SystemLink is extensible programmatically as well as through the UI
 - SDK for other languages like C#
 - I have built a C# library to move tag data to Seeq data store which works well

An example of the SystemLink web UI is shown below. This is a view of the tags generated by this example application.

	Path ↑	Current Value	Min	Max	Mean	Count	Updated
<input type="checkbox"/>	ERIO.AlarmThreshold	4.2	2.2	4.2	3.889296	11,295	September 24, 2017 2:14:26 PM PDT
<input checked="" type="checkbox"/>	ERIO.AWSAccessKey	AKIAJHAPAD6QJ2EDKQBA					September 24, 2017 2:14:26 PM PDT
<input type="checkbox"/>	ERIO.AWSBucket	mynicloud-demotest2					September 24, 2017 2:14:26 PM PDT
<input type="checkbox"/>	ERIO.AWSPhoneNumber	+15037066913					September 24, 2017 2:14:26 PM PDT
<input type="checkbox"/>	ERIO.AWSQueue	mynicloud-demotest2					September 24, 2017 2:14:26 PM PDT
<input type="checkbox"/>	ERIO.AWSRegion	3	0	3	2.569721	11,295	September 24, 2017 2:14:26 PM PDT
<input type="checkbox"/>	ERIO.CaptureAvailable	0	0	0	0	11,295	September 24, 2017 2:11:04 PM PDT
<input type="checkbox"/>	ERIO.CaptureSignal	0	0	1	0.302224	11,465	September 24, 2017 2:11:40 PM PDT
<input type="checkbox"/>	ERIO.CurrentAlarmState	0	0	1	0.053277	12,238	September 24, 2017 2:14:25 PM PDT
<input type="checkbox"/>	ERIO.EnableS3TDMSSignalCapture	1	0	1	0.287207	11,295	September 24, 2017 2:14:26 PM PDT
<input type="checkbox"/>	ERIO.EnableSNSAlarm	1	0	1	0.582736	11,295	September 24, 2017 2:14:26 PM PDT
<input type="checkbox"/>	ERIO.EnableSNSStageNotifier	1	0	1	0.665604	11,295	September 24, 2017 2:14:26 PM PDT
<input type="checkbox"/>	ERIO.EnableSQSNotification	1	0	1	0.653121	11,295	September 24, 2017 2:14:26 PM PDT
<input type="checkbox"/>	ERIO.EnableSystemLinkSignalCapture	1	0	1	0.996282	11,295	September 24, 2017 2:14:26 PM PDT
<input type="checkbox"/>	ERIO.Shutdown	0	0	1	0.00265	11,319	September 24, 2017 2:14:26 PM PDT
<input type="checkbox"/>	ERIO.SignalAmplitude	2.815266	0	4...	1.722781	13,129	September 24, 2017 2:14:25 PM PDT
<input type="checkbox"/>	ERIO.SignalCaptured	0	0	1	0.006326	11,381	September 24, 2017 2:11:38 PM PDT
<input type="checkbox"/>	ERIO.SignalWaveform	(*10^-3589132281.24586...					September 24, 2017 2:11:36 PM PDT
<input type="checkbox"/>	ERIO.StagePercentComplete	52.5	0	10...	11.155299	13,129	September 24, 2017 2:14:25 PM PDT
<input type="checkbox"/>	ERIO.State	Stage 2					September 24, 2017 2:14:25 PM PDT

AMAZON WEB SERVICES (AWS)

The AWS environment for SNS, SQS, and S3 must be setup for the AWS functions in this example to work. While AWS provides many different services, my experience is that setting up a new free account is straightforward on AWS. There is plenty of documentation and online information to support the process.

This document is not intended to provide a tutorial on AWS configuration but below are some useful links.



- ⚙ AWS General
 - Getting Started:
 - <https://aws.amazon.com/getting-started/>
- ⚙ SQS
 - What is Amazon SQS?
 - <http://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/Welcome.html>
 - Getting Started with Amazon SQS:
 - <http://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-getting-started.html>
 - How Amazon SQS Queues Work:
 - <http://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-how-it-works.html>
 - Best Practices for Amazon SQS:
 - <http://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-best-practices.html>
- ⚙ S3
 - What is S3?
 - <http://docs.aws.amazon.com/AmazonS3/latest/dev/Welcome.html>
 - Video Intro:
 - <http://docs.aws.amazon.com/AmazonS3/latest/dev/intro-managing-access-s3-resources.html>
 - Getting Started
 - <http://docs.aws.amazon.com/AmazonS3/latest/gsg/GetStartedWithS3.html>
 - FAQs
 - <https://aws.amazon.com/s3/faqs/>
- ⚙ SNS
 - What is SNS?
 - <http://docs.aws.amazon.com/sns/latest/dg/welcome.html>
 - Getting Started:
 - <http://docs.aws.amazon.com/sns/latest/dg/GettingStarted.html>

EXAMPLE SETUP & CONFIGURATION

Below are the steps taken to setup the environment needed to enable this demonstration.

They can be categorized in the following areas

- ⚙ Setup SystemLink server
- ⚙ Setup RIO target
- ⚙ Setup AWS services
- ⚙ Setup LabVIEW Development Environment

Note: When this document was authored SystemLink was in an Early Access Release (EAR) state. SystemLink details described in this document may have changed.



SETUP SYSTEMLINK SERVER

The first area of interest is [Installing and Configuring SystemLink Server and Clients](#). The following [www.ni.com](#) links are in this documentation space.

First make sure you have access to a SystemLink server. If not already setup then click on the [Setup SystemLink Server](#) link for more instruction.

SETUP RIO TARGET

For this example to work the SystemLink client must be installed on the target. In this case the target is an sbRIO eval board running Linux so navigate to the Linux [Install and configure SystemLink Client Installation](#) Procedure page for detailed documentation.

Make sure SystemLink support is setup on your RIO target. For more setup of SystemLink on the RIO client see the documentation page on [www.ni.com](#) which can be found [here](#).

Basic high-level steps for setting up the RIO target follow.

Add RIO Target to SystemLink

Navigate to the SystemLink server. If it is installed locally the default URL is: <https://localhost:9091/>.


Then navigate to the Systems Manager.



The Systems Manager should show a new “Discovered System”. Click on that link and you should see the newly configured SystemLink RIO target.

Systems Manager							
Dashboard > Discovered Systems							
Add Selected Add by Address							Filter
<input checked="" type="checkbox"/>	Name ↑	IP Address	Model Name	Serial Number	Master	Comments	Status
<input checked="" type="checkbox"/>	sbrio-9637-01bc4185-jetdev	10.223.142.25	sbRIO-9637	01bc4185			



Select the target and add it to the Systems manager with the link on top left. Once this process is complete it will show as  **Added** under the status column.

Now the sbRIO eval target has been registered with this SystemLink server as a connected system and all target SystemLink interactions will be with this connected server.

SETUP AWS SERVICES

This example documentation does not cover how to setup AWS services. Helpful links are provided in the [Amazon Web Services \(AWS\)](#) section. If you have specific questions please feel free to drop me a message on the contact info with this document.

SETUP LABVIEW DEVELOPMENT ENVIRONMENT

Setting up the LabVIEW development for this example is straightforward.

First, this example is built on the [LabVIEW RIO Evaluation Kit](#). To execute the code found in this project you will need to acquire the RIO Eval Kit. If you don't have one then this code can be used as an application example to be adapted for your specific needs.

Next open the LabVIEW project as described in the [The LabVIEW Project](#) section. Once opened make sure that the Windows based UI, real-time application, and FPGA application have a good run arrow.

The project FPGA reference in the real-time application currently points to the compiled bitfile so that may need to be pointed to your specific location to use the shipping compilation. Optionally you can point it to the project based FPGA reference but it will need to be recompiled.



Application Startup/Operation

Once all applications are executable then run them as indicated below.

1. First run the Windows UI application *ERIO Windows UI.vi*
 - a. It will create the SystemLink tags (if needed) and update them to starting values
2. Run the RIO application *ERIO RT Main.vi*
 - a. This application is driven from the SystemLink "supervisory settings" tags set from the UI
 - b. It will start the FPGA application
3. Primary simulation options
 - a. Turn the encoder knob to simulate conveyor movement
 - b. Adjust the waveform amplitude knob to reach alarm threshold
 - c. (if enabled) test SMS and SQS notification
 - i. Assume AWS has already been setup
 - d. Explore the UI settings and indications
 - i. AWS settings are read at startup so should not be changed once started

WHERE TO FROM HERE?

This example was built to the following expectations.

-  Provide a basic architecture example of an industrial application
-  Keep it relatively simple but still explore key technologies



- New “cloud connectivity” from National Instruments like SystemLink and AWS toolkit
- RIO application targets (real-time + FPGA)
- LabVIEW classes
- Public cloud integration (AWS)
- ⚙ Leave it a little rough for further development
- ⚙ Leave some obvious technologies out for further exploration

As mentioned previously, some technologies have been left out of this example but would be good candidates for further exploration. These include the following.

- ⚙ AWS IoT
 - A general purpose enabling technology for devices and the cloud
 - *DevNote:* experiment on connecting other devices into this example
- ⚙ SystemLink software packaging & deployment
 - *DevNote:* experiment on building the RIO application into a package and deploying to the RIO target
- ⚙ WebVI
 - *DevNote:* download and install [LabVIEW NXG version 2.0 beta](#), and then build a web based UI to this application
- ⚙ Make it real
 - *DevNote:* try some of the concepts here in your own application



CONCLUSION

I hope you find this example useful. I had fun building it and my intention is to provide code for a working example built on NI technologies that demonstrates the power of connected intelligence.

We hear a lot of talk about awesome technologies but eventually the rubber meets the road and for tangible benefit real working solutions that provide value are needed. I have found that good working examples can be hard to find and typically I can only find pieces of the answer. Hopefully you find this example as more than a piece of the answer and can not only provide answers to multiple “pieces” but also spark further contemplation about what is possible.

At [Endeavor Engineering](#) we are very excited about the possibilities from NI investments in cloud/edge connectivity and the IoT. SystemLink, LabVIEW Cloud Toolkit for AWS, LabVIEW NXG, and WebVI are the initial ones we are working with but look for these and yet-to-be-announced products to extend these capabilities.

We encourage you to check them out for yourself and if you want to get into your own solution we present this demonstration example as a starting point in getting to know them better.

Note they are in some form of early release so may change over time, but we think for the better.

Feel free to drop me a line if you have feedback or questions. I’m not sure how you came across this example but am interested in how I can adapt it to better support your exploration into Connected Intelligence.

<contact info on cover page>

–Jim