Platypus Sequencer Module

# Setup Instructions

1. This sequencer is dependent on the Tag Bus Data framework. As a first step you will need to download and install this framework. To do so, join the community where the code is hosted and following the download instructions. <https://decibel.ni.com/content/groups/crio-machine-control>
2. Next grab the sequencer code and move it to “C:\Program Files (x86)\National Instruments\LabVIEW 2014\vi.lib\NI\TBM Modules “. Alternatively, you can store it anywhere on disk you want and have the editor find it by going to Tools>>Edit Plugin Search Paths.
3. Go to the module you installed, and create your own ‘Supported User Actions List.vi’. This VI is currently where all of the application specific actions must live.
4. Create new actions by inheriting from the Action Parent found in the Sequencer Parent Classes folder. If you have multiple actions that are similar, it is likely worth the effort to create a project specific sample project or action template that you can use as a common starting point for each.

An example of using this sequencer can be found with the 3D Gears NI Week demo.

# Overview

This code implements a sequencer Tag Bus Module (TBM) within the Tag Bus Data framework. Like other Tag Bus Modules, this module is comprised of a configuration class, a runtime class, and an editor class. Unlike other TBMs, it also has a corresponding runtime API used to start, stop, and monitor sequences. A sequence is made up of Actions. An Action is like a TBM (it executes in a non-blocking fashion), except that its process method also has a first call input and a complete output. An Action also has a configuration class, but the Editor class is replaced with a single VI.

The configuration class inherits from a parent sequencer configuration class. This class is nearly identical to the parent engine configuration class, and provides an API to map data from the module channels to data within an Action. The main difference is that the sequencer configuration class allows multiple writers to the same channel.

The runtime class implements Init, Process, Process Safe-State, and Close. The process function has a set of defined sequences ready to run, and gets a command containing which sequences to execute. It can then be told to stop executing its current sequence.

The editor class implements a UI that lets users create arbitrary sequences from the list of supported actions, and configure the settings for each action individually. This information all gets stored in the main configuration editor file and can be loaded up at runtime.

# Known Issues

* The safe-state functionality currently only supports the execution of a single action. I think ideally this would support a specific safe-state sequence that would be configurable in the editor.
* Many places that should throw errors in the code are stubbed with #todo comments. The main reason for this is that the editor largely prevents these errors from being possible, but they should still be added for the programmatic configuration use-case.
* The ‘Supported user actions list.vi’ should be selected programmatically in the editor. This would prevent actions from being statically linked to the sequencer code. It would also allow different instances of sequencers to support different actions. Right now this VI will be reported as missing.
  + Also need to fix this for the case where the sequencer is loaded in the editor.
* The Process function currently receives its commands to execute a sequence using AMC. Ideally a deterministic method would be used instead, but this was done because it was fast and not detrimental to performance in my use-case.

# Needed Enhancements

* Instead of sequencing actions, this code would be much more powerful if it instead sequenced sequence objects. We could then make a ‘Step’ as a sequence object which would then contain one or more ‘Actions’. The Step would complete once all Actions completed. This would also allow use to add For Loops, Subroutines, etc.
* There is currently no way of mapping data in an Action to channels in the sequencer. The runtime code supports this, as well as the configuration code in the sequencer, but the UI code in the sequencer is not there.
* The sequencer could be greatly improved by better reporting features (that also execute deterministically). Currently an Action can either output a Boolean Finished or return an error. If there are different levels of success there is no way to report them.