Source guide: TestStand step types for VeriStand

Contents

[Intro 1](#_Toc448757049)

[Components 2](#_Toc448757050)

[TestStand Step Definitions 2](#_Toc448757051)

[Config time code modules 2](#_Toc448757052)

[Runtime code modules 2](#_Toc448757053)

[Install-time script 2](#_Toc448757054)

[Installer 2](#_Toc448757055)

[Build Process 3](#_Toc448757056)

[Modifying a step - An Example 3](#_Toc448757057)

# Intro

Before you work with this, refresh yourself on these concepts:

* Custom TestStand step types
* C#
* The VeriStand Client API

There are 3 main components to the step types

* The TestStand step definitions
* The config time code
* The runtime code

There are also a couple of other pieces the step types, including:

* An Installer
* An install-time script to rearrange the TS insertion palette

If you haven’t noticed yet, there are multiple branches of source code. That is because it is a manual process to link things against different versions of the assemblies.

For now, the code is lacking in uniformity. We’re working on that.

# Components

## TestStand Step Definitions

These are the TestStand step types as listed in the VeriStandTypes.ini file. These can be seen in TestStand (once you load it) by pressing ctl+T. These definitions don’t really implement much – but they tie all of the pieces together by telling TestStand what calls to make with what data at what times.

Main parts of this include the name, icon, etc., and the substep configurations. The substeps contain the configuration to the calls for both runtime and config time code modules. Here is also where the step variables are defined.

There are edit substeps, that specify configuration dialogs. Then there are pre- and post- steps, which specify runtime code. Nothing is special about the pre- or the post- designation in this case, since they are null adapter type steps. They simply execute in order from top to bottom in the substep list window. Then a couple of steps have an OnNewStep substep which runs when the step is added to the TS sequence.

## Config time code modules

The config time code modules are all C# code (for the steps that do have config dialogs). This code is all in the C# code in this project.

## Runtime code modules

There are substeps that make run time calls to LV code, to C# code, and directly to VS assemblies.

LV code modules include:

* MiscHelperVIs
  + This is miscellaneous LV code that helps manage errors and references
* RTSequenceVIs.lvlibp
  + This package has LV code deploys RT sequences
* Set Channels.lvlibp
  + This of course is LV code that sets channels
* Logging.lvlibp
  + LV code that does host side logging
* SilentVeriStand.exe
  + This is LV code built as an exe. It is code that runs the VS gateway without running the VS exe. That runs the main resources without running any UI elements.

## Install-time script

There is a C# script that runs as part of the installer. This is mainly to rearrange and add the step types in a nice way to the insertion palette (the left pane of TS by default). It also removes insertion palette items from previous versions of these step types, to maintain a clean look.

## Installer

The installer, of course, installs the step types. It installs the files, including:

* TS step definitions ini file
* LV code, mostly as PPLs
* C# assemblies that contain C# code from this project
* An icon file
* Silent Veristand.exe

# Build Process

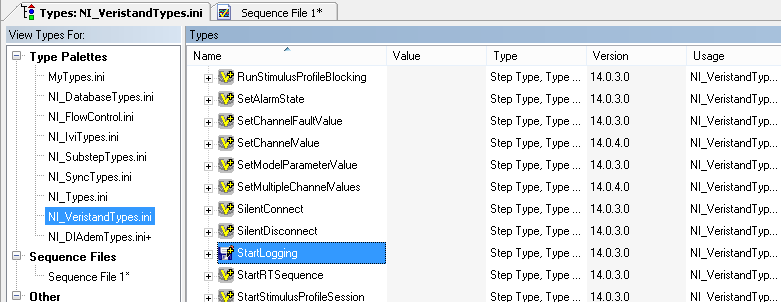
1. Build the LV components, including the .lvlibp’s and the Silent Veristand.exe. (There is also an icon file)
2. Place the files in their destination directory. For most of them, it’s in TestStand public docs 🡪 TypePalettes (you can find the path in the installer project). The SilentVeriStand.exe goes into the VeriStand.exe directory. Don’t worry about the C# code, since the installer will build that. Upon installation, it will be inserted into the GAC (Global Assembly Cache).
3. Build the whole Installer by building the “NI VeriStand 2014 Custom TestStand Steps.sln”. On second thought, that filename hasn’t been updated - though it doesn’t matter. The build process will build the script, scoop up the LV build files, and build the installer.

# Modifying a step - An Example

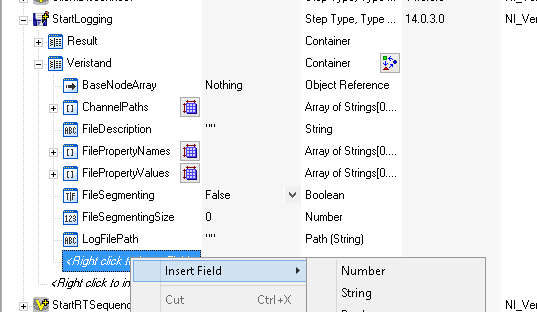
In this, let’s assume that we want to add a feature to a step. For example, let’s talk about adding a custom logging rate control to the Start Logging step. That will involve editing the step definition (in the ini file), the config dialog (in the C# assembly) and the runtime code (LV).

For clarity, I’ll show pictures, but I’m not actually making any modifications. Don’t be confused if they don’t appear changed from the way they are now.

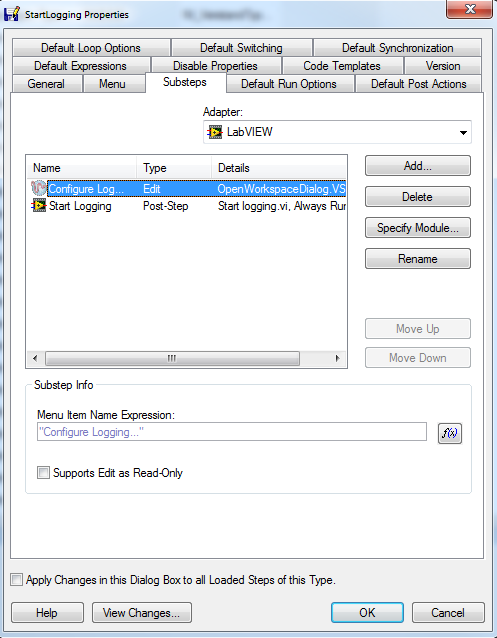
Let’s begin by investigating Start Logging’s step definition in the type palette (ctrl+T) to see some details.



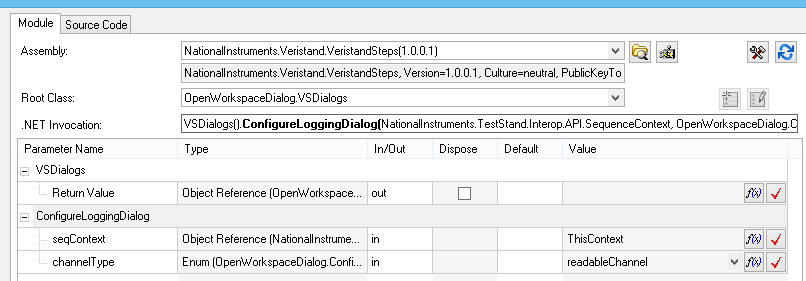
Since we want to add a new parameter to this step for logging rate, we will first need to add a new step property to the custom step. Add a new string property to the Start Logging step called “CustomLoggingRate” in the type palette. For reference, changes made to the type palette here are saved to the NI\_VeriStandTypes.ini file. For this new property, it makes sense to follow the current convention and put the new step property in the VeriStand container:



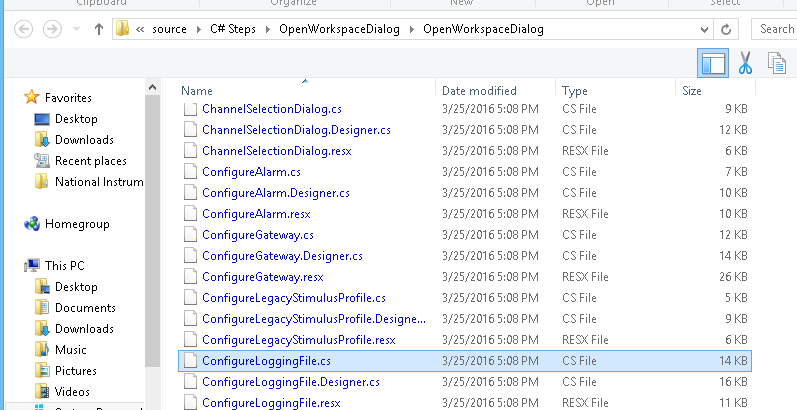
Next, to get an idea which code we need to modify to allow configuration of our new property, open Start Logging’s step properties and look at the substeps.



The Edit substep identifies the call that is made when the config button is pressed. Since we want to make an addition to the config dialog, we’ll need to modify this code. Clicking “Specify Module…” shows that this code refers to the “ConfigureLoggingDialog()” method somewhere inside the C# assembly.

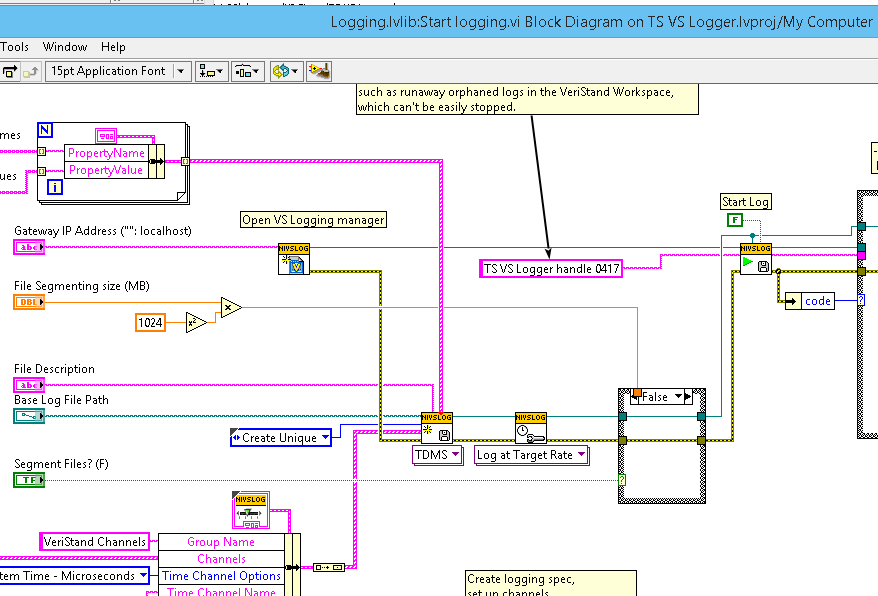


We know we’re going to be modifying C# code, so we need to open up the OpenWorkspaceDialog Visual Studio Solution and search it find where ConfigureLoggingDialog() lives. You’ll find that the file you’ll need to modify in this case is ConfigureLoggingFile.cs.

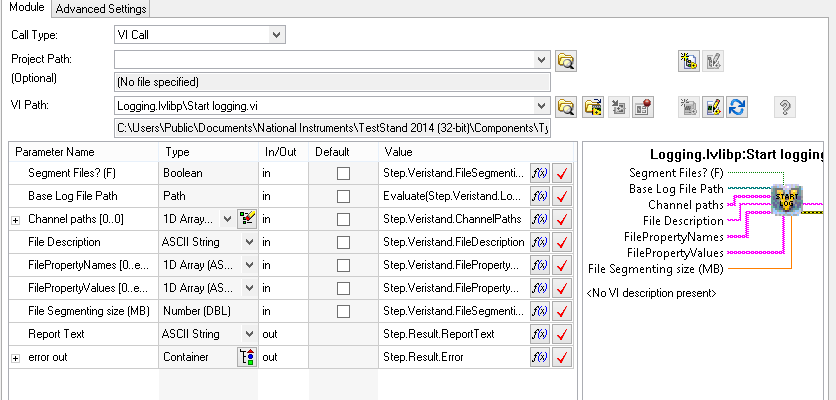


It’s important to know that the C# configuration dialog steps are only responsible for reading and writing to and from this step’s step properties. As mentioned above, the way this operates may differ a bit from implementation to implementation, but you can follow the general framework in place to get an idea of how and where to add in this new step property that we created. In general, we want to add our new property to the dialog GUI, add the code that handles retrieving the new property when the dialog initializes, and then code to save the control’s value back to its step property when the dialog closes. It’s worth noting that the GUI implementation *could* be skipped entirely if end users are comfortable with modifying your new step property in TestStand’s step property browser directly. The GUI implementation is there to provide convenience to the end user.

The last remaining piece is to actually implement the functionality that this new property represents. This means we’ll need to modify the code that runs when TestStand executes this step. We can find in the substep (following the process above) that the Post-Step runtime code call is made in Logging.lvlibp >> Start Logging.VI.

There is one subVI from the VeriStand API that needs to be added/changed to make the change for this example. You may also want to add a case structure and a boolean input to switch modes between logging at target rate versus a custom rate.

Make sure to add any new inputs and outputs to the connector pane. That is how they are tied into TestStand. Then, go back to the substep and reload the prototype. The new inputs will need to be added to the call there:



Map the new inputs and outputs to your new step variables by adding expressions to the boxes.

That should do it. Since the runtime code is in LV, we can test that portion without needing to rebuild the installer – just the lvlibp (and moving it to the built location of course).

Finally, you will need to rebuild the whole installer, using the steps in that part of the guide.