

User guide for the VI Launcher LabVIEW toolset

Rev. 2012.2

December 11th 2013

1	INTRODUCTION	2
2	VI LAUNCHER	3
3	REFERENCE	4
3.1	Close	4
3.2	Open	5
3.3	OpenAndRun	6
3.4	Run.....	7
3.5	WaitForInputData.....	8
4	DEPENDENCIES	8
5	LICENSING.....	9

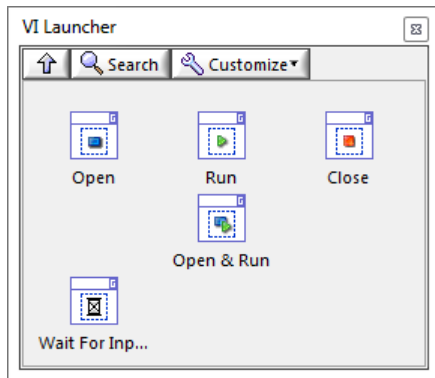
1 Introduction

The VI Launcher LabVIEW toolset simplifies the task of launching a VI dynamically and optionally transferring startup data to it. Dynamic launching is the process where you run a VI defined at runtime, contrary to a static VI defined at edit-time (e.g. an ordinary subVI dropped on the block diagram). VI Launcher features:

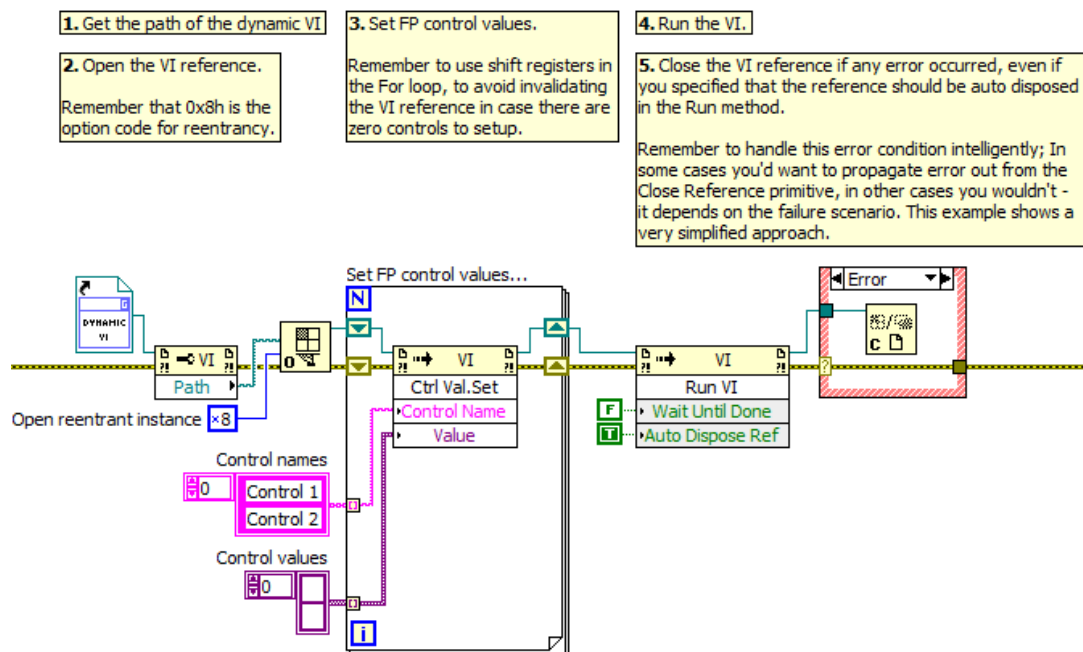
- This toolset lets you open a VI reference, setup input data to, and finally run that dynamic VI – all in either two separate operations or in a single function call if you prefer.
- Two different methods of inputting data are offered: the usual method where you setup values in some front panel controls with VI Server prior to actually running the dynamic VI, as well as an alternative method that lets you transfer data directly to the block diagram code without going through the front panel at all.
- The **Close** function optionally lets you wait for the dynamic VI to end execution before closing the VI reference. This also comes with an option of aborting the dynamic VI before the reference is closed, if the VI is still running when the timeout occurs.
- The dynamic VI to run can be specified by either path or by VI reference.
- With this toolset it's simpler, compared with the built-in methods, to select if a dynamic VI should run as a non-reentrant or as a reentrant instance.
- The toolset automatically handles many often forgotten cases, such as remembering to close the opened VI reference if the Run method fails after opening the VI reference.

This toolset has been developed for LabVIEW 2012 and is compatible with both older and newer versions as well. See section 5 “Licensing” for terms of use.

2 VI Launcher

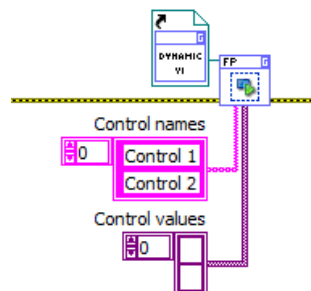


The most common built-in way of launching a VI dynamically is shown below. In this case it's a reentrant VI that includes setup of two front panel controls before it is run:



Even though the above code is elaborate and uses up quite a lot of block diagram space, it's still not complete if you want to be really thorough – error handling is lacking for instance.

Instead you can use this VI Launcher toolset. The example below shows the exact same launch as above:

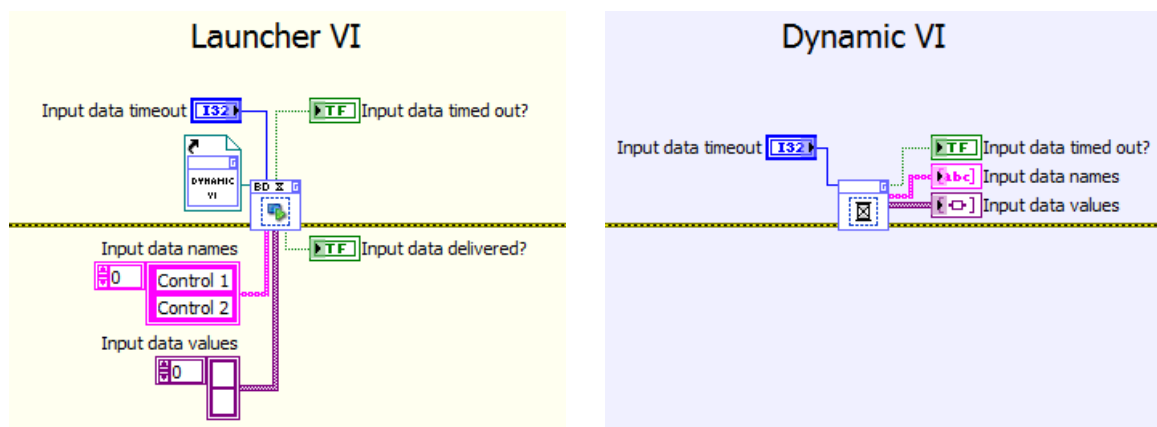


Notice the "FP" text on the **OpenAndRun** function icon above? This indicates that any data sent to the dynamic VI at the launch travels through its **Front Panel** controls. This is the simplest and most common

way built into LabVIEW to transmit data to a dynamic VI at launch, but it comes with a number of drawbacks:

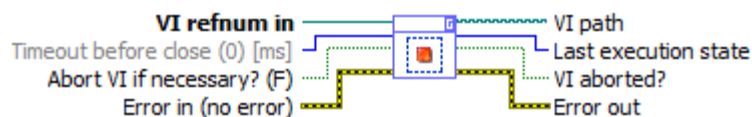
- You must have a UI control on the dynamic VI for each data element you wish to transfer to it at launch.
- The labels of these UI controls must remain constant. Or at least, if you change any of them you must also extend these changes to all locations you launch the dynamic VI from.
- All VI Server calls demand root loop access, which among other things mean they block, and can be blocked by, any user interface thread access. One consequence of this is that if your GUI blocks the user interface thread waiting on the operator, you cannot write to a front panel control of another VI with VI Server in parallel.
- There is no way to detect from the launcher if setup data was in fact read by the dynamic VI code.

The VI Launcher toolset presents an alternative way to transmit input data to a dynamic VI – a method that alleviates all the drawbacks of using the front panel, albeit imposing a single new one: you must use the **WaitForInputData** function in the dynamic VI to get the input data:



3 Reference

3.1 Close



Closes the reference to the dynamic VI that **VI refnum in** refers to and returns the **VI path**.

You can set **Timeout before close** to something else than the default value of 0 to let this function wait for the dynamic VI to leave "Run top level" execution state before the VI refnum is closed. Set **Timeout before close** to -1 to wait indefinitely for the dynamic VI to stop executing. **Timeout before close** has a granularity of 20 ms.

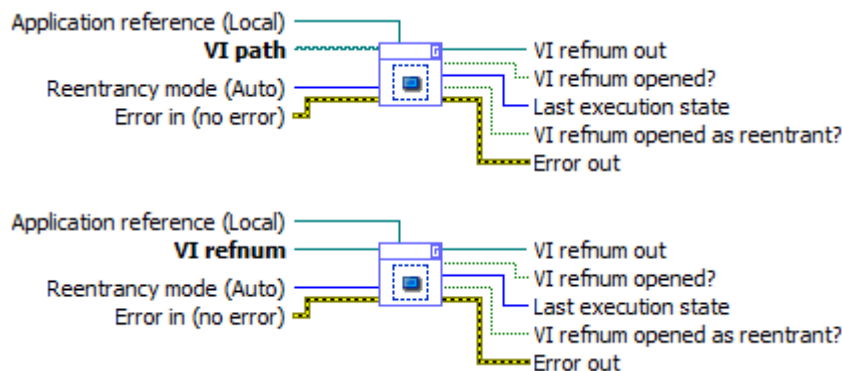
If the optional **Abort VI if necessary?** is True (default is False), and the dynamic VI's execution state is "Run top level" even after **Timeout before close**, this function aborts the dynamic VI before the VI refnum is closed.

Last execution state is the last value before the VI refnum was closed, but after an abort if such was executed.

The **VI aborted?** output will be True if the dynamic VI had to be aborted to get it out of "Run top level" execution state.

This VI is reentrant, and it runs even if there is an error on **Error in**. An error on **Error in** will not be overwritten by any error occurring in this function.

3.2 Open



This VI is polymorphic, it opens a reference to a dynamic VI either specified by path (**VI path**) or by reference (**VI refnum**) and returns the reference to the opened VI in **VI refnum out**. Remember to close the VI reference when you're done using it.

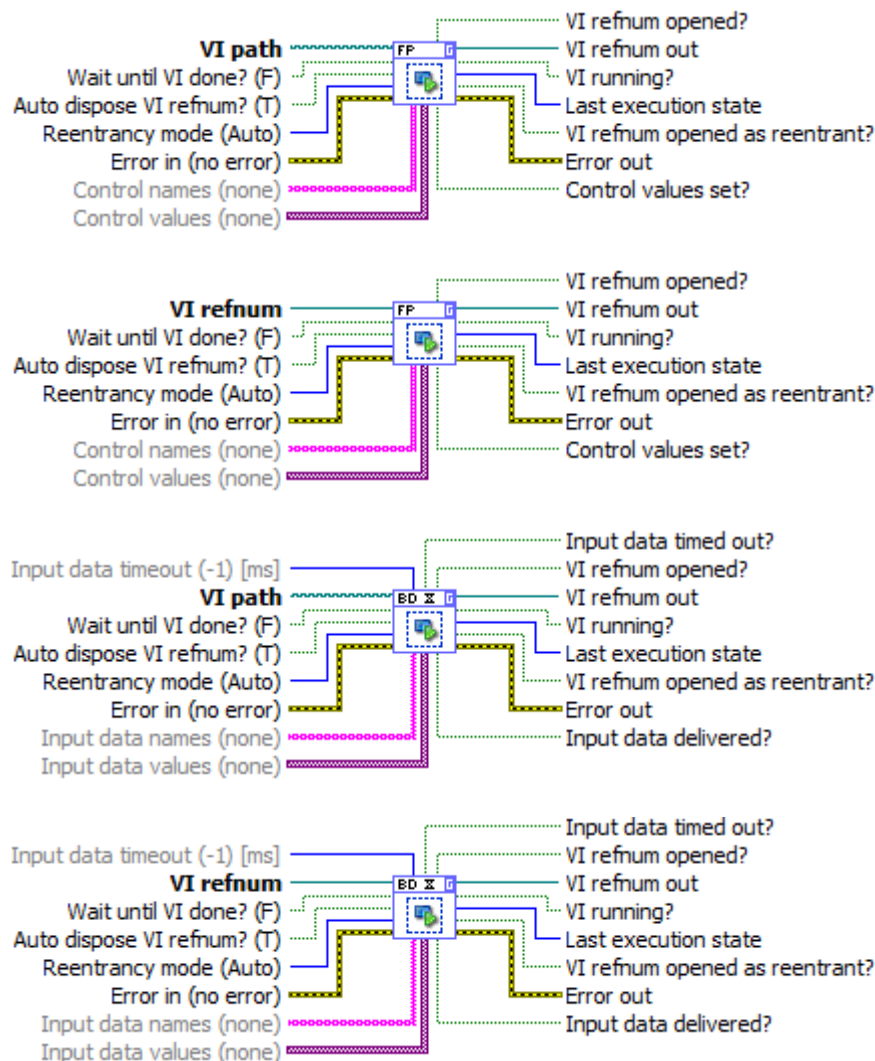
The optional **Application reference** can be used to specify another application instance to open the VI reference in. Use the Open Application Reference LabVIEW function to open another application reference.

Use the optional **Reentrancy mode** input to specify whether the dynamic VI should be opened as a reentrant or a non-reentrant instance. Set this to Auto (default) to first try as reentrant, and if that fails, as non-reentrant. The **VI refnum opened as reentrant?** output will reflect the instantiation mode.

The **VI refnum opened?** output will only be True if the dynamic VI's **Last execution state** is "Idle" or "Running".

This VI is reentrant, and it doesn't run if there is an error on **Error in**.

3.3 OpenAndRun



Opens a reference to a dynamic VI either specified by path (**VI path**) or by reference (**VI refnum**), runs the dynamic VI, and returns the reference to it in **VI refnum out**.

This function will wait until the dynamic VI is done if **Wait until VI done?** is True (default is False), and it will auto dispose the dynamic VI reference (hand it over to the running dynamic VI) if **Auto dispose VI refnum?** is True. Note that if this function fails to run the dynamic VI, it will close the opened VI reference itself, in which case **VI refnum opened?** will be False and **VI refnum out** will be invalid.

Use the optional **Reentrancy mode** input to specify whether the dynamic VI should be opened as a reentrant or a non-reentrant instance. Set this to Auto (default) to first try as reentrant, and if that fails, as non-reentrant. The **VI refnum opened as reentrant?** output will reflect the eventual instantiation mode.

The **VI running?** output will only be True if the dynamic VI's **Last execution state** is "Run top level". **VI running?** can be False even if the dynamic VI ran successfully – it might just have finished before this VI finished.

Optionally you can send input data to the dynamic VI when running it using one of two modes; Right click this VI and use the Select Type menu to choose input data transfer mode:

Input data via front panel

Set data on the front panel of the dynamic VI before running it by wiring **Control names** and **Control values**. **Control values set?** will be True only if all **Control names** were valid.

Input data via block diagram

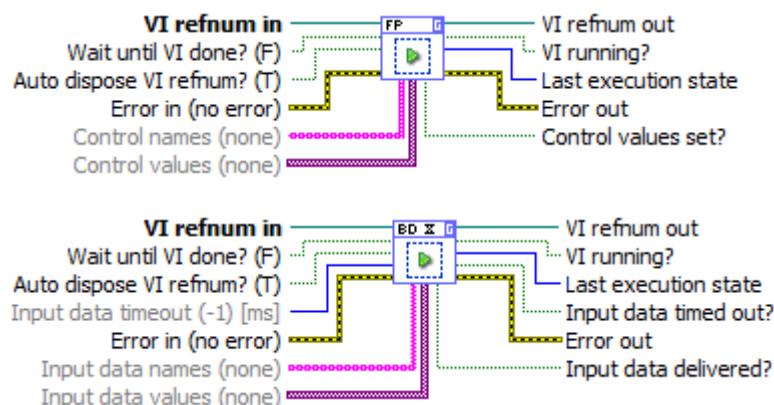
Transmit data to the dynamic VI after it has started by wiring **Input data names** and **Input data values**. If you use this mode the dynamic VI must use the WaitForInputData function to read the input data on the block diagram. The read in the dynamic VI must be finished before **Input data timeout** runs out. Set this timeout value to -1 (default) to wait indefinitely. If timeout does occur, **Input data timed out?** will be True. **Input data delivered?** will be True only if input data was in fact transmitted to the dynamic VI without any errors.

NOTE 1: When using the 'Input data via block diagram' mode the dynamic VI must use the WaitForInputData function even if you send zero input data. To avoid this requirement for the dynamic VI when not passing any input data, please use the 'Input data via front panel' mode in this case.

NOTE 2: The dynamic VI must have its front panel available at runtime for this function to work. The simplest way to ensure this, which also works in a built application without additional steps to be taken, is by enabling "Show front panel when called" in the "Window Appearance" section of the dynamic VI Properties.

This VI is reentrant, and it will run only if there is no error on **Error in**.

3.4 Run



Runs the dynamic VI specified by **VI refnum in**.

This function will wait until the dynamic VI is done if **Wait until VI done?** is True (default is False), and it will auto dispose the dynamic VI reference (hand it over to the running dynamic VI) if **Auto dispose VI refnum?** is True. Note that if this function fails to run the VI, **VI refnum in** will not be auto-disposed, in which case you retain the responsibility of closing that reference yourself.

The **VI running?** output will only be True if the dynamic VI's **Last execution state** is "Run top level". **VI running?** can be False even if the dynamic VI ran successfully – it might just have finished before this VI finished.

Optionally you can send input data to the dynamic VI when running it using one of two modes; Right click this VI and use the Select Type menu to choose input data transfer mode:

Input data via front panel

Set data on the front panel of the dynamic VI before running it by wiring **Control names** and **Control values**. **Control values set?** will be True only if all **Control names** were valid.

Input data via block diagram

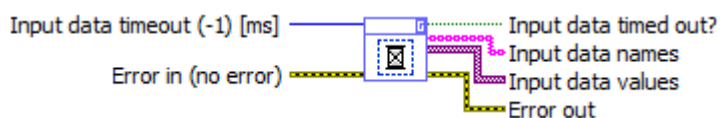
Transmit data to the dynamic VI after it has started by wiring **Input data names** and **Input data values**. If you use this mode the dynamic VI must use the WaitForInputData function to read the input data on the block diagram. The read in the dynamic VI must be finished before **Input data timeout** runs out. Set this timeout value to -1 (default) to wait indefinitely. If timeout does occur, **Input data timed out?** will be True. **Input data delivered?** will be True only if input data was in fact transmitted to the dynamic VI without any errors.

NOTE 1: When using the 'Input data via block diagram' mode the dynamic VI must use the WaitForInputData function even if you send zero input data. To avoid this requirement for the dynamic VI when not passing any input data, please use the 'Input data via front panel' mode in this case.

NOTE 2: The dynamic VI must have its front panel available at runtime for this function to work. The simplest way to ensure this, which also works in a built application without additional steps to be taken, is by enabling "Show front panel when called" in the "Window Appearance" section of the dynamic VI Properties.

This VI is reentrant, and it will run only if there is no error on **Error in**.

3.5 WaitForInputData



Waits for **Input data names** and **Input data values** sent by a launcher to a dynamic VI using the Run or OpenAndRun function.

The optional **Input data timeout** specifies how long time to wait for the input data. Set this to -1 (default) to wait indefinitely. If timeout does occur, **Input data timed out?** will be True.

This function does not have to be located in the Top-Level dynamic VI, but can be embedded anywhere in the dynamic VI hierarchy. Just be sure to read the input data before timeout of the Run/OpenAndRun function occurs, or else the input data will be lost.

This VI is reentrant, and it doesn't run if there is an error on **Error in**.

4 Dependencies

- GPower Error & Warning 2012.1 toolset or later.
- GPower Timing 2012.2 toolset or later.

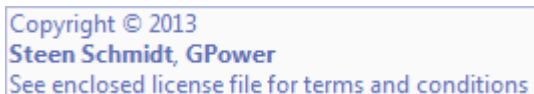
5 Licensing

This toolset is free and open source, and is covered by a BSD 2-clause license (see the `VIlauncher_License.txt` file included with the toolset). But why a license when it's free? The BSD license is one of the most unrestrictive licenses out there, and we use it mainly to disclaim our responsibility for that software which it covers. So you may use it for whatever you want, but we won't take any responsibility if it blows up on you. It by all probability won't blow up on you (we use this software ourselves), but if it does we won't take responsibility for your losses. We certainly would be glad to hear about any problems you might have run into though, so we can fix those issues. The BSD license in layman's terms grants you these rights:

- You may use this GPower software for free for both commercial and non-commercial applications. This includes both development of and inclusion in such applications.
- You are at liberty to employ any licensing scheme you wish on your final application. That this GPower software is open source and free does not mean your application using it has to neither be open source nor free. Your application can employ a proprietary license, you can charge money for it, it can be closed source etc.
- You are allowed to change the source code of this GPower software as you wish with no obligation to publish your changes in any way.

The BSD license does put two obligations on your shoulders though, even if you use this GPower software in modified form:

- You **must** leave the GPower copyright notice on the front panels of the VIs that make up the software. It will look something like this:



Copyright © 2013
Steen Schmidt, GPower
[See enclosed license file for terms and conditions](#)

It is not required that you include the front panels of the VIs of this GPower software in your built application. But if you do, or if you bundle the source code, then you must leave the mentioned copyright notice in place. If you're not including the front panels of the GPower software, then the copyright notices from those front panels obviously can't be present.

- You **must** preserve the unmodified license text file with the software, and it must reside somewhere your end-user would expect to find it (this is to preserve the disclaimer). If you include the GPower software source code you can just leave the license text file with those VIs. If you deliver binaries exclusively you can for instance save the license text file together with your other material rights documents, or you could paste the license text into the small-print section of your user guide.