Building a plug-in framework using classes and Packed Project Libraries… and use it on CompactRIO!

Jeffrey Habets

Wim Tormans

SPECIALISTS IN
VIRTUAL INSTRUMENTATION

LabVIEW • TestStand

Realtime • FPGA • Embedded • Object Oriented
UML • Consultancy • Framework • Architecture
Machine Vision • Machine Automation

Engineering support • Large Application development
• Project startup • Training • Integration

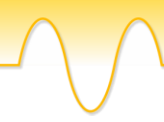
www.vi-tech.nl

- Reasons to use plug-ins
- Basics of a plug-in architecture
- Why we used Packed Project Library plug-ins on cRIO targets
- Software platform overview
- The Do's and Don'ts of PPL plug-ins on cRIO

Agenda

- **Reasons to use plug-ins**
- Basics of a plug-in architecture
- Why we used Packed Project Library plug-ins on cRIO targets
- Software platform overview
- The Do’s and Don’ts of PPL plug-ins on cRIO

Reasons to use plug-ins



Wikipedia: "A **plug-in** (or **plugin**, **add-in**, **addin**, **add-on**, **addon**, or **extension**) is a software component that adds a specific feature to an existing computer program. When a program supports plug-ins, it enables customization."

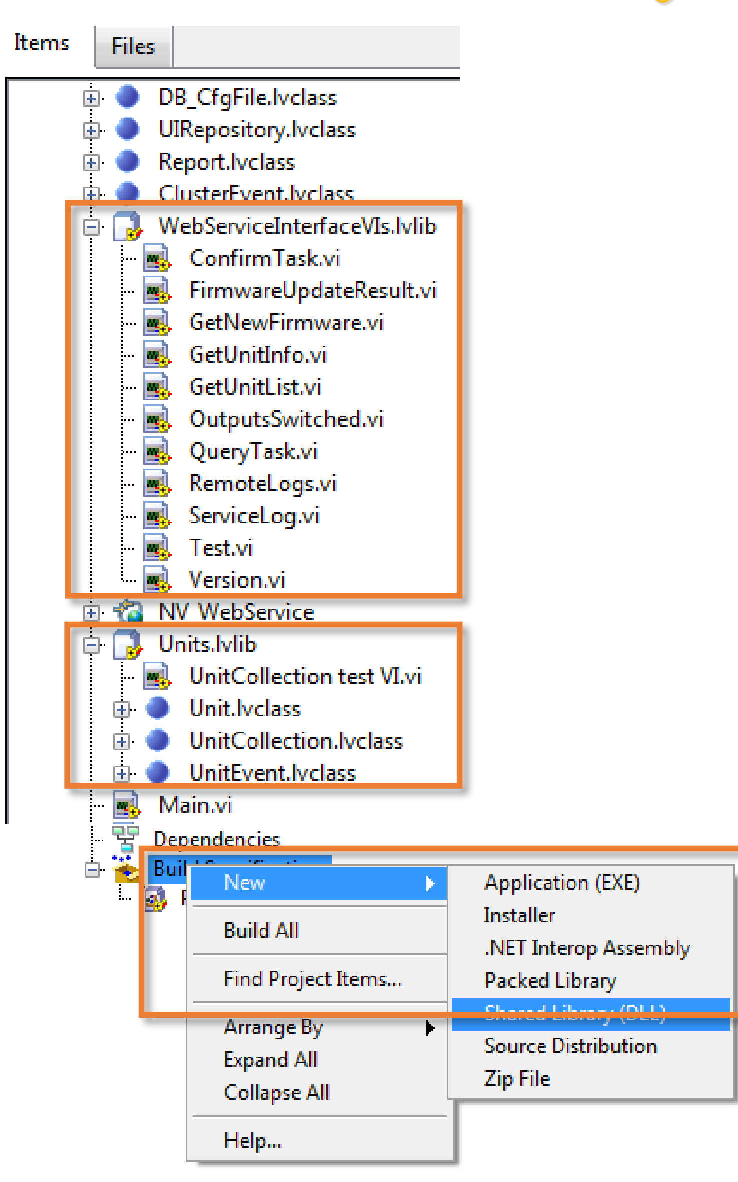
For example

- Enable third-party developers to create abilities which extend an application
- Support easily adding new features
- Reduce the size of an application
- Application is extensible without code modification to application

Reasons to use plug-ins

LabVIEW libraries (.lvlib) in a nutshell

- Are a way to modularize your code
- Provide namespacing (you can have VIs with the same name in different lvlibs in the same App)
- Give you scoping of code (Public, Private, Community)
- A class (.lvclass) is a special kind of library
- A XControl(.xctrl) is a special kind of library
- A Packed Project Library (.lvlibp) is a compiled version of a lvlib with all its contents



- Reasons to use plugins
- **Basics of a plugin architecture**
- Why we used packed project library plugins on cRIO targets
- Software platform overview
- The Do’s and Don’ts of PPL plugins on cRIO

Basics of a plug-in architecture

- Application operates independently of the plug-ins
- Plug-in typically needs the application to perform its job (not stand-alone)
- Plug-in needs to abide by the contract layed out by the application
- Important to get the contract 'right' the first time, think it through, really good!

Basics of a plug-in architecture - example in LabVIEW

Application

Plug-in manager

Loads available plugins

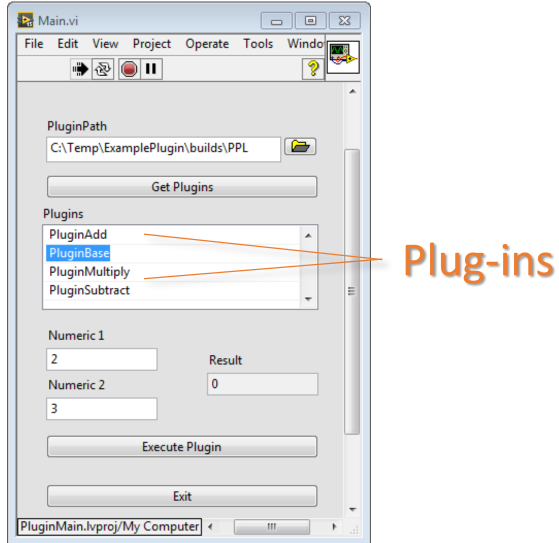
Plug-in interface

No code!

This is **all** that needs
To be in your application
(build .exe)

Basics of a plug-in architecture - example in LabVIEW

Application



Plug-ins

Implement the interface

Basics of a plugin architecture - example in LabVIEW

PluginMain.lvproj - Project Explorer

File Edit View Project Operate Tools Window

Items Files

- Project: PluginMain.lvproj
  - My Computer
    - PluginBase.lvlibp
      - PluginBase.lvclass
        - protected
        - PluginBase_Init.vi
        - CleanUp.vi
        - Execute.vi
    - Main.vi
    - Dependencies
    - Build Specifications
      - My Application

Computer ▸ Local Disk (C:) ▸ temp ▸ ExamplePlugin ▸

Search ExamplePlugin

Organize ▾ Open Include in library ▾ Share with ▾ New folder

Libraries: Documents, Music, Pictures, Subversion, Videos

Computer: Local Disk (C:), Data (D:), VITShared (\\vitnas) (S:)

Name	Date modified	Type	Size
builds	21-11-2016 12:42	File folder	
Plugins	21-11-2016 12:46	File folder	
Main.vi	21-11-2016 1:44	LabVIEW Instrume...	65 KB
PluginMain.aliases	21-11-2016 12:46	ALIASES File	1 KB
PluginMain.lvlps	21-11-2016 12:46	LVLPS File	1 KB
PluginMain.lvproj	21-11-2016 12:46	LabVIEW Project	11 KB

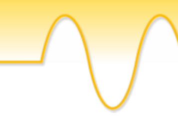
Plugins Date modified: 21-11-2016 12:46
File folder

- Reasons to use plugins
- Basics of a plugin architecture
- **Why we used packed project library plugins on cRIO targets**
- Software platform overview
- The Do’s and Don’ts of PPL plugins on cRIO

Why did we use PPLs plugins on cRIO targets

- Background info and requirements:
  - Customer has multiple production plants worldwide
  - Multiple production lines with multiple test stations
  - 1 Software (and Hardware) platform → maintainability
  - This platform should be modular and extendable
  - Previous production lines use cRIO (controlled by PXI)
  - cRIO should work standalone
  - Multiple types of DUTs, small differences in test.
  - Flexible way of adjusting / extending the code, in the right place

Why did we use PPLs plugins on cRIO targets



cRIO software platform should be modular and closed:

- Modular:

  Each component is separated and has an abstraction layer

  ➔ Plugins made with LabVIEW classes

- Closed:

  Developers can only make changes where this is allowed.

  Changes effective on the cRIO platform without rebuilding.

  ➔ Compiled code into a Packed Project Library

- Reasons to use plugins
- Basics of a plugin architecture
- Why we used packed project library plugins on cRIO targets
- **Software platform overview**
- The Do’s and Don’ts of PPL plugins on cRIO

Software platform overview

RT MAIN

DUT
SEQ PLUGIN
Sequence Interface
Handshake

UTILS
Device Base

Platform

Product Specific Plugins

DUT A
DUT B
StartComm.
StopComm.
INI FIle
PXI
Digital Input
PLC
Scanner
CANComm

Utils and devices:

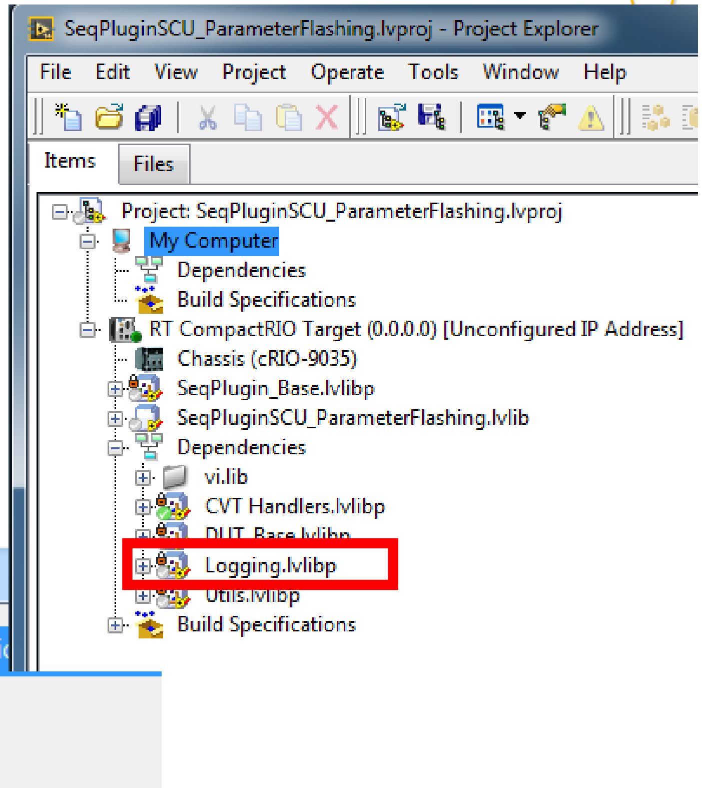
Agenda

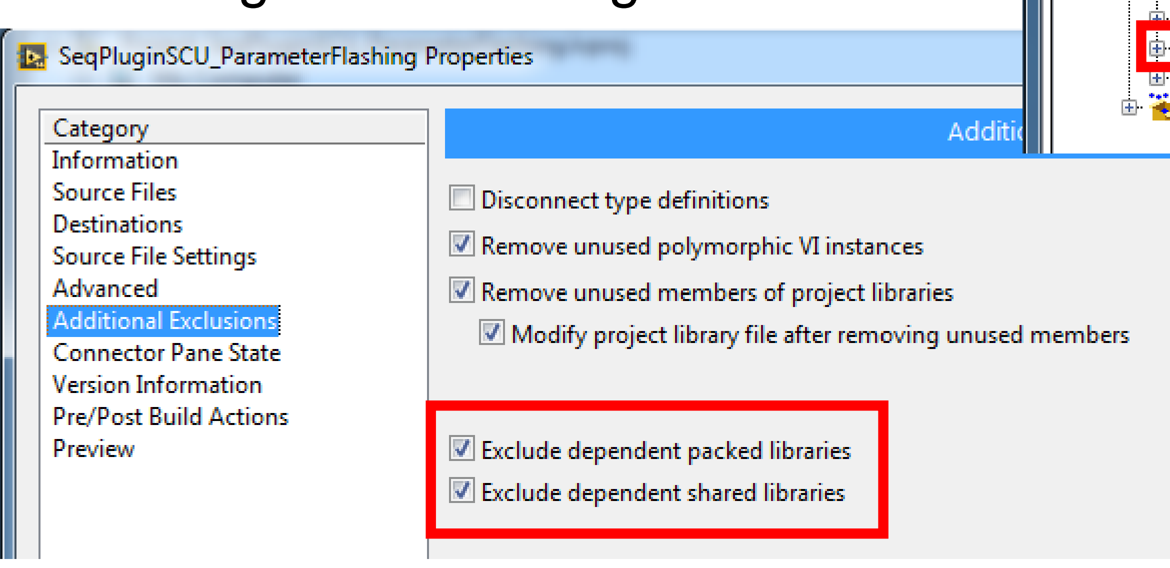
- Reasons to use plugins
- Basics of a plugin architecture
- Why we used packed project library plugins on cRIO targets
- Software platform overview
- **The Do’s and Don’ts of PPL plugins on cRIO**

The DO's and Don'ts of PPL plugins on cRIO

Use separate project files for each PPL

- Dependencies are shown
- Easy linking to correct PPL
- Exclude dependent PPLs in builds
- Configuration management





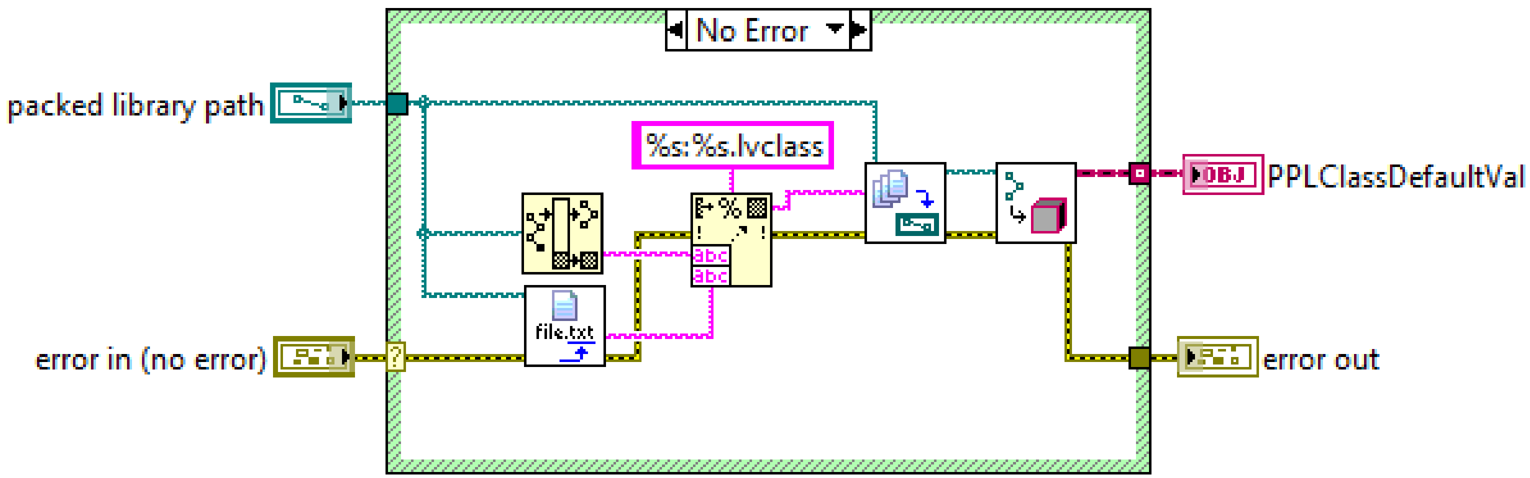
The DO's and Don'ts of PPL plugins on cRIO

Make naming conventions and stick to them !!

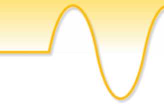
- Plugin architecture = Dynamic loading
- Keep the PPL name the same as the LabVIEW class

✓ "MyPlugin.lvlibp:MyPlugin.lvclass"

✗ "MyPlugin.lvlibp:MyPluginForTest.lvclass"



The DO's and Don'ts of PPL plugins on cRIO



Dynamic loading with dependent PPLs (ex. DUT with device)

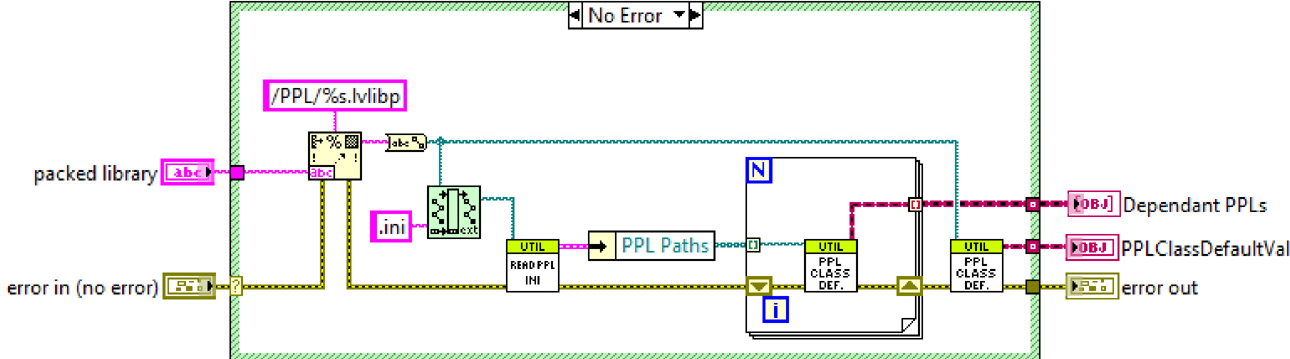
- First load your dependencies into memory

✔ Define a standardized method to load dependent PPLs

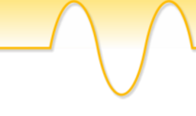
☒ Don't build your dependencies into the same PPL

→ Duplicates in memory, cross linking (ex. Base classes in main)

→ Exceptions – no dependencies to other PPLs



The DO's and Don'ts of PPL plugins on cRIO



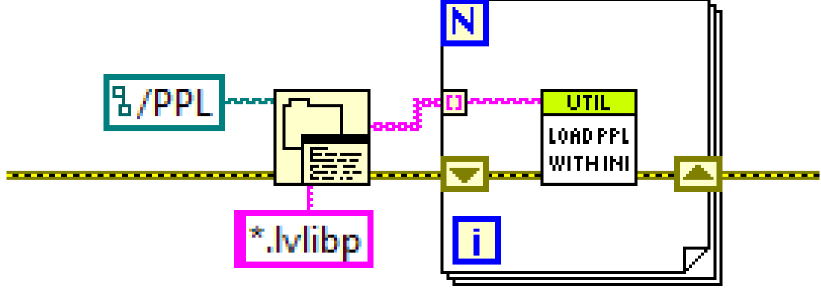
Preloading PPLs at startup saves time

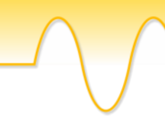
❌ Loading PPLs when they are needed, costs time at that moment.

If they have loading issues, you have to wait untill they are loaded

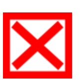
✅ Loading PPLs at start up does not cost extra time

in execution + you know if your PPLs have errors at startup





Rebuilding when a base PPL is changed (version change)



A new version of your base PPL breaks the executable and inheritance.

Rebuild your complete plugin structure

Rebuild your executable

→ check out the available tools

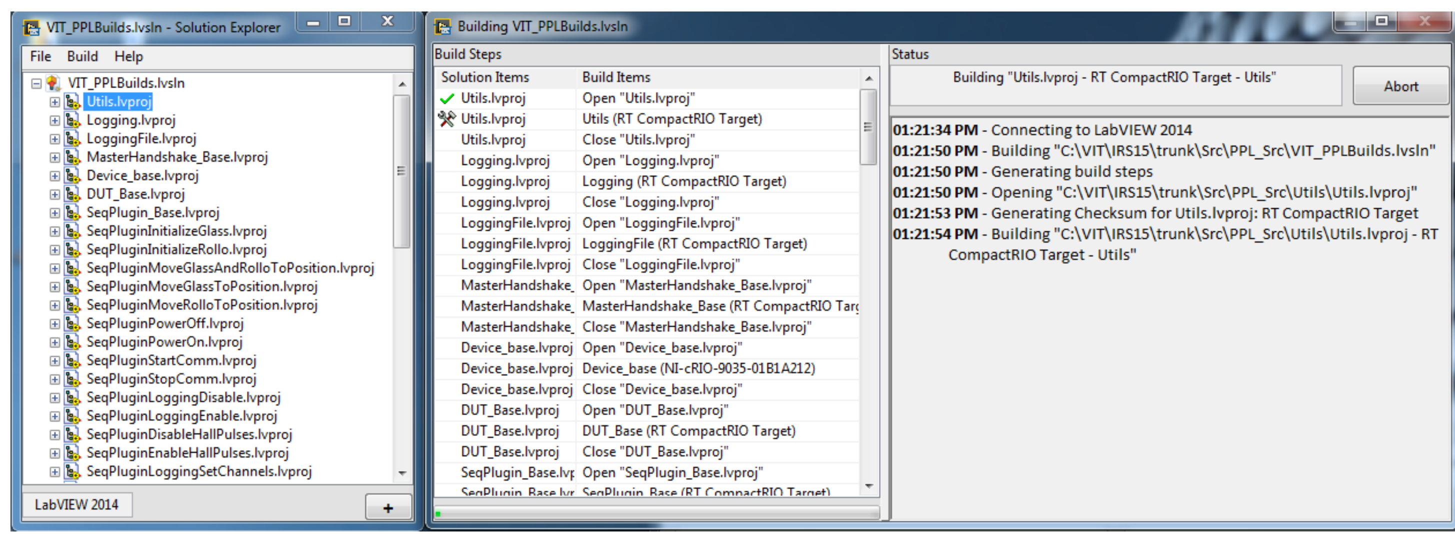
MGI Solution Explorer

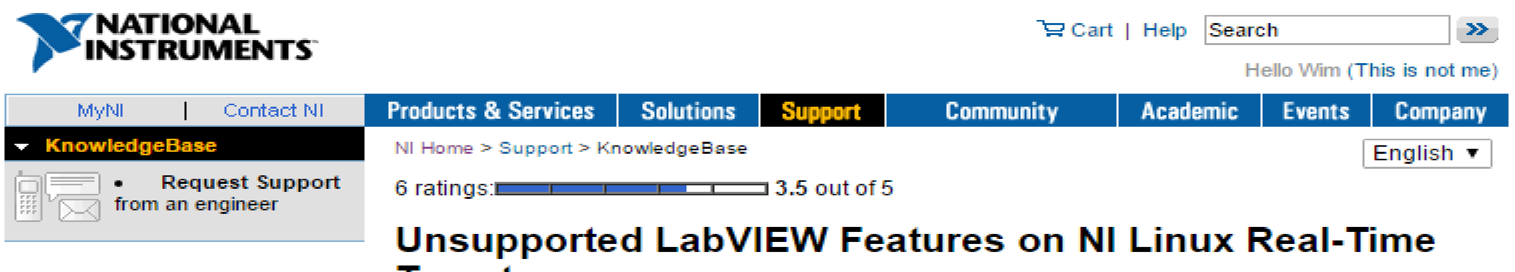
http://www.mooregoodideas.com/mgi-solution-explorer-2/

Automate your PPL build

Configure a list of project files in the correct order of building

Option to build for debug or release





Summary of unsupported features:

- Property nodes for UI properties
- Modifying Front Panel Objects of RT Target Vis
- Dynamically loading without first compiling for RT
- Debugging Reentrant VIs

Don't use UI property nodes on Linux RT: (not only in PPLs)

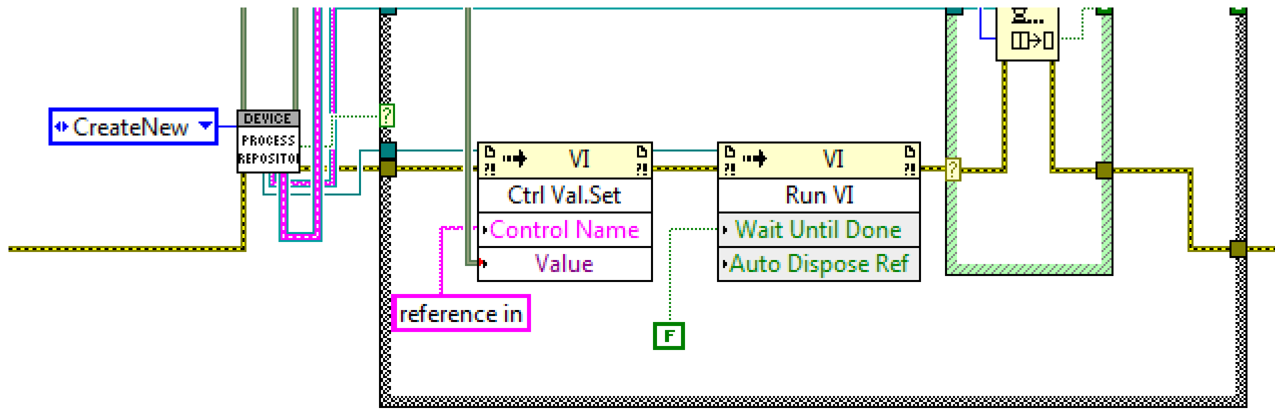
- ☒ There is no front panel for VIs that run on the RT target.

- ☒ In some cases, you can establish a front panel connection
  → use some unsupported LabVIEW features
  →The UI updates are asynchronous !!!

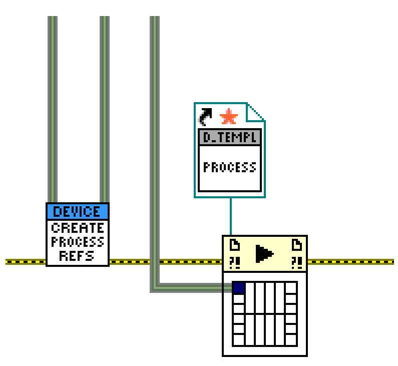
- ✓ Most RT applications do not need a UI for every VI
- ✓ Workaround with other types of controls

Starting active objects in plugins:

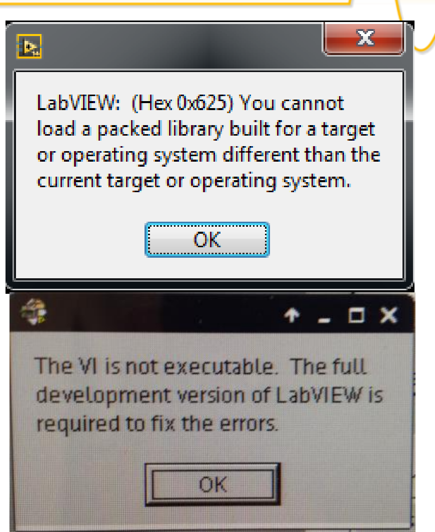
☒ Don't use the Set Control value method



☑ Use start asynchronous call

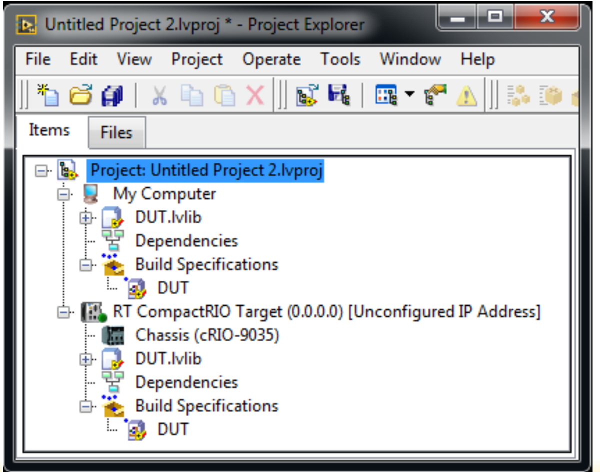


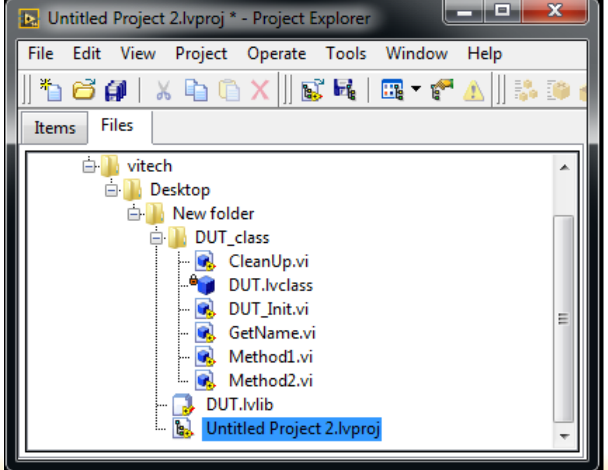
The DO's and Don'ts of PPL plugins on cRIO

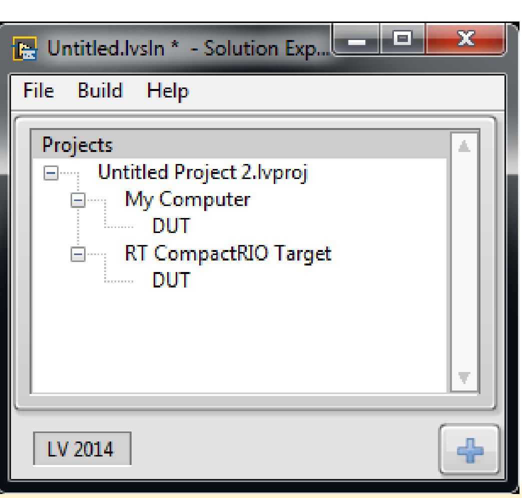


Loading a PPL on Windows and Real-time:

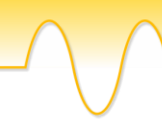
- ☒ Loading a PPL for RT on windows
- ☒ Loading a PPL for windows on RT
  - Vague error popup / Blocks deployment
  - Only visible on cRIO monitor
- ✓ Create separate build specs and build locations
- ✓ You can load the same lvlib in PC AND RT Target







The DO’s and Don’ts of PPL plugins on cRIO



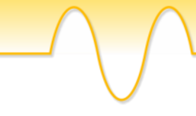
During development connect a monitor to your cRIO (if possible)

- ☒ Reentrant VI’s will not show when you run from PC
- ✓ On your cRIO display, they will show.

- ☒ Connecting to application (with PPLs) will cause errors
- ✓ Remote panel connection will work

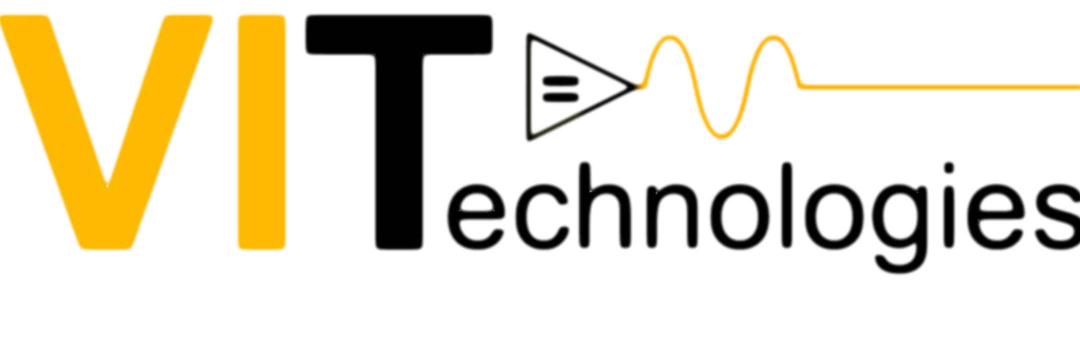
- ☒ Highlight execution crashes LabVIEW
- ✓ Solved in LabVIEW 2016

The DO's and Don'ts of PPL plugins on cRIO



Summary:

- Use separate project files for each PPL
- Make naming conventions
- Dynamic loading, standardized method for dependencies
- Preload PPLs at program startup
- Rebuild all PPLs and exe when a base plugin changed
- Automate your builds
- Asynchronous call to start active objects
- Separate builds for windows and RT
- Use a monitor on cRIO or remote panels



Visit our booth.