LabVIEW Project Provider Developer's Guide

Table of Contents

Overview

The LabVIEW project window has a plug-in architecture to allow for easy extension of the project window features and capabilities. A Provider is code that plugs into this framework to provide additional set of functionality to a project. This functionality could be

- global - that applies to all items in the project or
- item-specific – that applies to specific item types

Primary Provider vs. Secondary Provider

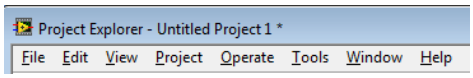
Any plug-in can be either a Primary or a Secondary Provider. *A primary provider* is responsible for putting one or more new item types into the project tree. A *secondary provider* adds additional functionality to existing items, but is not responsible for creating or managing those items. Hence a secondary provider does not cause things to be displayed in the tree, but simply attaches to items shown by other primary providers.

For example, the providers that enable addition of a RESTful Web service and a Zip file to the Build Specifications in the project tree are primary providers. Conversely, the Source Code Control provider and VI Analyzer are some examples of secondary providers, as they do not create any new items in the project tree.

More details on the differences in programming between a primary and a secondary provider will follow in later sections.

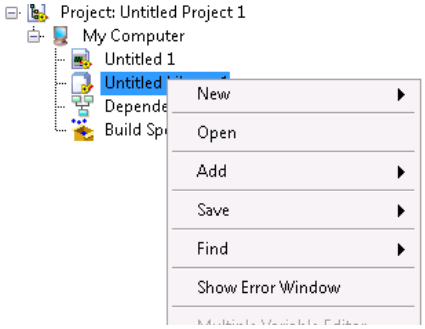
Features

1. **Menu items**



There are two basic types of menu items for a provider. First, there are global menu items, which appear in the menu bar of the project window. These menu items appear independent of the item selected in the tree. Second, a provider can specify menu items which apply to items of specific type in the project tree.

2. **Popup menu items**

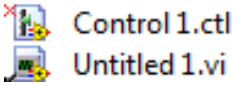


A provider can customize the popup menu that appears when an item is right-clicked. The menu can also be customized depending on the item selected and the state of the item.

3. **Toolbar buttons**

A provider can insert toolbar buttons into the toolbar of the project. These can be global buttons or specific to an item type. Global toolbar buttons are available regardless of which item is selected in the project tree. Item-specific toolbar buttons only show when an item of a specific type is selected in the tree.
Each toolbar button has a different icon. Toolbar buttons added by a specific provider can be removed from the toolbar by right clicking on the project toolbar and deselecting the appropriate provider.

4. **Modify the name or icon of an item in the tree**



A provider can modify the name or icon of an item in the tree. Only primary providers should assign a new name and a new icon to an item. Primary or secondary providers can also add an overlay to the existing icon of an item. Icon overlays can be used to indicate the status of an item. For example, an overlay can be used to indicate a VI is checked out from source code control.

5. **Save information related to an item**
A provider can store and retrieve data related to an item. A provider can add a new property or set the value of a property for an item and then either save this information to the project file on disk or keep it only in memory.

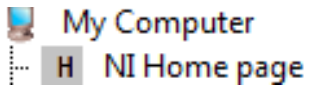
6. **Customize double click behavior for an item**
A provider can customize the actions that occur when an item of a specific type is double clicked on, e.g., it can call an external application or a VI based editor to view or edit the item.

The following features can be implemented by primary providers only:

7. **Drag and drop items to and within a project**
An item in the tree can be the source or the target for a drag-and-drop operation. The provider determines the desired behavior when doing both of these operations.

When an item of type *A* is being dragged onto an item of type *B*, the primary provider for *A* tells the tree that *A* is draggable and that this item is being a drag source. The primary provider for *B* is notified that it is being a drop target and an item of type *A* is being dropped onto it, so it responds accordingly.

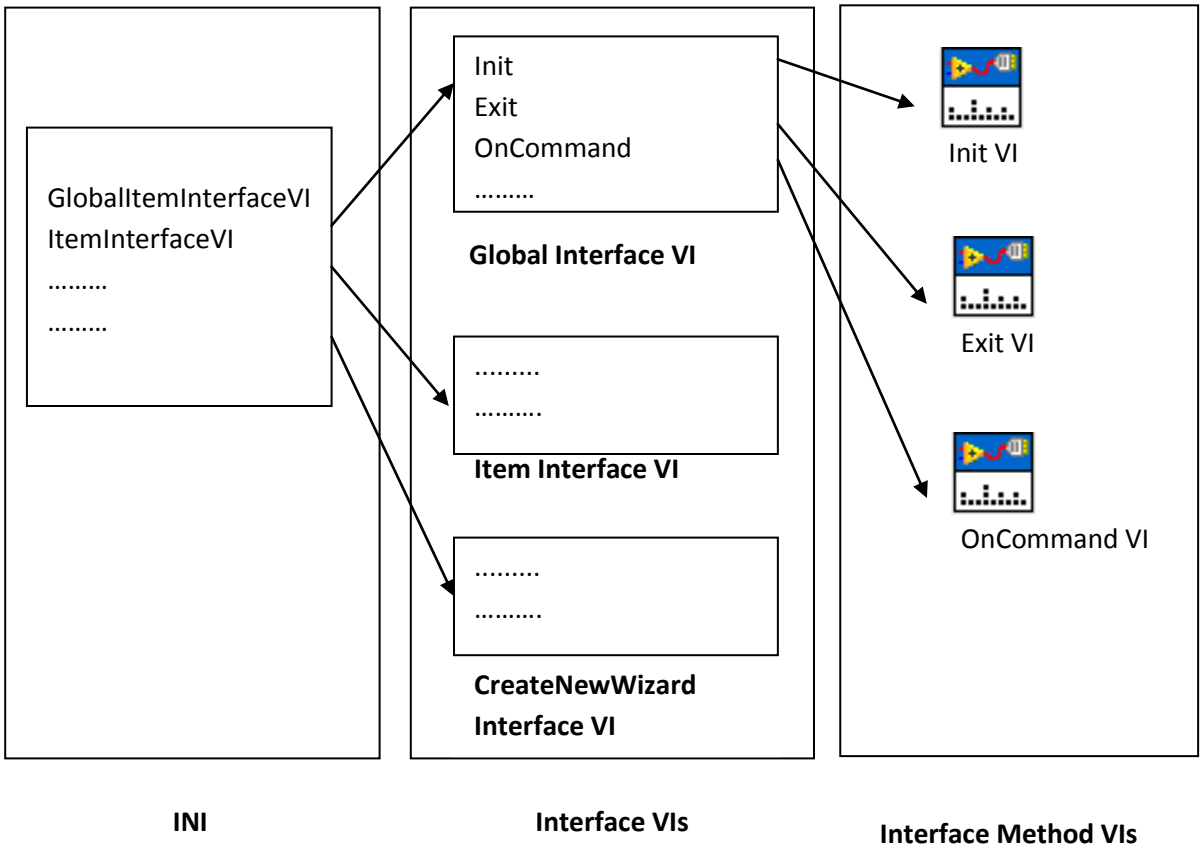
8. **Add items to the project tree**



Provider can provide a wizard to take users through the process of creating an item of the desired type in the tree.

Framework

A project provider is comprised of an INI file and a set of VIs. The INI file contains configuration information for the project. The VIs implement a set of interfaces that provide all the functionality of the provider.



I. INI File

Each provider contains an INI file that describes the supported type as well as the interfaces that are defined for that provider. Supported type represents the type ID (GUID) of the project item the provider applies to.

Format

The format of an INI file is:

```
[Provider]
Token=value
Token=value
  .
  .
```

Here is an example INI:

```
[Provider]
SupportedType={D60740D6-F254-4BBC-5675-8858F35B810E}
GlobalItemInterfaceVI=..\SecondaryProviderExample1\SecondaryProviderExample1_Global_Interface.vi
ItemInterfaceVI=..\SecondaryProviderExample1\SecondaryProviderExample1_Item_Interface.vi
ProviderInterfaceVI=..\SecondaryProviderExample1\SecondaryProviderExample1_Provider_Interface.vi
IsPrimary=0
LicenseName=None
InterfaceVersion=1.0
Signature=J7W9927AAAAAA5TBNTSSTTLWR29CNT9X
```

This INI file needs to be resigned. Refer section Signature for information on digitally signing the INI.

Supported Type/ Interface

A secondary provider needs to be able to specify which types of items it attaches to. There are two main ways for this to happen. The secondary provider gives the framework an explicit list of item types it wants to attach to by specifying their type IDs (GUIDs) in the INI file as supported type. The supported type token in a primary provider's INI file acts as the type ID of the item the primary creates. Another way for the secondary provider to attach to items is by specifying an interface identifier as the supported interface token in the INI file. In this case, the framework asks each primary provider whether it supports the specified interface and the framework attaches the secondary provider to each item whose primary provider supports the interface. On the primary provider side, the provider can specify the supported interface token in its INI to tell the framework that it supports that interface. Following are the supported interfaces and the items they apply to:

1. Build_Interface
   All builds i.e. the items under Build Specifications.
2. Analyzer_Interface
   All items supported by the VI Analyzer toolkit of LabVIEW.
3. Deploy_Interface
   All items that can be deployed, e.g. VIs, shared variable libraries, RESTful Web services.
4. SCC_Interface
   All items that can be saved using source code control. It applies to all default items permissible under My Computer in the project tree.

For example, a Source Code Control provider can add menu items for 'Check In' or 'Check Out' to all items which can be under source code control directly or have children in the project tree which are under source code control. This includes such things as VIs, documents of all kinds,

the project window itself, and others. The SCC Provider will tell the framework to only attach to items which implement the ' SCC_Interface ' interface. Primary providers will tell the framework that they implement the 'SCC_Interface' interface and the framework will attach the SCC secondary provider to the correct Items.

Tokens

INI files can contain the following tokens:

Token	***Description***
SupportedInterface	*Indicates which interfaces are supported by the provider.*
SupportedType	*Specifies a GUID that represents the type of project item. Must be formatted as following: {XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}*
Alias	*Is a unique string that like the GUID, is used to identify the type of a project item (applies to Primary providers only). The SupportedType token in a secondary provider can point to the Alias of the item type it wants to attach to instead of the GUID.*
IsPrimary	*1 – Primary provider (defines a new item type in the project). 0 – Secondary provider (adds functionality to existing item(s) in the project).*
ItemInterfaceVI	*Defines the VI that enumerates the "Item" interface.*
ProviderInterfaceVI	*Defines the VI that enumerates the "Provider" interface.*
GlobalItemInterfaceVI	*Defines the VI that enumerates the "Global" item interface.*
CreateNewWizardInterfaceVI	*Defines the VI that enumerates the "CreateNewWizard" interface.*
SourceControlInterfaceVI	*Defines the VI that enumerates the "SourceControl" interface.*
DeployInterfaceVI	*Defines the VI that enumerates the "Deploy" interface.*
BuildInterfaceVI	*Defines the VI that enumerates the "Build" interface.*
CreateNewWizardHost (Primary providers only)	*Defines which "New" menu to add to e.g., Source, Build.*
InterfaceVersion	*Specifies the interface version used by the provider. Currently, this must be set to **1.0**.*
Signature	*Digital signature – generated by NI.*

Signature

To get your INI files signed please contact the LabVIEW Partner Program (labviewpartnerprogram@ni.com) with your INI files and we will sign them for you. Do not modify the INI file in any way once it is signed or the provider will not load.

II. Interface VIs

The framework specifies a list of interfaces that must be implemented by providers. Each interface VI is a list of paths to dispatch VIs that point to each of the required methods that the providers must implement. Only the methods that the provider will provide need to be implemented**.** The VIs that implement these methods must conform to the particular connector description but the contents of the VI can be completely customized. Once a provider-based event occurs the provider framework will run the appropriate VI if a method has been customized for that event in the provider. For example if Item.OnSelect has been implemented, it will be called when project item is selected.

The following interfaces are available for a provider:

1. Build (Primary providers only)

Defines events that apply to builds, i.e. items under Build Specifications, in the project tree.

Interface Method VIs:

Invoke	Preview	IsRunnable	Run	OnDuplicate

2. CreateNewWizard (Primary providers only)

Defines events that occur while adding items of new types to the project tree.

Interface Method VIs:

Init	Invoke	Finalize	GetNewItemInfo	GetCreateNewWeight	IncludeItem

3. Global

Defines global-level events that are not tied into a specific item type.

Interface Method VIs:

Init	Exit	OnCommand	OnUpdateCommand

4. Item

Defines events that occur to individual items in the project tree.

Interface Method VIs:

Init	CanDragToExternalWindow	OnSelect	CanRename
Exit	WasDroppedOnItem	OnUnselect	OnRename
OnCommand	WasDroppedOnVI	OnDblClick	ValidateRename
OnDropFiles	GetCreateNewCategories	OnPopupMenu	PrefersFPHeap
CanDropExternalData	GetAddCategories	CanDoProperties	OnDoHelp

CanDropItem	OnWizardComplete	OnDoProperties	CanDoHelp
OnDropItem	NotifyChanged	CanDelete	
CanDragToProjectWindow	OnUpdateCommand	OnDelete	

5. ProjectDeployItem

Defines events that support deploying of an item in the project tree.

Interface Method VIs:

GetSupportedCommands	GetDeployState	NotifyCommandCompleted

6. Provider

Defines events that occur to multiple items in the project tree.

Interface Method VIs:

InitItems	OnSaveProject	LoadComplete
OnCommand	OnSaveForPrevious	LoadCompleteWithWarnings
Startup	OnSaveForPreviousEx	OnUpdateCommandBegin
Shutdown	OnSaveForPreviousWithWarnings	OnUpdateCommandEnd
OnPopUpMenu	NotifyChanged	

7. SourceControl

Defines events that apply to source code control functionality.

Interface Method VIs:

GetRequiredFiles	GetDependents	GetCallers

The interfaces available for providers to implement can be found at "%LabVIEW_Install_Directory%\resource\Framework\Providers\API\". They are stored as "%NameOfInterface%_Interface.ctl files. A brief overview of some of these interface methods can be found in the attached document "Interface VIs".

III. Directory Structure

All the VI- based providers and provider APIs can be found at "%LabVIEW_Install_Directory% */resource/Framework/Providers/".* This directory further contains following directories:

1. **API**

This directory contains type-definition controls defining the specifications for the various interface VIs. It also contains several helpful VIs that can be used from within the provider framework to perform frequent provider tasks, e.g., getting or setting project item properties.

2. **Common**
   This directory also contains reusable VIs that are useful to writing providers. They are simple VIs and unlike the API VIs do not call into the provider framework.

3. **GProviders**
   This directory was introduced in LabVIEW 2011. It is mandatory to have all the INI files in this directory for LabVIEW versions post-2010 for the providers to be loaded. This change is backward-compatible: a provider with its INI in the GProviders directory behaves the same way in a previous version of LabVIEW. It is highly recommended to follow this standard while building providers with any version of LabVIEW.

4. **Icons**
   This is the recommended location for all your provider icons. You can place icons at other locations as well but remember to specify the path with respect to the INI file location when setting the icons in provider VIs.

5. <***Provider***>
   Each provider should be located in a separate directory under "%LabVIEW_Install_Directory% /resource/Frameworks/Provider". This is not strictly enforced but is highly recommended.

IV. Working directory

The working directory for a provider is where its INI file is located. All relative paths to interface VIs in the INI should be specified with respect to this location.

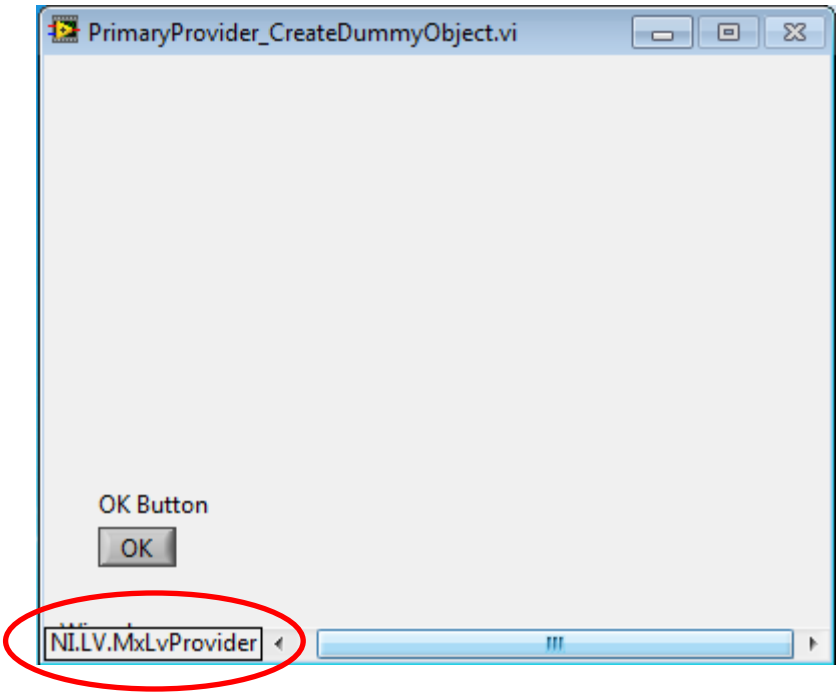
GProviders is the working directory for all providers in LabVIEW 2011. This should be kept in mind while specifying relative paths into API/ Common VIs, e.g., the icon file path should be relative to the GProviders directory and not the VI in which the provider is setting the icon.

V. API VIs

The "API" and the "Common" directories provide several reusable VIs that come handy while building providers, for example, VIs that set the icon of an item in the project tree, set the menu or the toolbar options. The VIs in the API directory call into the provider framework to implement their functionality. A descriptive list of these VIs can be found in the attached document "API VIs".

VI. Context

The VIs under provider framework run in the "NI.LV.MxLvProvider" context. This can be observed on the front panel of a VI running the provider framework, as shown below:

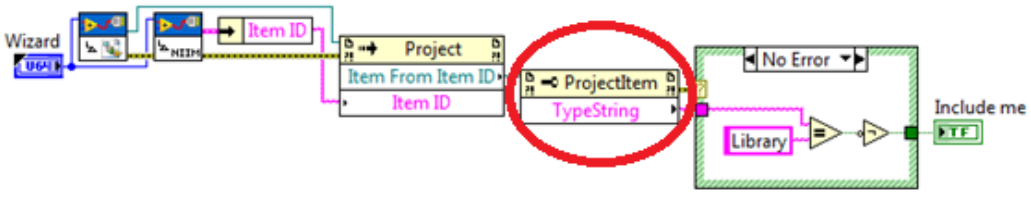


This can make the VIs behave differently than VIs that run under the "<ProjectName>/ My Computer" context. Also, the front panels of VIs to be executed under provider framework should be designed to not display this information to the user by hiding the scroll bars.

VII. GUIDs

GUID and Type strings identify items of a specific type. GUIDs can be used as supported types to build a provider for item of a specific type.

One example usage of the Type string in providers is to identify when to include the option to create the new item, e.g,



Here is a list of important GUIDs and Type strings.

Item type	Type String	GUID String
Virtual Folder	Folder	{D60740D6-F254-4BBC-5675-8858F35B820E}
Auto Populating Folder	Folder	{D60740D6-F254-4BBC-5675-8858F35B820E}
VI	VI	{D60740D6-F254-4BBC-5675-8858F35B810E}
Control	VI	{D60740D6-F254-4BBC-5675-8858F35B810E}
(Non-LabVIEW) file	Document	{D60740D6-F255-4BBC-5675-8858F35B820E}
LLB-file	Folder	{D60740D6-F254-4BBC-5675-8858F35B820E}
Lvlib file	Library	{ABC740D6-F254-4BBC-5675-8858F35B820E}
LabVIEW class	LVClass	{EFD740D6-F254-4BBC-5675-8858F35B820E}
Class datatype	Class Private Data	{64A9BF48-4C55-45DB-8F18-9C796DA0C113}
Property Definition Folder	Property Definition	{81C68620-8BE3-4643-B04A-4E83AA6363D2}
LabVIEW XControl	XControl	{2E4BD3AC-4E04-45C9-B6C5-AD138962C435}
XControl Method	Method VI	{DAABD3DC-4F44-4C3D-8BA3-E5D035A4F27A}
XControl Property	Property Folder	{93C4A07A-46E9-442F-AF0F-D6C6039546D1}
XControl Property Read VI	Property VI	{DAABD3DB-4F44-4C3D-8BA3-E5D035A4F27A}
XControl Property Write VI	Property VI	{DAABD3DB-4F44-4C3D-8BA3-E5D035A4F27A}
XControl ability VI	Ability VI	{DAABD3DA-4F44-4C3D-8BA3-E5D035A4F27A}

DAQmx Task	NI-DAQmx Task	{0A806145-1BF3-3A50-0B5E-F969F56C8E2A}
DAQmx Virtual Channel	NI-DAQmx Channel	{B5F05770-7C71-3A03-C9FB-F73F35629FC1}
DAQmx Scale	NI-DAQmx Scale	{BC977C97-1833-3D51-4EF6-D82E6838A8F3}
Shared Network Variable	Variable	{9BA597C5-4996-4622-B9BB-444431834D0D}
Hyperlink	Hyperlink	{CC472C20-0441-48DC-AF25-3E82ECC9376F}
Dependencies	Dependencies	{0D75D917-83D7-9871-AA09-A0FFD6A8099B}
Build Specifications	Build	{0C750917-83D7-9871-8908-BB4ED6A8099B}
Application exe	EXE	{9A75366A-79D3-4BFD-9532-E3070185C1E8}
DLL	DLL	{20A41099-3F2C-42C3-9544-7ABCC1E6CB0D}
Zip	Build Specifications	{51F0E16F-7FA4-4E9F-AE30-C81D9D0444B0}
My Computer	My Computer	{CEFE1B10-1732-4678-A70A-299293455410}
Source Distribution	Source Distribution	{15DA4F9E-0591-4AB1-A339-C3B4D54902D8}
Packed Library	Packed Library	{E4492117-CED3-4F31-9F8D-E2118AE04F12}

Example Providers

This section refers to the Example Providers that were included with this document. These are built in LabVIEW 2009 but are compatible with the new changes in LabVIEW 2011.

I. Secondary Provider 1

Name : SecondaryProviderExample1

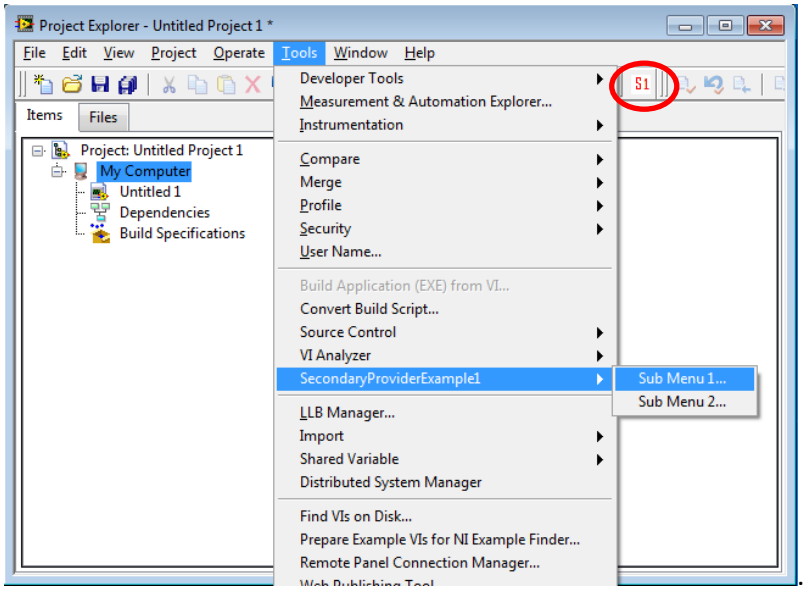
Features implemented : Global Menu items, global toolbar icon, item-specific popup menu, multiple item selection with popup menus.

Corresponding VIs

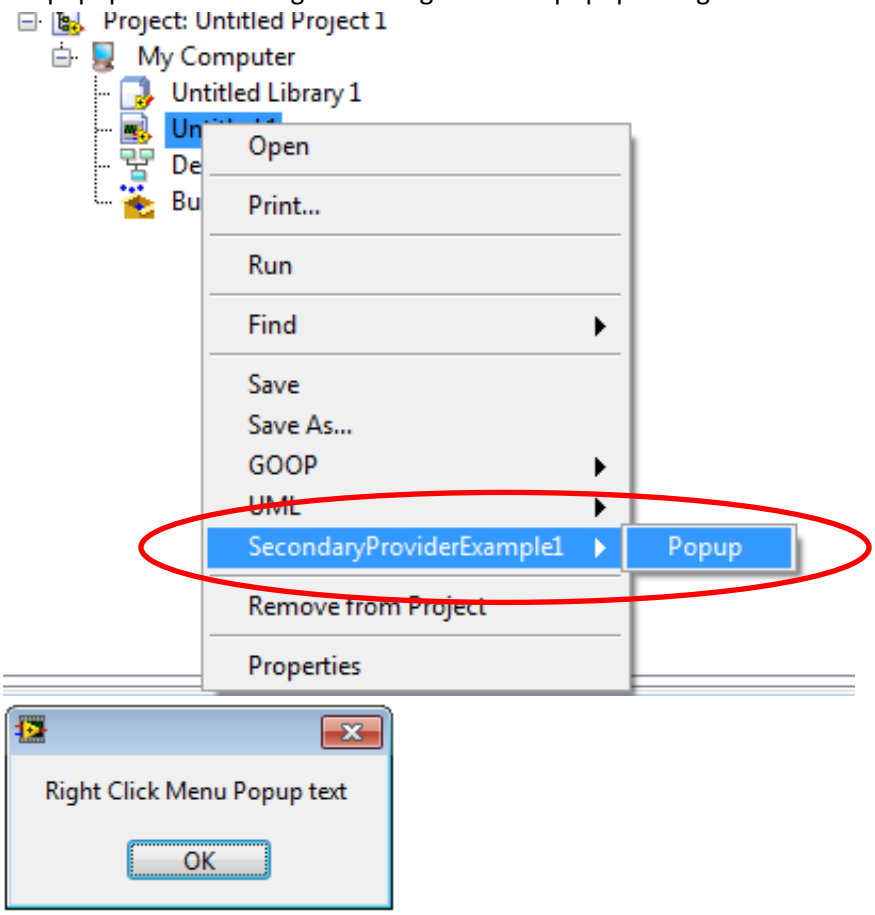
1. Global Menu items
   a. SecondaryProviderExample1_Global_Init.vi
   b. SecondaryProviderExample1_Global_OnCommand.vi
2. Global Toolbar Icon
   a. SecondaryProviderExample1_Global_Init.vi
3. Item specific popup menu
   a. SecondaryProviderExample1_Item_OnPopupMenu.vi
   b. SecondaryProviderExample1_Item_OnCommand.vi
4. Multiple item selection with popup menus
   a. SecondaryProviderExample1_Provider_OnPopupMenu.vi

Usage Instructions

1. Launch LabVIEW and create an empty project
2. Go to Tools and then SecondaryProviderExample1 in the menu bar. Click on Sub Menu 1 and Sub Menu 2 options to bring up customized popups.

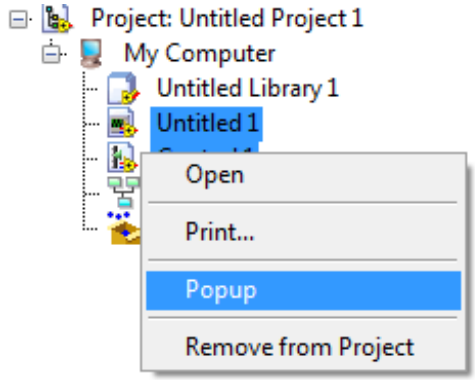


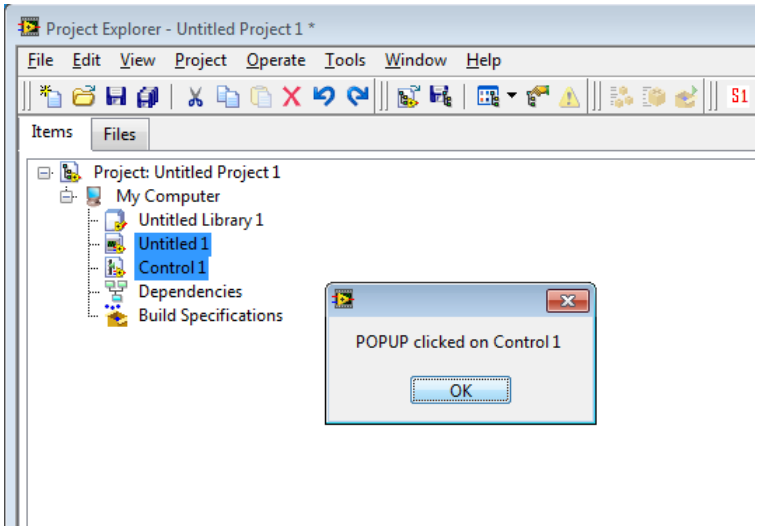
3. Add a VI and a control to the project. Right clicking on the VI or the control will show a popup menu. Clicking on it will generate a popup dialog.

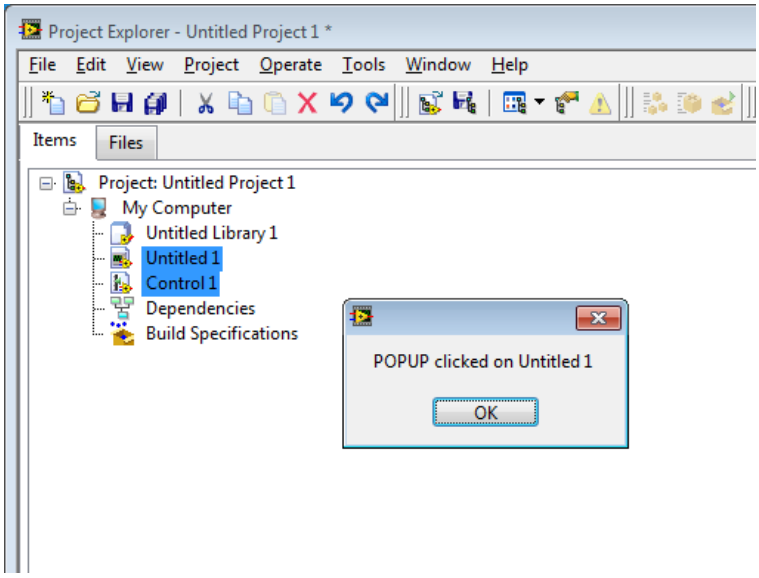


Note that the popup menu only shows up on right clicking on VIs and Controls. This is because in the INI file the SupportedType is set to only support items of type VI and Control.

4. Select multiple items of type VI or control and right click to generate a popup menu. You can still see the "Popup" menu option.







Note that clicking on "Popup" menu option when an item is selected invokes SecondaryProviderExample1_Item_OnPopupMenu.vi and when multiple items are selected invokes SecondaryProviderExample1_Provider_OnPopupMenu.vi.

II. Secondary Provider 2

Name : SecondaryProviderExample2

Features implemented : Item specific popup menu, icon overlays, reading and writing of item properties, persisting item properties.

<u>Corresponding VIs</u>

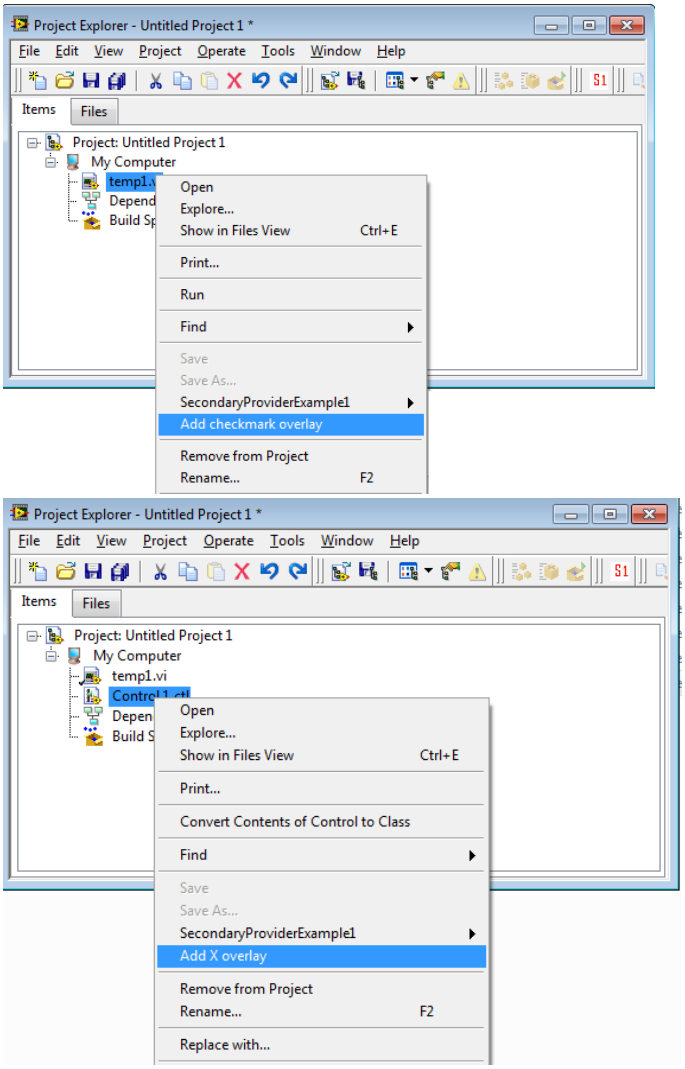
1. Item specific popup menu
   a. SecondaryProviderExample2_Item_OnPopupMenu.vi

b. SecondaryProviderExample2_Item_OnCommand.vi

2. Icon overlays
   a. SecondaryProviderExample2_Global_Init.vi
   b. SecondaryProviderExample2_Item_Init.vi
   c. SecondaryProviderExample2_Item_OnCommand.vi
3. Reading and writing of item properties; Persisting item properties
   a. SecondaryProviderExample2_Item_Init.vi
   b. SecondaryProviderExample2_Item_OnPopupMenu.vi

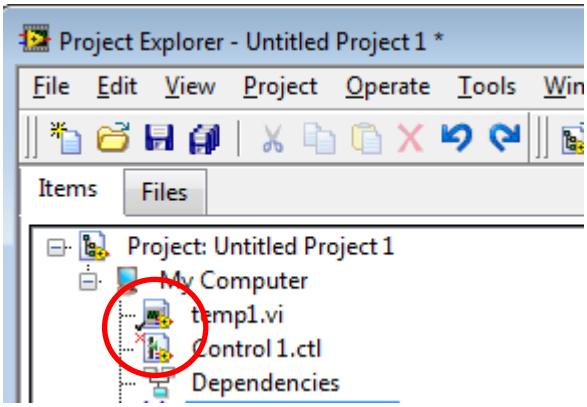
Usage Instructions

1. Launch LabVIEW and open a project. Add VIs and controls to the project and save them.
2. Now right click on a (saved) VI or a control in the project and you should see "Add checkmark overlay" or "Add X overlay" options respectively in the popup menu.

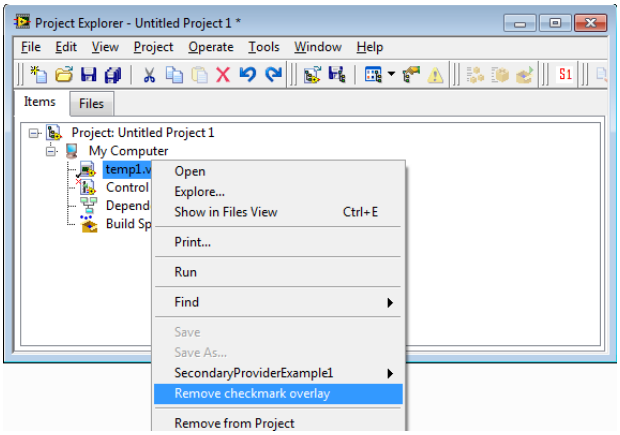


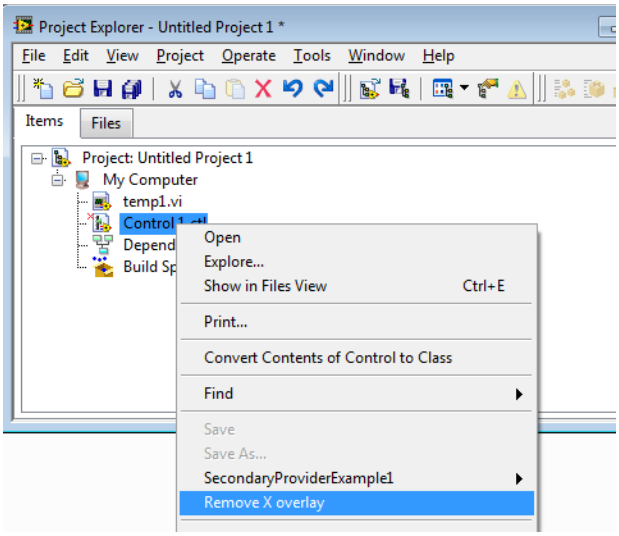
Note that the popup menu options would not be visible for unsaved VIs and controls.

3. Try these options and observe the icons overlays in items.



4. Right click again and this time the popup menu displays the option to remove the overlays for the VI or the control.





Note that the state of the VIs and controls (i.e. whether they are marked or not) is persisted with the project, and hence icon overlays are picked up on opening previously saved projects.

III. Primary Provider 1

Name : DummyPrimaryProvider

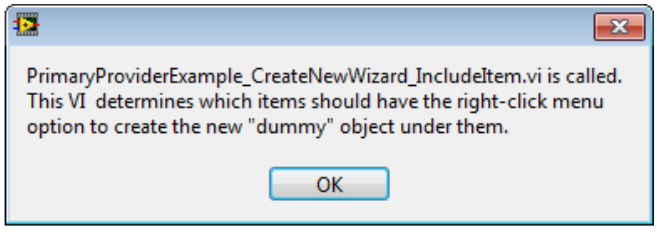
Features implemented : Adding a new item of type “Dummy” to the project tree, setting item name and icon, item specific popup menu, multiple items right click popup menu, item specific double click behavior, dragging and dropping.

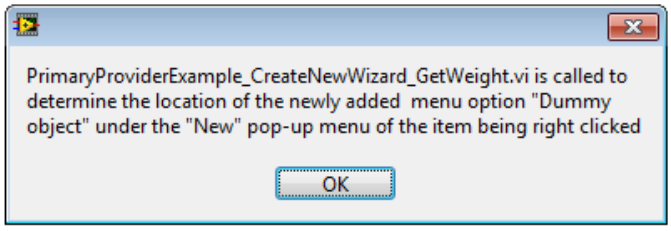
Corresponding VIs

1. Adding a new item type
   a. PrimaryProviderExample_CreateNewWizard_IncludeItem.vi
   b. PrimaryProviderExample_CreateNewWizard_GetWeight.vi
   c. PrimaryProviderExample_CreateNewWizard_Init.vi
   d. PrimaryProviderExample_CreateNewWizard_Invoke.vi
   e. PrimaryProvider_CreateDummyObject.vi
   f. PrimaryProviderExample_CreateNewWizard_Finalize.vi
2. Setting item name and icon
   a. PrimaryProviderExample_CreateNewWizard_Finalize.vi
   b. PrimaryProviderExample_Item_Init.vi
3. Item specific popup menu
   a. PrimaryProviderExample_Item_OnPopupMenu.vi
   b. PrimaryProviderExample_Item_OnCommand.vi
4. Multiple items right click popup menu
   a. PrimaryProviderExample_Provider_OnPopupMenu.vi
   b. PrimaryProviderExample_Provider_OnCommand.vi
5. Item specific double click behavior
   a. PrimaryProviderExample_Item_OnDblClick.vi
   b. PrimaryProviderExample_Item_OnCommand.vi
6. Dragging and dropping
   a. PrimaryProviderExample_Item_CanDragToProjectWindow.vi
   b. PrimaryProviderExample_Item_CanDropItem.vi
   c. PrimaryProviderExample_Item_OnDropItem.vi

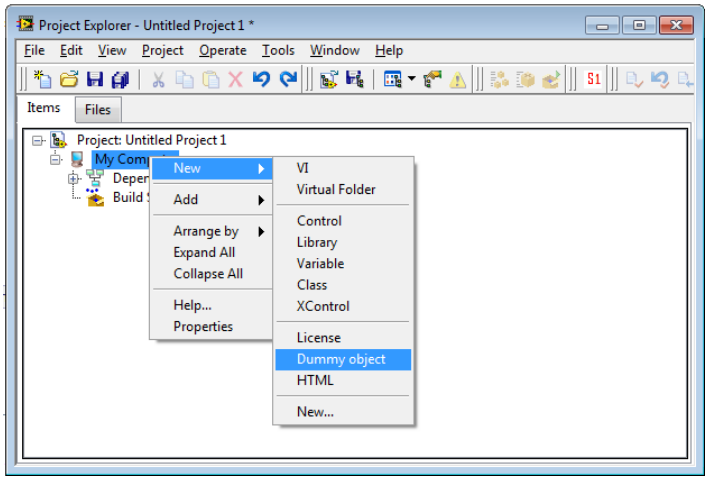
Usage Instructions

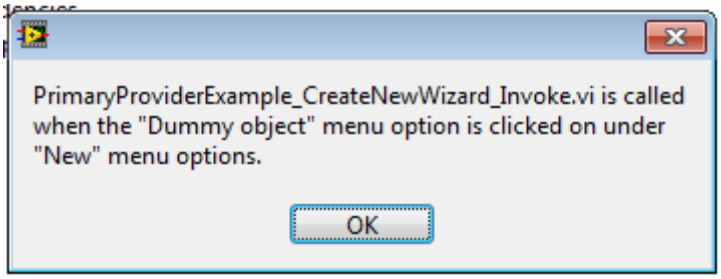
1. Launch LabVIEW and create an empty project
2. Right click on My Computer.



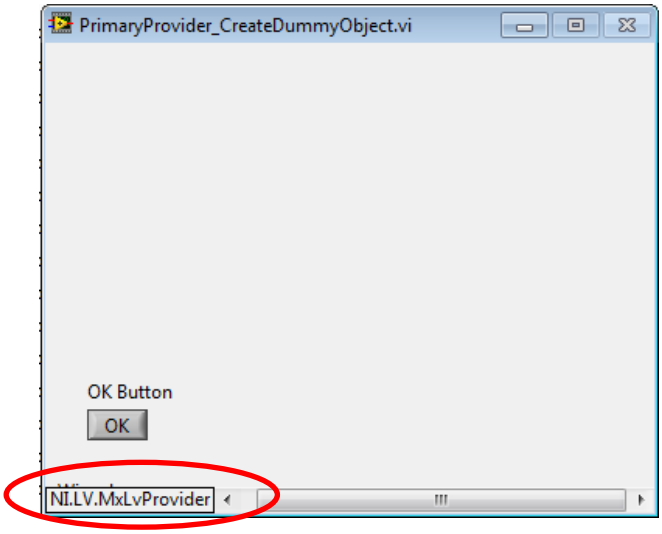


3. Select New > Dummy object

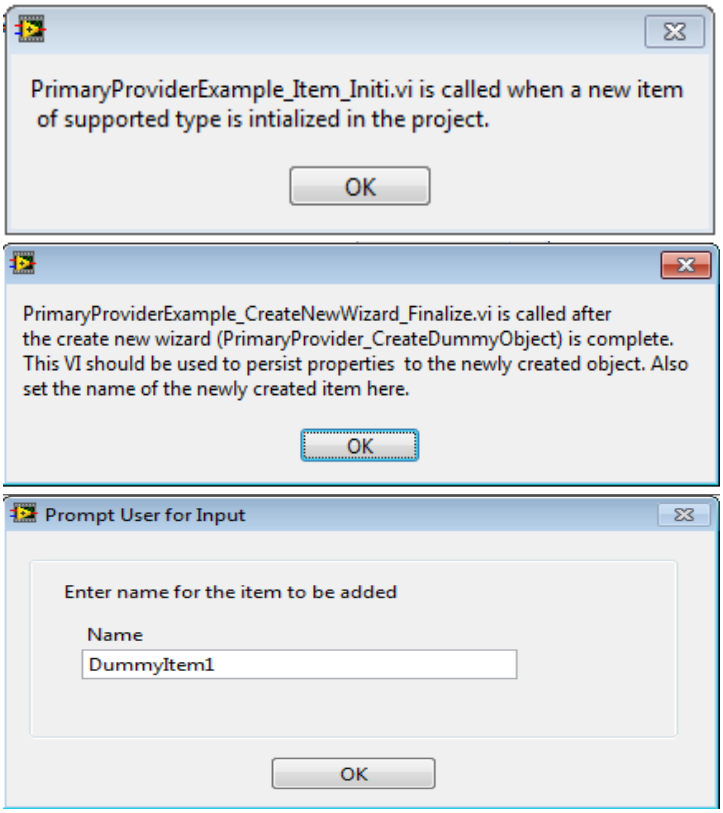




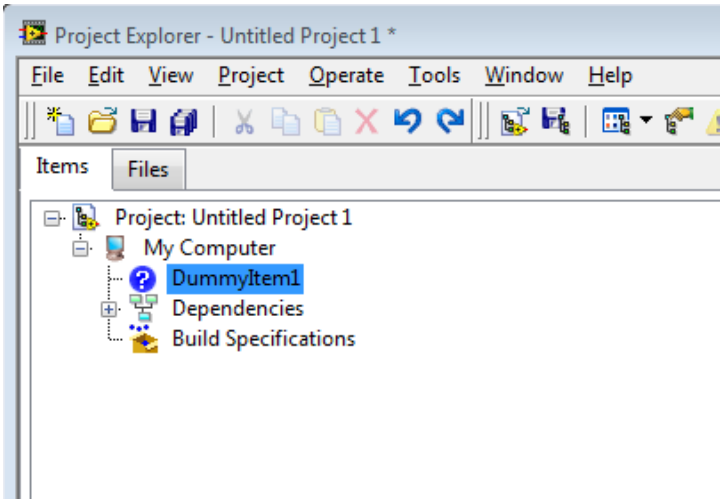
4. The custom VI (wizard) built to create the new item is called. Notice the context the VI is running in on the left bottom corner of the VI.



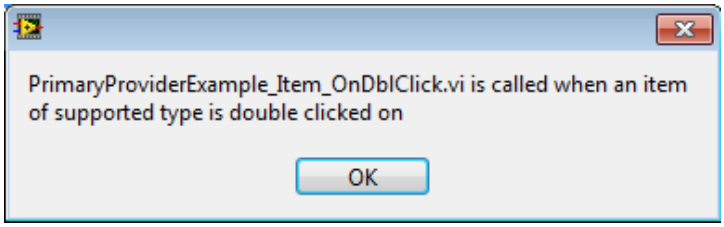
5. Click OK.



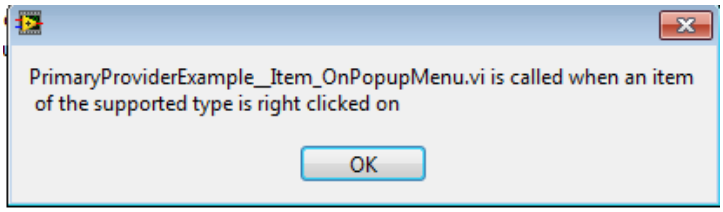
6. Notice the item "DummyItem1" is created in the project tree.

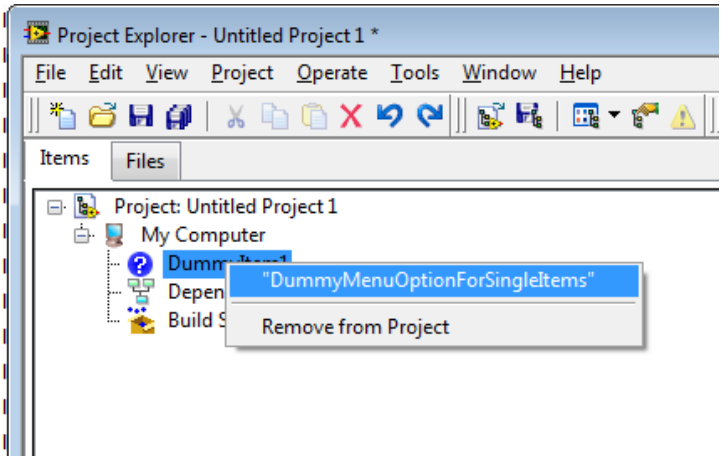


7. Double click on DummyItem1.

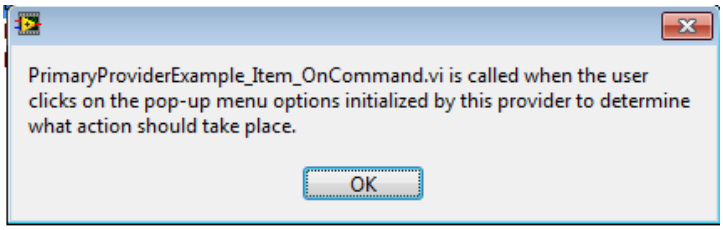


8. Right click on DummyItem1.

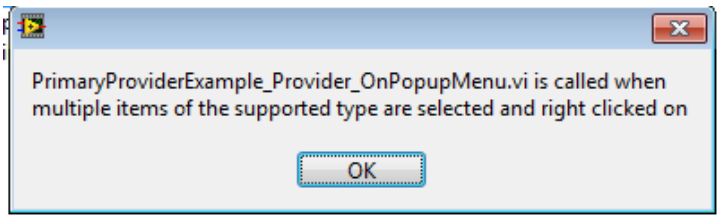


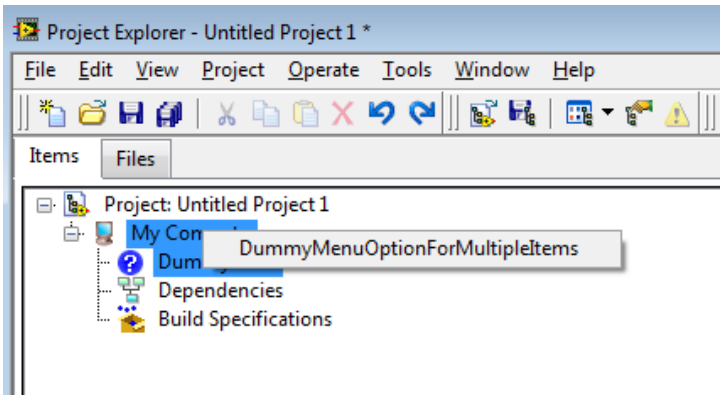


9. Select "DummyMenuOptionForSingleItems".

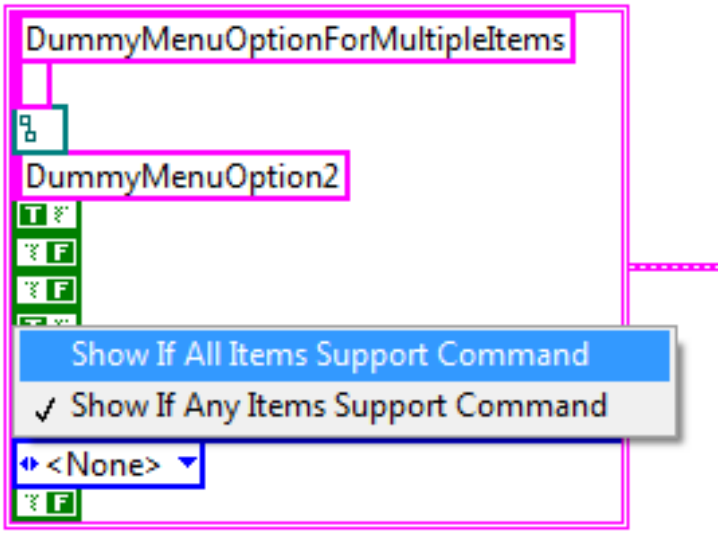


10. Select My Computer and DummyItem1 and right click

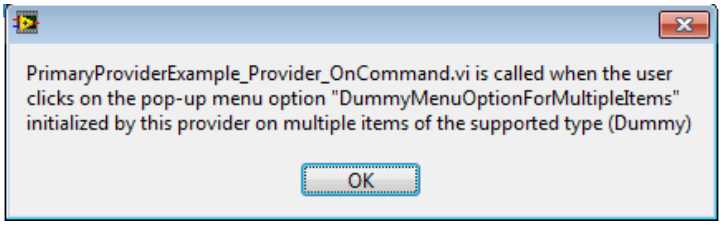




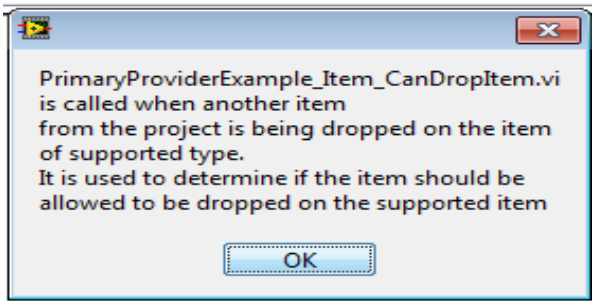
Note that the multiple item selection can be made to work only with item types supported by this provider (i.e. Dummy types) as well. To do this change the "PrimaryProviderExample_Provider_OnPopupMenu.vi" to "Show If All Items Support Command" from "Show If Any Items Support Command".

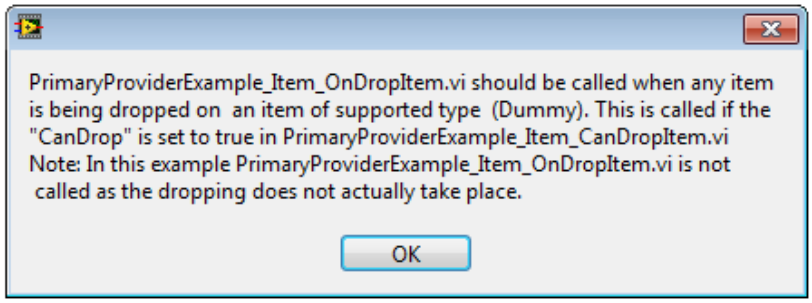


11. Select "DummyMenuOptionForMultipleItems".

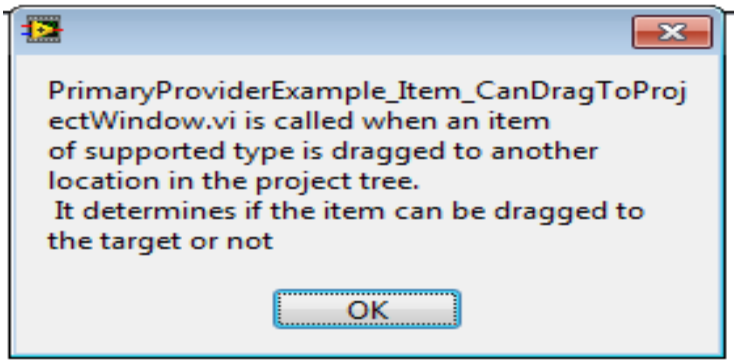


12. Add another item say a VI to the project. Drag the VI and try to drop it over DummyItem1.





13. Now do the reverse, drag the DummyItem1 to any other location in the project.



IV. Primary Provider 2

Name : PrimaryProviderExample2

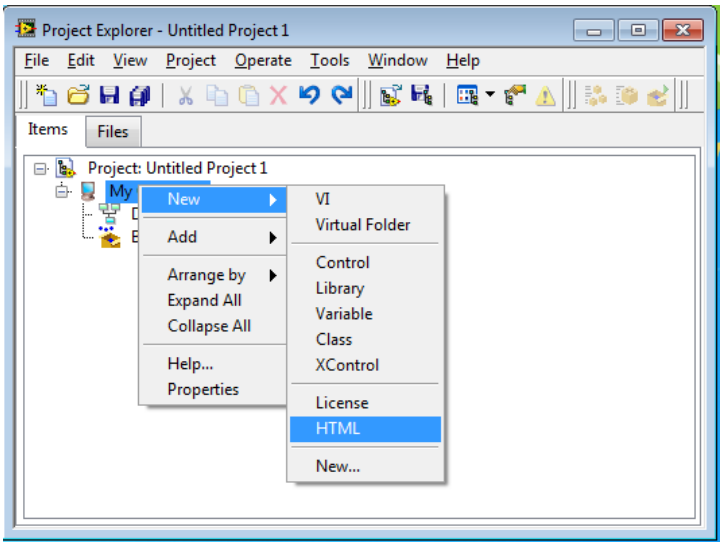
Features implemented : Adding a new item of type "HTML" object to the project tree, setting item name and icon, item specific right click menu options for this item type, custom behavior on double clicking on this item, customized drop behavior.

Corresponding VIs

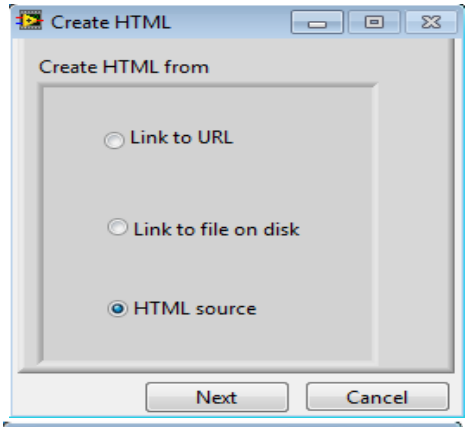
1. Adding a new item type
    a. PrimaryProviderExample2_CreateNewWizard_IncludeItem.vi
    b. PrimaryProviderExample2_CreateNewWizard_GetWeight.vi
    c. PrimaryProviderExample2_CreateNewWizard_Init.vi
    d. PrimaryProviderExample2_CreateNewWizard_Invoke.vi
    e. PrimaryProvider2_CreateHTMLFile.vi
    f. PrimaryProviderExample2_CreateNewWizard_Finalize.vi
2. Setting item name and icon
    a. PrimaryProviderExample2_CreateNewWizard_Finalize.vi
    b. PrimaryProviderExample2_Item_Init.vi
3. Item specific popup menu
    a. PrimaryProviderExample2_Item_OnPopupMenu.vi
    b. PrimaryProviderExample2_Item_OnCommand.vi
4. Item specific double click behavior
    a. PrimaryProviderExample2_Item_OnDblClick.vi
    b. PrimaryProviderExample2_Item_OnCommand.vi
5. Drag and drop behavior
    a. PrimaryProviderExample2_Item_CanDragToProjectWindow.vi
    b. PrimaryProviderExample2_Item_CanDropItem.vi
    c. PrimaryProviderExample2_Item_OnDropItem.vi

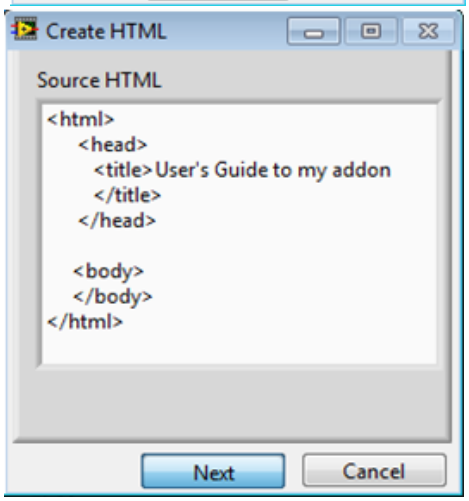
Usage Instructions

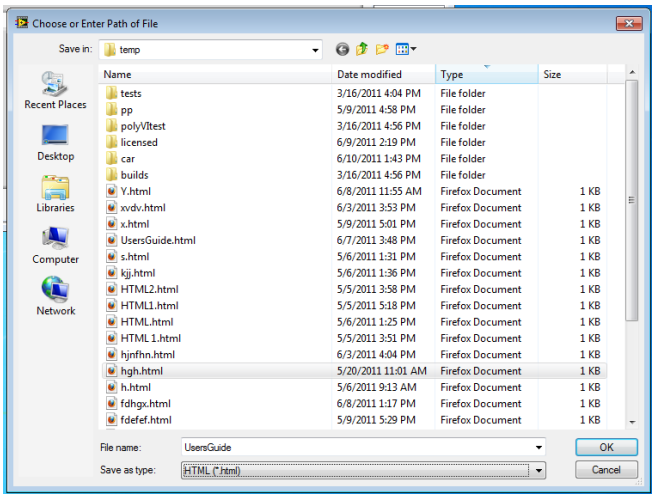
1. Launch LabVIEW and create an empty project
2. Right click on “My computer” or a virtual folder in the project tree. Select New > HTML.

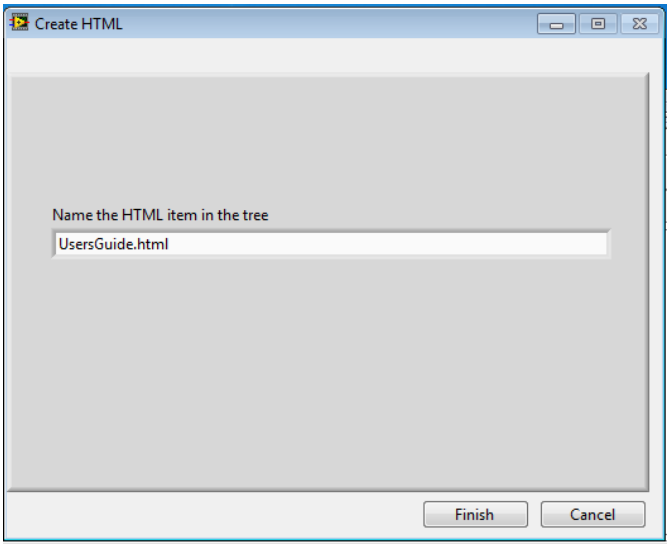


3. You have three ways to create an HTML item. Choose the option to create HTML from source.

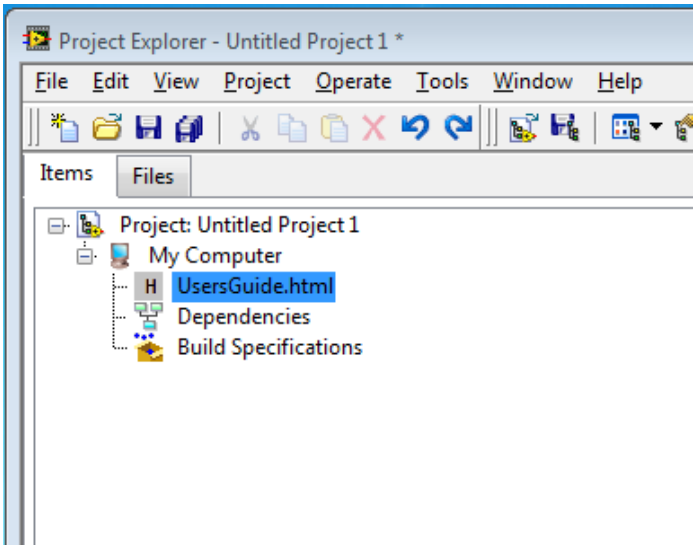




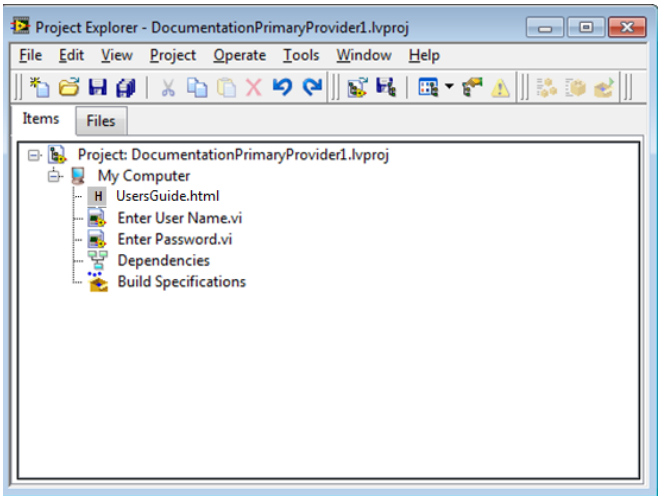




4. On clicking Finish you will see the HTML item added to your project

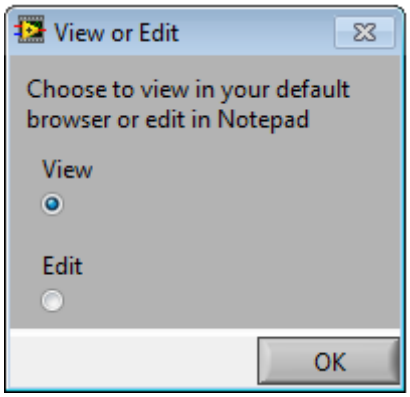


5. You can drag and drop the VIs and controls in your project onto the newly created HTML item to include snapshots of their front panel in the html.

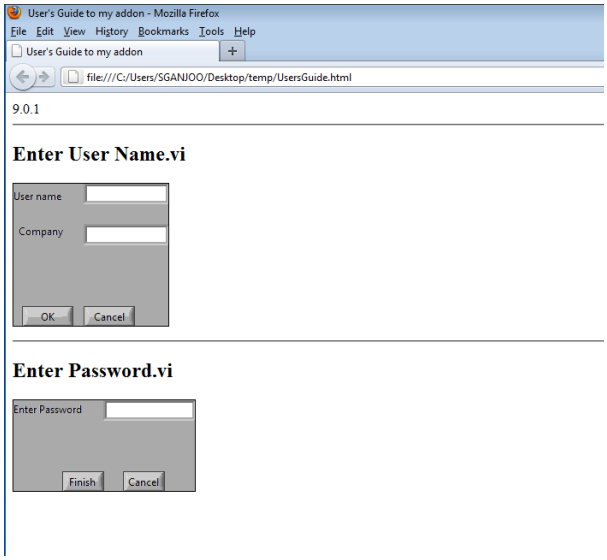


Drag and drop the VIs one by one onto the UsersGuide to update the HTML file.

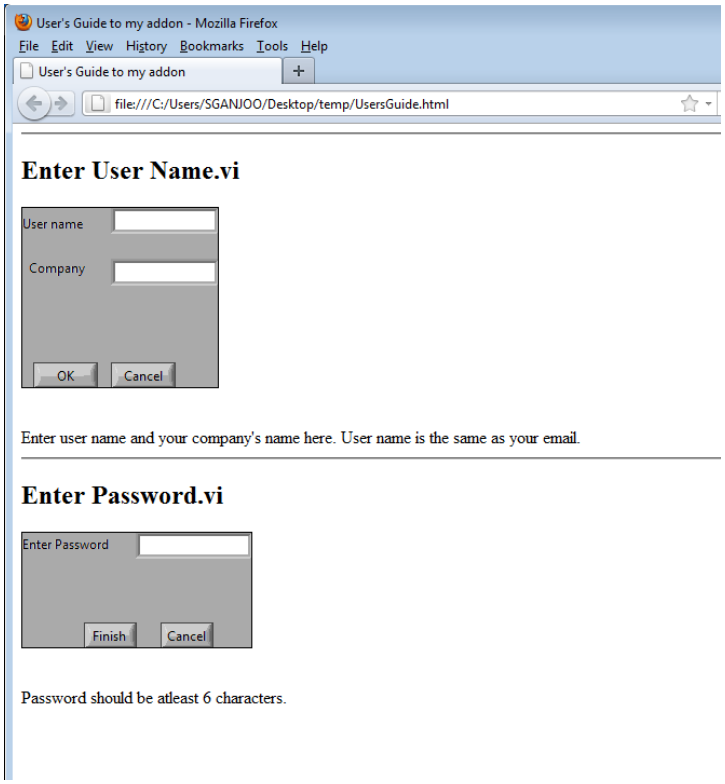
6. Double click on the HTML item "UsersGuide".



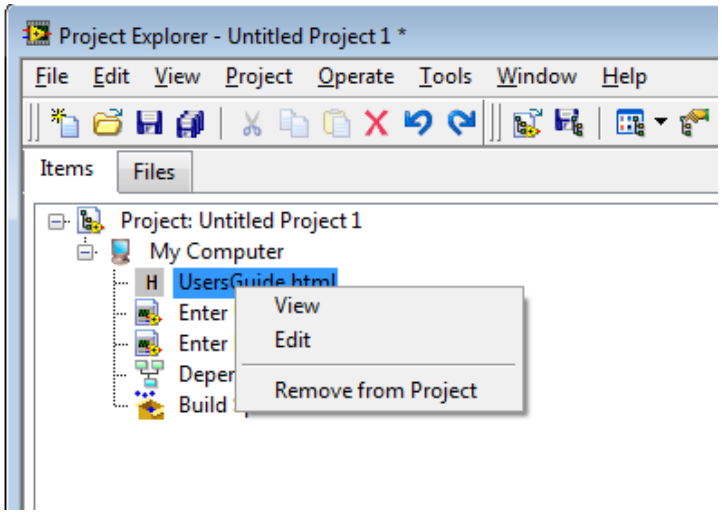
7. Select the option to "View" and click "OK" to open the HTML item in your default browser.



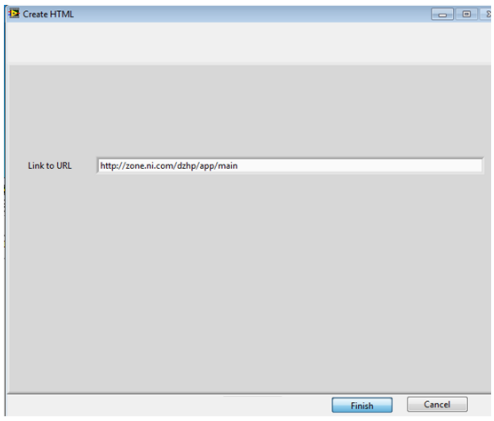
8. Double click on the UsersGuide item in the project again and this time select the option to "Edit". This opens the HTML source code in Notepad for editing. You can edit the html source code and add instructions. An example screenshot of the UsersGuide when viewed after editing is shown below.



9. You can also "View" or "Edit" the UsersGuide item by right clicking on it and choosing the appropriate popup menu option in the project explorer window.



10. You can also create an HTML item that points to a URL by choosing the option to “Link to URL” in the first step of the Create HTML wizard.



11. Another way to add an HTML item can be to choose the option to directly browse to an HTML file saved on your disk.

    Note you will only be able to edit the HTML if you have it saved on the disk and not if you are pointing to a URL on the Web. Also, to be able to drop VIs or controls on an HTML item, you have to have the HTML item saved to your disk.

Points to remember

1. **Paths should be constructed by getting the current VI's path and appending to it the relative path of the target and used to point to various VIs and files.**
2. **If passing relative paths to the provider API, e.g. icon file paths, these should be relative with respect to the working directory of the provider. The working directory for a provider is where its INI file is located at.**
3. **In LabVIEW 2011 all INI files should be under the "GProviders" directory, and hence it is the working directory for all providers. This change is backward compatible, hence this approach is highly recommended when building providers with any LabVIEW version.**
4. **The API VI "mxLvGenerateGuid.VI", located under the API directory, can be used to generate new GUID to be used as the value of the SupportedType token in the INI file of a new primary provider.**
5. **When making changes to the UI for an item, e.g. changing the name or the icon, call the mxLvUpdateUI with appropriate action (e.g. update name, icon, state, menus, etc.) for that item.**
6. **It is recommended to close all references that are opened in any of the VIs in the provider framework.**

Contacts

If you have any questions regarding the project provider framework, please contact the LabVIEW Partner Program at labviewpartnerprogram@ni.com.