**Actor Framework Project Providers**

**Design Reference**

**Overview**

This document describes the set of LabVIEW Project Providers that support the Actor Framework. It is intended to provide an overview of their structure, theory of operation, and implementation, so that future developers can effectively maintain and extend them.

**About Project Providers**

A project provider is a plug-in to the LabVIEW Project window that extends the window’s functionality. The new functionality can apply to all items in a project, or to specific item types. Primary providers add new item types to the project tree, while secondary providers add functionality to existing items.

The Actor Framework includes primary providers for creating actors and actor interfaces, and secondary providers for creating messages.

Provider development is outside the scope of this document. Users are referred to the *LabVIEW Project Provider Developer’s Guide*, which can be found at the LabVIEW Project Provider forum. See Reference Links, at the end of this document.

**About VI Scripting**

Project providers make extensive use of VI Scripting to do their work. A general discussion of VI Scripting is outside the scope of this document. Please see Reference Links, at the end of this document for links on how to get started.

You must enable VI scripting in your LabVIEW IDE to work effectively with these project providers. From any LabVIEW menu, select Tools » Options…, and then navigate to the VI Server page. Select “Show VI Scripting functions, properties and methods” as shown, and make sure “Display additional VI Scripting information in Context Help window” is selected.



We also recommend adding the following line to your LabVIEW INI file:

SuperSecretPrivateSpecialStuff=true

**Messages in Actor Framework**

A brief review of the types of message classes created by the providers is helpful in understanding the code artifacts the providers create.

In Actor Framework, a message is a class that inherits from **Message.lvclass**. It implements an override of **Do.vi**, a dynamic dispatch method of Message.lvclass. This message invokes a method of the message’s target actor. When created by the project provider, the message name is of the form <method name> Msg.lvclass. The message class itself includes as attributes all of the inputs to the method targeted by the message (except for the target actor and standard error input).

Messages created by the provider also include a method that writes data to an instance of the message class, and puts that object on the actor’s enqueuer. The name of this method is Send <method name>.vi.

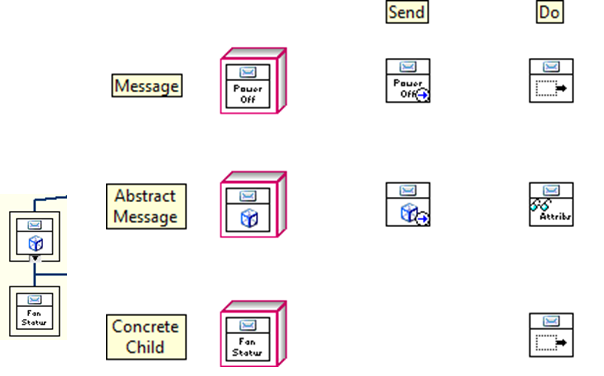
The providers will also create abstract messages and their child concrete implementations. (See the note about abstract messages immediately following this section.)

Abstract messages do not target a specific actor’s method. They include a set of attributes defined by the developer and a Send VI. The Send VI writes attributes to a child of the abstract message, and puts that object on an actor’s enqueuer. The child class is specified by an input to the Send VI. Abstract messages do NOT include an override of Do.vi. Abstract messages are named by the developer.

Abstract messages also provide a read accessor that returns all of the message’s class data. This VI used in the Do.vi methods of children of the abstract message.

A concrete child message inherits from an abstract message. Like the standard message, a concrete child implements a Do.vi that invokes a method of a specific target actor. Concrete children are named in the same fashion as standard messages.

The following diagram shows the artifacts created for each message type.

****

**A Note on Abstract Messages**

Abstract messages and their concrete children are **deprecated** in Actor Framework applications created in LabVIEW 2020 SP1 or later. Developers are strongly urged to use actor interfaces instead.

However, we will continue to maintain the abstract message provider code in the Actor Framework Project Provider so that developers can continue to work efficiently with older code bases.

**The Project Providers**

**Available Providers**

The suite consists of five individual providers. The two primary providers create actors or actor interfaces, while the three secondary providers create or refactor messages. The secondary providers each provide options for a single project item: an actor, an actor’s methods, or an existing message.

In theory, a single secondary provider can act on several types of project items, but this was shown during development to be difficult, so it was decided to create a different provider for each type.

Each provider is described individually, below.

Add Actor

****

Add Actor acts on Targets and Libraries, and creates a new actor under that target/library. When placing a new actor under a target, it creates a library for the actor, and includes a virtual method for messages created for that actor.

Add Actor does not create any messages, and does not invoke any messages from the core Message Maker library (see next section).

Add Actor Interface

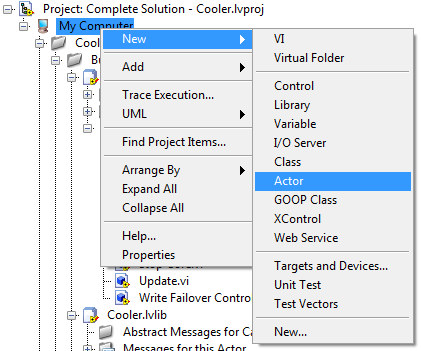
****

Add Actor Interface acts on Targets and Libraries, and creates a new interface under that target/library. When placing a new interface under a target, it creates a library for the interface, and includes a virtual method for messages created for that actor.

The option to create a new actor interface is not available in projects that contain no actors.

Add Actor Interface does not create any messages, and does not invoke any messages from the core Message Maker library (see next section).

Message Maker Provider



Message Maker Provider creates messages for one or more actor methods. Several methods may be selected, and the methods may span multiple actors.

If a single method is selected, the developer also has the option to create a child of an abstract message.

Actor Message Maker

Actor Message Maker Provider creates a message for each method in one or more actors. Several actors may be selected.

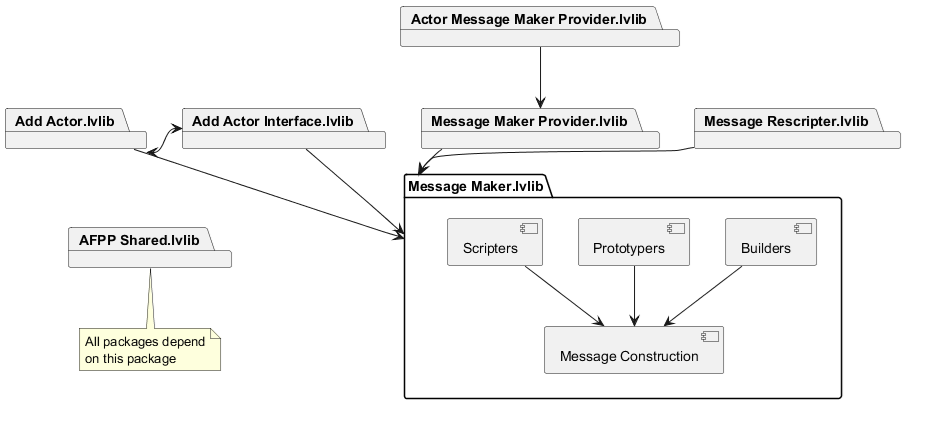
If a single actor is selected, the developer also has the option to create an abstract message.

Message Rescripter



Message Rescripter acts on an existing Actor Framework message. It rebuilds the existing message in place.

**Architecture**

****

**Populating Project Right-Click Menus**

**Executing Scripting Code**

**Error Handling**

**Creating Actors and Interfaces for Actors**

**Creating Messages**

**Message Templates**

The providers use template classes to build specific messages of a given type. Templates are included for standard, abstract, and concrete messages, as shown.

Note that these class templates are not part of a library, nor do they inherit from Message.lvclass. It is much easier to work with an uncoupled class than one that is linked to other code artifacts.

To create a message, the providers copy the appropriate template to the specified location, modify its VIs as necessary, change it to inherit from Message.lvclass, and then add it to the actor’s library.

The template to be used is selected based on the provider action invoked by the user.



**Required Data**

To create a message, the developer right-clicks on an actor or message and selects an option. The selected option determines the message type (and thus the starting template). The provider then then obtains additional information about the selected actor or method that is used to complete message construction:

1. Target Actor (reference or path): the actor for which the new message is being created.
2. Method Path: the path to the method to be invoked by the new message. In the case of abstract messages, the method path is the path to a prototype method (see Prototyping, below).
3. Class Name: the name of the new message class. For an abstract message, this is specified by the developer; otherwise, it is derived from the target method.
4. Parent Class Path: the path to the abstract parent class for this message (Concrete child only).

**Prototyping**

**Class Creation**

**Adding Member Data**

**Creating the Send VI**

**Creating the Receiver VI**

**Rescripting Messages**

**PPL Support**

**Reference Links**

**On Project Providers**

The LabVIEW Wiki entry is here:

<https://labviewwiki.org/wiki/Project_Provider_Framework>

The LabVIEW Project Providers forum:

<https://forums.ni.com/t5/LabVIEW-Project-Providers/bd-p/bymqyodmkc>

Written documentation and examples:

<https://forums.ni.com/t5/LabVIEW-Project-Providers/Project-Providers-Documentation/td-p/3492573>

A presentation by David Ladolcetta:

<https://www.youtube.com/watch?v=xXGro_DylHs>

The slide deck for that presentation:

<https://forums.ni.com/t5/Developer-Center-Resources/Customize-the-LabVIEW-Project-Explorer-Using-the-Project/ta-p/3532774>

**VI Scripting**

The LabVIEW Wiki entry has a number of great links:

<https://labviewwiki.org/wiki/VI_Scripting>

Trevor Christman’s introduction to VI Scripting:

<https://forums.ni.com/t5/Past-NIWeek-Sessions/Introduction-to-VI-Scripting-in-NI-LabVIEW/ta-p/3496554>

Darren Nattinger on VI Scripting:

<https://forums.ni.com/t5/Community-Documents/Don-t-Wait-for-LabVIEW-R-amp-D-Implement-Your-Own-LabVIEW/ta-p/3794651>