AZ Interface

version 1.0.0

Andrei Zagorodni

2018-05-08

Content

1 Introduction

AZ Interface (AZI) is a solution for implementing Java-like interface architecture in LabVIEW projects.

Contrary to other solutions providing Java-like interface architecture, **AZ interface** is simple while fulfilling basic programming demands.

1.1 Versions

1.1.1 Version 0.0.0-pre-alpha

First functional version of the toolkit.

The version was presented at European CLA summit in Madrid, 2018.

1.1.2 Version 1.0.0-alpha

The version basically differs from v.0.0.0.

This version is result of brainstorming at European CLA Summit 2018:

- first, the concept was presented as a regular lecture;
- second, pitfalls were extensively discussed/brainstormed with Stephen Loftus-Mercer, National Instruments.
- third, the lecture was repeated and many other experts participated in brainstorming.

I highly appreciate contribution of all participants of these sessions /Andrei Zagorodni

1.1.2.1 Release 1.0.0.0

First public release of **AZ Interface** software.

1.1.2.2 Release 1.0.0.1

Public release including few small fixes.

Main fix: Improved HD folder selection algorithm for newly created **AZ Interface**.

1.2 Shortenings and abbreviators

Abbreviator	Description
AZI	AZ Interface
HD	Hard Disk
[LabVIEW]	Location of LabVIEW in this computer; for example `C:\Program Files (x86)\National Instruments\LabVIEW 2016\`
OOP	Object-Oriented Programming
SW	Software; AZ Interface software

1.3 Concept of Interface in other languages

Concept of Interface was developed to substitute multiple inheritance in some object oriented languages (OOP languages). Probably the most known of them is Java.

Similarly to LabVIEW, a Java class can have only one parent class; i.e. class hierarchies have tree-like structures. *Java Interface* allows creating "cross-links" between trees; i.e. simulate multi-parent behavior. The concept is illustrated by Figure 1.

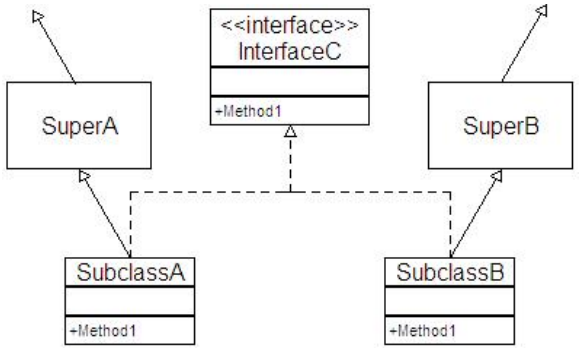

Figure 1 Interface in UML diagram

`SubclassA` and `SubclassB` belong to different hierarchical trees. `InterfaceC` provides common behavior to these classes without any effect on hierarchical positions of classes `SuperA` and `SuperB`.

Java Interface provides an own data type allowing to work at corresponding abstraction layer.

Java Interface can be considered as an *Abstract Class* having only abstract methods. Attributes are not allowed in *Interfaces*. Otherwise `InterfaceC` behaves exactly in the same way as any super-class.

2 AZ Interface Background Ideas

2.1 Solutions

AZ Interface (AZI) implements obvious solution utilizing capacity of *Call By Reference* node.

Each *AZI* is assembled as a native *LabVIEW class*. No class hierarchies are allowed between *AZI*-s.

Relation between *AZI* and *Class* applying the interface is defined by adding each other in list of *Friends* (*Community scope*).

2.2 Features

AZI-s allow creating abstraction levels independent on hierarchical structures of classes.

The *AZI-s* allow abstraction of functionality independently on implemented OOP model, if modern models are used; i.e. same methods of the same *AZI* can be applied in native LabVIEW classes, GOOP3 classes, GOOP4 classes, and G# classes.

LabVIEW code created with toolkit can be opened, edited and/or run in any LabVIEW computer without installation of the toolkit. The code is not limited to LabVIEW development environment; corresponding EXE-files can be run under conventional LabVIEW RTE. However, developer must take care about inclusion of invoked code in build specification (that is the same for any LabVIEW code invoked with *Call By Reference* node).

2.3 Limitations

- The code is not imperative at development time.
- No *AZI* hierarchy can be established.
- Current version is tested only for *My Computer* branch of current *LabVIEW Project*. Use of the toolkit with other targets was not tested yet. This limitation will be resolved in future.
- **AZ Interface Consistency tool** announced for v.0.0.0 is not included in v.1.0.0 package. Need in functions included in this tool disappeared due too altering of the whole concept. New tool will be created in future if new needs will be identified.
- Connector pane of *AZI methods* use terminal pattern 6x4x4x6 only. Altering the terminal pattern would cause errors that are difficult to fix.
- Connector pane terminals of each *AZI method* must be assigned before the method is applied in one of *classes*. Later changes could require significant efforts. *I am still thinking how to do such operations easier.*
- This version does not support method reentrancy. Setting corresponding VI-s as reentrant does not help. Clones of reentrant methods are executed consequently. *I am working to solve this limitation in future versions.*

3 System Requirements and Installation

3.1 Requirements

Current version of the toolkit is developed for LabVIEW 2016. No additional package is required.

3.2 Installation

No installer is supplied with current version of the toolkit. Files must be manually copied in corresponding LabVIEW directories.

3.2.1 File location

Files must be copied in different directories of LabVIEW. The table below refers to `[LabVIEW]` directory that means, for example,
`C:\Program Files (x86)\National Instruments\LabVIEW 2016\`

Content of the following source directories must be copied into corresponding target directories.

Supplied files	Target LabVIEW directory
`GProviders`	`[LabVIEW]\resource\Framework\Providers\GProviders\`
`Providers`	`[LabVIEW]\resource\Framework\Providers\`
`help`	`[LabVIEW]\help\`

3.2.2 Recompiling

In some cases files of the toolkit must be recompiled after the copying; f. ex. VIs must be re-saved accounting to new locations of sub-VIs.

Do to it open consequently two VIs. These VIs are used only for manual installation. Ignore messages concerning altered file locations. Order of opening could be important:

1. Open LabVIEW.
2. `[LabVIEW]\help\AZ Interfaces\_1_all_help_AZ_Interfaces.vi`
3. `[LabVIEW]\resource\Framework\Providers\AZ_Interfaces\_3_all_providers_AZ_Interfaces.vi`
4. Click menu File > Save All
5. Close all VI-s.
6. Restart LabVIEW.

4 Primary Functions of the Toolkit

4.1 Creating AZI

1. Right-click the **My Computer** or any **Virtual Folder** and select menu **AZ Interfaces** > **Create AZ Interface**.
2. **Create Interface** dialog will be opened.
3. Write name of new *AZI class*, use other input fields if needed.
4. Click **Create Interface**.

LabVIEW class will be created in selected location. Newly created *AZI* includes three members:

- `cast_to_Interface.vi` – community-scope utility method called only by corresponding methods of *AZI*-applying *classes*.
- `method_refs.ctl` – utility type definition that is part of *AZI* private data. The type definition is also used in automatically created methods of *AZI*-applying *classes*.
- `read_Object.vi` – method used for casting from *AZI* data type to data type of particular class. The method should usually be followed with node *To More Specific Class*.

4.2 Creating AZI method

1. Right-click the *AZI* class in LabVIEW project and select menu **AZ Interfaces** > **Create Interface method**.
2. Write name of the method in the opened dialog and click **Create method** button.
3. Open *Front Panel* of the newly created method.
4. Create necessary controls and indicators and connect them to terminal pattern of the VI. Do not alter the terminal pattern. Do not disconnect existing terminals. ATTENTION: altering terminals (number of terminals, they assigning in terminal pattern, data types) after overriding the method in *AZI*-applying *class*(es) will cause a need in extensive manual work. Thus be careful at this step.
5. Do not edit *Block Diagram* of the method.
6. Save the method.
7. Save the whole *AZI* (Select class in the project then right-click menu **Save > Save All (this Class)** or select menu **File > Save All**).

Block Diagram of the newly created method (see Figure 2) contains default code and terminals of user-created controls/indicators. This code will be automatically altered when the method is first applied in any *AZI*-applying *class*.

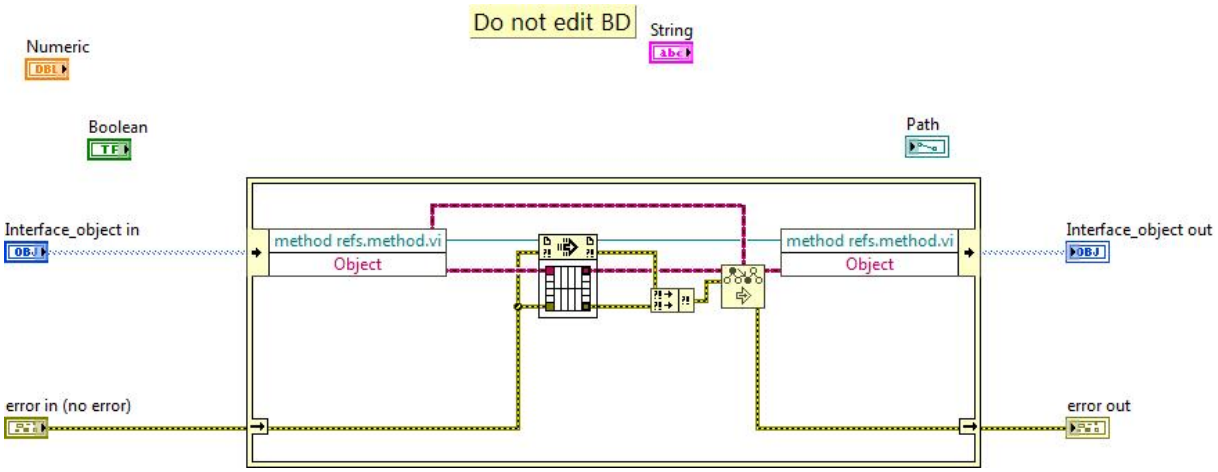

Figure 2 Example of *AZI method*

4.3 Applying AZI and AZI methods to Class

The same dialog is used for applying *AZI* to *Class* and for implementing *AZI Method* in the *Class*.

- Right-click the any class in the project then select menu **AZ Interfaces** > **Apply Interface**. The dialog appears listing all available *AZI*-s (Figure 3). Selection of an *AZI* in the list, populates list of methods belonging to the *AZI*.

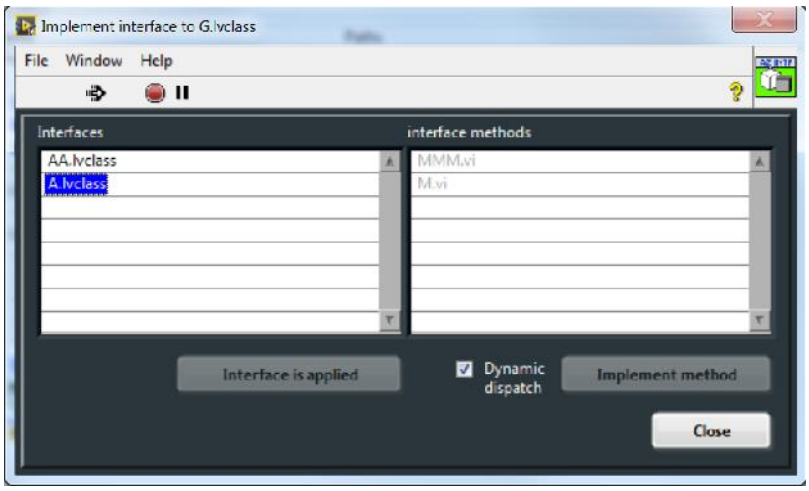

Figure 3 GUI used for applying *AZI* to *Class.*

4.4 Applying AZI to Class

- Select an item from *AZI* list at left-hand side. The list shows all *AZI*-s available in the *Project*.

- If selected *AZI* is already applied to the *Class*, button at bottom of the list is disabled. The button text is "Interface is applied" (see Figure 3). In this case select another *AZI*, continue working with methods (section 4.5), or click **Close**.
- Click button **Apply interface**.

The *Class* and the *AZI* get each other in their lists of *Friends*.

New method is added to the Class: `cast_to_aziName.vi`, where `aziName` is name of the *AZI*. This method is used for casting of corresponding *Object* to type of the *AZI*. In some sense the casting is similar to one performed by nodes *To More Specific Class* and *To More Generic Class*.

The method `cast_to_aziName.vi` is initially broken. It will be repaired automatically (its *Block Diagram* altered) with applying of first *AZI* method in the *Class*.

4.5 Implementing AZI methods in Class

- Select an item from *AZI* list at left-hand side (see Figure 3).
- List **interface methods** at right-hand shows available *AZI methods*.
- If selected *AZI* is not yet applied to the *Class*, button **Apply interface** at bottom of the list is enabled. In this case click button **Apply interface**, select another *AZI*, or click **Close**.
- Select method in the list **interface methods**. Methods already applied in this *Class* are disabled.
- Click button **Implement method**.

The method will be added in the *Class* supplied with necessary terminal pattern. *Block Diagram* of the method is initially empty. All coding of the method (including wiring of class terminals) must be performed manually.

Beside of the method, an utility method `util_aziName_cls_methodName.vi` is added in the *Class*. Name of the utility method contains name of the AZI (`aziName`) and name of the actual method (`methodName`). The method is created automatically and should not be altered.

5 How to Use

5.1 General example

Use of *AZI*-s can be illustrated by block diagram presented in Figure 4. Three classes are not hierarchically related while both apply the same *AZI*.

Objects belonging to three different OOP models are created (GOOP, G#, and Native LVClass) then processed at common abstraction level of the *AZI*. Finally, the objects are cast back to initial class types.

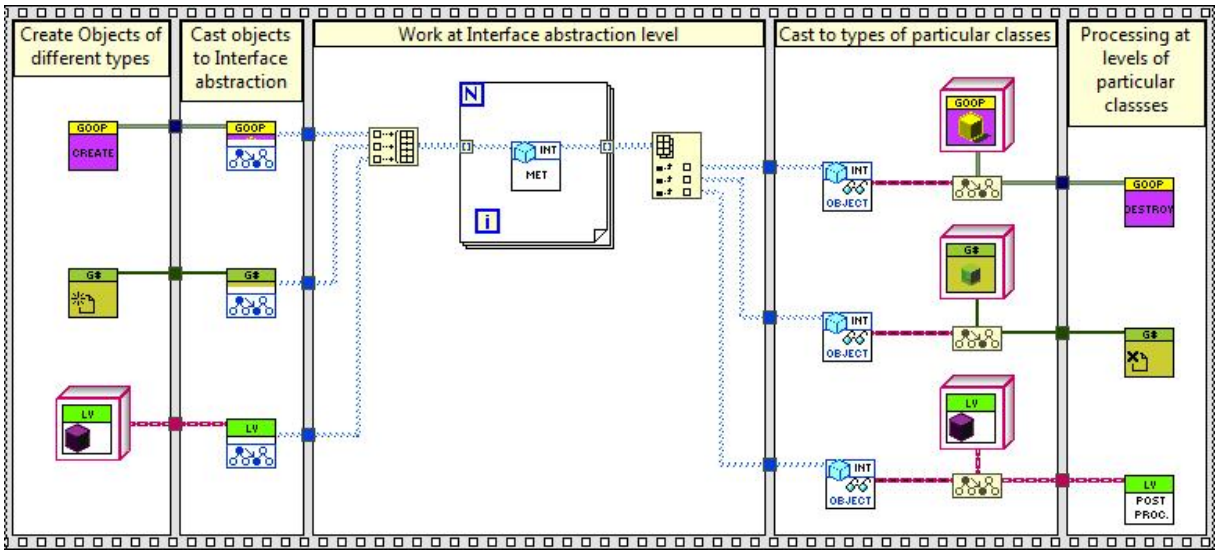

Figure 4 Example of *AZI* use.

5.2 Working with class hierarchies

5.2.1 Sub-classes of AZI-implementing class

Any child of an *AZI*-applying *class* can have corresponding *AZI method*(s). There is no need to apply *AZI* to each sub-class of the hierarchy. However, object terminals of corresponding methods must be *Dynamic Dispatch*.

5.2.2 Two AZI-implementing classes belonging to same hierarchy

A need in applying the same *AZI* to different *classes* of the same hierarchy is rare (see section 5.2.1). *At least I cannot identify such a need.* However; this can be done altering type of object terminals of all conflicting methods to *Dynamic Dispatch*.

5.3 Altering terminals of AZI method

Connector pane terminals of each *AZI method* must be assigned before the method is applied in one of *AZI*-applying *classes*. However, a need to alter the terminal signature could arise. Terminal signatures of the following must differ only by type of object terminals:

- *AZI* method must have object terminals of *AZI* type.
- Corresponding class methods must have object terminals of corresponding class types.
- Utility methods `util_aziName_cls_methodName.vi` (see section 4.5) must have object terminals of *LabVIEW Object* type in all classes.
- Corresponding element (*VI Refnum*) of `method_refs.ctl` (see section 4.1) belonging to the *AZI* must have object terminals of *LabVIEW Object* type; i.e. the

element must have the same signature as utility method `util_aziName_cls_methodName.vi`.

6 About and Contacts

Figure 5 About

6.1 License Agreement

1 Acknowledgement

CAREFULLY READ THE FOLLOWING TERMS AND CONDITIONS BEFORE USING THIS SOFTWARE. BY USING THIS FREEWARE VERSION YOU ACKNOWLEDGE THAT YOU HAVE READ THIS LIMITED WARRANTY, UNDERSTAND IT, AND AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS. YOU ALSO AGREE THAT UNLESS YOU HAVE A DIFFERENT LICENSE AGREEMENT SIGNED BY ANDREI ZAGORODNI YOUR USE OF THIS SOFTWARE INDICATES YOUR ACCEPTANCE OF THIS LICENSE AGREEMENT AND WARRANTY. IF YOU DO NOT AGREE TO THE TERMS OF THIS AGREEMENT, DELETE THE SOFTWARE FROM ALL STORAGE MEDIA.

2 License

This Freeware License Agreement (the "Agreement") is a legal agreement between you ("Licensee"), the end-user, and developer of AZ Interface Toolkit Andrei Zagorodni ("Developer") for the use of this software product ("Software"). Commercial as well as non-commercial use is allowed. By using this Software or storing this program or parts of it on a computer hard drive (or other media), you agree to be bound by the terms of this Agreement. Provided that you verify that you are handling the original freeware version you are hereby licensed to make as many copies of the freeware version of this Software and documentation. You can alter this Software in any way but Developer does not carry any responsibility for consequences.

If you alter and/or further develop this Software, documentation (including "help" and "about") must include reference to original Software, name of its Developer and his contacts.

3 Limited Warranty and Disclaimer of Warranty

The AZ Interface Toolkit EXPRESSLY DISCLAIMS ANY WARRANTY FOR THE SOFTWARE. THIS SOFTWARE AND THE ACCOMPANYING FILES ARE PROVIDED "AS IS" AND WITHOUT WARRANTIES AS TO PERFORMANCE OF MERCHANTABILITY OR ANY OTHER WARRANTIES WHETHER EXPRESSED OR IMPLIED, OR NONINFRINGEMENT. THIS SOFTWARE IS NOT FAULT TOLERANT AND SHOULD NOT BE USED IN ANY ENVIRONMENT WHICH REQUIRES THIS. NO LIABILITY FOR DAMAGES. In no event shall AZ Interface Toolkit or its suppliers be liable for any consequential, incidental or indirect damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or any other pecuniary loss) resulting of the use of or inability to use this SOFTWARE EVEN IF the Software HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. The entire risk resulting of use or performance of the SOFTWARE remains with you.

4 Copyright

Copyright (c) by Andrei Zagorodni.

6.2 Contacts

Andrei Zagorodni

andrei.zagorodni@novatorsolutions.se

Please write **AZI** or **AZ Interfaces** in subject line.

6.3 Support and communications

I shall appreciate feedback about bugs and bottlenecks identified in this SW.

I promise to read your emails and reply within reasonable time. However do not forget that the project is developed in my evenings and weekends. Thus the "reasonable time" will solely depend on my work load.

You are free to modify code of the software. However I do not promise to support the modified code.

Andrei Zagorodni