AZ Multi-inheritance

version 2.0

Andrei Zagorodni

2024-02-04

Content

1 Introduction

AZ Multi-inheritance (**AZM**) is solution and toolkit for implementing multiple inheritance in LabVIEW OOP projects.

The answer is based on native LabVIEW interfaces; thus, applicable to versions starting from LabVIEW 2020.

The current version of **AZM** works with **GOOP4** classes. Thus, this document is targeted to developers familiar with **GOOP4**.

Attention!

Multiple inheritance is an extremely powerful OOP technique. As any powerful programming technique, it opens possibility to create excellent architectural solutions as well as errors that are difficult to identify. Thus, using described tools (**AZM**) and approaches is recommended only to advanced users with significant OOP experience.

1.1 Conventions

1.1.1 Lexicon, shortenings, and abbreviators

Abbreviator	**Description**
Ancestor	Any class from which considered class inherits. The word is convenient to define indirect inheritance; i.e. parents of parents.
AZM	AZ Multi-inheritance
`[name of XYZ]`	User-selected name
BD	Block Diagram
enum	enumerated data type
G#	Alternative toolkit providing LabVIEW with by-reference OOP programming features.
GOOP	Concept and toolkit providing LabVIEW with by-reference OOP programming features.
i-Class	**Interface**-based class specific for **AZM**
IDE	Integrated Development Environment
Interface	The word is used in this document solely for LabVIEW OOP **Interfaces**.
`[LabVIEW]`	Location of LabVIEW in this computer; for example `C:\Program Files (x86)\National Instruments\LabVIEW 2020\`
LVLIB	LabVIEW Project Library
OOP	Object-Oriented Programming

Abbreviator	Description
Parent	Any class from which considered class inherits. The word is convenient to define direct inheritance; i.e. closest parents of the class.
RTE	Run-Time Engine
SW	Software; AZM software
VIPM	VI Package Manager

1.1.2 Font conventions

- **Bold** is used for anything that appears literally in a LabVIEW environment or in LabVIEW program. For example, for menu, labels that cannot be altered.
- *Italic* is used for terms and messages.
- `Constant Width` is used for values: paths, names, etc.
- **`Constant Width Bold`** is used for values: paths, names, etc. that cannot or must not be altered.
- `[ ]` – brackets surround selectable parts of paths, names, etc.
- *Green Italic* is used for my personal notes.

1.2 Versions

Version number consists of four values:

1. *version* - altered with major changes causing compatibility and/or conceptual issues;
2. *subversion* – altered with introduction of major changes;
3. *fix* – minor changes, f. ex. a bug fix or minor performance improvement;
4. *build* – has meaning only for developer; f. ex. allows accounting of development packages, special assemblies, etc.

Altered *version* or *subversion* can cause a need in reading updated manual, while altered *fix* or *build* does not affect the way of use.

1.2.1 Version 0.0.0-pre-alpha

First functional version of the toolkit. The version is prepared in connection to presentation at GDevCon #3, Amsterdam, 9th Sept 2022.

Attention! This version is proof of concept. It could undergo dramatic changes after obtaining feedbacks. *I highly appreciate any feedback /Andrei Zagorodni*

1.2.2 Version 1.0.0-alpha

First release of the toolkit.

1.2.3 Version 1.1.0-beta

New features:

- **New i-Class…** menu is added. The menu works with all three relevant project members: i-Classes, Interfaces, and GOOP4 classes. See 4.1.1.
- **Add Method…** menu is added. The menu works with all three relevant project members: i-Classes, Interfaces, and GOOP4 classes. See 4.3.

Improvements:

- Conversion of **Interface** to i-Class (see 4.1.2) automatically inserts parent constructors and destructors in newly created constructor and destructor.

1.2.4 Version 2.0.0

New features:

- **Icon editors** are available (borrowed from **GOOP Development Suite**); see chapter 5 for details.
- **Icon editors** belonging to GOOP menu branches can be used on i-Classes.

Improvements:

- Internal XML code of i-Classes is altered to be more compatible with GOOP Development Suite. This allows the use of **icon editors belonging to GOOP** menu branches.
- The toolkit is distributed as a single VIP-file and installed using **VI Package Manager (VIPM)**.

Bugfixes

- Some dialogs did not show custom menu. Fixed.
- Dialog **Add method** (section 4.3) could hang the whole LabVIEW IDE when working with large projects. The bug was observed in build 2.0.0.9 and fixed in build 2.0.0.10.

1.2.5 Versions of VIPM packages

The version, subversion and fix of each VIPM package corresponds to the version of AZM Toolkit. Strict correspondence between build numbers is not guaranteed.

1.2.6 AZM versions for different versions of LabVIEW

So far, the same AZM code can be installed in any version of LabVIEW starting from LabVIEW 2020.

2 AZ Multi-inheritance Background Ideas

2.1 Solutions

AZ Multi-inheritance (AZM) is based on following:

- Type-defined attribute cluster is stored as DVR (similarly to **GOOP4**).
- The DVR-s are mapped to object instances.
- The map key has LabVIEW **Interface** datatype.
- Only **GOOP4** objects are cast to the **i-Class** type (**i-Classes** are assembled by **AZM**).
- The map is stored in FGV.

2.2 Features

- AZM provides **GOOP4** classes with multiple parent classes called **i-Classes**.
- Conventional **GOOP4** ancestor class and ancestor **i-Classes** are two types of parentship.
- **GOOP4**, **i-Classes**, and **Interfaces** implemented as parents simultaneously do not create any conflict.
- LabVIEW code created with toolkit can be opened, edited, and run without installation of the toolkit. The code is not limited to LabVIEW development environment; corresponding EXE-files can be run under conventional LabVIEW RTE.

2.3 Limitations

- Current version is tested only for My Computer branch of LabVIEW Project. Use of the toolkit with other targets is implemented but not verified yet.
- **AZM** is applicable only to **GOOP4** classes. It can probably be used with **GOOP3** and G# classes but such feature has not been tested and not implemented yet.
- **AZM** cannot be used with native LabVIEW classes.
- Classes assembled by **AZM** (**i-Classes**) are abstract and cannot be instantiated.
- **i-Classes** and conventional **GOOP4** classes cannot be converted to each other.

3 System Requirements and Installation

3.1 Requirements

- The current version of **AZM** is developed for LabVIEW 2020. It is fully functional with following LabVIEW versions (currently tested up to LabVIEW 2023 that is the last version).
- **AZM** concept and toolkit cannot be downgraded to earlier LabVIEW versions.
- **VI Package Manager** is needed for installation.
- Installation of GOOP Development suite is recommended to enable icon editing (see chapter 5). **Open Source GOOP Development** suite is recommended.
  Installation of GOOP development suite is not required, however, I will be very surprised if one uses AZM toolkit without one of GOOP toolkits /Andrei Zagorodni

3.2 Installation

Starting from AZM v.2.0.0.0 the installation is carried out with conventional **VI Package Manager (VIPM)**.

If version 1.1 or older is already installed, manual removing of the older version is recommended prior installing the new version.

3.2.1 File locations

AZM files are located in different directories of LabVIEW. The table below refers to `[LabVIEW]` directory, for example to

`C:\Program Files (x86)\National Instruments\LabVIEW 2020\`

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

Target LabVIEW directory
`[LabVIEW]`**`\resource\Framework\Providers\GProviders\`**
`[LabVIEW]`**`\resource\Framework\Providers\AZ-MultiInheritance`**

Target LabVIEW directory
`[LabVIEW]`**`\resource\Framework\project\AZ-MultiInheritance`**
`[LabVIEW]`**`\help\AZ-MultiInheritance`**

4 Primary Functions of AZ Multi-inheritance toolkit

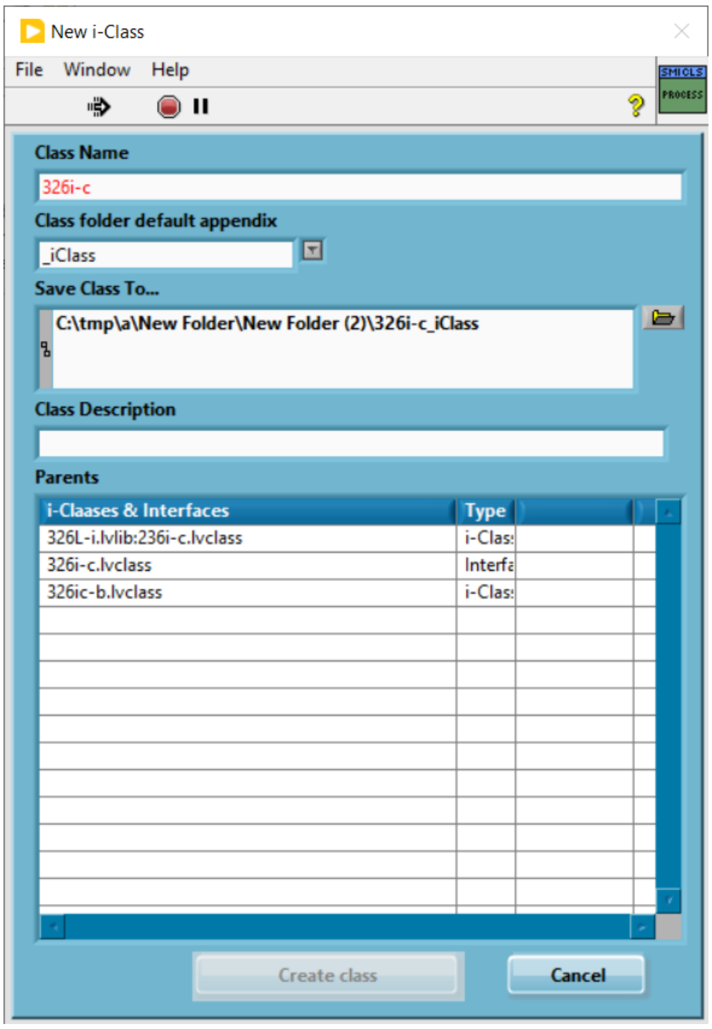
Note:

When working with **AZM** all involved files must be available for modifications. Remove write-protection from involved **i-Classes**, **GOOP4** classes, and all their members.

4.1 Creating i-Class

4.1.1 Creating new i-Class

1. Right-click **My Computer** or **Virtual Folder** and select menu **AZ-MultiIinheritance > New i-Class…**
2. **New i-Class** GUI will be opened.



Class Name
326i-c

Class folder default appendix
_iClass

Save Class To...
C:\tmp\a\New Folder\New Folder (2)\326i-c_iClass

Class Description

Parents

i-Claases & Interfaces	Type
326L-i.lvlib:236i-c.lvclass	i-Clas:
326i-c.lvclass	Interfa
326ic-b.lvclass	i-Clas:

Figure 1 GUI for creating new i-Class

3. Fill out the form and click **Create class**.
4. If possible, the class icon editor will be opened during the creation of the class (see chapter 5).

Notes:

- If a class with the selected name already exists in the project, the font of the field **Class Name** turns red.
- List **Parents** contains both **i-Classes** and **Interfaces**. The first column represents qualified names.

4.1.2 Converting Interface to i-Class

1. Select **Interface** existing in the project.
2. Right-click the **Interface** and select menu **AZ-MultiInheritance > Convert to class**.

This supplies the **Interface** be with two virtual folders and five new members:

- `utils/[Interface name]_Attributes.vi` – utility method; holding object attributes in uninitialized shift register (private).
- `utils/[Interface name]_GetAttributes.vi` – attribute accessor with functionality similar to corresponding member of **GOOP4** class (protected).
- `protected/ObjectAttributes.ctl` – cluster defining object attributes in a way similar to **GOOP4** (protected).
- `protected/[Interface name]_Create.vi` – object constructor similar to **GOOP4** (protected).
- `protected/[Interface name]_Destroy.vi` – non-dynamic-dispatch object destructor (protected).

Attention!

Conversion of an **Interface** to **i-Class** does not affect classes that already inherit from this **Interface**. Corresponding changes must be implemented in the code using **Consistency tool** (see section 4.4) or manually.

4.1.3 Creating i-Class within LVLIB

The current version of the toolkit does not allow one-step creating **i-Classes** as members of **LVLIB**-s. Use one of two methods instead:

- Create a new **i-Class** outside the **LVLIB** (see 4.1.1). The class should be located in the desirable directory of the HD but its location in the project does not matter. Then conventionally move the **i-Class** in the **LVLIB**.
- Create an **Interface within the LVLIB,** then convert the **Interface** to **i-Class** (see 4.1.2).

4.2 Setting i-Class as parent class

1. Right-click **GOOP4** class, **i-Class**, or **Interface** and select menu **AZ-MultiInheritance > Set parent**.
2. Parent-selection GUI is opened. The GUI is described in section 4.2.1.
3. Select parent and click **Set parent**.Inheritance from selected **i-Class** or **Interface** will be set.
4. If **i-Class** is set as a parent to **GOOP4** class or **i-Class** corresponding constructor and destructor will be added constructor and destructor BD-s.

Note:

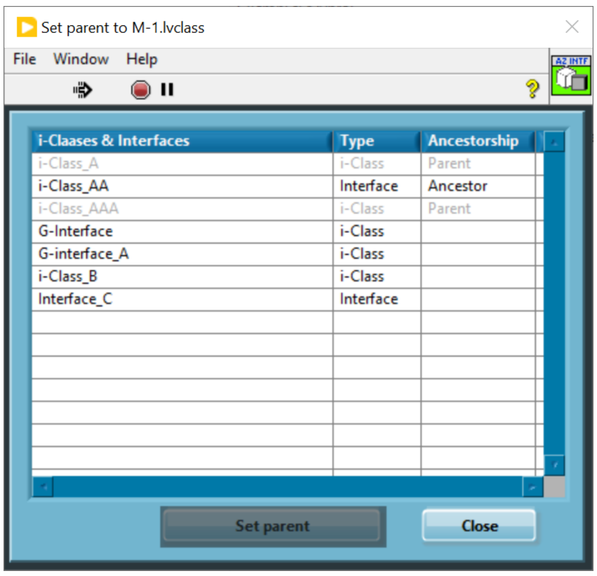
Contrary to destructors of **GOOP4** classes (`Destroy.vi`), destructors of **i-Classes** are not dynamic dispatch. Thus their names are unique: `[i-Class name]_Attributes.vi`.

Attention!

If the project contains **i-Classes**, this tool is highly recommended for creating parent-child relations even between **Interfaces**. It preserves calls to constructors and destructors of **i-Classes** if they are found somewhere up in the hierarchy.

4.2.1 Selecting parent

Parent selection GUI is shown in Figure 2.



i-Claases & Interfaces	Type	Ancestorship
i-Class_A	i-Class	Parent
i-Class_AA	Interface	Ancestor
i-Class_AAA	i-Class	Parent
G-Interface	i-Class	
G-interface_A	i-Class	
i-Class_B	i-Class	
Interface_C	Interface	

Figure 2 Example of newly created *AZI method*

- Column **i-Classes & Interfaces** shows all **Interfaces** in the project: both “as is” and **converted to i-Classes.**
- Column **Type** indicates this difference.
- **Parent** in Column **Ancestorship** shows if selected class already inherits from the **i-Class/Interface**. **Ancestor** in this column indicates that the inheritance is already exists but via one or more members in class hierarchy.

Difference between Parent and Ancestor is illustrated by Figure 3.

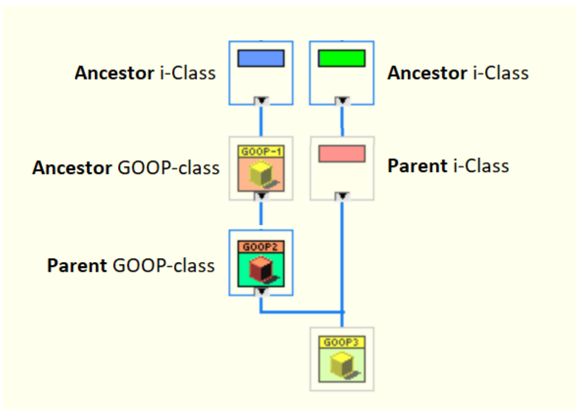


Figure 3 Example of newly created *AZI method*

Note:

If column **Ancestorship** contains word **Ancestor**, implementation of direct inheritance (changing ancestor to parent) is not forbidden. However, this does not add any functionality to the class while adding accessors to its constructor and destructor.

4.3 Creating new method

1. Right-click **GOOP4** class, **i-Class**, or **Interface** and select menu **AZ-MultiInheritance > Add Method…**
2. Corresponding GUI will be opened
3. If possible, icon editor will be opened during creatio of the method (see chapter 5).

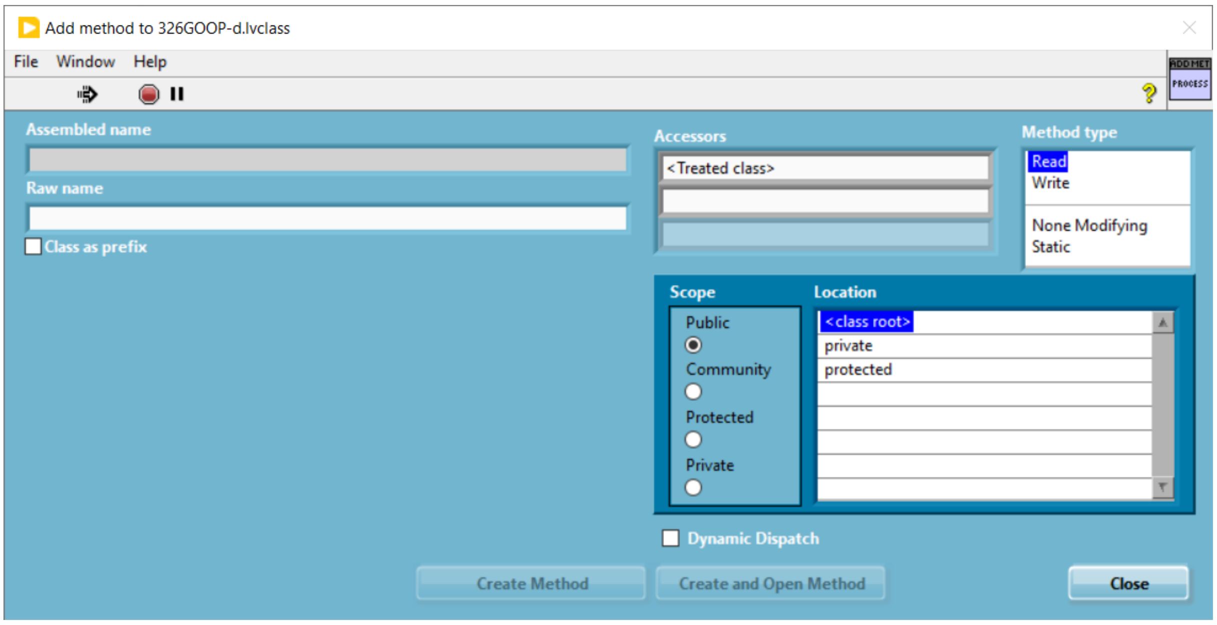


Figure 4 GUI for creating new method

The fields are:

- Indicator **Assembled name** shows the method name. The font turns red if the method already exists in the class.
- Control **Raw name** is used to input editable part of the name.
- Checkbox **Class as prefix** can be selected to prepend method name with class name.
- Selectors **Accessors** allow adding up to three attribute accessors in BD of the method. Contrary to GOOP4, where only single accessor of the treated class can be used, this tool allows using accessors of superclasses.
- Other controls do not differ from corresponding GOOP4 controls.

Note:

Field **Method Type** has higher priority than **Accessors**. If **Method Type** is set to **Read** or **Write** but list **Accessors** is empty, class own accessor is used for **i-Classes** and **GOOP4** classes.

4.4 Consistency control

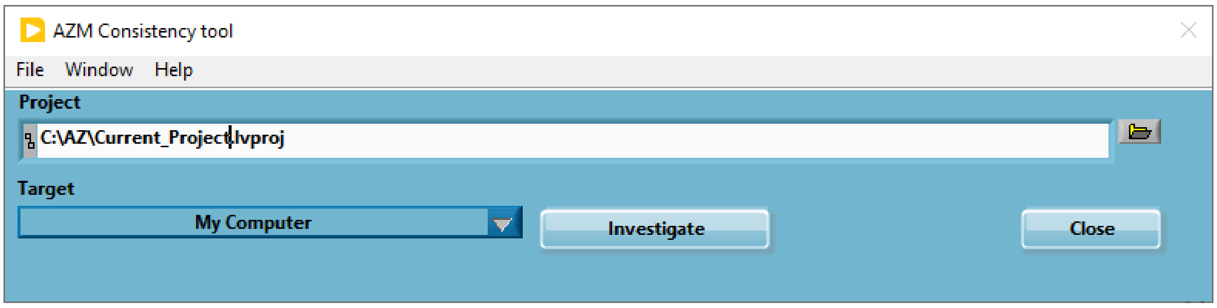
- Each **i-Class** class inheriting from ancestor **i-Class** must call one constructor and one destructor belonging to closest in the class hierarchy **i-Class**.
- Each **GOOP4** directly inheriting from **i-Class** must call one **i-Class** constructor per class constructor BD and one **i-Class** destructor in the destructor BD.

Consistency can be controlled with **AZM Consistency tool**.

Simple cases of inconsistency can be repaired with the same tool. More complicated cases must be fixed manually.

4.4.1 Using consistency tool

1. Select menu **Tools** > **AZ-MultiInheritance** > **AZM Consistency tool…** .
2. Use **opened GUI** to select project and target.


AZM Consistency tool

File Window Help

Project

C:\AZ\Current_Project.lvproj

Target

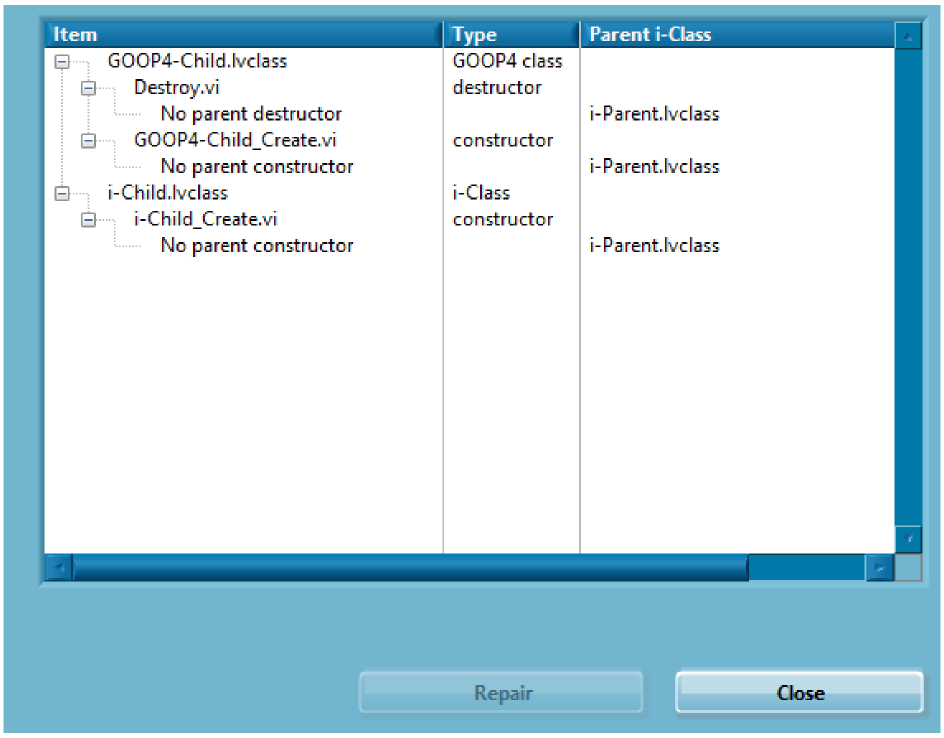
My Computer

Figure 5 First GUI of Consistency tool

3. Click **Investigate**.
4. List of inconsistencies is shown. Corresponding dialog is described in section 4.4.2.
5. Select inconsistency and click **Repair**.

4.4.2 Investigation results GUI

Inconsistency list is shown in Figure 6.



Item	Type	Parent i-Class
GOOP4-Child.lvclass	GOOP4 class	
Destroy.vi	destructor	
No parent destructor		i-Parent.lvclass
GOOP4-Child_Create.vi	constructor	
No parent constructor		i-Parent.lvclass
i-Child.lvclass	i-Class	
i-Child_Create.vi	constructor	
No parent constructor		i-Parent.lvclass

Figure 6 Example of investigation results

- First hierarchical level lists inconsistency-containing classes.
- First hierarchical level lists inconsistency-containing methods.
- The third level lists inconsistencies.

Column **Inconsistency** can have following values:

Inconsistency	Explanation	Expected action
Class has no constructor	Class constructor is missing.	**GOOP4** class can be repaired only manually while **i-Class** can be repaired automatically. The newly created **i-Class** constructor should be attended and further developed manually
Class has no destructor	Class destructor is missing.	**GOOP4** class can be repaired only manually while **i-Class** can be repaired automatically. The newly created **i-Class** destructor should be attended and further developed manually
No parent constructor	Constructor or accessor does not call **i-Class** accessor with option **Create**.	Can be repaired automatically.
No parent destructor	Destructor or accessor does not call **i-Class** accessor with option **Cleanup**.	Can be repaired automatically.

Parent has no constructor	Parent **i-Class** constructor is missing.	Can be repaired automatically selecting corresponding problem presented for the parent **i-Class** in the same GUI.
Parent has no destructor	Parent **i-Class** destructor constructor is missing.	Can be repaired automatically selecting corresponding problem presented for the parent **i-Class** in the same GUI.

5 Icon editors

AZ Multi-inheritance has no own icon editors. It borrows the editors from GOOP Development Suite if the suite is installed.

The compatibility table is provided below.

GOOP development suite	**Tested versions of VIPM packages**	**LabVIEW versions**	**i-Class icons**	**Method icons**
NI	1.1.87.94	2021 32-bit	No	Yes
Open source	1.2.72.135	2020 64bit	Yes	Yes

Unfortunately, testing all possible combinations is out of my capacity /Andrei Zagorodni

6 Using i-Classes

6.1 *Specificity of i-Class code*

6.1.1 Order of parent constructors and destructors

Calls of parent **i-Class** constructor and destructor are similar to corresponding calls of **GOOP4** class members:

- BD of child class constructor must call the parent constructor.
- BD of child class destructor must call parent destructor.

I.e., the constructor of any class inheriting from **i-Class** must call the constructor of the parent class

AZM toolkit adds parent **i-Class** constructors to child class constructor and parent **i-Class** destructor to child class destructor. This is true for both **GOOP4** and **i-Class** child classes.

If a class inherits from an Interface while the Interface inherits from an i-Class, calls to the constructor and destructor of the ancestor **i-Class** are added to child class BD-s. I.e., presence of the **Interface** "between" the child and the ancestor is ignored.

6.1.2 Order of parent constructors and destructors

If class inherits from multiple **i-Classes**, order of calls for the parent constructors/destructors is not established by AZM toolkit. If the order is important for particular application, it has to be established manually.

Calls of parent of **i-Class** constructors must be placed after utility `[child_class_name]_New.vi`.

6.1.3 Repeated calls of constructor and destructor

Multiple inheritance allows creating sophisticated class hierarchy. A common case is a class inheriting from an ancestor **i-Class** "twice or more" through different direct parents, see Figure 7. Repeated calls of **i-Class** constructor/destructor have no effect.

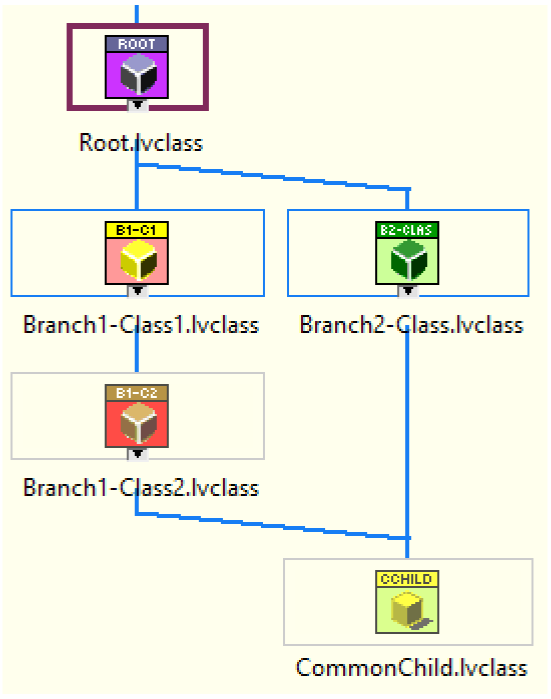


Figure 7 Example a class (`CommonChild`) inheriting from same ancestor (`Root`) two times.

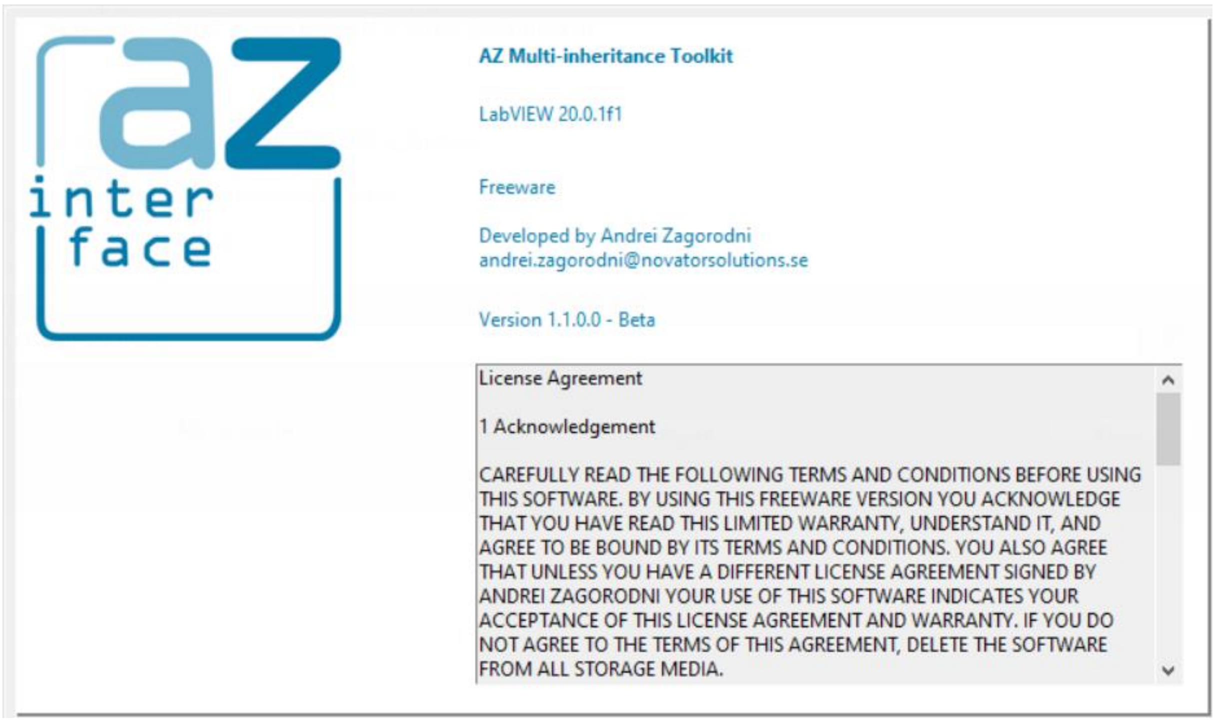
I.e., programmers should not worry about behavior of constructors/destructors in complicated hierarchies.

6.2 Recommendation

6.2.1 Code complexity

Multiple inheritance adds a new dimension in hierarchical structures of the classes**. Keeping i-Classes simple** is highly recommended. Following this rule should reduce efforts required to debug projects with multiple inheritance.

7 About and Contacts



AZ Multi-inheritance Toolkit

LabVIEW 20.0.1f1

Freeware

Developed by Andrei Zagorodni
andrei.zagorodni@novatorsolutions.se

Version 1.1.0.0 - Beta

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.

Figure 8 About

7.1 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.

7.2 Contacts

Andrei Zagorodni

`andrei.zagorodni@novatorsolutions.se`

Please write **AZI** or **AZ Multi-inheritances** in the subject line.

Website of the project:

https://azinterface.net

7.3 Support and communications

I shall appreciate any feedback on AZM Toolkit.

I promise to read your emails and reply within a reasonable time. However, the project is developed in my evenings and weekends. Thus the "reasonable time" will depend solely on my workload.

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

Andrei Zagorodni