

Dr. Holger Brand

GSI Helmholtzzentrum für Schwerionenforschung GmbH

Workshop

**Object-oriented Programming with LabVIEW**





[Creative Commons Lizenzvertrag](http://creativecommons.org/licenses/by-nc-sa/3.0/de/)  
Workshop: Objekt-orientiertes Programmieren mit LabVIEW von Dr. Holger Brand steht unter einer [Creative Commons Namensnennung - Nicht-kommerziell - Weitergabe unter gleichen Bedingungen 3.0 Deutschland Lizenz](http://creativecommons.org/licenses/by-nc-sa/3.0/de/). Über diese Lizenz hinausgehende Erlaubnisse können Sie unter [tt-info@gsi.de](http://creativecommons.org/choose/tt-info@gsi.de) erhalten.

Table of contents

[Overview 3](#_Toc345317914)

[Exercise 1: Unpack and open the project 4](#_Toc345317915)

[Exercise 2: Create a library 5](#_Toc345317916)

[Exercise 3: Create a class 6](#_Toc345317917)

[Create new class 6](#_Toc345317918)

[Change inheritance 7](#_Toc345317919)

[Add attributes 8](#_Toc345317920)

[Add items to ini-file 8](#_Toc345317921)

[Exercise 4: Add *Methods* 9](#_Toc345317922)

[Access rights 9](#_Toc345317923)

[Exercise 4.1 Data access methods 9](#_Toc345317924)

[Exercise 4.2 Work with LabVIEW Objects 10](#_Toc345317925)

[Exercise 4.3 *Overwrite*-Methods 10](#_Toc345317926)

[Exercise 4.4: *Dynamic Dispatch*-Methods 12](#_Toc345317927)

[Übung 5: Weitere *Interface*-Klassen 13](#_Toc345317928)

[Referenzen 14](#_Toc345317929)

[Notizen 15](#_Toc345317930)

[Notizen 16](#_Toc345317931)

[Notizen 18](#_Toc345317932)

# Overview

This workshop extends the talk *Introduction to Obejct-oriented Programming with LabVIEW* held at VIP 2012 in the series of lectures in LabVIEW Power Programming with exercises.

This course will apply LabVIEW classes to read and write configuration data which are often saved in ini-files. In this project the complete configuration data is stored in an instance of class *Configuration*. Sections are implemented as classes derived from the base class *Section*. The access to storage medium is abstracted in the base class *Interface*, derived classes implement the concrete access. The *Ini-File* class already exists and is used during these exercises. More Interface classes can be implemented to access XML files or databases[[1]](#footnote-1).

During the workshop you will familiarize yourself with the class library first. Figure 1 shows the project and class library. Then you will learn how to create a new class for your own section. You will change the inheritance of that new class to make it a derived class of *Section*. Next you will add attributes to your class where the item data becomes stored and you overwrite the dynamic dispatch VI’s of the ancestor class to read and write the new items by using the Base class *Interface*. Finally you will tackle the challenge to add new functions by creating new dynamic dispatch and corresponding overwrite VI’s.

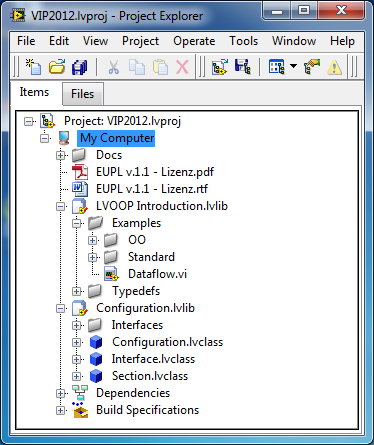
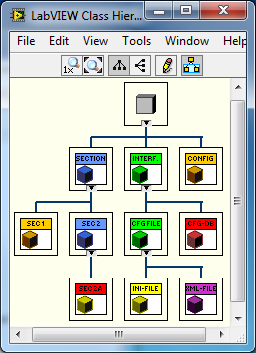
 

Figure 1: Project & Class Hierarchy

# Exercise 1: Unpack and open the project

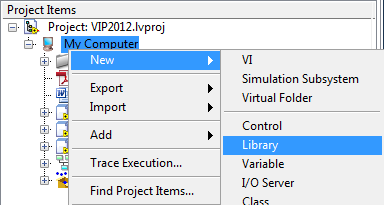
This chapter describes how to unpack the project files (if not already installed) and to familiarize yourself with the LabVIEW project.

1. Unpack the file *VIP 2012 LVOOP Workshop.zip* using its context menu  
   → Choose a new folder *<Project folder>[[2]](#footnote-2)*.
2. Go to the folder *<Project folder>\VIP2012*
3. Double click *VIP2012.lvproj* to open the project.
   1. Alternatively start LabVIEW 2012  
      *Windows Start→All Programs→National Instruments LabVIEW 2012*
   2. Select Menu*→File→Open Project…*
   3. Navigate to project file *VIP2012.lvproj* and click OK to open the project.
4. The project should contain some documents as well as license files and of course the LabVIEW libraries that contain the classes and application examples.
   1. *Doc*  
      Talk and this manual.
   2. *Configuration.lvlib*  
      This LabVIEW library contains necessary base classes for this project.
      1. *Configuration.lvclass*, This class contains an Interface and after reading the complete configuration data.
      2. *Interface.lvclass* and its derived classes, especially the already existing derived class *Ini-File.lvclass* which implements the access to an ini-file.
      3. *Section.lvclass*, ancestor class for all application specific derived classes of *Section*.
   3. *LVOOP Introduction.lvlib*  
      This LabVIEW library contains example VI’s, classes and ini-files.
      1. Reading of configuration data using the standard VI’s and typedefinitions you find in the virtual folder: *Examples\Standard*
      2. You find the example for reading configuration data with the object-orientiented approach in the virtual folder: *Examples\OO*. There you can also find the derived classes of *Section.lvclass* used in this example.
         1. *Section 1.lvclass*
         2. *Section 2.lvclass*
            1. *Section 2a.lvclass, a* derived class of *Section 2.lvclass*.
   4. *DB\_Configuration.lvlib* is not used in this workshop. It contains a database design with example configuration data for a SQL database accessed via ODBC.
5. You need to adopt the *ClassPath*- items to he actual paths.
6. Execute the examples to familiarize yourself with the problem. You can *Drag&Drop* the corresponding ini-file from the Project to the path controls on the front panel. Do you recognize the differences?

# Exercise 2: Create a library

You should best save the results of your exercises in a new folder, e.g. *<Project folder>\yourName*.

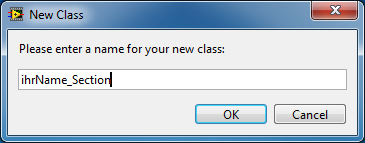
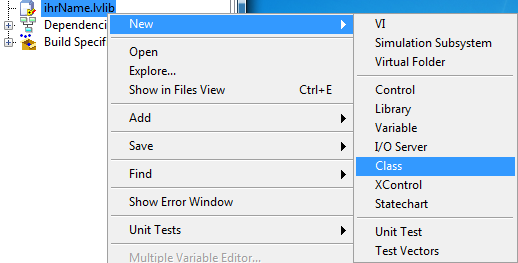
In the end of the workshop you can copy the complete *<Project folder>* to a USB-Stick in order to take it home to be used as base for more exercises or reuse with your own projects.

1. Create a new library in the project.  
   (Context menu of *My Computer>New>Library*) 
2. Save this library to *<Project folder>\yourName\yourName.lvlib*  
   (Context menu of *Untitled Library 1>Save>Save as…*)
3. Open the properties of the library  
   (Context menu of *yourName.lvlib>Properties*)
   1. In the category *General Settings* you can write your initials in the *VI Icon Template*. This template is used as base for new VI‘s.
   2. Enter a brief description in the category *Documentation*.
   3. Close the dialog and save the project.

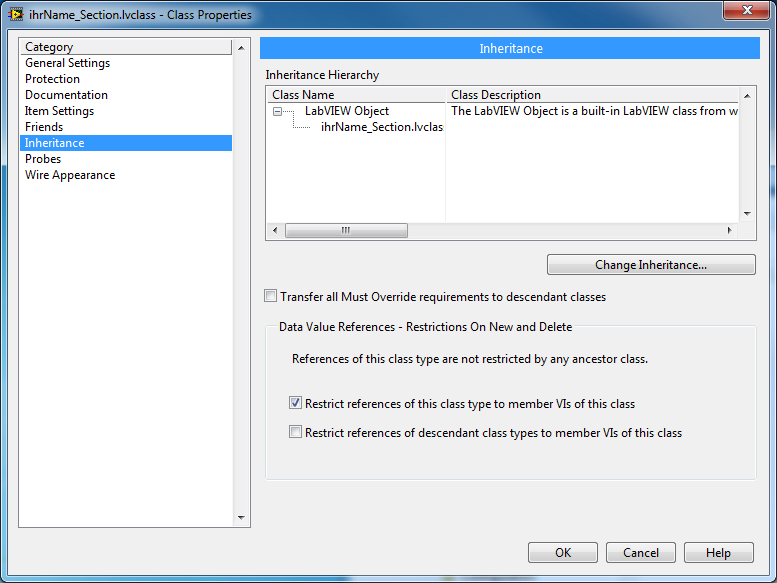
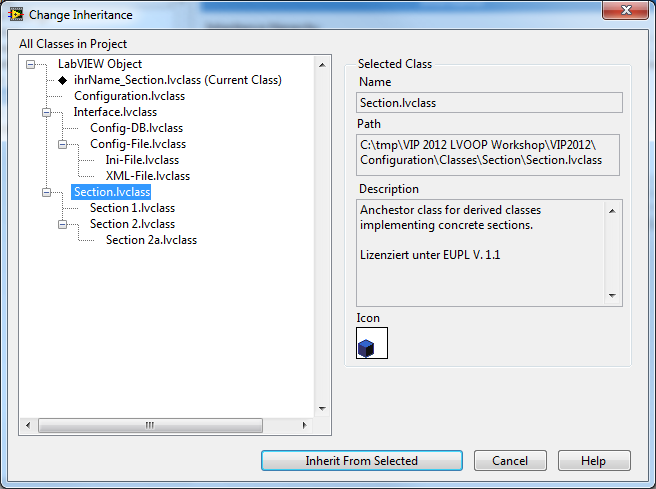
# Exercise 3: Create a class

In this exercise you create your own class. It should model your own section, so it needs to inherit from *Section.lvclass*. You will add the necessary attributes to *Class private data* and overwrite the die *dynamic dispatch*-VI‘s of the ancestor class to read and write your own items. You will extend the existing configuration file with a new section and its items. You will test it with the already existing example-VI’s.

## Create new class

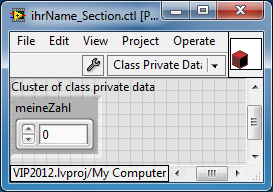
1. Create a new class named *ihrName\_Section.lvclass* in your library.  
   (Context menu of *ihrName.lvlib>New>Class*) 
2. Open the properties of the new class  
   (Context menu of *yourName.lvclass>Properties*)
   1. In the category *General Settings* you can write your initials in the *VI Icon Template*. This template is used as base for new VI‘s.
   2. Enter a brief description in the category *Documentation*.
   3. Close the dialog and save the project.

## Change inheritance

1. Reopen the properties of class.
2. In the category *Inheritance* you can inspect the inheritance hierarchy of your class. A new class always inherits from *LabVIEW Object*. 
   1. For this time keep all options unchanged. The meaning of those is not part of this introductory workshop.
   2. Click the button *Change Inheritance*. Select the desired ancestor class in the Dialog, in this example it is *Section.lvclass*. Click *Inherit From Selected* to change the inheritance hierarchy.
   3. Close the dialog and save the project.
3. Open the LabVIEW Class Hierarchy (*Menu>View*). You can find your class in the inheritance tree blow the class *Section.lvclass*.

## Add attributes

Expand your class and open *yourName\_Section.ctl*. (This corresponds to the type definition with standard LabVIEW programming.)

1. Add your desired attributes from the controls palette to the *Cluster of class private data* and name them with meaning, e.g. a *Numeric* Control of type *double* with name *myNumber*. 
2. You should also fill *Description* and *Tip* for all attributes, refer to context menu.
3. You can also fill the *VI-Description* in the *Control Properties*.
4. Close this *Control* and save the project.

## Add items to ini-file

Open *LVOOP Introduction.lvlib:Examples\OO\Demo\_OO.ini* and save a copy in your own folder, e.g. *<Project folder>\yourName\yourName.ini*.

1. Add a new section.
2. Enter the *ClassPath* to your new section class.
3. For each attribute in your *class private data* add an *Item* with value.
4. The result could look like this[[3]](#footnote-3):  
   [yourName\_Section\_0]  
   ClassPath="<Project folder>\yourName\yourName\_Section\yourName\_Section.lvclass"  
   myNumber=987.654  
   [yourName\_Section\_1]  
   ClassPath="<Project folder>\yourName\yourName\_Section\yourName\_Section.lvclass"  
   myNumber =876.543
5. Save and close the file.
6. Add the new ini-file to the library as example and save the project.

# Exercise 4: Add *Methods*

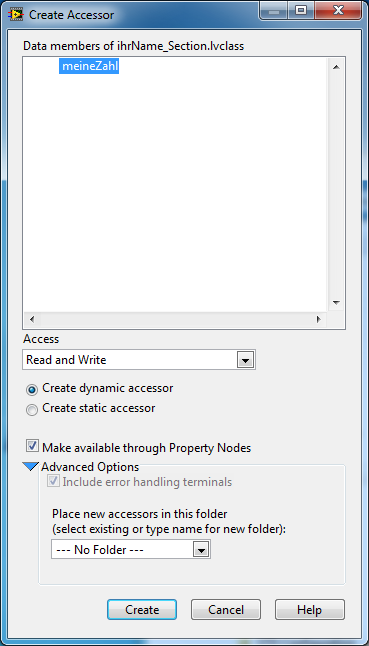
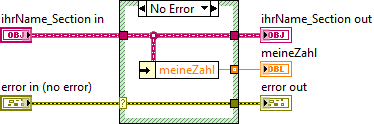
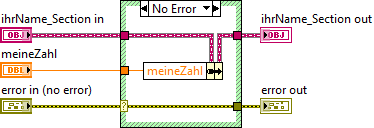
## Access rights

**Attributes of a class are always declared as private**. One cannot use *Un-/Bundle By Name* to access class attributes in VI‘s, that do not belong to the class itself. The calling VI would not be executable. Therefore the class needs to provide accessor-VI’s that have the appropriate access scope. The access scope can also be selected[[4]](#footnote-4) for all other class VI’s as desired.

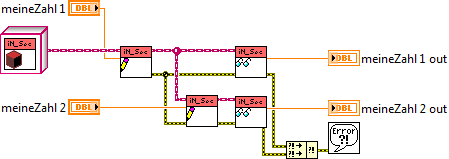
* *Public*: Can be used by all VI‘s.
* *Protected*: Can be used by VI’s of the class itself and derived classes.
* *Private*: Can be used by VI’s of the class itself only.
* *Community*: Can be used by VI’s of the class itself and their friends[[5]](#footnote-5).

## Exercise 4.1 Data access methods

Create VI’s to read or write attribute data for your class.

1. Select in context menu of your class *New>VI for Data Member Access…*
   1. Select the desired attributes[[6]](#footnote-6) in the dialog, e.g. *myNumber* in this example.
   2. Select the access method: *Read and Write*.
   3. Keep the other option unchanged.
   4. Click *Create*. The wizard will create the corresponding VI‘s.
   5. Provide icon and description for new VI’s and their controls and indicators on the front panel elements.
   6. Save new VI’s with default names.
   7. Their block diagrams should look similar to the following:   
      

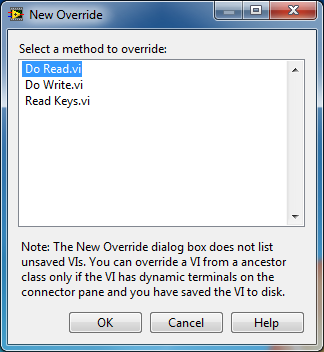
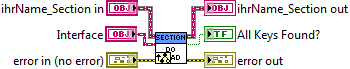
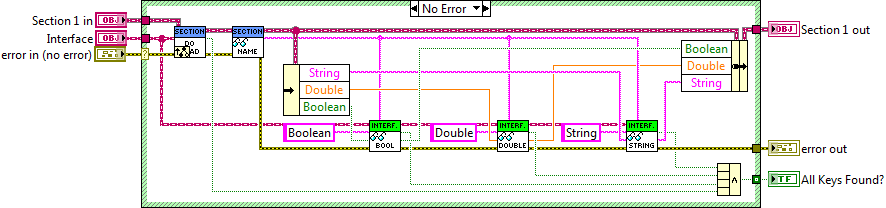
## Exercise 4.2 Work with LabVIEW Objects

1. Create a standard VI within your library, e.g. *Test\_DataAccess.vi.*
2. Drag & drop your class from the project explorer onto the block diagram of the new VI. By this means you create an object of that class with default data.
3. Drag & drop corresponding read and write accressor Vis onto the block diagram.
4. Wire the nodes, so that you write data to the object and read it back.
5. Create the corresponding control and indicator on the front panel and wire them to the accessor VIs.
6. The VI could look like this:  
    
7. Set a number on the front panel and execute the VI. Observe the result.
8. Extend the VI by forking the object wire after the first write VI, set another value and display both values. Observe the result.  
   
9. Set different numbers on the front panel and execute the VI. Observe the result.
10. Remember the effect of dataflow! The object is cloned at the wire fork.
11. Modify the VI with respect to your own ideas, e.g. use two loops.
12. Ask questions.

## Exercise 4.3 *Overwrite*-Methods

Add *Overwrite*-VI’s for the *dynamic dispatch*-VI’s defined by the ancestor class *Section.lvclass*.

* *Do Read.vi* reads data from the medium (Interface) and stores them in the object attributes.
* *Do Write.vi* writes data, that are stored in the object attributes to the medium (Interface).
* *Read Keys.vi* converts object attributes to *strings* und appends them together with their key name to an array.

1. Select in context menu of class *New>VI for Overwrite…*
2. Select *Do Read.vi* and click the OK button.  
    
   1. The *Wizard* creates an *Overwrite*-VI as starting point for your implementation. It calls the corresponding VI of its *ancestor* class, e.g. Do Read.vi.  
      
   2. Use your own class Vis and those of its ancestor class als well as methods of the base class *Configuration.lvlib:Interface*, in order to read your items.
   3. You can refer to corresponding example classes in *LVOOP Introduction.lvlib:*  
      
3. Repeat step 1 and 2 also for *Do Write.vi* und *Read Keys.vi*.
4. Execute *Read Configuration OO.vi* in *LVOOP Introduction.lvlib* again. You should be able to find your own item values in the displayed array.
5. Save a copy of this VI and modify it to your needs, e.g. change the value of an item and write the configuration back to a new ini-file.

## Exercise 4.4: *Dynamic Dispatch*-Methods

Create new methods analog to *Configuration.lvclass:Read Keys.vi* respectively *Section.lvclass:Read Keys.vi* to readout and display all *items* of type *double* as *double* values.

**Hint**: Create a *static dispatch*-VI in *Configuration.lvclass* und a *dynamic dispatch*-VI in *Section.lvcass as well as* neseccary *overwrite*-VI’s in the derived classes.

# Exercise 5: More *Interface* classes

**Homework:** You can create derived classes of *Interface.lvclass* to store configuration data in aother file based formats, e.g. XML (*eXtensible Markup Language*), or in database. Corresponding base classes are alread prepared in *Configuration.lvlib*.

* *Config-DB.lvclass*
* *Conig-File.lvclass*
  + *XML-File.lvclass*

Various Document Type Definitions (DTD) are possible in the framework of XML. So, you should derive your own *XML-Interface* class from *XML-File.lvclass* ancestor class.

Since there are a lot of different databases available you should derive your own *DB-Interface* class from *Config-DB.lvclass* ancestor class. SQL scripts and utility VI’s are prepared in *DB\_Configuration.lvlib*, to create an example Oracle database and to insert and drop data which can serve as a starting point for your own configuration database.

# References

The following list provides starting points for the object-orientiented programming in LabVIEW.

* LabVIEW Menue>Help>LabVIEW Help... -> Contents -> Fundamentals -> LabVIEW Object-Oriented Programming
* LabVIEW Menue>Help>Find Examples -> Browse by Task -> Fundamentals -> Object-Oriented
* [LabVIEW Object-Oriented Programming: The Decisions Behind the Design](http://zone.ni.com/devzone/cda/tut/p/id/3574)
* [LabVIEW Object-Oriented Programming FAQ](http://zone.ni.com/devzone/cda/tut/p/id/3573)
* [Applying Common OO Design Patterns to LabVIEW](http://decibel.ni.com/content/docs/DOC-2875)
* [HGF Baseclass Library](http://wiki.gsi.de/cgi-bin/view/NIUser/HGFBaseClassLibrary)
* [Mobile Agent System](http://wiki.gsi.de/cgi-bin/view/NIUser/LVMobileAgentSystem)
* [Actor Framework](https://decibel.ni.com/content/docs/DOC-17193)
* [Measurement Abstraction and Model-View-Controller (MVC) Project with Actor Framework in LabVIEW](https://decibel.ni.com/content/docs/DOC-21441)

# Notice

# Notice

# Notice

1. A library containing an ODBC-Interface and Tools is already prepared. [↑](#footnote-ref-1)
2. The concrete project folder to be used will be announced in the beginning of the workshop. [↑](#footnote-ref-2)
3. Please remember the decimal delimiter defined in System Settings>Regional Settings [↑](#footnote-ref-3)
4. *Access Scope* is available in context menu of elements in a LabVIEW library or class. [↑](#footnote-ref-4)
5. Friends must be declared in the class properties. [↑](#footnote-ref-5)
6. In each VI created by the wizard only one element is used. If you would like to read or write more attributes at once, select one attribute only and modify the VI manually. [↑](#footnote-ref-6)