SimpleOOP – Opensource OOP Plugin

Einführung

SimpleOOP brings PureBasic an easy to use OOP support without complicated syntax and a special emphasis on clear and simple code.

SimpleOOP is Opensource (see License.txt) to avoid problems with future Purebasic versions and nobody has to worry that the plugin does not work anymore because of a new Purebasic update.

To install SimpleOOP, simply run the "SimpleOOP Installer.exe". After the installation you can start working immediately. The installer does all the installation.

For the first start, it’s the best to read this manual and also try the Examples in the supplied Examples Folder, where all commands are explained, too.

greets
Sirhc.ITI

Manual Installation – PureBasic IDE

- Rename the file "PBDebugger.exe" to "_PBDebugger.exe" in the folder "PureBasic\Compilers\". And afterwards copy the supplied "PBDebugger.exe" in the folder.

- Copy "SimpleOOP.res" into "PureBasic\Residents\"

- Copy "SimpleOOP.exe" to a convenient place and then create the following entries in the IDE

Configure Tools: (SimpleOOP.exe)

1.
- Name: SimpleOOP Debug Start
- Arguments: DebugStart "%CompileFile" "%CompileFile" "%Path" Format
- Working Directory:
- Event to trigger the tool: Before Compile/Run
- Settings: "Wait until tool quits" activated

2.
- Name: SimpleOOP Debug Stop
- Arguments: DebugStop "%CompileFile" "%Executable" "%Path" Format
- Working Directory:
- Event to trigger the tool: After Compile/Run
- Settings: "Wait until tool quits" deactivated

3.
- Name: SimpleOOP Compile Start
- Arguments: CompileStart "%CompileFile" "%CompileFile" "%Path" ""
- Working Directory:
- Event to trigger the tool: Before Create Executable
- Settings: "Wait until tool quits" activated

4.
- Name: SimpleOOP Compile Stop
- Arguments: CompileStop "%CompileFile" "" "%Path" ""
- Working Directory:
- Event to trigger the tool: After Create Executable
- Settings: "Wait until tool quits" activated

Info

- You can leave out the argument "Format", if you want to see the converted code in the Standalone GUI Debugger

- You can add the SimpleOOP Keywords in the IDE Settings under "Custom keywords", "folding" and "identation"

- While holding the "Control“ key during start of the Compilation process, the converted code will be opened afterwards

- Copy "SimpleOOP ClassViewer.exe" to a convenient place and then create the following entries in the IDE

Configure Tools: (SimpleOOP ClassViewer.exe)

1.
- Name: SimpleOOP ClassViewer
- Arguments: "%TempFile”
- Working Directory:
- Event to trigger the tool: Menu Or Shortcut (Strg + 1)
- Settings: "Wait until tool quits" deactivated

2.
- Name: SimpleOOP ClassViewer AutoCompleteOnBackslash
- Arguments: AutoCompleteOnBackslash
- Working Directory:
- Event to trigger the tool: Editor Startup
- Settings: "Wait until tool quits" deactivated

Info

- You can leave out the tool "SimpleOOP ClassViewer AutoCompleteOnBackslash", if you don’t want automatic Autocomplete when typing a "\"

- You can add the SimpleOOP Keywords in the IDE Settings under "Custom keywords", "folding" and "identation"

Manual Installation – jaPBe IDE

- Rename the file "PBDebugger.exe" to "_PBDebugger.exe" in the folder "PureBasic\Compilers\". And afterwards copy the supplied "PBDebugger.exe" in the folder.

- Copy "SimpleOOP.exe" to a convenient place and then create the following entries in the IDE

Werkzeuge konfigurieren: (SimpleOOP.exe)

1.
- Name: SimpleOOP Debug Start
- Arguments: DebugStart "%CompileFile" "%CompileFile" "%Path" Format
- Working Directory:
- Event to trigger the tool: Before Compile/Run
- Settings: "Wait until tool quits" activated

2.
- Name: SimpleOOP Debug Stop
- Arguments: DebugStop "%CompileFile" "%Executable" "%Path" Format
- Working Directory:
- Event to trigger the tool: After Compile/Run
- Settings: "Wait until tool quits" activated

3.
- Name: SimpleOOP Compile Start
- Arguments: CompileStart "%CompileFile" "%CompileFile" "%Path" ""
- Working Directory:
- Event to trigger the tool: Before Create Executable
- Settings: "Wait until tool quits" activated

4.
- Name: SimpleOOP Compile Stop
- Arguments: CompileStop "%TempFile" "" "%Path" ""
- Working Directory:
- Event to trigger the tool: After Create Executable
- Settings: "Wait until tool quits" activated

Werkzeuge konfigurieren: (SimpleOOP ClassViewer.exe)

5. (Werkzeug wird Manuel aus dem Menü Gestartet)
- Name: SimpleOOP ClassViewer
- Arguments: "%TempFile”
- Working Directory:
- Event to trigger the tool: Menu Or Shortcut (Strg + 1)
- Settings: "Wait until tool quits" deactivated

Info

- You can leave out the argument "Format", if you want to see the converted code in the Standalone GUI Debugger

- You can add the SimpleOOP Keywords in the jaPBe IDE Settings under "Custom keywords", "folding" and "identation"

- While holding the "Control“ key during start of the Compilation process, the converted code will be opened afterwards

ClassViewer

The ClassViewer has two functions, first is to show all classes of the source code with all included files. And on the other hand it’s an Autocomplete tool that shows the class of the selected object.

You can start the ClassViewer manual in the IDE. The ClassViewer lists all classes and his Attributtes, Methodes etc.

If you move your cursor in the code behind a Backslash (e.g. "*Obj\", "Parent\", "This\"), then the Classviewer shows you an Autocomplete list, if the selected object has a class.

If you have activeted the IDE Tool "SimpleOOP ClassViewer AutoCompleteOnBackslash", then the Autocomplete list automatically opens after typing a Backslash, if the selected object has a class.

The autocomplete lists shows only the content of the class that is visible at the current scope in the source code, it means private or protected methods/attributes will be automatically be shown/hidden. This even works with nested objects (e.g. "This\obj\obj2\").

If the autocomplete/class list is open, you can simply select an entry by writing the name or "Arrow up/down" on your keyboard. Double click or press return, to insert the entry at the current position in the IDE.

If you start the ClassViewer when you’re behind "NewObject.Class" with the cursor, then the init parameter will be inserted in the IDE, if they exist.

By right clicking on a Method, Attribut or Procedure in the ClassViewer, you can jump to the line in the source code.

By right clicking on a node in the ClassViewer, it will collapse all other nodes and expand the current one.

Polymorphie

In SimpleOOP you have the possibility to overwrite methods (Attributes can't be overwritten). The child method always overwrites the parent method. When you overwrite methods, keep in mind that the return type and parameters are the same. It's possible not to do that, but that could lead to problems… e.g. if a parent class wants to call a method which is being overwritten by a child method, but the parameters of the child method has been changed, that could course IMA's. You also can't overwrite a parent method which has a different scope type e.g. both methods must be public or protected, a public method can't overwrite a protected and vice versa.

Keywords

- Singleton
- Class/EndClass
- Abstract
- Private/BeginPrivate/EndPrivate
- Protect/BeginProtect/EndProtect
- Public/BeginPublic/EndPublic
- Method/MethodReturn/EndMethod
- Parent
- This
- NewObject
- FreeObject
- _ (Multiline support)

Attributes

Attributes are like variables in a structure. Arrays, Maps and Lists are supported in classes, too.

Example:

```
Class MyClass
    Val.l
    A.q
    B.f
    String$
    Map Tools()
    List MyList()
    Array MyArray(5)
EndClass
```

Singleton

- Makes the class static
- On every NewObject, it returns the first initiated object
- The constructor Init() will only be called on the first NewObject
- FreeObject will be ignored, the destructor Release() will not be called

Example:

```
Singleton Class MyClass
        Method Test()
                ; Code
        EndMethod
EndClass
```

Class/EndClass

- Like Structures, supports Extends
- Methods will be written directly in the class
- Prototypes can be written directly in the class
- You can set a startup value for attributes

Example:

```
Class MyClass
        String$
        Val.l = 123
        List MyList()
        Map MyMap()
        Array NewArray(5)

        Public Method Test()
                ; Code
        EndMethod

        Prototype.i Proc(Num.d, *Pointer)

EndClass
```

Abstract

- For creating of abstract methods
- Abstract methods must be overwritten by their child methods, otherwise you can't initiate the class

Example:

```
Class MyClass
        String$
        Val.l

        Abstract Test()
        Public Abstract DoThat(Val.l, String$)

EndClass
```

Private/BeginPrivate/EndPrivate

- For private attributes and methods in classes
- Private attributes or methods can only be called in their own class
- On default, all objects in a class are private, this keyword is basically only for a better overview

Example:

```
Class MyClass

    String$ ; Is private, too

    BeginPrivate
        Val.l
    EndPivate

    Private Method Test()
        ; Code
    EndMethod

    BeginPrivate
        Method Proc()
            ; Test
        EndMethod
    EndPrivate

EndClass
```

Protect/BeginProtect/EndProtect

- For protected attributes or methods in classes
- Protected attributes or methods can only be called from their own class or a child class

Example:

```
Class MyClass

    Protect String$

    BeginProtect
        Val.l
    EndProtect

    Protect Method Test()
        ; Code
    EndMethod

    BeginProtect
        Method Proc()
            ; Test
        EndMethod
    EndProtect

EndClass
```

Public/BeginPublic/EndPublic

- For public attributes or methods in classes
- Public attributes or methods can be called from everywhere in the code

Beispiel:

```
Class MyClass

    Public String$

    BeginPublic
        Val.l
    EndPublic

    Public Method Test()
        ; Code
    EndMethod

    BeginPublic
        Method Proc()
            ; Test
        EndMethod
    EndPublic

EndClass
```

Method/MethodReturn/EndMethod

- Like procedures
- You can work with the keyword "This" inside
- Methods are written directly in the class

Beispiel:

```
Method.s Test()
    ; Code
    MethodReturn "Hallo"
EndMethod
```

Parent

- To call a parent method of a method
- Please note that when other methods are called in the parent method, the parameters of the parent methods must be the same like the current ones (See chapter Polymorphie)

Example:

```
Class Second Extends First
    Method.s Test()
        Parent\Test() ; Calls Test() in Class First
    EndMethod
EndClass
```

This

- To access the own class content in methods
- You can als write *This

Example:

```
Method.s Test()
    Debug This\Val
EndMethod
```

NewObject

- To create class objects
- supports Objects, Arrays, Lists, Maps…
- Objects have to start with an "*"
- The constructor method "Init()" will be called, if exists
- You can pass parameters to "Init()" after "NewObject"
- You have to declare the class that will be initiated after "NewObject"

Example:

```
*Object.MyClass = NewObject.MyClass

Dim *Array.MyClass(10)
    For I=0 To 10
        *Array (I) = NewObject.MyClass()
    Next

    NewList *Liste.MyClass()
        AddElement(*Liste())
            *Liste () = NewObject.MyClass

*Object.MyClass = NewObject.MyClass("String", 123)
```

FreeObject

- Deletes the object
- The destructor method "Release()" will be called, if exists
- Sets the object automatically to 0
- Please note, if you free "This", the object outside the scope, will not be set to 0, because the "This" Pointer is handed by "byVal"

Example:

```
*Object  = FreeObject
*Array(3) = FreeObject
This = FreeObject

ForEach *Liste()
    *Liste () = FreeObject
    DeleteElement(*Liste())
Next
```

_ (Multiline support)

- To split a command to multiple lines
- You can split a command on as much lines you want to
- If an error occurs on a splitted line, the debugger will highlight the first

Example:

```
OpenWindow(0, 0, 0, 640, 480, _
                "PureBasic Window", _
                #PB_Window_SystemMenu |  _
                #PB_Window_MinimizeGadget | _
                #PB_Window_MaximizeGadget | _
                #PB_Window_ScreenCentered)

Repeat
        Event = WaitWindowEvent()

        If Event = #PB_Event_CloseWindow
                End
        EndIf
ForEver
```