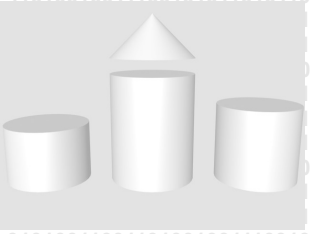


ODABA^{NG}

Quick Starter Guide

Summary

The document provides necessary steps to get a project implemented quickly. It includes quick start of the ClassEditor as well as a short starter guide for the designer.

run Software-Werkstatt GmbH
Weigandufer 45
12059 Berlin

Tel: +49 (30) 609 853 44
e-mail: run@run-software.com
web: www.run-software.com

Berlin, May 2013

Table of Contents

1 Introduction

ODABA

ODABA is a terminology-oriented database system that allows storing objects and methods as well as causalities . As terminology-oriented database, ODABA supports complex object types (user-defined data types) defined in a terminology model, which reflect application relevant concepts.

ODABA applications are characterized by high flexibility. In addition to object type or context hierarchies, ODABA supports multifarious relations between object instances (master and detail relations, relations between independent object instances and others). This way, behavior of objects in the real world can be represented considerably better than in relational database systems.

ODABA supports event-driven applications concerning the graphical user interface as well as the database level. Thus, application design is tightly related to the experts or customers problem, since it refers to the same names and concepts as being defined by subject matter experts. This enables ODABA to solve highly complex jobs in administrative and knowledge areas.

Platforms

ODABA supports windows platforms (from Windows 95 up to Windows8) as well as UNIX platforms and Mac OS (. ODABA supports 64 and 32 bit technologies.

ODABA also runs well in heterogeneous client/server environments or with Internet servers.

Interfaces

ODABA supports several technical interfaces:

- C++, .Net as application program interface (this allows e.g. using ODABA in C# or VB scripts and applications)
- ODABA Script Interface (OSI) for accessing data via a script language, which is similar to C# or JAVA.
- Multiple storage support for using relational databases for storing ODABA data
- XML for supporting data exchange with complex data structures
- OIF (object interchange format), flat files and ESDF (extended self delimiter fields) for accessing data provided in external file formats
- Document exchange support for importing or exporting data from/to open office or Microsoft office documents.

Tools

ODABA provides a number of database maintenance tools, but also development tools in order to provide terminology model definitions, data model specifications, application design and others.

To support just-in-time documentation, all ODABA tools provide extended documentation facilities, which are the base for generating system and WEB documentation, but also online help systems.

License agreements and development support

License regulations contain the agreements for using ODABA. Practically, one may use ODABA as long as publishing products as Open Source products under GPL, again. Besides the GPL there is a commercial license available, which allows using ODABA for creating products, which do not require a GPL.

ODABA does not require run-time licenses for ODABA applications, except some special advanced features (in this version it is the RDBMS controller for running ODABA on relational database systems). Usually, advanced features become part of the open source agreement after a few years.

One may use the GPL in order to develop other software licensed under GPL. In order to develop commercial software not licensed as GPL product, one may get a commercial development license.

Note, the the GPL allows creating any sort of GPL projects, i.e. projects for own purposes as well as well as commercial products (as long as those are published under GPL, again). In order to develop non-GPL products, a commercial development license is required, which can be ordered at RUN-Software.

One may also join the ODABA Software Development Foundation, which supports the further ODABA development and which gives you a chance to participate in strategic and short term decisions.

2 Installation

For installing ODABA follow the steps described in the installation manual. The Installation Guide describes the main steps for installing ODABA and tools. This document is delivered together with the installation. The installation guide also includes a quick start manual for creating a new project or running the sample project.

Source code installation consists of three commands:

- `configure` - setting installation options
- `make` - compile ODABA and GUI framework
- `make install` - move binaries to run-time environment

Since ODABA 11.1 autotools are supported. Therefore, configuring build and install of ODABA is now similar for all package.

2.1 Installing ODABA under Linux

The following types of installations are provided for Linux:

- Source code (**odaba-nn.n.n.tar.bz2**)
- Documentation (**odaba-doc-nn.n.n.zip**)
- Additional plugins (**odaba-plugins-nn.n.n.tar.bz2**)
- ODABA sample and development databases (**odaba-dat-nn.n.n.zip**)

where **nn.n.n** refers to the current version number (e.g. 12.3.0).

Documentation, which is also provided online, might be installed also on a local folder.

Source package

For Linux, no binaries are delivered. Source packages have to be compiled with the required compiler.

Documentation

Documentation, which is also provided online, might be installed on a local directory. In order to start the HTML documentation for ODABA or ODABA GUI, one has to start

- **.../HTMLDoc/odaba/index.html** or
- **.../HTMLDoc/odabagui/index.html**

Plugin packages

Plugin packages provide several user controls for the ODABA GUI framework. Plugin packages are provided as source packages and have to be compiled for the used compiler version.

Databases

In order to install databases, the database package may be copied from the .zip file.

Beside several sample databases the package contains the ODABA development databases, which contain the complete ODABA and GUI framework code, as well as design and documentation. This provides direct access to ODABA development resources and allow customizing tools for specific requirements.

2.1.1 Compile and install the Linux Package

In order to install the Linux package, **odaba-12.0.0.tar.bz2** (or any other version) may be downloaded from http://sourceforge.net/p/odaba/. The package contains the complete documentation and the sources and procedures to build the software.

After unpacking and copying the files to local directories, one may compile the system. In order to build the basic database libraries and tools, only the **base** sources need to be build. In order to build the GUI framework and ODABA GUI tools, QT 4 has to be installed as development-package.

ODABA can be installed on any Linux (Suse, Fedora, Gentoo, Arch). In order to install ODABA, you have to

- unpack the installation file
- compile the source-package for your platform
- roll a package and use the facilities provided by you distribution to install

After installing ODABA, menu items for starting ODABA Tools are provided in the applications menu.

Please read the INSTALL file provided with the source package as it contains hints about the dependencies your system has to provide.

Unpack installation file

In order to unpack the installation file, one may simply call

```
$ tar xjf odaba-[version].tar.bz2
```

where *version* is the current version number of the ODABA release (e.g. **12.0.0**). The result is the ODABA-source tree in a directory 'odaba-[version]'.

Notes: I requires about 100 MB of free disk space to unpack ODABA.

2.1.2 Install documentation

In order to run the documentation system locally, download **odaba_doc-12.0.0.zip** (or a higher version) from http://sourceforge.net/p/odaba. and unpack the compressed file into a local directory.

```
$ cp usr/share/doc/odaba/HTMLDoc.tar.bz2 /tmp
$ cd /tmp
$ tar xjf HTMLDoc.tar.bz2
```

After installing you will find a set of .pdf documents in the **doc/PDFDoc** folder in the installation folder. The WEB browser documentation can be started from **doc/HTMLDoc/odaba/index.html** of from **doc/HTMLDoc/odabagui/index.html** (GUI framework and tools documentation).

For uninstalling documentation, one may simply remove the documentation directory.

2.2 Installing ODABA under MS Windows

Three types of installations are provided for windows:

- Binaries (**odaba-win-nn.n.n.zip, odaba-win-nn.n.n.msi**)
- Source code (**odaba-nn.n.n.zip**)
- Documentation (**odaba-doc-nn.n.n.zip, odaba-doc-nn.n.n.msi**)
- Additional plugins (**odaba-plugins-nn.n.n.zip**)
- ODABA sample and development databases (**odaba-dat-nn.n.n.zip, odaba-doc-nn.n.n.msi**)

where **nn.n.n** refers to the current version number (e.g. 12.3.0).

Binary package

Binary packages are available as .msi (installation file) and .zip file. When using the .msi file it may become necessary to uninstall ODABA before re-installing. Since ODABA installation does not require the Windows registry, one may also simply copy the content of the .zip file into a folder.

Binaries are provided as MS C++ v 10.

Source package

The source package should be used, when another compiler version than MS VS 10 is required. In this case, the source package has to be recompiled with the required compiler version. Beginning with version 7 all MS C++ compiler versions might be used.

Documentation

Documentation, which is also provided online, might be installed on a local folder. In order to start the HTML documentation for ODABA or ODABA GUI, one has to start

- **.../HTMLDoc/odaba/index.html** or
- **.../HTMLDoc/odabagui/index.html**

Plugin packages

Plugin packages provide several user controls for the ODABA GUI framework. Plugin packages are provided as source packeges and have to be compiled for the used compiler version.

Databases

The database package is also provided as .msi installation file. Nevertheless, no specific installation is required and the databases may be simply copied from the .zip file, as well.

Beside several sample databases the package contains the ODABA development databases, which contain the complete ODABA and GUI framework code, as well as design and documentation. This provides direct access to ODABA development resources and allow customizing tools for specific requirements.

2.2.1 Installing ODABA binaries under MS Windows

ODABA can be installed on a Microsoft operating system (Windows 2000, XP, Vista, etc). Depending on the installation type ODABA requires about 150 MB space on hard disk (without sources) and about 500 MB with source installation.

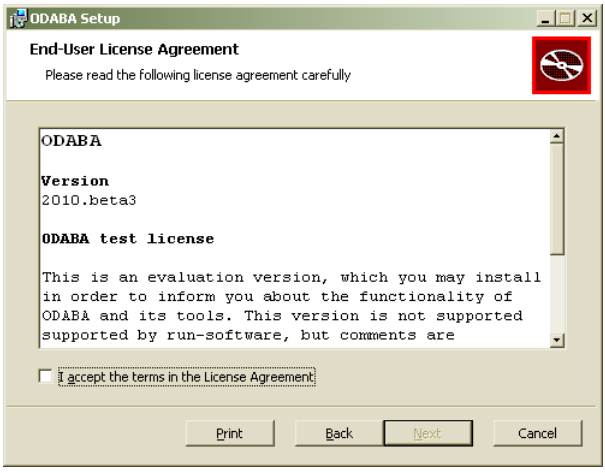
After installing ODABA, a menu item **ODABA** has been installed in the Start menu, which provides several sub items in order to browse the ODABA development databases, documentation and design. This allows you, e.g. to customize the design for your specific requirements.

Notes: Before installing a new release or version of ODABA it is suggested to uninstall the previously installed version.

Running installation

In order to install ODABA, one may run the installation by double clicking the **odaba-win-*version*.msi** installation file, where version is the current release number (e.g. 10.0.0). When ODABA has already been installed, it is necessary to uninstall ODABA before re-installing.

While installing, you are requested to "sign" the license agreement. The license is GPL and allows you using ODABA as long as everything you produce with the help of ODABA is open source, again. In order to provide commercial products, you have to apply for a commercial developer license.



After installing ODABA one may run the 8 steps sample installation or make any kind of project.

2.2.2 Install documentation

In order to run the documentation system locally, one may download **odaba_doc-12.0.0.msi** or **odaba_doc-12.0.0.zip** (or any other version) from http://sourceforge.net/p/odaba. One may either unpack the .zip file into a local folder or install the .msi file.

After installing a set of .pdf documents has been copied to the doc/PDFDoc folder in the installation folder. The WEB browser documentation can be started from **doc\HTMLDoc\odaba\index.html** of from **doc\HTMLDoc\odabagui\index.html** (GUI framework and tools documentation).

In order to uninstall documentation, you just remove the installation folder.

2.2.3 Compile Windows version

In order to install the Windows package by recompiling the system, **odaba-12.0.0.zip** (or a higher version) may be downloaded from http://sourceforge.net/p/odaba. The package contains the complete documentation and the sources and procedures for rebuild the software. Besides, the package contains ODABA development databases for examining ODABA resources (source code, GUI resources).

After unpacking and copying the files to local directories, one may compile the system. In order to build the basic database libraries and tools, only the **base** sources need to be build. In order to build the GUI framework and ODABA GUI tools, QT 4.4 (Qt 4.7 or 4.8 suggested) must have been installed.

ODABA can be installed on several MS Windows platforms (from Windows 2000/XP to Windows 7). In order to install ODABA, the following steps have to be performed

- unpack the installation file
- Configure installation
- compile the source package with the required compiler
- install binaries, i.e. setup the ODABA folder

After installing the package successfully, one may create your own projects or load the ODABA databases in order to view ODABA resource details.

The standard installation does not include .Net support. In order to build the .Net interface see **.Net features** (sub topic).

Unpack installation file

The installation is provided as simple .zip file, which can be unpacked easily by calling e.g.

`unzip.exe odaba-`*`nn.x.x.`*`zip`

`nn.x.x` stands for current version, subversion and release number. A directory with the name `odaba-`*`nn.x.x`* will be created at the current location. After unpacking one may configure and compile the system with the required compiler.

2.2.4 .Net features

In order to use the .Net interface, the .Net library has to be created. An model procedure for installing a .Net library is provided in the next sub topic (**.Net installation**). The .Net library ODABA-net.dll may be referenced in MS Visual Studio .Net projects by referencing the ODABA-net.tlb type library, which has been created with the installation.

Using the interface in VB script programs (e.g. MS Office macros) causes some problems, since the ODABA interface uses many names, which are reserved names for VB Script. A simple workaround is providing a wrapper library as C# or VB interface library, which is described in sub topic **.Net wrapper library**.

In order to use MS Office extensions (document generation), additional resources are required, which are not part of the installation. How to make those resources available is described in sub topic **MS Office document generation**.

2.2.4.1 .Net installation

In order to install the ODABA .Net interface, ODABA-net plug-in has to be installed (????). The plug-in includes the dot net-connector, which is a generic solution for creating .Net interfaces from C++ applications.

Building .Net libraries is not part of the default installation. When one has compiled the Windows version, for using the .Net interface it is necessary to build the .Net libraries. The example below shows a procedure, which just has to be adapted to the current environment.

```
@setlocal
@if "%VS_ENVIRONMENT%" == "defined" @goto Start
@call "%VS100COMNTOOLS%vsvars32.bat"
@set S10_ENVIRONMENT=defined
@set ODABAROOT=...                  --> set the current odaba source root
folder from the installation
@set ODABA_DIR=...odaba             --> Replace with the odaba binary
directory
rem  the dotnet-connector installation is required
(https://github.com/git-e/dotnet-connector)
```

```
@set DNCROOT=...dotnet-connector   --> Replace with dotnet-connector
installation directory
@set ROOT=%ODABAROOT%\opa\opi\odaba

rem Compile native part
:native
@if not exist %ROOT%\lib\dnc @mkdir %ROOT%\lib\dnc
@pushd %ROOT%\lib\dnc
del /F /Q *.obj
cl /Zi /RTCscu /GR /I%DNCROOT\include% /I"%ODABA_DIR%\include"
/Il:\opa /c /EHsc %ROOT%\qlib\*.dnc.cpp
@popd                %DNCROOT\include%

rem 3. Compile managend part
:managed
@if not exist %ROOT%\lib\dnc-cli @mkdir %ROOT%\lib\dnc-cli
@pushd %ROOT%\lib\dnc-cli
del /F /Q *.obj
cl /Zi /GR /AI %DNCROOT%\build /I%DNCROOT%\include /I"%ODABA_DIR
%\include" /I%ODABAROOT%\opa /c /clr /wd4355 /wd4490 %ROOT
%\qlib\*.dnc.cpp
@popd

rem 4. Link the whole thing
:link
@if not exist %ROOT%\exe @mkdir %ROOT%\exe
@pushd %ROOT%\exe
rem : build .res
l:\odet\IncreaseID.exe %ROOT%\tpl\odaba-net_version.h %ROOT%\rc\odaba-
net.h build
copy %ROOT%\rc\odaba-net.h           %ROOT%\temp\version.h
copy l:\bat\version\win32_version.rc %ROOT%\temp\odaba-net.rc
rc %ROOT%\temp\odaba-net.rc
rem : link with signing
::link /KEYFILE:l:\bat\run-dll.key %ROOT%\lib\dnc\*.obj %ROOT%\lib\dnc-
cli\*.obj %ROOT%\temp\odaba-net.res %ODABA_DIR%\lib\opa.lib %DNCROOT
%\build\dotnet-connector.lib /OUT:odaba-net.dll /NODEFAULTLIB:LIBCMT
/DEBUG /DLL
rem : link without signing
link %ROOT%\lib\dnc\*.obj %ROOT%\lib\dnc-cli\*.obj %ROOT%\temp\odaba-
net.res %ODABA_DIR%\lib\opa.lib %DNCROOT%\build\dotnet-connector.lib
/OUT:odaba-net.dll /NODEFAULTLIB:LIBCMT /DEBUG /DLL
copy odaba-net.dll %ODABA_DIR%\.
regasm %ODABA_DIR%\odaba-net.dll /tlb:odaba-net.tlb
@popd
```

2.2.4.2 .Net wrapper library

For using the ODABA .Net interface from within MS VB Script, a wrapper library has to be provided for several reasons. The example below shows the wrapper library used for in MS Word macros for document generation.

The **`ODatabase`** class Provides functions that allow opening an ODABA database by means of an ini-file. Moreover, it provides functions for reading current option

settings (`GetOption()`) and providing extent property handles (`GetProperty()`).

The **`OProperty`** class provides property handle access functions. Since several function names used in ODABA are reserved names in VB, some function names had to be changed (e.g. `next()` became `ReadNext()`).

Finally, the **`OValue`** class provides enhanced value access to values stored in the database.

The example below is part of the .Net project provided for MS Office support.

```
Option Strict Off
Option Explicit On

<System.Runtime.InteropServices.ProgId("ODatabase_NET.ODatabase")> Public
Class ODatabase

    Public ds As odaba.DataSource
    Public db As odaba.Database
    Public dbo As odaba.ObjectSpace
    Public index As Integer
    Public stati As Integer
    Public Function Open(ByRef inifile As String, ByRef section As
String) As Boolean

        Dim location As String
        Dim language As odaba.Option

        Open = True
        location = " (inifile='" & inifile & "', section='" & section &
"') "

        odaba.Application.initialize(inifile, section,
odaba.ApplicationTypes.ConsoleApplication)

        ds = New odaba.DataSource
        ds.open(section)

        dbo = ds.objectSpace
        db = ds.database
        If dbo Is Nothing Or Not dbo.isValid Then
            MsgBox("DataSource" & location & "could not be opened")
            Open = False
        End If

        language = New odaba.Option("DSC_Language")
        If language.toString() = "" Then
            language.assign("English")
        End If
    End Function

    Public Function Close() As Boolean

        'UPGRADE_NOTE: Object odaba may not be destroyed until it is
garbage collected. Click for more: 'ms-
help://MS.VSCC.2003/commoner/redir/redirect.htm?keyword="vbup1029"'
```

```
        If Not IsNothing(db) Then
            db.close()
        End If
        If Not IsNothing(dbo) Then
            dbo.close()
        End If
        If Not IsNothing(ds) Then
            ds.close()
        End If

        Close = True

    End Function
    Public Function GetOption(ByRef opt_name As String) As String
        Dim opt As odaba.Option
        opt = New odaba.Option(opt_name)
        GetOption = opt.toString()
    End Function
    Public Function GetProperty(ByRef ph As OProperty, Optional ByRef
prop_path As String = "") As OProperty

        Dim oprop As OProperty
        Dim cond As Boolean

        oprop = New OProperty
        cond = False
        If prop_path <> "" Then
            If ph Is Nothing Then
                cond = oprop.Open(dbo, prop_path, odaba.AccessModes.Read)
            Else
                cond = oprop.Open(ph, prop_path)
            End If
        End If
        If cond Then
            GetProperty = oprop
        Else
            MsgBox("Could not open property handle for '" & prop_path &
"'")
            GetProperty = Nothing
        End If

    End Function
End Class
<System.Runtime.InteropServices.ProgId("OProperty_NET.OProperty")> Public
Class OProperty
    Public prop As odaba.Property
    Public Function Count() As Integer
        Count = prop.count
    End Function
    Public Function Open(ByRef oprop As OProperty, ByVal path As String)
As Boolean
        prop = New odaba.Property(oprop.prop, path)
        Open = True
    End Function
    Private Function Open(ByRef oprop As odaba.Property, ByVal path As
String) As Boolean
        prop = New odaba.Property(oprop, path)
        Open = True
```

```
    End Function
    Public Function Open(ByRef db As odaba.ObjectSpace, ByVal osi_path As
String, ByVal accmode As odaba.AccessModes) As Boolean
        prop = New odaba.Property(db, osi_path, accmode)
        Open = True
    End Function
    Public Function Read(ByRef key As String) As Boolean
        Dim skey As New odaba.Key(key)
        Read = False
        If prop.tryGet(skey) Then
            Read = True
        End If
    End Function
    Public Function Read(ByVal num As Integer) As Boolean
        Read = False
        If prop.tryGet(num) Then
            Read = True
        End If
    End Function
    Public Function ReadFirst(Optional ByVal read_opt As Boolean = True)
As Boolean
        ReadFirst = prop.first(read_opt)
    End Function
    Public Function ReadLast(Optional ByVal read_opt As Boolean = True)
As Boolean
        ReadLast = prop.last(read_opt)
    End Function
    Public Function ReadNext(Optional ByVal read_opt As Boolean = True)
As Boolean
        ReadNext = prop.next(read_opt)
    End Function
    Public Function ReadPrevious(Optional ByVal read_opt As Boolean =
True) As Boolean
        ReadPrevious = prop.next(read_opt)
    End Function
    Public Function OProperty(ByRef path As String) As OProperty
        Dim oprop As New OProperty

        If oprop.Open(prop, path) Then
            OProperty = oprop
        Else
            OProperty = Nothing
        End If
    End Function
    Public Function OValue(ByRef path As String) As OValue
        Dim oval As New OValue
        If oval.Open(prop, path) Then
            OValue = oval
        Else
            OValue = Nothing
        End If
    End Function
    Public Function Top() As Boolean
        prop.top()
        Top = True
    End Function
    Public Function ODABAProperty() As odaba.Property
        ODABAProperty = prop
```

```
    End Function
End Class
<System.Runtime.InteropServices.ProgId("OValue_NET.OValue")> Public Class
OValue
    Public value As odaba.Value
    Public Function Open(ByRef oprop As OProperty) As Boolean
        value = oprop.prop.value()
        Open = True
    End Function
    Private Function Open(ByRef prop As odaba.Property) As Boolean
        value = prop.value()
        Open = True
    End Function
    Public Function Open(ByRef oprop As OProperty, ByRef path As String)
As Boolean
        value = oprop.prop.value(path)
        Open = True
    End Function
    Public Function Open(ByRef prop As odaba.Property, ByRef path As
String) As Boolean
        value = prop.value(path)
        Open = True
    End Function
    Public Function AsBoolean() As Boolean
        AsBoolean = False
        If AutoSelect() Then
            AsBoolean = value.toBool
        End If
    End Function
    Public Function AsInteger() As Integer
        AsInteger = 0
        If AutoSelect() Then
            AsInteger = value.toInteger
        End If
    End Function
    Public Function AsPlainText() As String
        AsPlainText = ""
        If AutoSelect() Then
            AsPlainText = value.toString.toPlainText
        End If
    End Function
    Public Function AsString() As String
        AsString = ""
        If AutoSelect() Then
            AsString = value.toString
        End If
    End Function
    Public Function AutoSelect() As Boolean
        AutoSelect = False
        value.property.autoSelect()
        If value.property.selected Then
            AutoSelect = True
        End If
    End Function
End Class
```

2.2.4.3 MS Office document generation

A typical way for generating MS Office documents is using MS Word macro features. The **Terminus** application provides document generation actions, which may work, however, only when the required document templates have been installed. Document templates are not part of the ODABA installation and might be modified or rewritten according to specific requirements.

Typically, MS Office document templates refer to an initial document, which provides style definitions and title page for the document to be generated. This might easily be replaced by a more appropriate one. Moreover, styles might be changed, but not the style names, which are referred to from within the macros. Available macros and its resources are described below.

In order to support scripting languages as MS VB Script, ODABA .Net libraries have to be extended by a wrapper library. The .Net project and other resources required for MS document generation are available at following locations:

- MS Word helper functions

http://www.odaba.com/content/downloads/demos/odabaWordHelper.zip

- Document template

http://www.odaba.com/content/downlads/demos/DocumentTemplates.zip)

Those are just demos for showing, how to get out some documents from an ODABA database. On the other hand, these demos are used by RUN for generating documentation from Terminus specifications. Usually, document templates are installed in the template directory (**/usr/share/odaba/template** under Linux and ...**odaba/template** under MS Windows).

Document templates are provided as **.dot** files for reference documentation (**ReferenceDocu.dot**), Terminology Model documentation (**TerminologyModel.dot**) and hierarchical topic documentation (**TopicsDocu.dot**). Document style definitions are provided for these templates in .doc files with appropriate names.

Notes: MS Office document generation by means of VB Script macros is one possible way, but it is rather slow and difficult to maintain. A better way is using Open Document templates, which generate documents that are accessible in MS Office as well as in LibreOffice.

MS Word helper functions

The MS Developer Studio 2010 solution provides an ODABA wrapper supporting ODABA database access and a few MS Word function for opening and closing word documents. Before compiling the solution, references for odaba-net.dll and dotnet-connector.dll have to be updated.

The wrapper library works with all MS Office versions from office 1997-2003 upwards. It has not been tested with older versions.

When opening a document (**ODocument**::Open()), an ini-file is required, that contains document and template name. The ini-file is, usually, generated when calling MS Word macros from within **Terminus**. Otherwise, an ini-file has to be provided, which contains a path the document to be created (option name passed in *docname*) and a path for a template for initializing the document (option name passed in *templatename*).

Two more functions (**ODocument**::Find() and **ODocument**::ReplaceText()) are available for convenience.

The odabaDBInterface file provides the ODABA database access function wrapper for accessing the database (**ODatabase**), for property handle support (**OProperty**) and for value access (**OValue**).

2.2.4.3.1 Call creating MS Office document

In order to call creating an MS Office document, MS Word has to be invoked. Since the technology for creating MS Office documents is based on MS Word macros (VB Script), the template document has to be called and executed. Since this is a different process, an ini-file has to be created and passed to the MS Word template (macro).

All information requested is passed via an ini-file, which is usually hard-coded in the document template macro. The document template examples provided in the .tpl directory of the installation folder shows how to open a database by means of an ini-file.

When calling an MS Office template (.dot), the location for the template file has to be passed to the function call (GetTemplatePath() is just a symbolic function call in the example, which returns the complete path for the document template file). Other option file variables as location for output file or root object instance for document have to be set in additional option variables as requested by the document template.

```
// generate document in separate process
bool ...fragment(Property &ph) {
// root object instance is selected in ph

// create ini-file for CreateDocument
  fstream ini_file;
  ini_file.open ("test.ini", fstream::out | fstream::app);

  ini_file << "[SYSTEM]" << std::endl;
  ini_file << "DICTIONARY=" << Option("SYSDB").toString().data() << endl;

  ini_file << "[DOCU]" << std::endl;
  ini_file << "DICTIONARY=" << Option("SYSDB").toString().data() << endl;
  ini_file << "RESOURCES=" << Option("RESDB").toString().data() << endl;
  ini_file << "DATABASE=" << Option("DATDB").toString().data() << endl;
  ini_file << "ONLINE_VERSION=YES" << endl;
  ini_file << "ACCESS_MODE=Write" << endl;
```

```
  ini_file << "NET=YES" << endl;
  ini_file << "ODABA_ROOT=" << Option("ODABA_ROOT").toString().data() <<
endl;
  ini_file << "CTXI_DLL=" << Option("CTXI_DLL").toString().data()<< endl;
  ini_file << "TRACE=" << Option("TRACE").toString().data() << endl;
  ini_file << "DSC_Language=" << Option("DSC_Language").toString().data()
endl;
// create option variables for template options
  ini_file.close();

  odaba::String   path(Option("ODABA_ROOT"));
  path += "/CreateDocument.exe";
// depending on template reqirements additional optione might be set
  ph.instanceContext().executeShell("open",GetTemplatePath());

  return true;
}
```

Notes: In the example above, the the document template has to "know", where the ini-file has been stored.

2.3 Version upgrade

When the ODABA schema version changes, which may happen with a new major version release, resource databases and databases referring to ODABA system data types (typically **__OBJECT** or **DSC_Topc**), have to be upgraded. The following paragraphs describe the upgrade procedure required for the latest version.

In order to upgrade a resource database (dictionary), the ode.sys database containing extended system resource definitions is required with old and new version. The installation contains an appropriate old version database (ode.**sys_V21**) preceded by the old version number. The new system database is the **ode.sys** shipped with the installation.

Before starting the upgrade procedure (Upgrade.cmd as described below), it is suggested to create a copy of the old database (here we use the extension **_old**). When copying the database terminates successfully, one may replace your dictionary with the upgraded resource database. From the example below, one may simply create your upgrade procedure by replacing %DB% and storing the Upgrade.cmd and Upgrade.ini file.

In order to upgrade an application database, which is rarely necessary, nearly the same procedure might be used. Just the DICTIONARY has to be replaced by old and new resource database, as shown in the comment lines in the ini-file. In this case, %DB% refers to the application database to be upgraded while %RDB% refers to the resource database, which already had been upgraded.

Under Linux a script is provided that evaluates the current database version and upgrades it to a temporary directory:

```
(/usr/local/share/odaba/upgrade /path/to/data.base)
```

```
// Upgrade.cmd, current directory is odaba
// DB stands for the database location
copy|cp %DB% %DB%_old
CopyResDB Upgrade.ini OLDDB NEWDB
PackDB %DB%_new -p:percent

// when completed successfully, replace DB
copy|cp %DB%_new %DB%

// Upgrade.ini (resource database) - required ini-file for copy
[SYSTEM]
DICTIONARY=ode.sys
DISABLE_CONTEXT=YES

[CopyResDB]
PLATFORM_INDEPENDENT=YES // required for non-windows databases

[OLDDB]
DICTIONARY=ode.sys_V21
DATABASE=%DB%_old
SYSTEM_VERSION=21
NET=NO
ACCESS_MODE=Write
DISABLE_CONTEXT=YES

[NEWDB]
DICTIONARY=ode.sys
DATABASE=%DB%_new
SYSTEM_VERSION=22
NET=NO
ACCESS_MODE=Write
DISABLE_CONTEXT=YES
PLATFORM_INDEPENDENT=YES
```

3 8 Steps to run an ODABA GUI application

This is a short instruction guide for starters in order to create a database and build a simple GUI application. The Manual refers to essential steps without being a detailed documentation of different ODABA features. The manual does not follow the development guide lines, which explicitly suggest beginning with the terminology model, but it demonstrates the essential technical steps for developing an application.

The example starts with installing the software, defining a database schema, creating and evaluating test data, building a GUI application and creating an application context library.

The following steps are to be executed:

- (0) - Install the software (binary or sources)
- (1) - Create **Sample** project
- (2) - Load schema to database
- [(3) - Update schema]
- (4) - Create test data
- (5) - Evaluate test data
- (6) - Design a GUI application
- (7) - Create application context library
- (8) - Run application

In order not to run into problems when trying the sample, one should name the sample project **Sample**. Only in this case, scripts are copied to the proper location (**Sample/osi** directory). When using installation defaults, one may simply perform the steps below. More details are described in the following topics.

```
# default installation
(0) $(HOME)/odaba-10.0.0$ ./configure
(0) $(HOME)/odaba-10.0.0$ make
(0) $(HOME)/odaba-10.0.0$ sudo make install

# linux commands for performind 8 steps to an ODABAS GUI application
(1) /usr/local$ LD_LIBRARY_PATH=lib ./lib/odaba/tools/CreateProject
(2) $(HOME)/odaba/Sample$ ./ODL.sh
(3) $(HOME)/odaba/Sample$ ./ClassEditor.sh
(4) $(HOME)/odaba/Sample$ ./OSI.sh
(5) $(HOME)/odaba/Sample$ ./OShell.sh
                          ODABA>call 'osi/Sample.osh'
                          ...
                          Sample_dat/Company>quit
(6) $(HOME)/odaba/Sample$ ./Designer.sh
(7) $(HOME)/odaba/Sample$ ./ClassEditor.sh
(8) $(HOME)/odaba/Sample$ ./Main.sh
```

3.1 Create new project

Creating the sample project differs slightly under Linux and Windows. In any case, the project should be called **Sample** in order to be compatible with the subsequent steps. The way to install the sample under Linux and Windows is described in the corresponding topic below.

In order to create a new project, **CreateProject** may be called, which allocates project resources in a local directory. Project resources created are configuration (ini) files and procedures for running different ODABA tools. Following procedures will be generated to the project directory:

- **OShell.sh** - shell for examining databases
- **OSI.sh** - running ODABA script file
- **ODL.sh** - running database definition script (ODL file)
- **ClassEditor.sh** - starting ODE ClassEditor tool
- **Designer.sh** - starting ODE GUI designer
- **Terminus.sh** - starting ODE terminology builder

Under MS Windows, instead of **.sh** extension **.cmd** is used. In addition, a configuration file for running the tools mentioned above has been generated in the project's root directory.

- **ode.ini** - configuration file for ODE and command line tools

The configuration file contains database locations and application setting in appropriate sections. In the example below, the configuration file for a LINUX project is shown. The configuration file for Windows differs in file locations, only. When creating a project with the name "Sample", some script files are copied to the project sub directory **Sample/osi**:

- **Sample.odl** - sample database schema definition (ODL file)
- **Sample.osi** - ODABA script file for generating test data
- **Sample.osh** - **OShelll** script for evaluating generated data

In order to build the sample database, one may call **ODL.sh** for defining the Sample database schema, **OSI.sh** for creating Sample test data in a test database Sample.dat and **OShell.sh** for evaluating test data. Running the complete program up to test data generation takes not longer than 3 minutes.

Details for each step are explained in the subsequent topics.

```
; generated confiruration file - linux
[SYSTEM]
DICTIONARY=/usr/local/share/odaba/ode.sys

; GUI framework (ODE) section - ODE tools
[code]
ODABA_ROOT=/usr/local/
SYSDB=/usr/local/share/odaba/ode.sys
RESDB=/usr/local/share/odaba/ode.dev
DATDB=/home/testuser/odaba/Sample/Sample.dev
TRACE=/home/testuser/odaba/Sample/
PLATFORM_INDEPENDENT=YES

NET=YES
SYSAPPL=YES
ONLINE_VERSION=YES

PROJECT_DLL=Designer
CTXI_DLL=AdkCtxi

; OShell section
[OShell]
ODABA_ROOT=/usr/local/
DSC_Language=English
DefaultEncoding=ASCII
TRACE=/home/testuser/odaba/Sample/

; data source section for resource database
[Sample_dev]
DICTIONARY=/usr/local/share/odaba/ode.sys
RESOURCES=/usr/local/share/odaba/ode.dev
DATABASE=/home/testuser/odaba/Sample/Sample.dev
PLATFORM_INDEPENDENT=YES
NET=YES
ACCESS_MODE=Write
ONLINE_VERSION=YES

; data source section for application database
[Sample_dat]
DICTIONARY=/home/testuser/odaba/Sample/Sample.dev
DATABASE=/home/testuser/odaba/Sample/Sample.dat
PLATFORM_INDEPENDENT=YES
NET=YES
ACCESS_MODE=Write
ONLINE_VERSION=YES
```

3.1.1 Create the Sample project (Linux)

For the following topic, it is supposed that ODABA had been installed in the default location (**/usr/local**). When it had been installed in another place, one simply has to replace **/usr/local** references by the location that had been passed in **--prefix** when configuring the package. In order to load dynamic libraries properly, the library path has to be defined.

```
/usr/local$ LD_LIBRARY_PATH=lib ./lib/odaba/tools/CreateProject
```

CreateProject allocates project resources in a local project directory below **/usr/home/***user_name***/odaba/***project_name*. The shell files created are configured for calling services for the local project. In order to share project resources, it may be useful to place resources on proper locations and update shell and configuration files by changing locations for referenced resources and server type, when this becomes necessary..

```
$ LD_LIBRARY_PATH=/usr/local/lib /usr/local/lib/odaba/tools/CreateProject

2012-03-05 17:50:48 - Running /usr/local/lib/odaba/tools/CreateProject
with:

Setting up ODABA project environment ...
Enter ODABA path  [/usr/local] :
Enter project name  [] : Sample
Enter project path  [/home/testuser/odaba/Sample] :
Use ClassEditor Y(es)/N(o)  [YES] :
Use Designer Y(es)/N(o)  [YES] :
Use Terminus Y(es)/N(o)  [YES] :

Current project settings ...
ODABA path          : /usr/local
Project name        : Sample
Project path        : /usr/home/testuser/odaba/Sample
Use ClassEditor     : YES
Use Designer        : YES
Use Terminus        : YES
  ... enter (a)ccept/(c)ancel/(r)epeat [a] :

/usr/home/testuser/odaba/Sample/ode.ini created
/usr/home/testuser/odaba/Sample/OShell.sh created
/usr/home/testuser/odaba/Sample/OSI.sh created
/usr/home/testuser/odaba/Sample/ODL.sh created
/usr/home/testuser/odaba/Sample/ClassEditor.sh created
/usr/home/testuser/odaba/Sample/Designer.sh created
/usr/home/testuser/odaba/Sample/Terminus.sh created
```

Notes: Never forget to set the LD_LIBRARY_PATH to the **lib** directory in odaba root (/usr/local/lib) before calling **CreateProject**.

3.1.2 Create the Sample project (Windows)

The `CreateProject()` function can be called from the ODABA root directory (installation path), which may be "`C:\Program Files\odaba`" or something similar when relying on the default or the location, which had been defined in **--prefix** when configuring the system.

```
...>C:\Program Files\odaba\CreateProject
```

`CreateProject()` allocates project resources in a local project directory below **C:\Documents and Settings***user_name***\odaba***project_name*. The command files created are configured for calling services for the local project. In order to share project resources, it may be useful to place resources on proper locations and update shell and configuration files by changing locations for referenced resources and server type, when this becomes necessary..

```
12-11-07 - Running CreateProject with:

Setting up ODABA project environment ...
Enter ODABA path  [L:\odet\] :
Enter project name  [] : Sample
Enter project path  [C:\Documents and Settings\my_user\odaba\Sample] :
E:\Sample
Use ClassEditor Y(es)/N(o)  [YES] :
Use Designer Y(es)/N(o)  [YES] :
Use Terminus Y(es)/N(o)  [YES] :

Current project settings ...
ODABA path          : L:\odet\
Project name        : Sample
Project path        : E:\Sample
Use ClassEditor     : YES
Use Designer        : YES
Use Terminus        : YES
  ... enter (a)ccept/(c)ancel/(r)epeat [a] :

E:\Sample\ode.ini created
E:\Sample\OShell.cmd created
E:\Sample\OSI.cmd created
E:\Sample\ODL.cmd created
E:\Sample\ClassEditor.cmd created
E:\Sample\Designer.cmd created
E:\Sample\Terminus.cmd created

Start ODE application ClassEditor Y(es)/N(o) [YES] : n
```

Notes: When the first prompt does not display an absolute path to the ODABA system folder location (second line in the example), you have to enter the absolute path to the ODABA folder explicitly. Otherwise, one may just press enter.

3.2 Load database schema

In order to simplify the schema definition, we provide the example as listed below. The example schema describes companies (**Company**), which have got employees (**Employee**), which are persons (**Person**) and cars (**Car**). The cars of a company might be assigned to be used by one or more employees of the company.

Before loading the schema one may change the dictionary folder, which refers to the position where you did create your Sample development database (dictionary). When the definition file Sample.odl had not been copied to the OSI folder in the project folder, the sample might be copied from this page or from the documentation folder. More explanations for the schema definition you will find in the topics below (Sample schema notes).

In order to load the schema, simply call **ODL.sh** (LINUX) or **ODL.cmd** (Windows). In order to update the schema, one may extend the ODL script and reload it. A more comfortable way, however, is using ODE tools (**ClassEditor**) as being described in **Quick Starter Guide/9 Steps to develop a GUI application/View or update schema definitions**.

When using the generated defaults, the dictionary will be created in the project's root directory (e.g. ~/Sample/Sample.dev). In order to place it somewhere else, the DICTIONARY path in the ODL script (osi/Sample.odl) has to be changed.

```
DICTIONARY = 'Sample.dev'; // sample resources

UPDATE SCHEMA Sample {

// Car class definition
  CLASS Car PERSISTENT (  KEY IDENT_KEY pk(cid); ) {
    ATTRIBUTE {
            CHAR(10)   cid;
            STRING(40) type;
            INT(2)     number_of_seats = 4;
    };

    RELATIONSHIP {
            Company       SECONDARY  company  INVERSE cars;
            SET<Employee> SECONDARY  users    ORDERED_BY (pk UNIQUE)
INVERSE used_cars;
    };
  };

// Person class definition
  CLASS Person PERSISTENT
  ( KEY { IDENT_KEY pk (pid); sk (name); };
    EXTENT MULTIPLE_KEY owner Persons ORDERED_BY (pk UNIQUE NOT_EMPTY,
sk); )
  {
    ENUM Sex {
                male = 1,
                female = 2,
                undefined = 0
```

```
    };

    STRUCT Address PERSISTENT {
                STRING(6)  zip;
                STRING(40) city;
                STRING(80) street;
                STRING(6)  number;
    };
    ATTRIBUTE {
      NOT_EMPTY CHAR(16)   pid;
                STRING(40) name;
                STRING(40) first_name[3];
                DATE       birth_date;
                Sex        sex = male;
                bool       married = false;
                INT(10,2)  income;
      TRANSIENT INT(3)     age SOURCE( (Date() - birth_date)/365.25 );
    };
    REFERENCE   Address  location;
    REFERENCE   STRING   notes[4000];

    RELATIONSHIP {
                SET<Person>         children      BASED_ON Persons
                                                  INVERSE parents;
                Person   SECONDARY  parents[2]    BASED_ON Persons
                                                  INVERSE children;
                Employee SECONDARY  employee      INVERSE person;
    };
  };

// Employee class definition
  CLASS Employee PERSISTENT : Person person BASED_ON Person::Persons
INVERSE employee
  ( KEY  IDENT_KEY pk(pid); )
  {
//    ATTRIBUTE   INT(10,2)  income;

    RELATIONSHIP {
                Company SECONDARY   company       BASED_ON Company
                                                  ORDERED_BY (pk UNIQUE)
                                                  INVERSE employees;
                Car     NO_CREATE   used_cars[2]  BASED_ON .company.cars
                                                  ORDERED_BY (pk UNIQUE)
                                                  INVERSE users;
    };
  };

// Company class definition
  CLASS Company PERSISTENT ( KEY IDENT_KEY pk(name); ) {
    ATTRIBUTE   NOT_EMPTY STRING(200)  name;

    RELATIONSHIP {
                SET<Employee>       employees BASED_ON Employee
                                              ORDERED_BY (pk UNIQUE)
                                              INVERSE company;
                SET<Car>      OWNER cars      ORDERED_BY (pk UNIQUE)
                                              INVERSE company;
    };
```

```
  };

// Global extent definition
  EXTENT Company UPDATE MULTIPLE_KEY OWNER Company
               ORDERED_BY ( pk UNIQUE NOT_EMPTY );
  EXTENT Employee UPDATE MULTIPLE_KEY OWNER Employee
               ORDERED_BY ( pk UNIQUE NOT_EMPTY );
};
```

Notes: Schema loading should work without problems when the Sample project had been created. When creating another project, one may have to update the ODL.sh or ODL.cmd file in order to set proper location for the schema file. Moreover, one has to make sure, that the dictionary path in the .odl file is correct.

3.2.1 Load Schema (Linux)

Typically, the dictionary is stored in the project root (e.g. ~/`Sample`). In the example, the script file has been stored in a sub folder OSI (~/`Sample/osi/Sample.odl`). When generating another project than Sample, an appropriate ODL script file has to be provided.

```
~/odaba/Sample$ ./ODL.sh
```

This procedure is properly initialized when generating the Sample project but has to be adapted to specific requirements when generating other projects. The options -R and -C is suggested in order to check the schema after being imported and mark data types as checked and ready.

The load protocol is written to console as shown in the example below.

```
2010-09-23 21:02:33 - Running /usr/local/lib/odaba/tools/ODL with:
  ini-file: Sample.odl
  script file:
  Options used: -C: -R:
New database created at '/home/my_user/odaba/Sample/Sample.dev'
... checking all persistent structures
  Address ... checking structure
  Address...checked successfully
  Car ... checking structure
  Car...checked successfully
  Company ... checking structure
  Company...checked successfully
  Employee ... checking structure
  Employee...checked successfully
  Person ... checking structure
  Person...checked successfully
... checking all extent definitions
  Company ... checking extent
  Company... checked successfully
  Employee ... checking extent
  Employee... checked successfully
  Persons ... checking extent
```

```
  Persons... checked successfully
... setting version number for all structures
```

Notes: When there is no odaba.ini stored in the ODABA system folder or when it contains an invalid reference to the ode.sys database, instead of the message

```
New database created at 'Sample.dev'
```

the following message will appear:

```
Undescribed Error : 98 in LDBHandle::Open
(Sample/Sample.dev,InputArea,ik_name,,,)
```

which can be ignored. When the input file contains invalid characters or one has edited it, then sometimes errors like

```
symbol 'sc_element' accepts empty expressions
SOS Error :
Error at line 2, column 15
No match for 'schema_dcl' at: ... Sample {
in: sc_definition
in: sc_element
in: sc_elements
in: ODL
```

occur. Please make sure that the input script is valid.

3.2.2 Load Schema (Windows)

Typically, the dictionary is stored in the project root (e.g. `E:\Sample`). In the example, the script file has been stored in a sub folder OSI (E:\Sample\osi\Sample.osi).

```
...>E:\Sample\ODL.cmd
```

This procedure is properly initialized when generating the Sample project but has to be adapted to specific requirements when generating other projects. The options -R and -C is suggested in order to check the schema after being imported and mark data types as checked and ready.

The load protocol is written to console as shown in the example below.

```
2010-09-23 21:02:33 - Running C:\Programs\odaba\ODL.exe with:
  ini-file: E:\Sample\osi\Sample.odl
  script file:
  Options used: -C: -R:
New database created at 'e:/Sample/Sample.dev'
... checking all persistent structures
  Address ... checking structure
  Address...checked successfully
  Car ... checking structure
  Car...checked successfully
  Company ... checking structure
  Company...checked successfully
  Employee ... checking structure
  Employee...checked successfully
  Person ... checking structure
  Person...checked successfully
... checking all extent definitions
  Company ... checking extent
  Company... checked successfully
  Employee ... checking extent
  Employee... checked successfully
  Persons ... checking extent
  Persons... checked successfully
... setting version number for all structures
```

Notes: When there is no odaba.ini stored in the ODABA system folder or when it contains an invalid reference to the ode.sys database, instead of the message

```
New database created at 'e:/Sample/Sample.dev'
```

the following two messages will appear:

```
Undescribed        Error        :        98        in        LDBHandle::Open
(e:/Sample/Sample.dev,InputArea,ik_name,,,)
```

which can be ignored.

When the input file contains invalid characters after being edited, errors like

```
symbol 'sc_element' accepts empty expressions
SOS Error :
Error at line 2, column 15
No match for 'schema_dcl' at: ... Sample {
in: sc_definition
in: sc_element
in: sc_elements
in: ODL
```

may occur. Please make sure that the input script is valid.

3.2.3 ODL Script comments

The sample schema demonstrates different ways of defining schema elements. In order to define persistent type information, complex data types have to defined as `CLASS` rather than as `STRUCT`. In order to distinguish better between keywords and names, we used capital letters for keywords in the scripts. Keywords can be written in lower and upper case letters. More details, you will find in **OSI - ODABA Script Interface**.

The example defines three global and one local complex data type. `Person`, `Car` and `Company` are global types. The sequence of definition is not important since the semantic check is performed after loading the schema updates for the complete schema stored in the dictionary.

Dictionary location

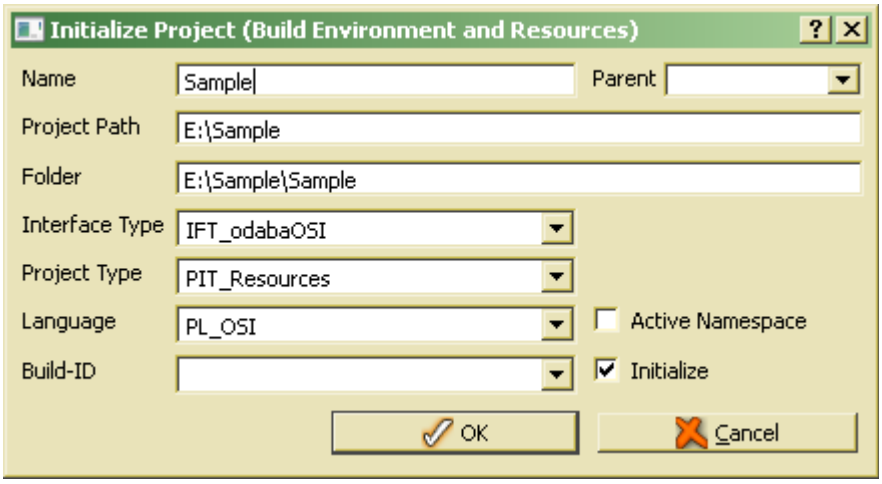
The dictionary location might be passed as parameter when calling ODL or within the script. Usually, it is more flexible passing the dictionary location as parameter when calling ODL. For the example, we used the internal specification for the dictionary as shown below.

In order to get proper error messages, it is suggested to call ODL with an ini-file, that refers to the system dictionary in the system section. Then, one may refer to the dictionary in the script of by defining a `DATASOURCE` in the script, which refers to a section in the ini-file.

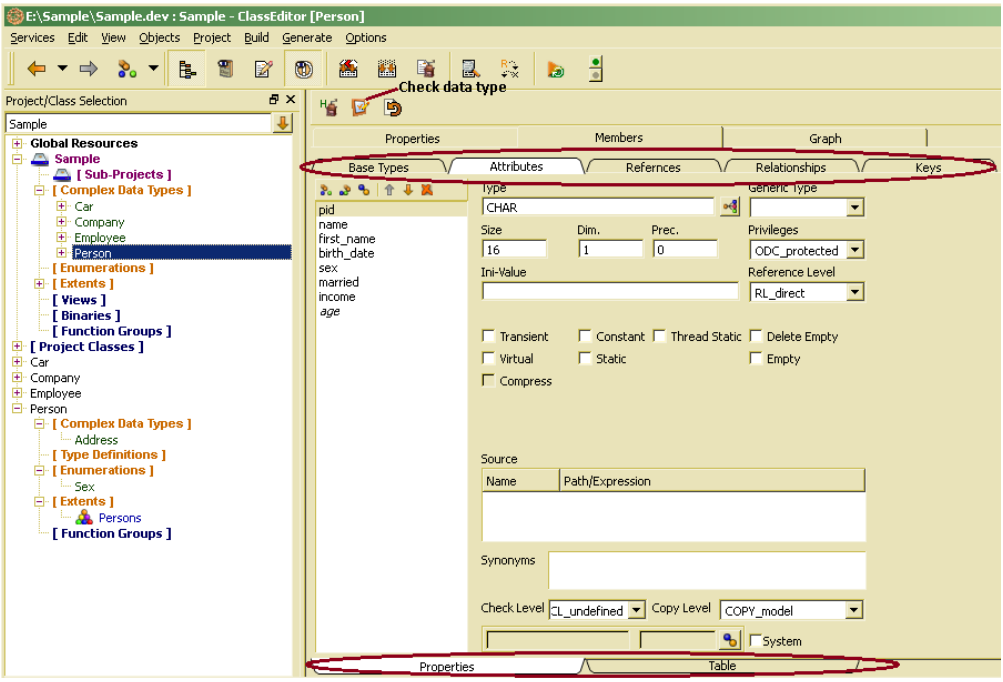
```
DICTIONARY = 'Sample/Sample.dev'; // sample resources
```

3.3 View or update schema definitions

After loading the schema successfully, one may start the ClassEditor in order to view the result or update the schema, but one may also skip this step and continue immediately with creating test data. After starting the ClassEditor the first time, you are prompted to initialize the resource project.



For first test it is suggested to use OSI interface (**IFT_odabaOSI**), which is the default setting, when the project has been initialized as OSI project before. We suggest using OSI interface in order to build fast prototypes. Since OSI provides a script interface, compiling and linking is not necessary. After clicking **OK**, ClassEditor resources will be initialized which may take up to 1 minute. After a while, the ClassEditor opens and displays project classes. The default view shows the class view, i.e. the three persistent sample classes. In order to view structure (complex data type) properties, **Sample/Structures** has to be expanded in the tree.



Structure details are displayed in the right side property window. After selecting a structure in the tree, general structure properties are displayed in the property window on the right side. In order to view structure properties, you click the **Members** tab on top. Structure properties are grouped by type which can be selected by tabs marked on top in the picture above. On bottom, there one might change between detail and list view. The list view also provides a complete member list with all structure properties.

The class view (tree bottom), allows viewing/updating class extensions of the structure definition as extents, local structures, enumerations etc.

After updating structure definitions, the checked and ready state for the definition will be reset. In order to check the structure definition one may click the check button in the structure window tool bar. In order to prepare the project for production, you have to call **Database/DB Version/Production Phase** from the application menu.

3.3.1 Initializing the ClassEditor (Linux)

ODE tools like ClassEditor require a configuration file as **ode.ini**, which has been generated to the project's root directory (~/`Sample/ode.ini`). This is referenced in the **ClassEditor.sh** command in the project root directory:

```
~/odaba/Sample$ ./ClassEditor.sh
```

3.3.2 Initializing the ClassEditor (Windows)

ODE tools like ClassEditor require a configuration file as **ode.ini**, which has been generated to the project's root directory (`E:\Sample\ode.ini`). This is referenced in the **ClassEditor.sh** command in the project root directory:

```
...>E:\Sample\ClassEditor.cmd
```

3.4 Create test data

For creating test data for the sample database, one may use the OSI script as shown below. Comment on the example you will find in the subtopic **OSI Script comments**. After creating the Sample project, one may simply call **OSI.sh** (LINUX) or **OSI.cmd** (Windows), which have been generated to the project directory. When generating other projects, the generated scripts have to be revised in order to check file location for the called **.osi** script file.

Dictionary and database are expected in the project root directory. Otherwise, `DICTIONARY` and `DATABASE` have to be updated in the script file.

When calling the script, a new database will be created at the location defined in `DATABASE` in the script. Before rerunning the script, the database should be deleted, since the script is not prepared for being called repeatedly.

In order to change the amount of data to be created, one may modify the script slightly (see also comments). The number of persons to be created can be passed as parameter in the `CreatePersons(`*`person_count`*`)` function call. The default is 1000.

The dictionary must exist at the location specified in the script file for `DICTIONARY`. The script contains several `Message()` calls that write protocol information to the console. The protocols listed in the Linux and Windows specification are the output from the tests running on our test machine.

```
DICTIONARY='Sample.dev';
DATABASE='Sample.dat';

bool main ( ) {
VARIABLES
    global int    comp_count   = 5;
PROCESS
    CreateCompanies();
    CreatePersons();
    InitCollections();
    AssignChildren();
}

void CreateCompanies ( ) {
VARIABLES
    int           comp_number  = 0;
    int           count        = 0;
    int           car_counts[5];
    string        key;
    SET<Company> &companies   = Company;
    SET<Car>     &cars        = companies.cars;
PROCESS
// create 5 companies
Message("Starting company transaction: " + (string)Time() );
companies.objectSpace.beginTransaction();
    SystemClass::RandomNumbers(car_counts,20); // initialize number of
cars per company between 0 and
```

```
    companies.insert("My company");
    companies.insert("Your company");
    companies.insert("NO company");
    companies.insert("Any company");
    companies.insert("Best company");
    comp_count = companies.count;
    while ( comp_number < comp_count ) {
        companies.get(comp_number);
        count = 0;
        while ( count < car_counts[comp_number] ) {
            key = 'C' + (string)comp_number + '-' + (string)(++count);
            cars.insert(key);
        }
        ++comp_number;                                  // only prefix
operators are supported
    }

Message("Storing company transaction: " + (string)Time() );
companies.objectSpace.commitTransaction;
Message("Stopping company transaction: " + (string)Time() );
}

void CreatePersons ( int person_count = 1000 ) {
VARIABLES
    int           count = 0;
    SET<Person>  &persons     = Person::Persons;
PROCESS
Message("Starting person transaction: " + (string)Time() );
persons.objectSpace.beginTransaction();

    while ( count < person_count )
        persons.insert('P' + (string)(++count)).SetAttributes();

Message("Storing person transaction: " + (string)Time() );
persons.objectSpace.commitTransaction;
Message("Stopping persons transaction: " + (string)Time() );
}

void Person::SetAttributes ( ) {
VARIABLES
    int         count = 0;
    int         number;
    int         comp_number = 0;
    int         first_name_letters[10];
    int         last_name_letters[5];
    int         years;
    int         months;
    int         days;
    string      upper  = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    string      lower = "abcdefghijklmnopqrstuvwxyz";
PROCESS
// initialize gender
    SystemClass::RandomNumbers(number,2);
    sex = number + 1;    // 1: male, 2:female

// initialize birth date
    SystemClass::RandomNumbers(years,100);
    SystemClass::RandomNumbers(months,12);
```

```
    SystemClass::RandomNumbers(days,28);
    birth_date = (string)(years + 1910) + '-' + (string)(months + 1) +
'-' + (string)(days + 1);

// initialize family state
    SystemClass::RandomNumbers(number,2);
    if ( age >= 18 )                // initial value is false
        married = number;

// init first first name
    count = 0;
    SystemClass::RandomNumbers(first_name_letters,26);
    first_name[0] += upper.subString(first_name_letters[0],1);
    while ( ++count < 10 )          // number of random letters in
first_name_letters
        first_name[0] += lower.subString(first_name_letters[count],1);

// init last name
    count = 0;
    SystemClass::RandomNumbers(last_name_letters,26);
    name += upper.subString(last_name_letters[0],1);
//    while ( ++count < 5 )          // number of random letters in
last_name_letters
    while ( ++count < last_name_letters.propertyDefinition.collectionSize
)  // number of random letters in last_name_letters
        name += lower.subString(last_name_letters[count],1);
    save();

// initialize employee
    if ( age >= 16 && age < 65 ) {
        RandomNumbers(comp_number,comp_count);
        RandomNumbers(number,9500);
        Company.get(comp_number).employees.insert(pid).{ income = number
+ 500; };
    }
}

void InitCollections ( ) {
VARIABLES
    global set<Person>   &f20 &= Person::Persons;
    global set<Person>   &f40 &= Person::Persons;
    global set<Person>   &f60 &= Person::Persons;
    global set<Person>   &f80 &= Person::Persons;
    global set<Person>   &f100 &= Person::Persons;
    global set<Person>   &m20 &= Person::Persons;
    global set<Person>   &m40 &= Person::Persons;
    global set<Person>   &m60 &= Person::Persons;
    global set<Person>   &m80 &= Person::Persons;
    global set<Person>   &m100 &= Person::Persons;
    global set<Person>   &c10 &= Person::Persons;
    global set<Person>   &c30 &= Person::Persons;
    global set<Person>   &c50 &= Person::Persons;
    global set<Person>   &c70 &= Person::Persons;
PROCESS
// set filter for global male and feemale collections by age
    f20.filter('age <= 20 && sex == "female"');
    f40.filter('age > 20 && age <= 40 && sex == "female"');
    f60.filter('age > 40 && age <= 60 && sex == "female"');
```

```
    f80.filter('age > 60 && age <= 80 && sex == "female"');
    f100.filter('age > 80 && age <= 100 && sex == "female"');
    m20.filter('age > 20 && sex == "male"');
  m40.filter('age > 20 && age <= 40 && sex == "male"');
    m60.filter('age > 40 && age <= 60 && sex == "male"');
    m80.filter('age > 60 && age <= 80 && sex == "male"');
    m100.filter('age > 80 && age <= 100 && sex == "male"');
    c10.filter('age <= 10');
    c10.first(); // locate first
    c30.filter('age > 10 && age <= 30');
    c30.first(); // locate first
    c50.filter('age > 30 && age <= 50');
    c50.first(); // locate first
    c70.filter('age > 50 && age <= 70');
    c70.first(); // locate first
}

bool AssignChildren ( ) {
VARIABLES
    set<Person>          &father;
    int                   age_group = 100;
PROCESS
Message("Create child/parent relations: " + (string)Time() );

    while ( age_group > 20 ) {
        switch ( age_group ) {
            case 100 : father &= f100;
                       break;
            case  80 : father &= f80;
                       break;
            case  60 : father &= f60;
                       break;
            case  40 : father &= f40;
                       break;
        }
        if ( father.next() ) {
            father.AssignChild(age_group);
        } else
            age_group -= 20;
    }
Message("Created: " + (string)Time() );
}

bool Person::AssignChild ( int age_group ) {
VARIABLES
    set<Person>               &mother;
    set<Person>               &child;
    int                        child_count;
    int                        cage_group = 100;
PROCESS

    SystemClass::RandomNumbers(child_count,7);
    if ( !child_count )
      leave;

    while ( age_group > 20 ) {
        switch ( age_group ) {
            case 100 : mother &= m100;
```

```
                        break;
            case  80 : mother &= m80;
                        break;
            case  60 : mother &= m60;
                        break;
            case  40 : mother &= m40;
                        break;
        }
        if ( !mother.next ) {
            age_group -= 20;
        } else
            break;
    }
    while ( child_count > 0 && age_group >= 40) {
        --child_count;
        switch ( age_group ) {
            case  100: child &= c70;
                        break;
            case  80 : child &= c50;
                        break;
            case  60 : child &= c30;
                        break;
            case  40 : child &= c10;
                        break;
        }
        if ( !child.located ) {
            age_group -= 20;
            continue;
        } else {
            children.insertReference(child);
            if ( mother.located )
                mother.children.insertReference(child);
            child.next();
        }
    }
}
```

3.4.1 Create test data (Linux)

Typically, the dictionary is stored in the project root (e.g. ~/`Sample`). In the example, the script file has been stored in a sub directory osi (~/`Sample/osi/Sample.osi`). When generating another project than Sample, an appropriate OSI script file has to be provided.

```
~/odaba/Sample$ ./OSI.sh
```

This procedure is properly initialized when generating the Sample project but has to be adapted to specific requirements when generating other projects.

The load protocol is written to console as shown in the example below.

```
2010-10-01 16:23:17 - Running /usr/local/lib/odaba/tools/OSI with:
  ini-file: Sample.osi
  script file:
New database created at '/home/testuser/odaba/Sample.dat'
Starting company transaction: 19:42:08.51
Storing company transaction: 19:42:08.79
Stopping company transaction: 19:42:08.84
Starting person transaction: 19:42:08.87
Storing person transaction: 19:42:24.92
Stopping persons transaction: 19:42:25.39
Create child/parent relations: 19:42:25.54
Created: 19:42:41.17
```

Notes: When there is no odaba.ini stored in the ODABA system folder or when it contains an invalid reference to the ode.sys database, instead of the message

```
New database created at 'Sample.dat'
```

the following message will appear:

```
Undescribed Error : 98 in LDBHandle::Open (Sample.dat,InputArea,ik_name,,,)
```

which may be ignored.

Due to random name generation some errors may appear on console and/or in the error log file as:

```
Error 64 in Property::insert (Error in Field: Company )
Error 152 in Property::insertReference (Error in Field: children of Struc-
ture: Person )
```

this only means that the script created two persons with the same name and the database rejected the data. Nevertheless, one may safely continue with the next steps.

3.4.2 Create test data (Windows)

Typically, the dictionary is stored in the project root (e.g. E:\Sample). In the example, the script file has been stored in a sub directory OSI (E:\Sample\osi\Sample.osi). When generating another project than Sample, an appropriate OSI script file has to be provided.

```
...>E:\odaba\Sample\OSI.cmd
```

This procedure is properly initialized when generating the Sample project but has to be adapted to specific requirements when generating other projects.

The load protocol is written to console as shown in the example below.

```
2010-10-01 16:23:17 - Running C:\Programs\odaba\OSI.exe with:
  ini-file: E:\Sample\osi\Sample.osi
  script file:
New database created at 'E:\Sample\Sample.dat'
Starting company transaction: 19:42:08.51
Storing company transaction: 19:42:08.79
Stopping company transaction: 19:42:08.84
Starting person transaction: 19:42:08.87
Storing person transaction: 19:42:24.92
Stopping persons transaction: 19:42:25.39
Create child/parent relations: 19:42:25.54
Created: 19:42:41.17
```

Notes: When there is no odaba.ini stored in the ODABA system folder or when it contains an invalid reference to the ode.sys database, instead of the message

```
New database created at 'e:/Sample/Sample.dat'
```

the following message will appear:

```
Undescribed Error : 98 in LDBHandle::Open (Sample.dat,InputArea,ik_name,,,)
```

which may be ignored.

Due to random name generation some errors may appear on console and/or in the error log file as:

```
Error 64 in Property::insert (Error in Field: Company )
Error 152 in Property::insertReference (Error in Field: children of Struc-
ture: Person )
```

This only means that the script created two persons with the same name and the database rejected the data. Nevertheless, one may safely continue with the next steps.

3.4.3 OSI Script comments

The OSI script example creates test data for the sample database. This is, perhaps, not the most typical use case, but it also demonstrates the principle use of OSI scripts. For an OSI script, the database is the memory, i.e. no queries are necessary, since collections and instances are just there. On the other hand, OSI behaves like a normal programming language (similar to C# or Java). Thus, database access becomes very flexible and is rather familiar to programmers. More details, you will find in **OSI - ODABA Script Interface** in the ODABA Online Documentation.

Data source

Static or global OSI scripts require a data source, which defines the database the script is working on. In the example, the data source is defined in the script file by defining the path for `DICTIONARY` and `DATABASE`. One might also define the data source in an ini-file or data catalog and referring to the data source name by the `DATASOURCE` keyword.

```
DICTIONARY='Sample/Sample.dev';
DATABASE='Sample/Sample.dat';
```

Notes: On Windows you have no common place for the Databases and Dictionaries. In contrast Linux provides a Directory in /var/lib/odaba. Therefore you have to change the provided examples or consequently ignore the presumption and write the ini-files for yourself.

3.5 Evaluate test data

Some examples for accessing data in an ODABA data base via OShell commands and embedded OSI scripts have been provided within the OShell script shown below in the example. The first lines in the script contain OShell script statements for producing simple statistics. The second part contains embedded OSI script in order to provide more complex statistics. Detailed comments to the example you will find in **OShell Script comments**.

After creating the Sample project, one may simply call **OShell.sh** (LINUX) or **OShell.cmd** (Windows), which have been generated to the project's root directory. When generating other projects, the generated scripts have to be revised in order to check file location for the called **.osh** script file. OShell allows opening data sources defined in a data catalog or in a configuration file passed to the OShell function. The generated configuration file **ode.ini** contains data source sections for the resource database (e.g. [**Sample_dev**]) and the application database (e.g. [**Sample_dat**]) prefixed with the name of the project created. By default, dictionary and database are expected in the project's root directory. Otherwise, data sources in the **ode.ini** file have to be updated.

After processing the command passed in the script file, the OShell remains open and the data source is still active. The data collection currently selected is `Company`. You may enter further OShell commands or `q[uit]` in order to leave OShell.

```
cd Sample_dat

cc Company
li
fa p cars.count
fa p employees.count

cc /Employee
count
sf "sex == 'male'"
relativeCount

cc '/Person::Persons'
count
osi do
VARIABLES
    int         child_count;
    int         distance = 0, dist = 0;
    int         min_dist = 100, max_dist = 0;
    int         dcount = 0;
PROCESS
    filter('age > 65 && employee.count == 0');
    Message('Persons probably retired (over 65 and not employed): ' +
relativeCount);

    top();
    while ( next() )
```

```
        child_count += children.count;
    Message('Total number of children: ' + child_count);

    filter("");
    top();
    while ( next() )
        while ( children.next() ) {
            dist = age-children.age;
            if ( min_dist >= dist )  min_dist = dist;
            if ( max_dist <= dist )  max_dist = dist;
            distance += age-children.age;
            ++dcount;
        }
    Message('Average age distance between children and parents: ' +
distance/dcount);
    Message('   Minimum/maximum distance is: ' + (string)min_dist + '/' +
(string)max_dist);
end

cc /Company
osi do
VARIABLES
    int(10,2)     sum_income;
PROCESS
    top();
    while ( next() ) {
        sum_income = 0;
        while ( employees.next() )
            sum_income += employees.income;
        Message('Total income for "' + name + '" is: ' + sum_income);
    }
end
```

3.5.1 Evaluate test data (Linux)

OShell is called with a configuration file **ode.ini** stored in the project's root directory (~/Sample/ode.ini). When the Sample project had been generated, the **osi** sub directory contains an oshell script **Sample.osh** (~/Sample/osi/Sample.osh), which may be called from the OShell:

```
~/odaba/Sample$ ./OShell.sh
```

```
ODABA>call 'osi/Sample.osh'
```

The load protocol is written to console as shown in the example below.

```
12-11-08 - Running /usr/local/lib/odaba/tools/OShell with:
  ini-file: ode.ini
  script file:
ODABA>call 'osi/Sample.osh'
  Any company
  Best company
  My company
  NO company
  Your company
  cars.count=9
  cars.count=5
  cars.count=15
  cars.count=4
  cars.count=13
  employees.count=98
  employees.count=110
  employees.count=96
  employees.count=77
  employees.count=81
  count returns: 462
  relativeCount returns: 241
  count returns: 1000
Persons probably retired (over 65 and not employed): 400
Total number of children: 1051
Average age distance between children and parents: 41.318873668188736
   Minimum/maximum distance is: 11/69
Total income for "Any company" is: 436170.00
Total income for "Best company" is: 543902.00
Total income for "My company" is: 485852.00
Total income for "NO company" is: 369282.00
Total income for "Your company" is: 371598.00
Sample_dat/Company>
```

Notes: Data has been generated randomly, i.e. figures may differ.

3.5.2 Evaluate test data (Windows)

OShell is called with a configuration file **ode.ini** stored in the project's root directory (e.g. `E:\Sample\ode.ini`). When the Sample project had been generated, the osi sub directory contains an `oshell` script **Sample.osh** (`E:\Sample\osi\Sample.osh`), which may be called from the OShell:

```
...>E:\odaba\Sample\OShell.cmd
```

```
ODABA>call 'osi/Sample.osh'
```

The load protocol is written to console as shown in the example below.

```
12-11-08 - Running C:\Programs\odaba\OShell.exe with:
  ini-file: ode.ini
  script file:
ODABA>call 'osi/Sample.osh'
  Any company
  Best company
  My company
  NO company
  Your company
  cars.count=9
  cars.count=5
  cars.count=15
  cars.count=4
  cars.count=13
  employees.count=98
  employees.count=110
  employees.count=96
  employees.count=77
  employees.count=81
  count returns: 462
  relativeCount returns: 241
  count returns: 1000
Persons probably retired (over 65 and not employed): 400
Total number of children: 1051
Average age distance between children and parents: 41.318873668188736
   Minimum/maximum distance is: 11/69
Total income for "Any company" is: 436170.00
Total income for "Best company" is: 543902.00
Total income for "My company" is: 485852.00
Total income for "NO company" is: 369282.00
Total income for "Your company" is: 371598.00
Sample_dat/Company>
```

Notes: Data has been generated randomly, i.e. figures may differ.

3.5.3 OShell Script comments

The OShell script in the example demonstrates some typical features. Besides elementary script commands as changing data sources (cd) or data collections (cc), OShell supports embedded OSI scripts. When you enter script data directly via console, blocks (beginning with `.. .do` or `... begin`) are processed only after terminating the block with an `end` command. More details one may find in **Database Utilities**.

Selecting data source and collection

The first command (**cd**) selects the data source (`Sample`) as being defined as section in the ini-file. Thus, one may switch between different data sources by defining different data sources in the ini-file. After the data source has been opened, a collection (extent) is selected (**cc**). After selecting a data collection, one may browse data in the data collection or change to subsets (references or relationships).

```
cd Sample
cc Company
```

3.6 Design a GUI application

After importing and testing the database schema, a GUI application can be build in order to access the test data via a GUI example. The following steps demonstrate the typical activities necessary for creating and running an ODABA GUI application.

Notes: The GUI framework is a bit unstable and tools sometimes abort. Usually, what you have done so far is already saved and one may simply restart the **Designer** and continue.

3.6.1 Starting Designer

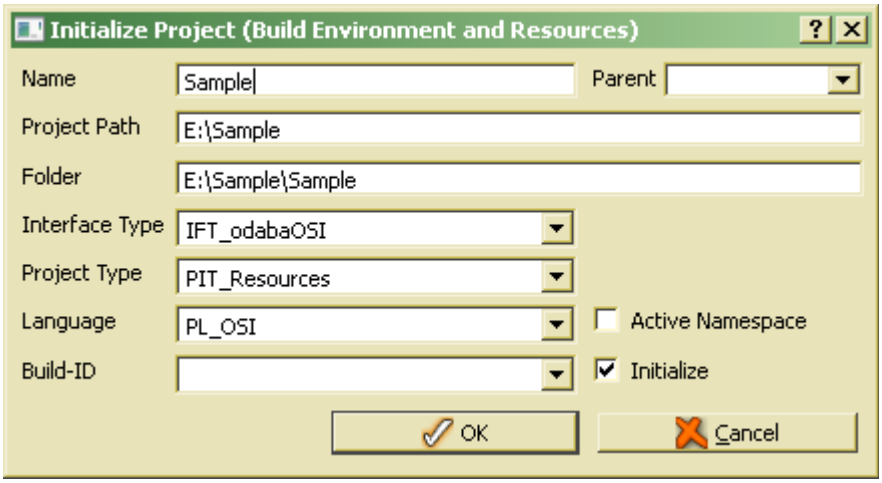
In order to start the Designer, the procedure **Designer.sh** (LINUX) or **Designer.cmd** (Windows) generated by `CreateProject` may be called from the project's root directory.

When starting ODE tools the first time, they will load several resources from the system resource database, which might be used for development but also updated by the user. Thus, starting the Designer the first time, you will be asked to initialize a project.

How to start the Designer under Linux or Windows you will find in the following subtopic.

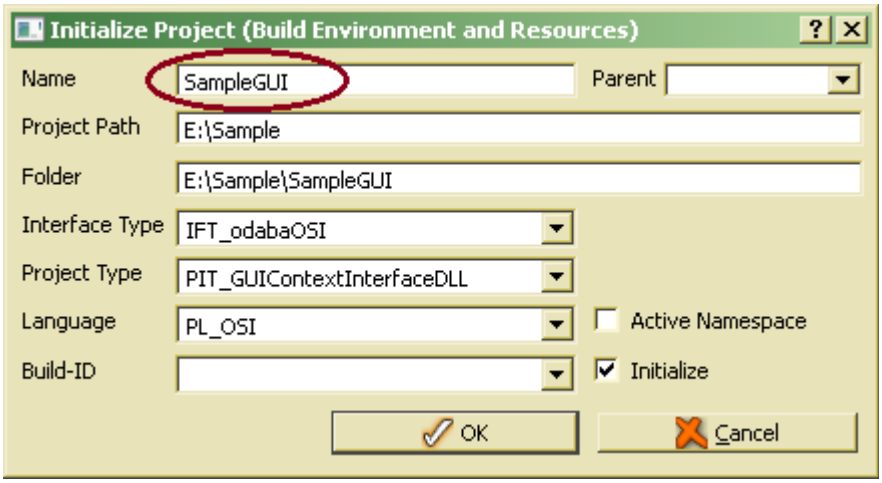
When the Designer starts up, it requires to initialize a project. The project name, e.g. **Sample**, is usually used for the resource project which is identical with the schema name when loading a database scheme from an ODL file. In order to initialize a GUI context interface project, one should define a name different from the resource project (e.g. **SampleGUI**).

When ClassEditor resources have not yet been initialized (by previous call of ClassEditor), first, ClassEditor resources will be initialized. For initializing ClassEditor resources you are prompted to initialize the resource project.



For first test it is suggested to use OSI interface (**IFT_odabaOSI**), which is the default setting.

In order to create a GUI application, one should create a GUI context interface project for implementing application rules (project type **PIT_GUIContextInterfaceDLL**). This allows providing actions and application rules implemented as event handlers. For creating a GUI context class interface, the following settings should be selected:



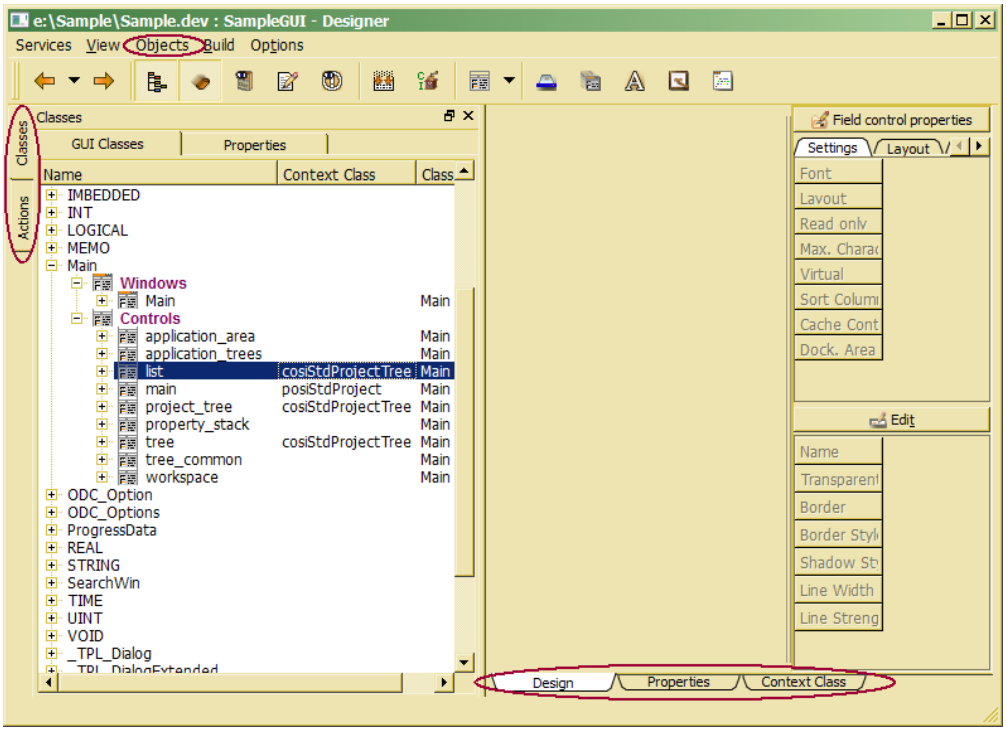
Instead of using the the C++ interface type (**IFT_odabaInterfaceCPP**), for first test it is suggested to use OSI interface (**IFT_odabaOSI**), which is the default setting, when project resources have been initialized as OSI project before. We suggest using OSI in order to build fast prototypes. Since OSI provides a script interface,

compiling and linking is not necessary. Later one may change the interface type in the project settings in order to change OSI classes to C++ or C# classes. This might become necessary in order to increase performance, e.g. when handling events on row level for large lists or tables (`doBeforeDataSet()`).

The build system (C++ and C# only) one may select in the Build-ID drop list. **vs** is preselected when compiling with MS Visual Studio compiler. **gcc** (GNU compiler) is the suggested build environment for Linux applications. The build environment might be changed later in the project.

After confirming the settings (by clicking **OK**) it will take some time (up to two or three minutes) until the project has been initialized as GUI development project. Several Designer resources are copied from the system database in order to support application design.

Two movable docking lists shown on left side provide direct access to main design resources (GUI classes and controls, and actions), which might be tabbed or arranged below each other. The Designer opens with the class view. One may change between Design, Properties and Context Classes view by selecting the corresponding tab on the bottom of the work area. Below the work area, an output area will be opened in order to show process messages.



On the left side, the Designer shows two navigation trees. In order to design windows or controls, the class tree should be used. Event actions combining actions with events are accessible via the action tree. More design resources are provided via the the Options menu in the main menu.

The three tabs below the work area allow switching between different views. The **Design** view will be activated, when selecting a control in the class tree for design. The **Properties** view displays properties of the currently selected entry in the class or action tree. When a window, control or field that is associated with a context class has been selected in the class tree, the Context Class view shows the implemented handlers and actions in the associated context class.

3.6.1.1 Calling Designer (Linux)

ODE tools like Designer require a configuration file as **ode.ini**, which has been generated to the project's root directory (~/`Sample/ode.ini`). This is referenced in the **Designer.sh** command in the project root directory:

```
~/odaba/Sample$ ./Designer.sh
```

3.6.1.2 Calling Designer (Windows)

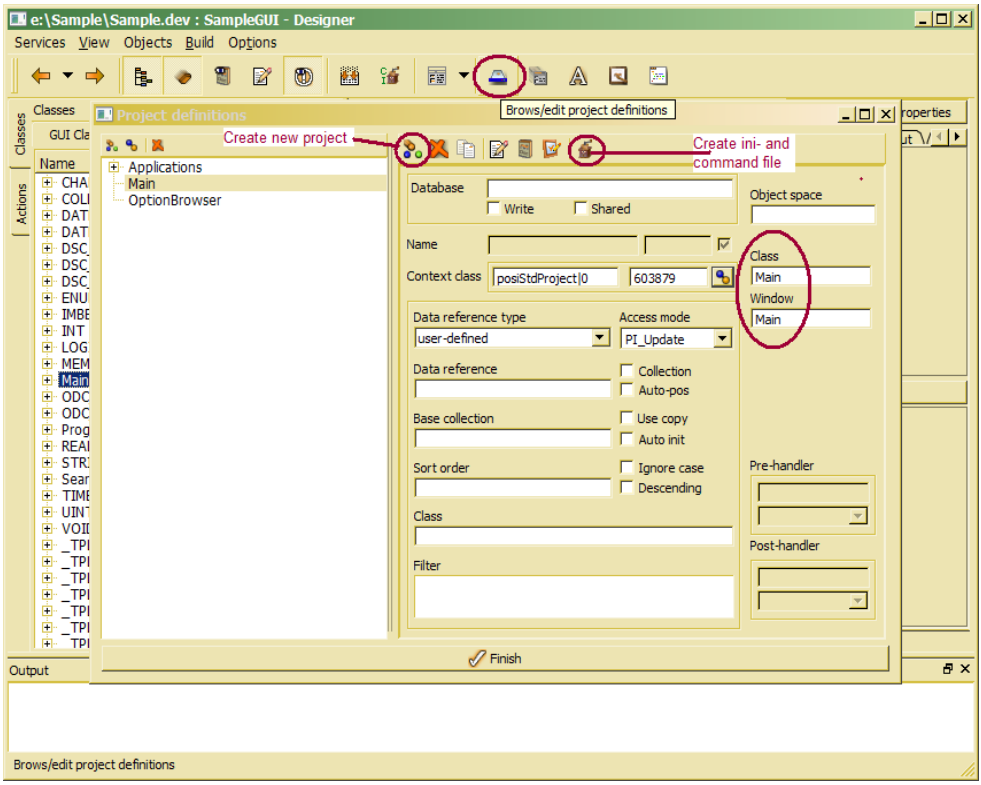
ODE tools like Designer require a configuration file as **ode.ini**, which has been generated to the project's root directory (`E:\Sample\ode.ini`). This is referenced in the **Designer.sh** command in the project root directory:

```
...>E:\Sample\Designer.cmd
```

3.6.2 Create project resource

First, an application or project should be defined. In the **Designer**, a project refers to an executable GUI application. In order to edit project definitions, click the **Browse/edit project definitions** button in the designer toolbar. When initializing **Designer** resources, a **Main** project has already been created and might be used. A new project can be created by clicking the **Create** button in the project dialog toolbar.

Check the class and the window reference in the project definition, which refers to the project window to be called when starting up the project. In the example, this is the window `Main` in class **`Main`** (**`Main`**`::Main`). The **Data source type** is typically set to user-defined.



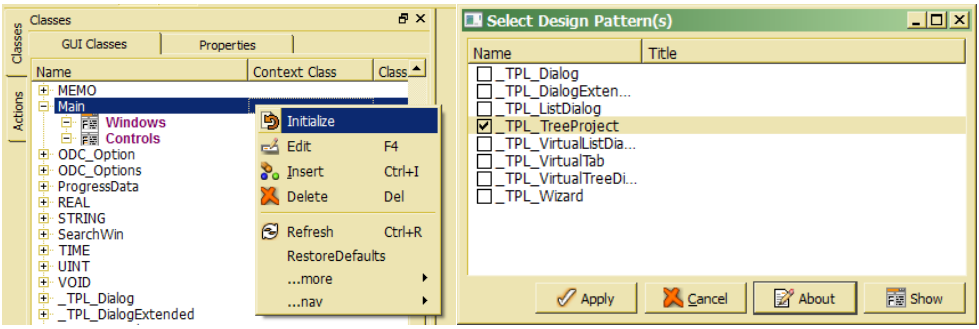
Now, one may generate external resources for calling the application. Therefore, click the Generate project files button in the project dialog toolbar (titled "Create ini- and command file" in the picture above). The application configuration file (here **Main.ini**) and a command file (**Main.sh** for LINUX or **Main.cmd** for Windows) are written to the project's root directory (see also messages in the output area.).

Additional details about defining project resources are described in **Project resource definition**.

Notes: In order to generate project resources, a project must have been selected in the project list (Main). When no project is selected, nothing will happen when pressing the "create ini- and command file" button.

3.6.3 Initialize project window from design pattern

The ODABA **Designer** provides several design patterns in order to standardize and simplify application design. Design patterns can be selected from a list, which pops up when selecting **Initialize** from the context menu after selecting the design class to be initialized. Typically, one starts with designing the project window referenced in the project defined before (e.g. `Main` window in class **`Main`**).



When the pattern selection dialog pops up, one may select one or more class patterns for initializing. Standard design pattern classes start with _TPL_ and can be designed as any other class in order to provide standard design rules for an application.

In order to initialize the main application window, one should select **_TPL_TreeProject** from the pattern list, which initializes an application with a main tree on the left side and a dynamic work area on the right side. In order to select the right pattern, one may select a pattern and click the **Show** button, which displays the current pattern design. More information about how to use the pattern one may get when clicking the **About** button.

After selecting the pattern(s) that should be used for the selected design class by activating the check box(es) in the list one may click **Apply** in order to initialize your class. This may take some seconds. After initializing, Windows and Controls shown after expanding the selected class should have become expandable. When this is not the case, you should refresh the class tree explicitly (context menu **Refresh**). Since the tree project template fits to the Sample project data model, one may now call the generated application start procedure (**Main.sh** or **Main.cmd**) in order to view Sample data in the GUI application, supposed, that OSI has been selected as default application interface language.

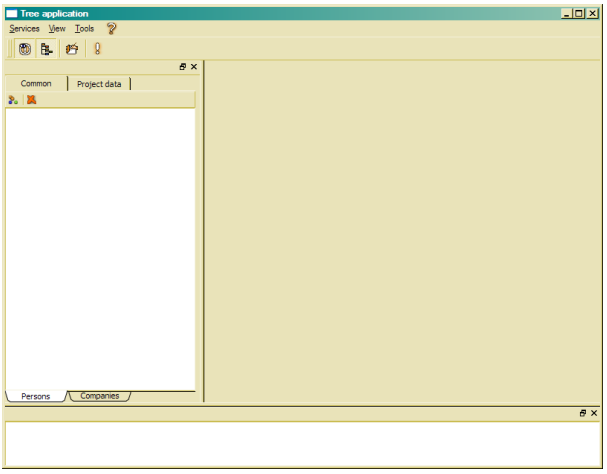
In the next step, one may expand the **Windows** and **Controls** region in order to start windows and control design (see also **Dialog design** and **Control design**).

Notes: The tree project template requires that design classes for object types displayed in the tree are initialized later with the **`_TPL_VirtualTab`** template.

3.6.4 Standard application elements

The standard application that has been initialized from the **_TPL_TreeProject** pattern can be considered as a proposal. It contains several elements, which one may not want. In this case, they can be simply removed.

After selecting the Main window in the class tree (**Main**/**Windows**/Main), one may call **Test window** from the context menu in order to get a first impression how your application looks like. The application consists of following elements, which might be removed or updated:



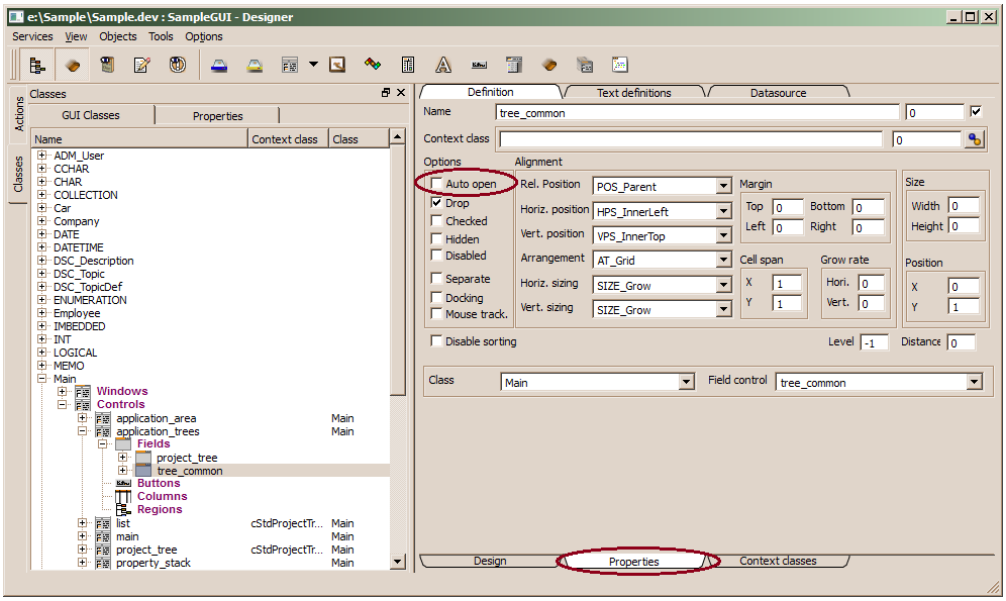
- Application Menu
- Application toolbar
- Application area

The application area contains the

- Application tree (left)
- output or message area (bottom)
- workspace area (right)

The sample contains two tree controls arranged in a tab (**Main**::application_trees). In the test mode one may check the design elements (switching tabs, resizing etc.), but you cannot enter data. In order to access data, data sources assigned to controls have to be set.

Controls in a window or control are referenced via **Fields**, which provide context specific information for displaying the control (position in the parent control, size, texts etc). Moreover, Fields combine the data source with a control, that displays the data. Fields and other elements defined within a control do have an Auto-open property, which allows to switch off an element temporarily.

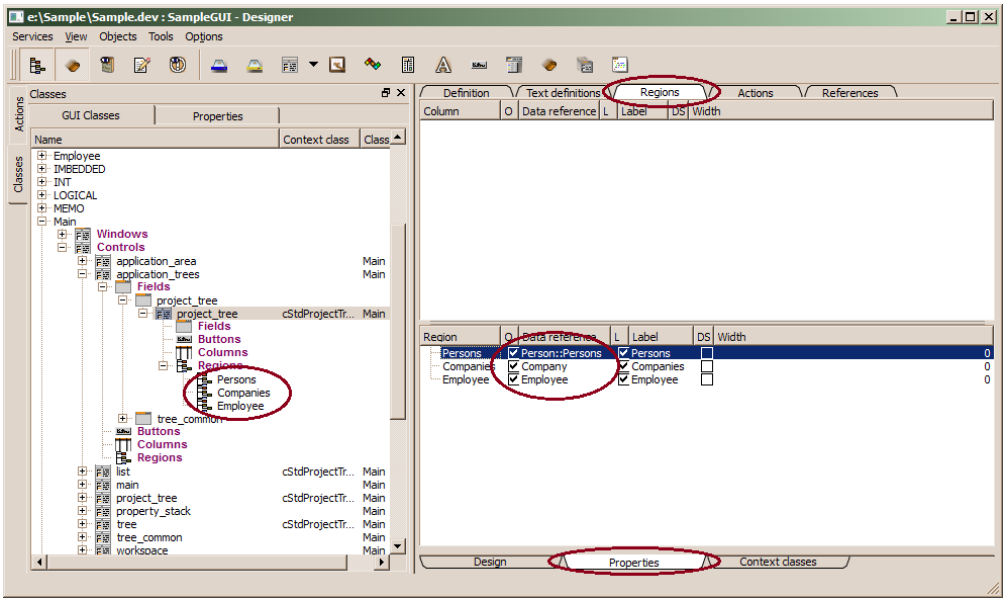


In order to deactivate, e.g. the second tab, you switch to the **Properties** view (down tab), select the field `tree_common` in control application trees and deactivate the Auto-open check box (right). When you test the application again (**Test window**), the tree area shows only one tabs.

Notes: One should never call Design for the Main window (**`Main`**/**`Windows`**/`Main`) or the main application control (**`Main`**/**`Windows`**/`Main/main`) since this may cause strange side effects.

3.6.5 Defining data sources

The next step, usually is to assign data sources to fields and regions in the control. In the example, the project tree should show two regions, for **`Person`**`::Persons` and **`Company`**`::Companies`. In order update region data sources, one may select the **`Main`**`::project_tree` control and select **Regions** on the right side property window.



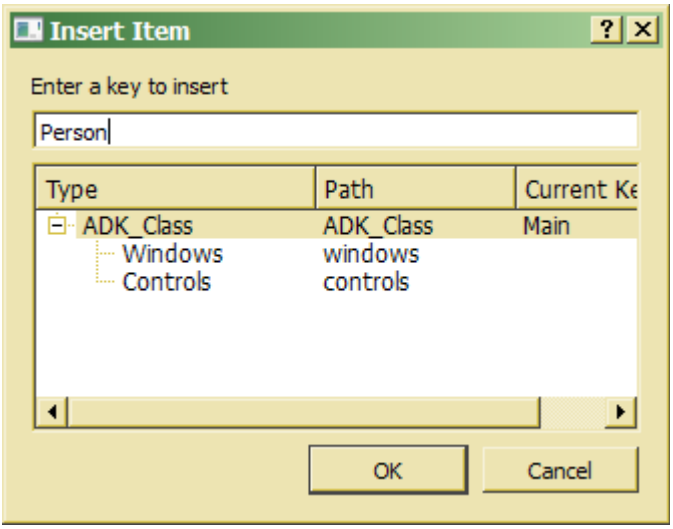
The data source defined in the template might not be correct. Since `Persons` is a defined extent in **`Person`**, the data reference has to refer to a scoped name (as shown above).

More details for defining data sources you will get in the property view when you select one of the regions in the class tree on the left side.

Notes: After you have checked the data sources, you may test the application by running it. The GUI framework provides sufficient functionality, that you will see already some data in the application (see **Run the application**).

3.6.6 Create new design classes

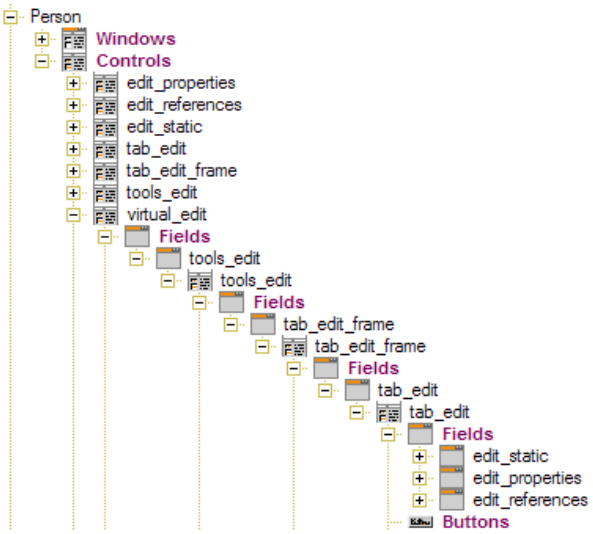
The Data types defined in the project object models are not yet visible as design classes. In order to display data for the data types, complex data types have to be added to the designer class list. In order to create design classes for existing data types, one may select **Insert** from the class tree context menu. Then, a tree insert dialog pops up.



Enter the data type name, check that **`ADK_Class`** is selected in the type tree below the name field in the insert dialog and press **OK**. After creating the design classes for **`Person`**, **`Employee`**, **`Car`** and **`Company`**, you should initialize those by selecting the class and clicking **Initialize** in the class tree context menu (see **Initialize design class**). When the template selection window appears, you should now select **`_TPL_VirtualTab`** for each class.

3.6.7 Virtual controls

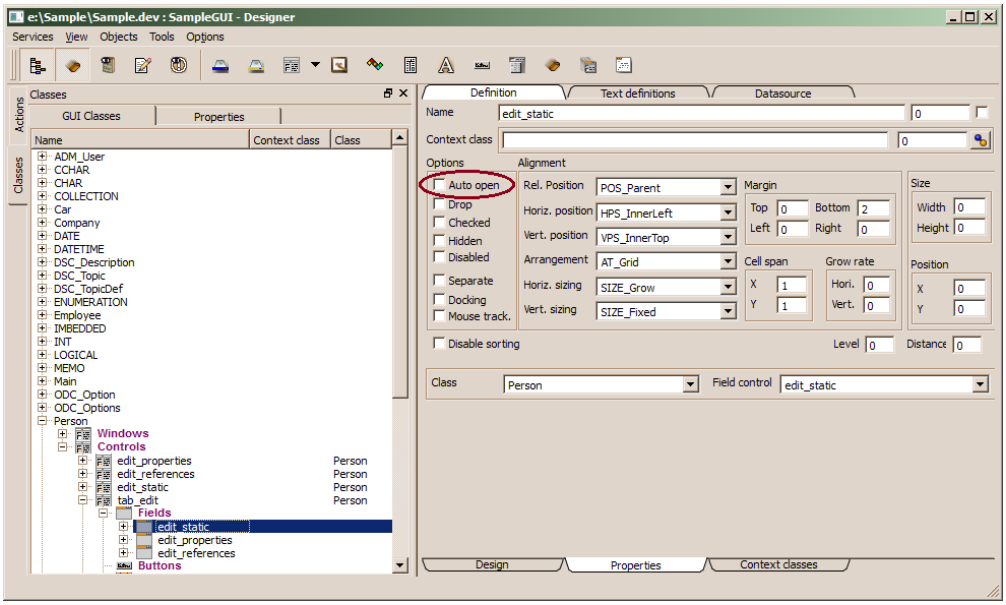
Virtual controls are controls, that automatically pop up in the work area (right side of the application) when selecting an entry in the application tree. The control to be shown on the work area depends on the complex data type of the instance selected in the tree. Thus, defining a virtual control in the work area (here `virtual_edit`) will automatically look for `virtual_edit` controls in the class of the selected instance and in its base classes.



By initializing the class with the **_TPL_VirtualTab** template, the preconditions for implementing virtual tab controls are given. The project tree requires a virtual control with the name `virtual_edit`. The template also initializes an `Edit` window, which will be displayed when selecting edit from the context menu in a list or tree. The virtual edit control refers to a `tools_edit` control, which provides a toolbar for the tab control referenced below. Even though the control is rather complex, the only thing to do is to design three controls.

Each `tab_edit` controls contains a static area (e.g. **`Person`**`::edit_static`), which will be displayed on each tab level. The first tab page shows the object's properties (**`Person`**`::edit_properties`) and the second tab shows references (**`Person`**`::edit_references`). In order to make data visible, those three controls should be designed or removed. The `tools_edit` control also defines a window toolbar, which might be updated or removed. Some additional `...frame` controls have been added, which are necessary for technical reasons.

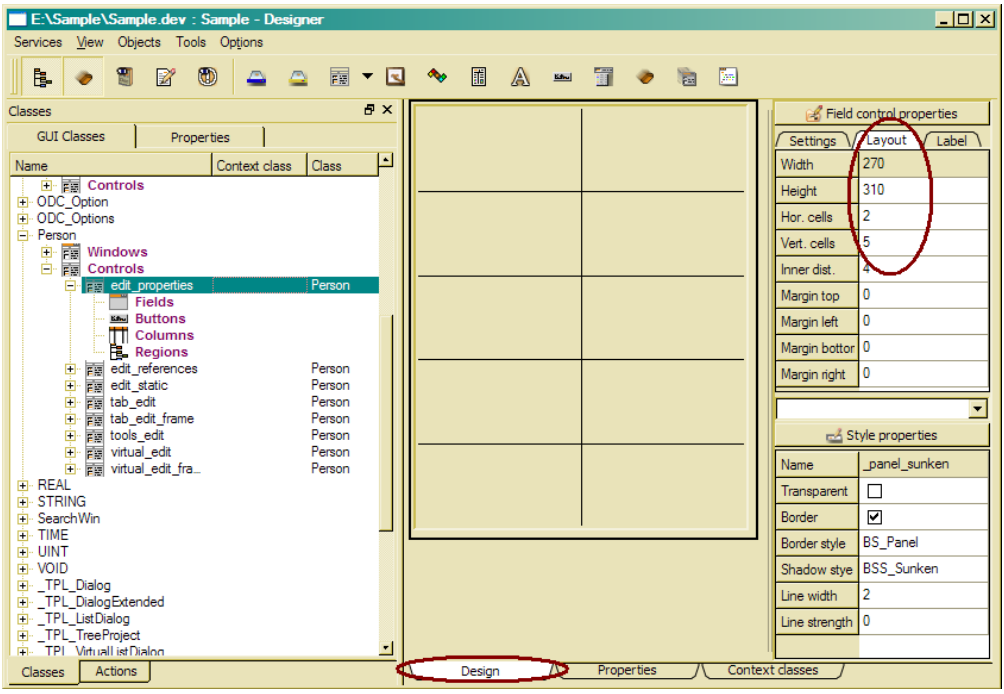
In order to disable elements (e.g. the `edit_static` control in the `tab_edit` control, you only need to switch off the **Auto-open** option after switching to the **Properties** view) (see also **Standard application elements**).



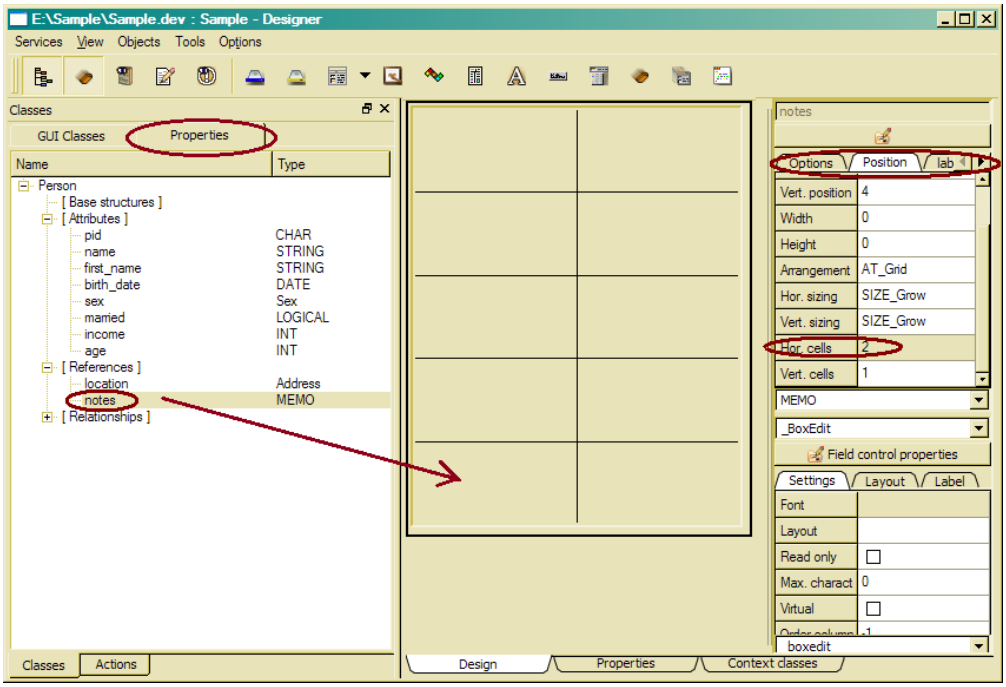
Notes: Unfortunately, this feature is not yet fully implemented, i.e. there is still a small interface necessary. Thus, a context class has to be defined for the tree (in this case **`cStdProjectTree`**, which implements a `doAfterSelect()` handler. This handler still requires three lines of code in order to inform the work area about the change in the tree selection. In the initialized database, there you will find the **`cStdProjectTree`** context class, which provides a corresponding `doAfterSelect()` handler.

3.6.8 Design a complex control

In order to design the `edit_properties` and `edit_references` control, one may select the field `edit_properties` and Design from the context menu. Then, the right side window changes to **Design** view and a 5 x 2 grid control will be displayed on the design area.

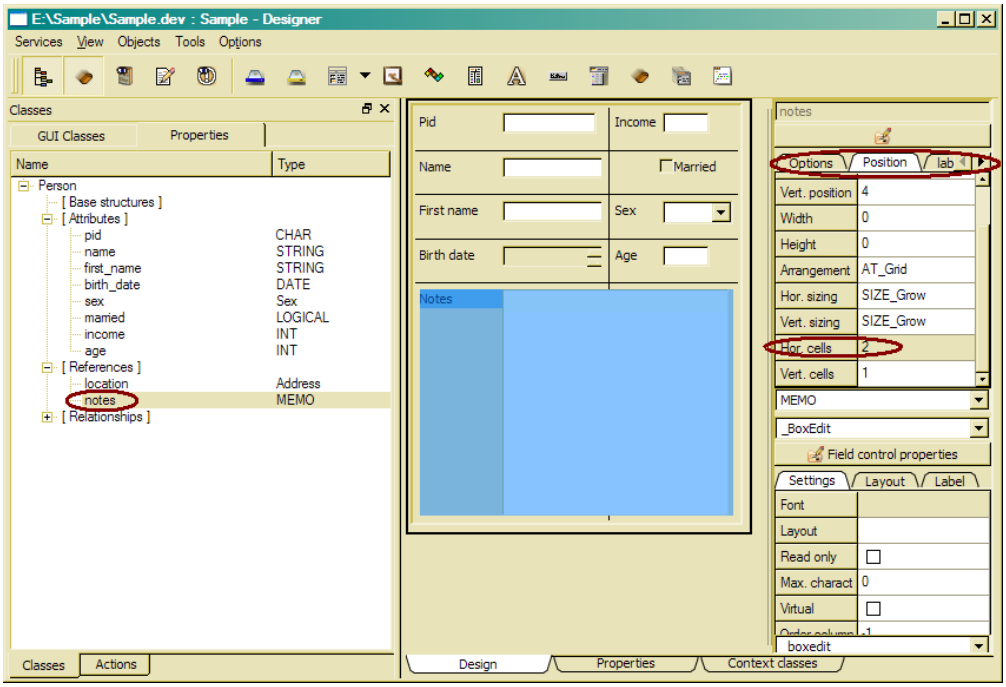


Then you change the class tree to Properties view (tab above the class tree) and expand to **Person/Attributes**. Now, you drag the attributes to a grid cell in the design window and drop them there.



Essential control properties are size and grid dimensions (vertical and horizontal cells). Those might be updated in the property view, but also directly here in the design view in the property lists on the right side of the designer window, which display relevant properties for designing controls, fields and buttons.

After you have dragged the attributes you want to display in the application, one may change field properties in the property table right of the designer area, e.g. set horizontal cells to 2 for the `notes` field.



Details about different design properties you will find in the Control Design reference guide (**Reference documentation/Design objects/Control design**). After designing the `edit_properties` control, one may design the `edit_references` control by dragging relationships and references to the control grid cells.

Now, one may repeat this procedure for **`Employee`**, **`Company`** and **`Car`** and your application will allow viewing, creating and updating objects in the Sample database.

Notes: When changing properties in the property window (right), changes become visible after you have left the field you just edited.

3.7 Define application rules

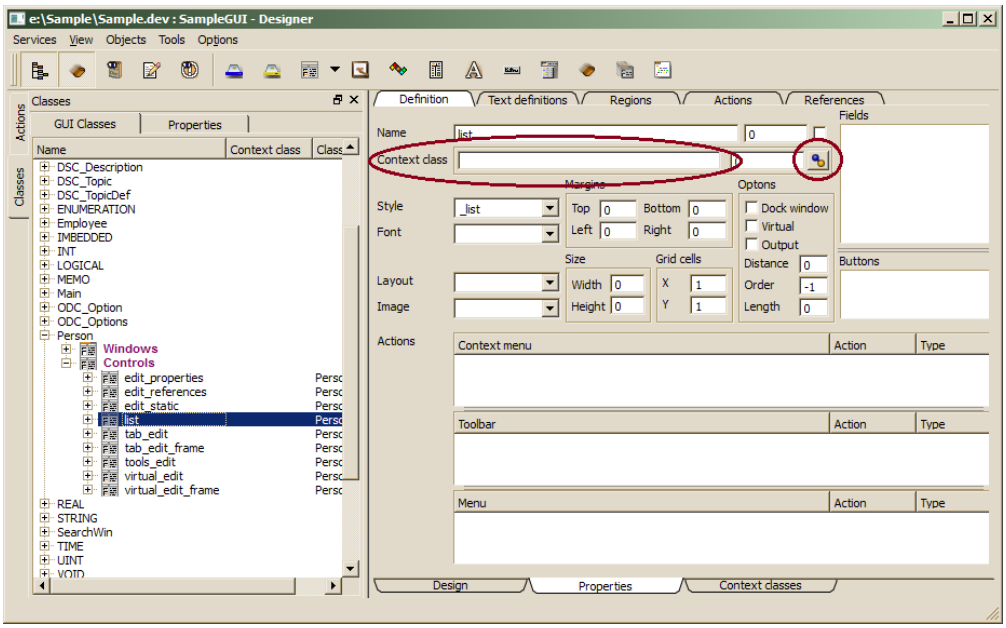
Application rules are, typically, defined on control or field level. Since a control defines common behavior for a GUI element, the field defines the behavior of a GUI element in a certain context (form or window). Thus, one may associate a control context class defining the behavior for the GUI element with a field, but also with a control.

When a context is defined for the control referenced by the field, but also for the field, the field context class will overload the control context class completely, i.e. control context behavior will not be activated at all. In order to specialize the control context behavior in a field context class, one may define a field context class which inherits from the control context class.

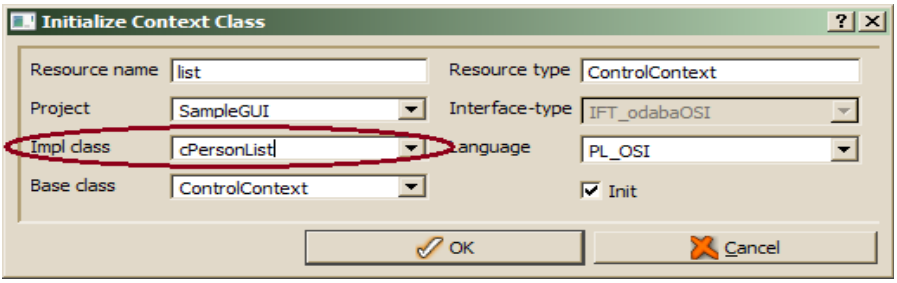
After implementing the handlers and functions required, the context class library has to be created (see **Create application rules library (context class library)**), except the class has been implemented as OSI class.

3.7.1 Create new context class

In order to associate a context class with a control or field the name of the context class has to be set in the context class field in the control or field property window.



When associating an existing context class, you need just enter the name in the context class field. In order to create a new context class, it is suggested to use the associate button on the right side if the context lass field. After pressing the button, a context class initialization dialog pops up:



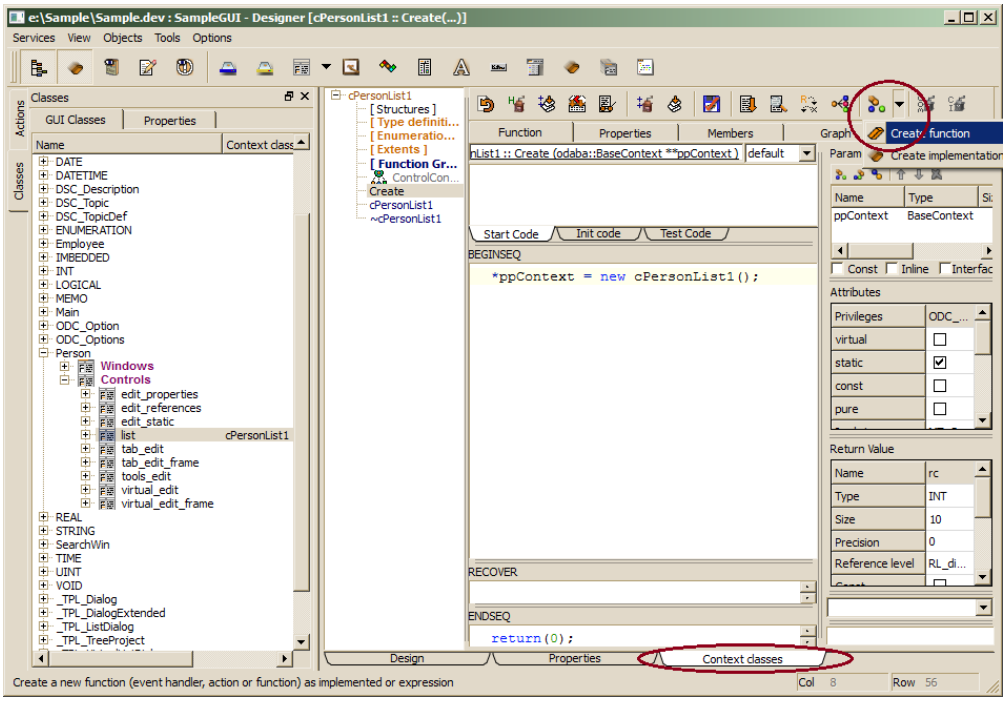
Usually, values in the fields are set properly from the project definition and only the name of the implementation class needs to be entered in the `Impl.class` field. In order to associate an existing context class, one may select one from the drop list or enter a new class name.

The base class, the context class inherits from is usually the **`ControlContext`** class. In order to change the base class, one may select an existing context class from the drop list, which, directly or indirectly, has to inherit from the Control context class.

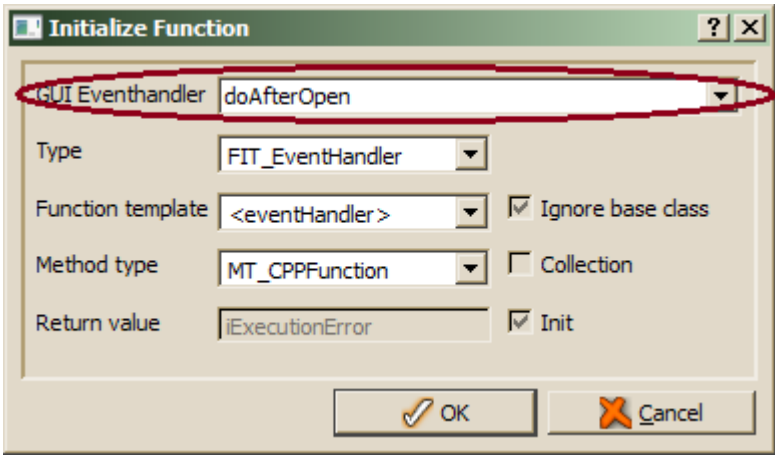
When creating a new context class, the class is initialized be creating default functions depending on the programming language used (e.g. Create function, constructor and destructor). In order to avoid initializing the context class, the initialize option has to be switched off.

3.7.2 Edit context class

In order to implement a GUI context class, one may change to the **Context class** tab in the **Designer** or edit the class in the **ClassEditor**.



The function list displays the functions already implemented or generated. In order to create new functions, one may use the context menu on the function list or the create button in the class toolbar. When selecting the Create function menu item from the toolbar menu (as shown in the picture above), a function initialization dialog pops up:



By default, event handler function type (**FIT_EventHandler**) is selected fot the context function to be created. In this case, one may select one of the event handler names defined in the drop list above. When changing the function implementation type, the drop list disappears and an input field allows you entering your function name. The Method type is selected according to the settings in the project. In order to test new functionality, one may select OSI (**MTT_OSIFunction**) rather than C++ or C#. When compiling the class, OSI functions are not be generated into the class source file. Thus, one may test the new function immediately without compiling the class and creating a new context library version. Later, you might change the implementation type and create a C++ or C# function from the OSI function.

While creating the function it will be initialized according to the handler function declaration or the function template selected from the **Function template** list. When the same function has been implemented in a base class, the function will be initialized according to the function definition in the base class. In order to avoid initializing from base class functions, the **Ignore base class** option should be switched on.

Notes: One might mix OSI with other programming languages as long as both are strictly separated, i.e.as long as OSI functions do not call C++ or C# functions and reverse. The only exceptions are actions called via `executeFunction()`, which supports mixed processing.

3.7.3 [Create application rules library (context class library)]

The application that has been defined using the **Designer** has already implemented some default application rules. Depending on the selected programming language, application rules are provides as OSI, C++ or C# classes. When implementing rules in C++ or C# classes, those have to be provided in a so-called appli-

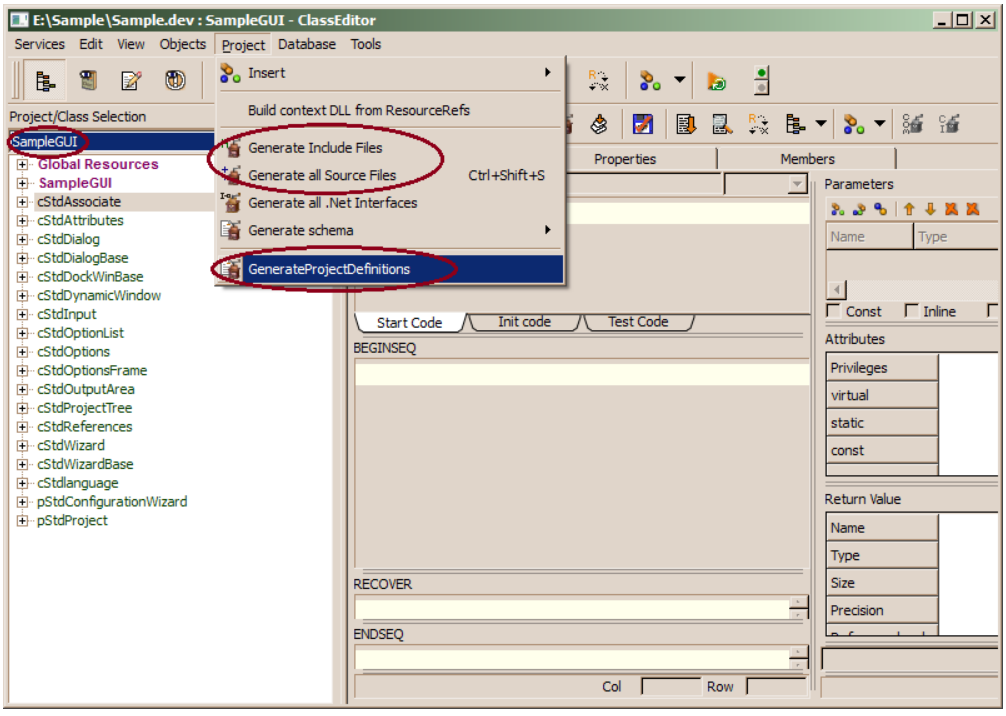
cation context library. In order to create the context library, one has to compile the **SampleGUI** context project, which is not necessary for OSI implementations. Hence, the following topics might be skipped for OSI projects.

In order to create the application context library, the **ClassEditor** has to be started (Windows: `ClassEditor.cmd`, Linux: `ce Sample`). There one may update the application context interface, create external resources and build the application context interface.

Notes: This step is necessary for C++ or C# projects, but not for OSI projects.

3.7.3.1 Update external resources

ODE generates external resources in order to call compiler and other services. When project definition changes, you are usually prompted for updating external resources. When the GUI context project is not yet selected after starting the ClassEditor, one may select it from the project list (drop list above the class tree, left side).

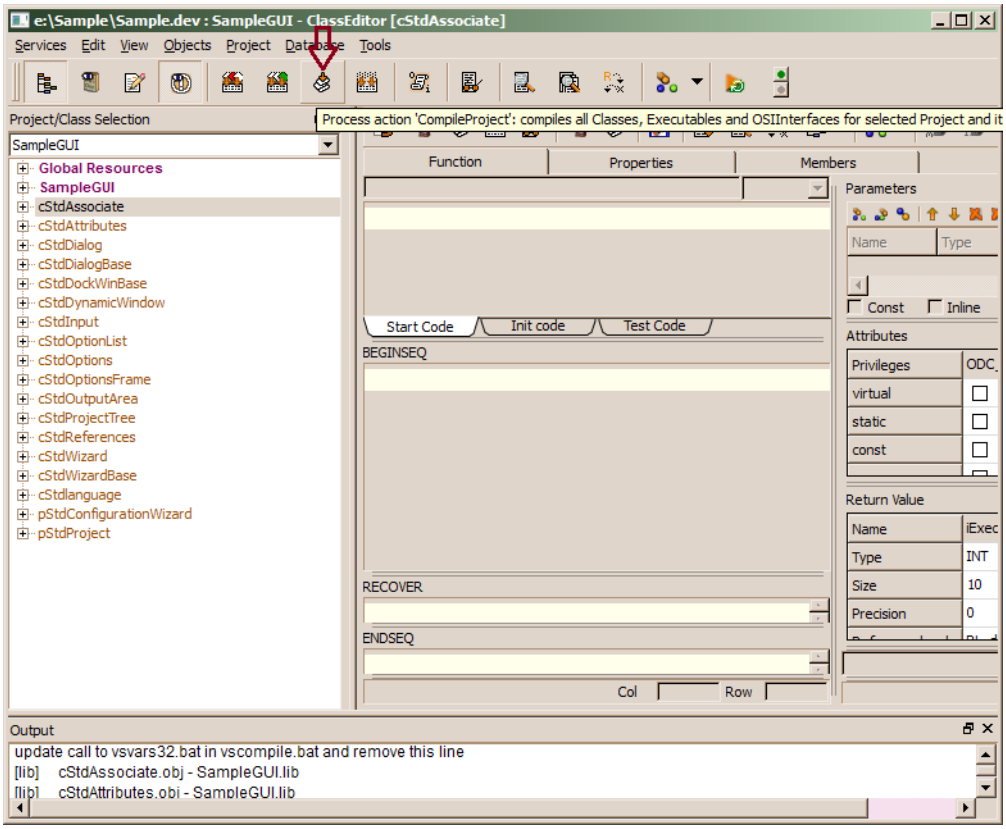


Normally, source code and header files have already been created in **Sample/SampleGUI/qlib** and **Sample/SampleGUI/h**. When this is not the case, one may generate sources and header files by clicking the appropriate menu items.

Generate project definitions creates command files for compiling and linking project resources. This is, usually, done automatically.

3.7.3.2 Compile application context classes

After creating external resources, one may compile the project by selecting the "Compile all classes" action:



The compiler output will be displayed in the output area on the bottom of the ClassEditor. Since output will be displayed when the process terminated, i.e. after compiling all classes, it may take a while, until the message appears.

Another way for compiling the project is going straight to the **Sample/SampleGUI/bat** folder, where you will find procedures for compiling and linking the project (**gcc** for Linux and **vs** for Windows):

`*_compileProject.bat`

3.7.3.2.1 Create context library (Windows)

In order to create the context library in a Windows system make sure, that you have activated compiler options. The procedures delivered are prepared for MS Developer Studio 10. When you are using another compiler version, you need to update the command files in the **odaba\bat folder**.

Now, one may call compile and link procedures:

```
vs_compileProject.bat
```

from the **Sample\SampleGUI\bat** folder.

3.7.3.2.2 Create context library (Linux)

In order to create the context library in a Linux system make sure, that you have installed a compiler and the -dev packages of ODABA. The procedures delivered are prepared for a system-installed gcc (g++). When you are using another compiler or you want to change something in the defaults, you need to update the command files in the **~/odaba/projects/bat** folder. You can reset the changes by removing the ~/odaba/projects/bat folder and restarting the ClassEditor.

Now, one may call compile and link procedures:

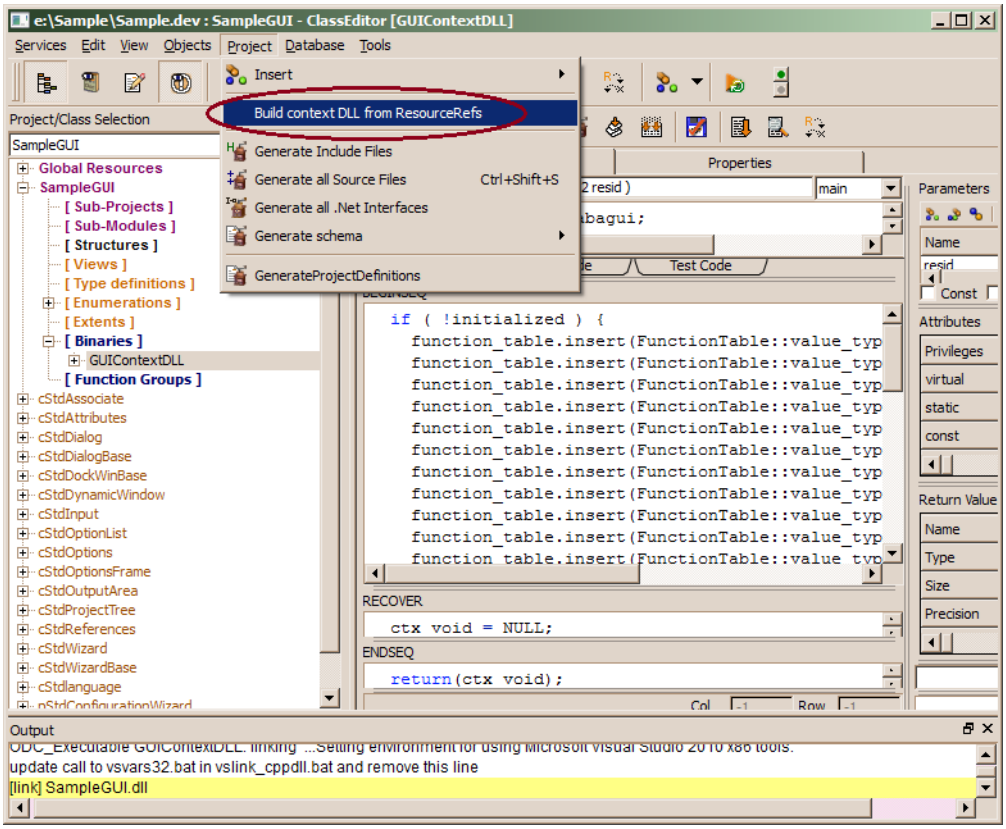
```
gcc_compileProject.bat
```

from the **~/odaba/projects/Sample/SampleGUI/bat** folder.

3.7.3.3 Update application context interface

The GUI framework requires a context interface, which connects GUI resources to context classes. This interface can be generated from within the **ClassEditor**. Before generating the context interface, it is suggested to compile all classes which need to be compiled.

After all context classes have been compiled successfully, the context interface might be build. When selecting **Project / Build context interface from ResourceRefs** from the main menu, the source for the interface will be generated.



The generated code can be checked when selecting **GUIContextDLL** in the class tree. The library will be created and stored to the project library folder (**Sample/SampleGUI/exe**). Moreover, it will be copied to the **Sample** folder where the application is looking for the library. When the Sample folder does not contain this library, you should copy it manually:

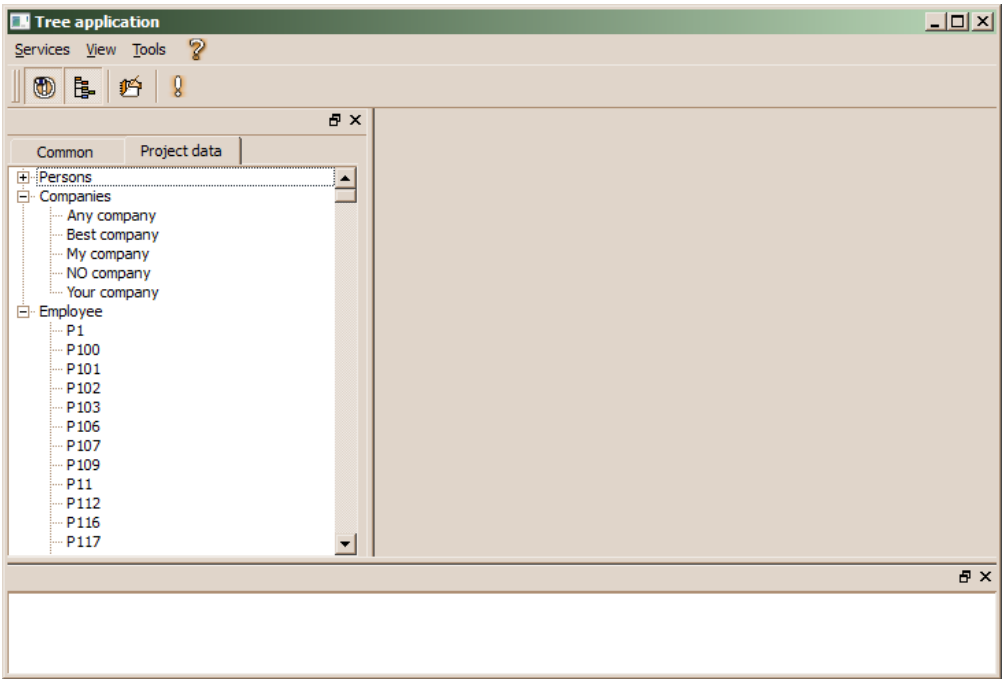
```
Sample/SampleGUI/exe/SampleGUI.dll --> Sample/SampleGUI.dll
```

Now, you are ready to run the Sample application.

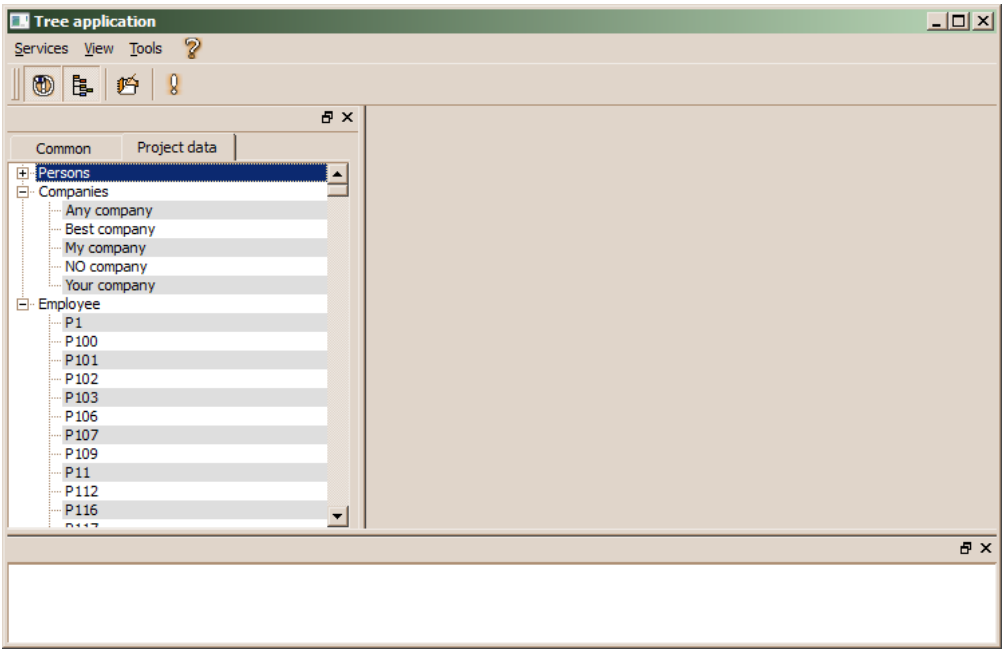
3.8 Run the application

In order to start the application, one may call the start procedure that had been generated to the project root (**Main.sh** for Linux or **Main.cmd** for Windows in the example).

The application may run with or without context library. Running the application without context library just displays the data in the project tree. You may delete or insert objects, but you cannot see details unless you 'Edit' the object via context menu. OSI applications do not need a context library, but refer to context class function stored in the resource database.



The tree/list context menu provides the standard functionality supported for the tree without application context library, When running the application after the application context library has been created, the application looks a bit different:



The alternating colors in the list and the object details on the right side are result of caused by the application context library.

Notes: Main is just the name that had been used for the sample application. Any other application name might be used instead when creating the GUI application (project). See also **Design a GUI application**/**Create GUI application**.

3.8.1 Run the Application (Linux)

To run the application execute the **Main.sh** that has been generated when creating application resources (**Design a GUI application**/**Create GUI application**). The command file is located in the project's root directory:

```
~/odaba/Sample$ ./Main.sh
```

3.8.2 Run the application (Windows)

To run the application execute the **Main.cmd** that has been generated when creating application resources (**Design a GUI application**/**Create GUI application**). The command file is located in the project's root directory:

```
...>E:\Sample\Main.cmd
```

Notes: If you are missing the command file, please review the notes in '**Design a GUI application**/**Create GUI application**' as they contain vital information and require your interaction.