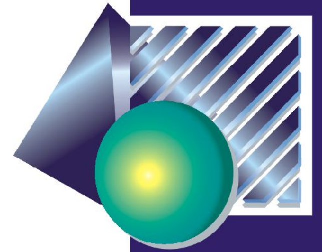


# ClassiX®

## ClassiX®

## Tutorial 2

## Content

**1. ABSTRACT** .......... **2**
**1.1 Course preparations** .......... **2**

**2. MODULARISATION** .......... **3**
**2.1 What is a module?** .......... **3**
**2.2 Modularisation of the application** .......... **4**
2.2.1 All modules in one file .......... 4
2.2.2 Dividing modules into different files .......... 7
2.2.3 Reloading external connections .......... 7
**2.3 Summary** .......... **8**

**3. INHERITANCE** .......... **9**
**3.1 Derivating a module** .......... **9**
**3.2 Derivating again** .......... **12**
**3.3 Source code and inheritance** .......... **14**
**3.4 Specifications** .......... **15**
**3.5 Summary** .......... **15**

**4. REVIEW** .......... **16**

**5. PROGRAM LISTINGS** .......... **17**
**5.1 Modularised project in a file (chapter 2.2.1)** .......... **17**
**5.2 Project split into different files (chapter 2.2.2)** .......... **19**
5.2.1 Person.cxp .......... 19
5.2.2 allperson.mod .......... 20
5.2.3 relation.mod .......... 22
5.2.4 ftree.mod .......... 23
5.2.5 controlledrel.mod .......... 24
5.2.6 Ableitung – advancedrel.mod (chapter 3.2) .......... 24
**5.3 Batch files** .......... **25**
5.3.1 tutorial2-person.bat .......... 25
5.3.2 tutorial2.bat .......... 25

Copyright © ClassiX Software GmbH
ClassiX, CyberEnterprise, InstantView and AppsWarehouse are registered trademarks by the ClassiX Software GmbH
All rights are reserved. Subject to alterations.
All remaining hard- and software names are registered/copyrighted the rights of which rest exclusively with the Originator.

# 1. Abstract

After the first tutorial, you now have an idea of how to create a simple application with InstantView®. The second tutorial guides the way to larger projects. Topic is the InstantView® module concept which forms the base for the entire AppsWarehouse®.

## 1.1 Course preparations

You will find the current tutorials with accordant program files on our homepage and on the Infothek CD respectively. You will need the corresponding zip archive *cx_Freeware.zip* (see tutorial 1) to work through this tutorial without any major problems.

You will find the main files for this tutorial in the directory **\Classix\Freeware\Projects\Tutorial2**. In addition, all modules that you will develop in this tutorial are in *Classix\Freeware\AppsWH\Freeware\Tutorial2.* The file *tutorial2.cxp.end* presents the final state of the project file without access to external module.
The project file *Person.cxp.end* however, demonstrates the work with external modules. It requires the accordant modules in the *AppsWH* directory as well as the file *tutorial.ext* in the *\Classix\Freeware\system* directory. Just by simply removing the ending ".end", we could already produce the final state (for demonstration purposes).

For demonstration purposes, we have implemented the first two tutorials in a complete program with other standard modules. You can start these via *\Classix\Freeware\Projects**Freeware.bat***. Just try!

Copyright © ClassiX Software GmbH
ClassiX, CyberEnterprise, InstantView and AppsWarehouse are registered trademarks by the ClassiX Software GmbH
All rights are reserved (1993 – 2007). Subject to alterations.
All remaining hard- and software names are registered/copyrighted the rights of which rest exclusively with the Originator.

# 2. Modularisation

Now the small example has grown so much, that it seems appropriate to ask for clarity and structure of the program code. How can we divide an application's program code into clear and small entities? InstantView®code is organized in modules. Modules are components of the highest level in the ClassiX® system. The module concept has fundamental relevance for

- the creation and maintenance of large applications
- rapid development; a module library – the AppsWarehouse® – makes sure, that it is possible to start the development with prefabricated components
- adjustments and modifications; inheritance develops new module variants – without changing the original program code

The following will show you,

- what a module is
- how to divide an application into modules

## 2.1 What is a module?

Although we haven't actually dealt with modules, our application consists of a module (named *GLOBAL*). There is no InstantView®code outside of a module! Instead of working with the automatically generated default module, we want to define the modules explicitly. For this, we use the statement *module*.

They have all characteristics of object-orientation:

- **Encapsulation:** Different modules communicate via messages which form the interface to the outside. Messages are the only bridge between different modules. A module A doesn't know variables, sub-routines or window objects of another module. To develop an application out of different modules, only the accepted messages need to be known.
- **Inheritance:** Further modules can be derivated from a module. The derivated module inherits the window resources and activities, which are defined in the basic module to complement these or to overwrite them by redefinition.
- **Polymorphy:** All inherited activities are polymorph (analog virtual functions in C++).

In the first tutorial, we suggested to organize the cooperation of windows via messages. Now we definitely have to work with messages! There is no other option to communicate between different modules. In the Basics (Tutorial 0), paragraph 2.3.3 deals with the object characteristics of InstantView® modules more detailed.

Copyright © ClassiX Software GmbH
ClassiX, CyberEnterprise, InstantView and AppsWarehouse are registered trademarks by the ClassiX Software GmbH
All rights are reserved (1993 – 2007). Subject to alterations.
All remaining hard- and software names are registered/copyrighted the rights of which rest exclusively with the Originator.

## 2.2 Modularisation of the application

To demonstrate modules, we will now take apart the program created in tutorial 1 and split the single parts into files.
This is how larger, more complex projects remain clear and well-structured.

### 2.2.1 All modules in one file

Now we want to divide the application from the previous tutorial (`tutorial1.cxp`) into modules. For this, please copy
`tutorial1.cxp` from `\Classix\Freeware\Projects\Tutorial1` to `…\Projects\Tutorial2` and rename it into `tutorial2.cxp`.

Here an overview of the future modules:

| module | function |
|---|---|
| Person | edit persons |
| AllPersons | show all persons from database |
| Relation | edit parent-child-relationships |
| FamilyTree | 3-generation presentation |
| | Tree diagram |

Now the windows can only be opened in their own module, and we have to introduce accordant messages for the activation:

| module | window | message |
|---|---|---|
| AllPersons | PersList | SHOW_ALL_PERSONS |
| Relation | Relationships | PARENT_CHILD_RELATIONSHIP |
| FamilyTree | Family | FAMILY_TREE_1 |
| | FTree | FAMILY_TREE_2 |

For style reasons, we drag the definitions for variables and sub-program statements into the accordant module action list. There is no semantic difference to the old solution (agreement in the window object action lists).

After the modifications, `tutorial2.cxp` looks like this:

```
Module(Person)
[
  Msg(SHOW_ALL_PERSONS, PARENT_CHILD_RELATIONSHIP, FAMILY_TREE_1, FAMILY_TREE_2)

  Msg(PERSON_SELECTED, PERSON_CHANGED)

  Var(person)

  Define(EnterStateI)  // Button "Save" deactivated
        Lock(,SaveBtn);

  Define(EnterStateII)  // Button "Save" deactivated &
                        // alert to detect user input
```

Copyright © ClassiX Software GmbH
ClassiX, CyberEnterprise, InstantView and AppsWarehouse are registered trademarks by the ClassiX Software GmbH
All rights are reserved (1993 – 2007). Subject to alterations.
All remaining hard- and software names are registered/copyrighted the rights of which rest exclusively with the Originator.

```
        EnterStateI Alert(StartWindow);

  Define(EnterStateIII)  // Button "Save" is active
        Unlock(,SaveBtn);
]

// Window to display and edit 1 person object
Window(StartWindow, ACCEPT_DROP, 0, 0, 600, 110, T("Personenverwaltung","personal file"))
[

  DROP: Drop SendMsg(PERSON_SELECTED)
  PERSON_SELECTED: -> person, person FillWindow, EnterStateII
]
{
  Menu
  {
    Item(T("alle Personen", "show all persons"))
    [
      SELECT: SendMsg(SHOW_ALL_PERSONS)
    ]
    Item(T("Verwandschaftsbeziehungen", "Family relationships"))
    [
      SELECT: SendMsg(PARENT_CHILD_RELATIONSHIP)
    ]
    Item(T("3 Generationen", "3 Generations"))
    [
      SELECT: SendMsg(FAMILY_TREE_1)
    ]
    Item(T("Stammbaum", "Family Tree"))
    [
      SELECT: SendMsg(FAMILY_TREE_2)
    ]
  }

 Prompt(10, 6, T("Name:"))

  .  .  .

 Button(SaveBtn, NON_SELECTABLE, 10, 70, 120, T("Speichern","Save"))
 [
  SELECT: person ifnot { CreatePersObject(CX_PERSON) -> person }
          person DrainWindow, EnterStateI
          person SendMsg(PERSON_CHANGED)
 ]
 Button(220,70,120,T("Neu","New"))
 [
  SELECT: ClearWindow, NULL -> person, EnterStateII
 ]
}


/*******************************************************************************/


Module(AllPersons)
[
  Msg(SHOW_ALL_PERSONS)

  SHOW_ALL_PERSONS: OpenWindow(PersList, 1)
]

// show all persons in the database
Window(PersList, 580, 53, 360, 79, T("Alle Personen", "All persons"))
{
  .  .  .
```

Copyright © ClassiX Software GmbH
ClassiX, CyberEnterprise, InstantView and AppsWarehouse are registered trademarks by the ClassiX Software GmbH
All rights are reserved (1993 – 2007). Subject to alterations.
All remaining hard- and software names are registered/copyrighted the rights of which rest exclusively with the Originator.

```
}

/*****************************************************************************/

Module(Relation)
[
  Msg(PARENT_CHILD_RELATIONSHIP)

  PARENT_CHILD_RELATIONSHIP: OpenWindow(Relationships, 1)
]

// window to edit parent - child relationships
Window(Relationships, 50, 135, 361, 100, T("Verwandtschaftsbeziehungen", "Family
relationships"))
{
   ObjectCombobox(Parent, 58, 5, 224, 37)
   [
    INITIALIZE: FindAll(CX_PERSON) FillObox
    SELECT:    GetObject FillWindow

   ]

   .  .  .

}

/*****************************************************************************/

Module(FamilyTree)
[
  Msg(FAMILY_TREE_1, FAMILY_TREE_2)

  Var(maleSymbol, femaleSymbol)

  Define(Symbol)
     Copy(sexEnum) if femaleSymbol else maleSymbol;

  FAMILY_TREE_1: OpenWindow(Family, 1)
  FAMILY_TREE_2: OpenWindow(FTree, 1)
]

// shows grand parents and children of a person selected
Window(Family, 412, 135, 395, 100, T("3 Generationen", "3 Generations"))
{
}

// display family relationships as a tree
Window(FTree, 10, 10, 400, 200, T("Stammbaum ", "Family Tree"))

  .  .  .
}
```

Copyright © ClassiX Software GmbH
ClassiX, CyberEnterprise, InstantView and AppsWarehouse are registered trademarks by the ClassiX Software GmbH
All rights are reserved (1993 – 2007). Subject to alterations.
All remaining hard- and software names are registered/copyrighted the rights of which rest exclusively with the Originator.

### 2.2.2 Dividing modules into different files

This was certainly only the first step. The modules don't have to be saved together in one file. They can be divided into several different files. This is the only possibility to deal with larger, more complex applications. To overlook the situation, the following plan applies to the file locations:
General modules of the Classix®system are usually stored in *\Classix\xxx\AppsWH,* whereas *xxx* stands for a project name. Modules that are required for the project are put into an accordant sub-directory: *\Classix\xxx\AppsWH\xxx*. If there are further divisions, continue according to this method.

*\Classix\Freeware\AppsWH\Freeware\xxx* applies for this tutorial (xxx stands for a tutorial directory, which contains even more specific modules that are modified due to inheritance).

The project is now supposed to get divided into the following modules. The files are stored in *\Classix\Freeware\AppsWH\Freeware*.

| module | file | remarks |
|---|---|---|
| Person | person.cxp (muss noch erstellt werden) | Starter program |
| AllPersons | allpersons.mod | |
| Relation | relation.mod | |
| FamilyTree | ftree.mod | |

### 2.2.3 Reloading external connections

We divide the InstantView® code according to this pattern. If we start `person.cxp` afterwards, the connection to the other modules is missing. The messages SHOW_ALL_PERSONS, PARENT_CHILD_RELATIONSHIP, FAMILY_TREE_1 or FAMILY_TREE_2 get sent, but there is no more recipient. A bridge to the source code is missing in the other files – and exactly this "connector" is defined with the statement *Extern*.

We include `person.cxp` into the following lines:

```
Extern(AllPersons, File(allpersons.mod), triggeredBy(SHOW_ALL_PERSONS))
Extern(Relation, File(relation.mod),triggeredBy(PARENT_CHILD_RELATION-
SHIP))
Extern(FamilyTree, File(ftree.mod), triggeredBy(FAMILY_TREE_1,
                                                   FAMILY_TREE_2))
```

The extern statement describes, where to find a module in which source file; and which message causes the reloading of the source file. After loading the module, the message gets sent to this module. The "reloading" is transparent: Therefore there is no difference in the conduction between a loaded module and a module that has not been loaded, yet.

The statement always conducts exactly as in our example at the end of the last tutorial. (`tutorial1.cxp`). But now we have gained much more clarity.

Copyright © ClassiX Software GmbH
ClassiX, CyberEnterprise, InstantView and AppsWarehouse are registered trademarks by the ClassiX Software GmbH
All rights are reserved (1993 – 2007). Subject to alterations.
All remaining hard- and software names are registered/copyrighted the rights of which rest exclusively with the Originator.

For large projects, it is common to collect all external references in one own file. Referring this to our example we have:

File `tutorial.ext`

```
Msg(SHOW_ALL_PERSONS, PARENT_CHILD_RELATIONSHIP, FAMILY_TREE_1,
FAMILY_TREE_2)

Extern(AllPersons, File(allpersons.mod), triggeredBy(SHOW_ALL_PERSONS))
Extern(Relation, File(relation.mod),triggeredBy(PARENT_CHILD_RELATION-
SHIP))
Extern(FamilyTree, File(ftree.mod), triggeredBy(FAMILY_TREE_1,
                                                 FAMILY_TREE_2))
```

and in `person.cxp` only

```
#include "tutorial.ext"

Module(Person)
[
   . . .
]
```

If there is no complete path indicated in the file clause (the extern statement) – and this is usually the case – the source file gets searched in all directories that are predetermined via environment variable CX_PATH.

## 2.3 Summary

In this chapter, we have learned about the ClassiX® module concept and divided our exemplary application in modules.

We have learned,

- how to modulise an existing application
- that encapsulation makes modules become independent from each other
- that modules can be coupled via messages only
- that applications are built with modules

Copyright © ClassiX Software GmbH
ClassiX, CyberEnterprise, InstantView and AppsWarehouse are registered trademarks by the ClassiX Software GmbH
All rights are reserved (1993 – 2007). Subject to alterations.
All remaining hard- and software names are registered/copyrighted the rights of which rest exclusively with the Originator.

# 3. Inheritance

We have seen that applications are made of modules. In the everyday life it makes sense to use prefabricated **AppsWarehouse**® modules as much as possible.
A new module can be derivated from an already existing one, at first inheriting all characteristics from its basic module. With this, ClassiX® offers an elegant way of adjusting modules to certain specifics.

> **Remark:** In this matter, we can also talk about specialisation. The derivated model is adjusted for a particular purpose while the basic module always represents a more abstract solution.

## 3.1 Derivating a module

We want to upgrade the function „edit parent-child-relationship“ in a way that doesn't allow for cyclic relationships anymore. The application is supposed to report a problem, whenever a person becomes his or her own direct descendant. This can be realised via derivattion. The module **relation** remains unchanged.
We introduce a new source program file (named `controlledrel.mod`).
At first, it only contains:

Module(ControlledRelation) : Relation

Matching to this, we change the extern statements in `tutorial.ext`:

```
Msg(SHOW_ALL_PERSONS, PARENT_CHILD_RELATIONSHIP, FAMILY_TREE_1,
FAMILY_TREE_2)

Extern(AllPersons, File(allpersons.mod), triggeredBy(SHOW_ALL_PERSONS))
Extern(Relation, File(relation.mod))
Extern(ControlledRelation, File(controlledrel.mod),
                              triggeredBy(PARENT_CHILD_RELATIONSHIP))
Extern(FamilyTree, File(ftree.mod), triggeredBy(FAMILY_TREE_1,
                                                 FAMILY_TREE_2))
```

Starting `person.cxp` now and opening the monitor window bei dem Editierfenster für die Eltern-Kind-Beziehungen, we can see the new module name **ControlledRelation**. The functionality of course hasn't changed so far.

Copyright © ClassiX Software GmbH
ClassiX, CyberEnterprise, InstantView and AppsWarehouse are registered trademarks by the ClassiX Software GmbH
All rights are reserved (1993 – 2007). Subject to alterations.
All remaining hard- and software names are registered/copyrighted the rights of which rest exclusively with the Originator.

Derivating, all this gets inherited:

- the window objects
- the InstantView® variables
- all action lists,
  meaning action list of the module and of the window objects
- the sub-program statements

We remodel our derivated module as follows:

```
Module(ControlledRelation) : Relation
[
  Var(parent, tmp)

  Declare(ReportError)

  Define(CheckRelations)
     Get(children) -> tmp
     tmp if { tmp iterate { Dup parent @ if { ReportError
                                               cancel
                                             }
                            CheckRelations
                          }
            };

   Define(ReportError)
      "Rekursion . . ." Attention;
]



Window(Relationships, 50, 135, 361, 100,
           T("Verwandtschaftsbeziehungen", "Family relationships"))
{

   Button(RelBtn, GREEN, 91, 20, 135, 8, T("hat die Kinder",
                                                        "has children"))
    [
      SELECT: SendMsg(, SUPER)
              GetObject(, Parent) -> parent, parent CheckRelations
    ]
}
```

We have changed the button action list and its appearance:
green text stands for more security while editing relationships.

The assignment to the window objects in the basic module is controlled via name equity. This is why the button appears in his parent window here in the derivated module again. Consequently, the button has to be already named in the basic module. We still have to change the line `relation.mod` accordantly:

```
       Button(RelBtn,  91,  20,  141,  T("hat  die  Kinder",  "has
children"))
```

Copyright © ClassiX Software GmbH
ClassiX, CyberEnterprise, InstantView and AppsWarehouse are registered trademarks by the ClassiX Software GmbH
All rights are reserved (1993 – 2007). Subject to alterations.
All remaining hard- and software names are registered/copyrighted the rights of which rest exclusively with the Originator.

To make the derivation of new modules easier, it make sense to clearly name all window objects.

Via derivattion, we obtain a module with elements that either

- are adopted from the basic module (inherited) or
- have been added to the new, derivated module, or
- elements, that overwrite the accordant counterpart in the basic module due to a redefinition

The button definition got overwritten to change the color, but also to call – in case of the system event SELECT the sub-program statement *CheckRelations* in the action list. We overwrite the reaction onto SELECT.

One option would have been

```
 Button(RelBtn, GREEN, 91, 20, 135, 8, T("hat die . . .
[
  SELECT: GetObject(, Parent) Copy(sexEnum)
          // set back reference according to parent gender
          if "mother" else "father" BackRefName(, children)
          GetObject(, Parent) DrainWindow
          GetObject(, Parent) -> parent, parent CheckRelations
]
```

If we want to avoid repeating the basic module statement sequence, we can trigger it with *SendMsg(, SUPER)*:

```
 Button(RelBtn, GREEN, 91, 20, 135, 8, T("hat die . . .
[
  SELECT: SendMsg(, SUPER)
          GetObject(, Parent) -> parent, parent CheckRelations
]
```

SUPER indicates: transfer the event or the message to the inherited action lists! In contradiction to the “normal” form of the statement *SendMsg*, the first parameter has to be left out. *SendMsg(, SUPER)* only refers to the just received message or to the just caused event.

If we now start `person.cxp` and try to make a person object directly or indirectly his or her own child, we receive a warning and the statement sequence is cancelled with *cancel*.

But didn’t we make it too easy for ourselves, if the family relationship gets checked only after the statement *DrainWindow* in the basic module. What is in the database after entering a recursive structure? You will get a detailed answer in paragraph 3 (transactions) in tutorial 3. Here we only observe: *cancel* also prevents, that database changes get adopted. The simple solution shown here, is correct.

Copyright © ClassiX Software GmbH
ClassiX, CyberEnterprise, InstantView and AppsWarehouse are registered trademarks by the ClassiX Software GmbH
All rights are reserved (1993 – 2007). Subject to alterations.
All remaining hard- and software names are registered/copyrighted the rights of which rest exclusively with the Originator.

## 3.2 Derivating again

The tutorial now wants to demonstrate a two-stage derivattion, as well.

We need a motivation for the next upgrading:
Just imagine the database contains so many objects that are connected via relations, that it is possible to indirectly and accidentally create a recursion. Now, it would be nice if the problem report shows, which objects (persons) form a cycle!

From the module **ControlledRelation,** we want to proceed with the module **AdvancedRelation**!
The window gets a status line, where we show the discovered cycle.
Next, the new module needs to be introduced in `tutorial.ext`:

```
 Extern(Relation, File(relation.mod))
 Extern(ControlledRelation, File(controlledrel.mod)) : Relation
 Extern(AdvancedRelation, File(advancedrel.mod),
        triggeredBy(PARENT_CHILD_RELATIONSHIP)) : ControlledRela-
tion
```

The essential upgrading is that the sub-program statement *CheckRelations* lists all objects of a just examined branch in a vector. In case of an error, the status line shows all objects currently contained by the vector. The sub-program statements *CheckRelations* and *ReportError* are inherited, which means that with each call, they cover the definition of the initial object. This also refers to calls in the basic module. We will get back to this soon.

Nevertheless, the inherited code from the basic module is not wrong. It's just incomplete. Just like in the button action list in the previous paragraph, we don't want to repeat it here. There, we could explicitly refer to the inherited action list with *SendMsg(, SUPER)*. Here, the call ***super::****CheckRelations* does the same job.

*CheckRelations* in the basic module (**ControlledRelation**) contains a recursive call. The idea of redefinition by *CheckRelations* in the module **AdvancedRelation** is based upon the collection of objects at a recursive descent in a vector. And this only works, when *CheckRelations* of the module **AdvancedRelation** is called again from *super::CheckRelations*.

Calls of a sub-program statement are polymorph.

For a better understanding – the listing with comments:

```
Module(AdvancedRelation) : ControlledRelation
[
  Var(object, vector, reportString)

  Define(CheckRelations)
     -> object, object vector Insert // keep track of object
     object super::CheckRelations
     object vector Remove  // remove object because this branch
                           // has proven to be acyclic
```



Copyright © ClassiX Software GmbH
ClassiX, CyberEnterprise, InstantView and AppsWarehouse are registered trademarks by the ClassiX Software GmbH
All rights are reserved (1993 – 2007). Subject to alterations.
All remaining hard- and software names are registered/copyrighted the rights of which rest exclusively with the Originator.

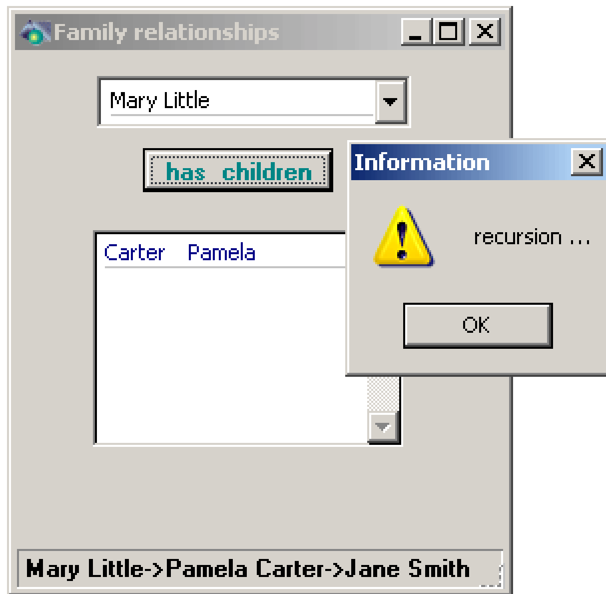
```
      ;

   Define(ReportError)
      "" -> reportString
      vector Revert  // revert sequence
      iterate { String
                Index if { // insert "->" except for the 1st one
                           "->" reportString + -> reportString
                         }
                reportString + -> reportString
              }
      reportString PutValue(, report)
      super::ReportError
      "" PutValue(, report);
]



Window(Relationships, 50, 135, 361, 123,
            T("Verwandtschaftsbeziehungen", "Family relationships"))
{

   Button(RelBtn, CYAN, 91, 20, 135, 8, T("hat die Kinder",
                                          "has  children"))
   [
     SELECT: [] -> vector, SendMsg(, SUPER)
   ]

   Statusbar
   {
     String(report, 2, 1, 500)
     Attach(report, RIGHT, STRETCH, 2)
   }
}
```

The following table shows, how the inheritance concept shows how the inheritance concept "collects" the window objects:

| application | module | | |
|---|---|---|---|
| | AdvancedRelation | ControlledRelation | Relation |
| ObjectCombobox **parent** | | | **defined** |
| button **RelBtn** | **overwritten** | **overwritten** | **defined** |
| ObjectList **children** | | | **defined** |
| statusbar + string **report** | **defined** | | |

We could make a very similar diagram for the action lists of window objects and modules. Here, every single reaction to a system event is inherited.

Copyright © ClassiX Software GmbH
ClassiX, CyberEnterprise, InstantView and AppsWarehouse are registered trademarks by the ClassiX Software GmbH
All rights are reserved (1993 – 2007). Subject to alterations.
All remaining hard- and software names are registered/copyrighted the rights of which rest exclusively with the Originator.

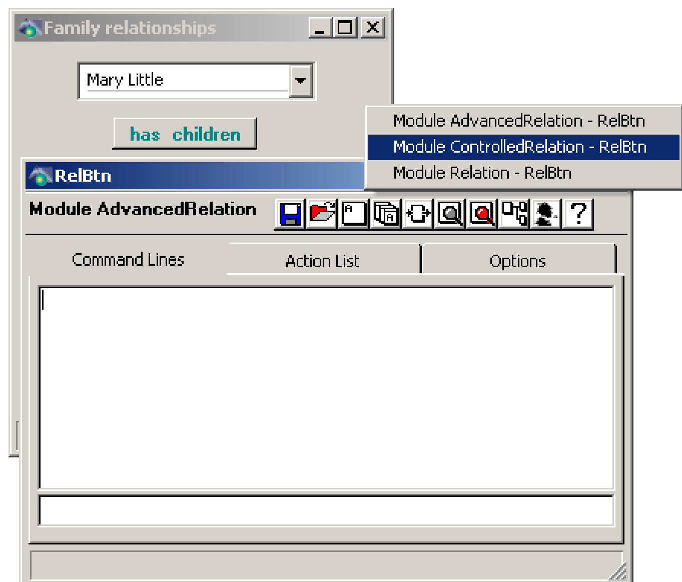



Example:

Pamela Carter cannot be assigned as a child, because Mary Little is the daughter of Jane Smith and grandchild of Pamela Carter.

## 3.3 Source code and inheritance

Inheritance plays a very important role for the AppsWarehouse®.
The collecting character causes that the windows of an InstantView® code application develop from different source files. With this, the monitor window helps in connection with the CodeWright® editor maintain the overview. On every derivattion level, it is possible to jump into the source code.




The figure shows the monitor window with the button *RelBtn* from the application example.

Copyright © ClassiX Software GmbH
ClassiX, CyberEnterprise, InstantView and AppsWarehouse are registered trademarks by the ClassiX Software GmbH
All rights are reserved (1993 – 2007). Subject to alterations.
All remaining hard- and software names are registered/copyrighted the rights of which rest exclusively with the Originator.

## 3.4 Specifications

- Please start `person.cxp` and open the monitor window to conduct some InstantView® applications:

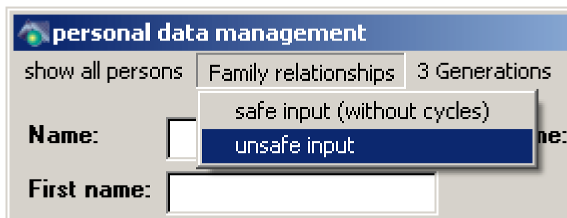
  `Module(AdvancedRelation)`

  adds the accordant object CX_MODULE to the stack and the application

  `Get(inherited)`

  causes the basic module. You can also see such objects, when the menu "family relations" has not been activated, yet – meaning the accordant modules are not loaded, yet! Please look up the reason for this in the Basics (tutorial 0) in paragraph 2.3.3.3.

- In `person.cxp` we want to modify the menu in relations to the entry of family relationships in the following way:




With the bottom menu entry, it is shall be possible again, to enter cycles as well. We can see how the tree diagram of a cyclic structure looks like!
Try to understand, why it is not possible to just call the module **Relation**[1]. Derivate a new module **UnsafeRelation** from the module **Relation**, and complete the extern statements in `tutorial.ext.`

## 3.5 Summary

In this paragraph, we have introduced the inheritance concept for InstantView® modules.

We have learned,

- how a module gets derivated from an already existing one
- that an inheriting module only needs to define the modifications
- that the inherited elements can be overwritten
- how to call the inherited InstantView® code in action lists and sub-program statements explicitly

[1] In den Grundlagen (Tutorial 0) Abschnitt 2.3.3.1 wird darauf eingegangen.

Copyright © ClassiX Software GmbH

ClassiX, CyberEnterprise, InstantView and AppsWarehouse are registered trademarks by the ClassiX Software GmbH
All rights are reserved (1993 – 2007). Subject to alterations.
All remaining hard- and software names are registered/copyrighted the rights of which rest exclusively with the Originator.

# 4. Review

In this tutorial, you have got to know the concept of modularisation. You have learned that the single modules are independent and communicate amongst each other. Additionally, some modules have been upgraded via inheritance.

At this stage, we want to point out again, that these modules have been implemented in Freeware – “small”, complete program environment with other modules - in an easy way.

In Tutorial 3 we want to introduce you to the versatile world of AppsWH. We will show you, which modules will be provided in the full version and how to implement these.

Copyright © ClassiX Software GmbH
ClassiX, CyberEnterprise, InstantView and AppsWarehouse are registered trademarks by the ClassiX Software GmbH
All rights are reserved (1993 – 2007). Subject to alterations.
All remaining hard- and software names are registered/copyrighted the rights of which rest exclusively with the Originator.

# 5. Program listings

Here you see the entire code from this tutorial. For an easier understanding, we provided it with some comment lines ( // ).
At first, the *un*modularised version and then the single splitted files.

## 5.1 Modularised project in a file (chapter 2.2.1)

```
// tutorial2.cxp - Tutorial II
// Copyright (c) 2007 ClassiX Software GmbH, Hamburg (Germany)
// All rights reserved

// $Header: //iv/Freeware/Projects/Tutorial2/tutorial2.cxp#3 $ $DateTime: 2007/09/18 10:23:04 $ $Author: Bohl $


// Window to display and edit 1 person object

Window(StartWindow, ACCEPT_DROP, 0, 0, 600, 110, T("Personenverwaltung","personal file"))
[
  Msg(PERSON_SELECTED, PERSON_CHANGED)       Var(person)

  Define(EnterStateI)  // Button "Save" deactivated
    Lock(,SaveBtn)
  ;

  Define(EnterStateII)  // Button "Save" deactivated &
                        // alert to detect user input
          EnterStateI Alert(StartWindow)
        ;

  Define(EnterStateIII)  // Button "Save" is active
          Unlock(,SaveBtn)
        ;

  DROP: Drop SendMsg(PERSON_SELECTED)
  PERSON_SELECTED: -> person, person FillWindow, EnterStateII
]
{
  Menu
  {
    Item(T("alle Personen", "show all persons"))
    [
      SELECT: OpenWindow(PersList) // opens the 2nd window
    ]
    Item(T("Verwandtschaftsbeziehungen", "Family relationships"))
    [
      SELECT: OpenWindow(Relationships)
    ]
    Item(T("3 Generationen", "3 Generations"))
    [
      SELECT: OpenWindow(Family)
    ]
    Item(T("Stammbaum", "Family Tree"))
    [
      SELECT: OpenWindow(FTree)
    ]
  }

  Prompt(10, 6, T("Name:"))
  String(CX_PERSON::name, TOOLTIP("Name der Person", "Name of the Person"), 110, 6, 191)
  [
    ALTERED: EnterStateIII
  ]

  Prompt(10,17,T("Vorname:", "First name:"))
  String(CX_PERSON::firstName, 110, 17, 191)
  [
    ALTERED: EnterStateIII
  ]

  Prompt(10, 30, T("Geburtstag:", "Birthday:"))
  Date(CX_PERSON::dateOfBirth, 110, 30, 120)
  [
    ALTERED: EnterStateIII
    Var(tmp) // Hilfsvariable
    NON_CURRENT:
    SELECT: CreateTransObject(CX_PERSON) -> tmp
             tmp DrainWindow, tmp Call(Age) PutValue(, age)
  ]

  Prompt(10, 43, T("Alter:", "Age:"))
  String(CX_PERSON::Age()~age, NO_DRAIN, 110, 43, 191)

  Prompt(310, 6, T("Rufname:","Nickname:"))
  String(CX_PERSON::nickName, 410, 6, 150)
  [
    ALTERED: EnterStateIII
  ]

  Enumeration(CX_PERSON::sexEnum, 405, 30, 100, 35)
```



Copyright © ClassiX Software GmbH
ClassiX, CyberEnterprise, InstantView and AppsWarehouse are registered trademarks by the ClassiX Software GmbH
All rights are reserved (1993 – 2007). Subject to alterations.
All remaining hard- and software names are registered/copyrighted the rights of which rest exclusively with the Originator.

```
  [
    ALTERED: EnterStateIII
  ]

  Button(SaveBtn, NON_SELECTABLE, 10, 70, 120, T("Speichern","Save"))
  [
    SELECT: person ifnot { CreatePersObject(CX_PERSON) -> person }
                   person DrainWindow, EnterStateI
                   person SendMsg(PERSON_CHANGED)
  ]
  Button(220,70,120,T("Neu","New"))
  [
    SELECT: ClearWindow, NULL -> person, EnterStateII
  ]
}

// show all persons in the database
Window(PersList, 580, 53, 360, 79, T("Alle Personen", "All persons"))
{
  Header(ListBoxHeader, HIDDEN, 0, 0, 0, 0, ListBox)
  {
    Prompt(name, 0, 0, T("Name", "Name"))
    Prompt(firstName, 0, 0, T("Vorname", "First name"))
    Prompt(dateOfBirth, 0, 0, T("Geburtstag", "Birthday"))
  }
  ObjectListView(ListBox, AUTO_POSITION, SELECT_MULTIPLE, 10, 5, 328, 40)
  [
    INITIALIZE: "CX_PERSON::name" SetFormat
                "CX_PERSON::firstName" SetFormat
                "CX_PERSON::dateOfBirth" SetFormat
                 FindAll(CX_PERSON) FillObox
    SELECT: GetObject SendMsg(PERSON_SELECTED)
    PERSON_CHANGED: UpdateObox
  ]
  Attach(ListBox, RIGHT, STRETCH, 10)
  Attach(ListBox, BOTTOM, STRETCH, 5)

}

// window to edit parent - child relationships
Window(Relationships, 50, 135, 361, 100, T("Verwandtschaftsbeziehungen", "Family relationships"))
{
  ObjectCombobox(Parent, 58, 5, 224, 37)
  [
   INITIALIZE: FindAll(CX_PERSON) FillObox
   SELECT:    GetObject FillWindow
  ]

  Button(91, 20, 141, T("hat die Kinder", "has  children"))
  [
    SELECT: GetObject(, Parent) Copy(sexEnum)
            // set back reference according to parent gender
            if "mother" else "father" BackRefName(, children)
            GetObject(, Parent) DrainWindow
  ]

  ObjectList(CX_PERSON::children, ENTIRE, ACCEPT_DROP, AUTO_POSITION, 55, 37, 221, 45)
  [
    INITIALIZE: [ "CX_PERSON::name" COLOR BLUE ] SetFormat
                [ "CX_PERSON::firstName" COLOR BLUE ] SetFormat
    DROP:          FillObox
    DELETE:        OboxDel
  ]
}

// shows grand parents and children of the person selected
Window(Family, 412, 135, 395, 100, T("3 Generationen", "3 Generations"))
{
  Prompt(20, 5, T("Vater:", "Father:"))
  String(CX_PERSON::father, 20, 13, 158)

  Prompt(220, 5, T("Mutter:", "Mother:"))
  String(CX_PERSON::mother, 220, 13, 157)

  Prompt(20, 28, T("Person:"))
  ObjectCombobox(Person, 112, 28, 175, 37)
  [
    INITIALIZE: FindAll(CX_PERSON) FillObox
    SELECT: GetObject FillWindow
  ]

  Prompt(20, 43, T("Kinder:", "Children:"))
  ObjectList(CX_PERSON::children, ENTIRE, 112, 43, 174, 37)
}


// display family relationships as a tree
Window(FTree, 10, 10, 400, 200, T("Stammbaum ", "Family Tree"))
[
  Var(maleSymbol, femaleSymbol)

  Define(Symbol)
    Copy(sexEnum) if femaleSymbol else maleSymbol;


  INITIALIZE: CreateTransObject(CX_BITMAP) -> maleSymbol
              "CX_ROOTDIR\\BMP\\male.bmp" maleSymbol Put
              CreateTransObject(CX_BITMAP) -> femaleSymbol
              "CX_ROOTDIR\\BMP\\female.bmp" femaleSymbol Put
```

Copyright © ClassiX Software GmbH

ClassiX, CyberEnterprise, InstantView and AppsWarehouse are registered trademarks by the ClassiX Software GmbH
All rights are reserved (1993 – 2007). Subject to alterations.
All remaining hard- and software names are registered/copyrighted the rights of which rest exclusively with the Originator.

```
]
{
  ObjectCombobox(Person, 5, 5, 380, 37)
  [
    INITIALIZE: FindAll(CX_PERSON) FillObox
    SELECT: GetObject FillObox(, tree)
  ]

  ObjectTree(tree, 5, 15, 380, 170)
  [
    INITIALIZE: "CX_PERSON::call(Symbol)" SetFormat
                "CX_PERSON::name" SetFormat
                "CX_PERSON::firstName" SetFormat
                [ "CX_PERSON::children" NODE ] SetFormat
  ]
}
```

## 5.2 Project split into different files (chapter 2.2.2)

Here you see the entire code from this tutorial. For an easier understanding, we provided it with some comment lines ( // ).

### 5.2.1 Person.cxp

```
// person.cxp - Tutorial II
// Copyright (c) 2007 ClassiX Software GmbH, Hamburg (Germany)
// All rights reserved

// $Header: //iv/Freeware/Projects/Tutorial2/Person.cxp#3 $ $DateTime: 2007/09/18 10:23:04 $ $Author: Bohl $


Module(PersonalFile)
[
  Msg(SHOW_ALL_PERSONS, PARENT_CHILD_RELATIONSHIP, FAMILY_TREE_1, FAMILY_TREE_2)

  Msg(PERSON_SELECTED, PERSON_CHANGED)       Var(person)

  Define(EnterStateI)  // Button "Save" deactivated
        Lock(,SaveBtn);

  Define(EnterStateII)  // Button "Save" deactivated &
                        // alert to detect user input
        EnterStateI Alert(StartWindow);

  Define(EnterStateIII)  // Button "Save" is active
        Unlock(,SaveBtn);
]
// Window to display and edit 1 person object

Window(StartWindow, ACCEPT_DROP, 0, 0, 600, 110, T("Personenverwaltung","personal file"))
[
  DROP: Drop SendMsg(PERSON_SELECTED)
  PERSON_SELECTED: -> person, person FillWindow, EnterStateII
]
{
  Menu
  {
    Item(T("alle Personen", "show all persons"))
    [
      SELECT: SendMsg(SHOW_ALL_PERSONS)
    ]
    Item(T("Verwandtschaftsbeziehungen", "Family relationships"))
    [
      SELECT: SendMsg(PARENT_CHILD_RELATIONSHIP)
    ]
    Item(T("3 Generationen", "3 Generations"))
    [
      SELECT: SendMsg(FAMILY_TREE_1)
    ]
    Item(T("Stammbaum", "Family Tree"))
    [
      SELECT: SendMsg(FAMILY_TREE_2)
    ]
  }

  Prompt(10, 6, T("Name:"))
  String(CX_PERSON::name, TOOLTIP("Name der Person", "Name of the Person"), 110, 6, 191)
```

Copyright © ClassiX Software GmbH

ClassiX, CyberEnterprise, InstantView and AppsWarehouse are registered trademarks by the ClassiX Software GmbH
All rights are reserved (1993 – 2007). Subject to alterations.
All remaining hard- and software names are registered/copyrighted the rights of which rest exclusively with the Originator.

```
  [
    ALTERED: EnterStateIII
  ]

  Prompt(10,17,T("Vorname:", "First name:"))
  String(CX_PERSON::firstName, 110, 17, 191)
  [
    ALTERED: EnterStateIII
  ]

  Prompt(10, 30, T("Geburtstag:", "Birthday:"))
  Date(CX_PERSON::dateOfBirth, 110, 30, 120)
  [
    ALTERED: EnterStateIII
    Var(tmp) // Hilfsvariable
    NON_CURRENT:
    SELECT: CreateTransObject(CX_PERSON) -> tmp
            tmp DrainWindow, tmp Call(Age) PutValue(, age)
  ]

  Prompt(10, 43, T("Alter:", "Age:"))
  String(CX_PERSON::Age()~age, NO_DRAIN, 110, 43, 191)

  Prompt(310, 6, T("Rufname:","Nickname:"))
  String(CX_PERSON::nickName, 410, 6, 150)
  [
    ALTERED: EnterStateIII
  ]

  Enumeration(CX_PERSON::sexEnum, 405, 30, 100, 35)
  [
    ALTERED: EnterStateIII
  ]

  Button(SaveBtn, NON_SELECTABLE, 10, 70, 120, T("Speichern","Save"))
  [
    SELECT: person ifnot { CreatePersObject(CX_PERSON) -> person }
            person DrainWindow, EnterStateI
            person SendMsg(PERSON_CHANGED)
  ]
  Button(220,70,120,T("Neu","New"))
  [
    SELECT: ClearWindow, NULL -> person, EnterStateII
  ]
}

#include "tutorial.ext"
```

## 5.2.2 allperson.mod

```
// allpersons.mod - Tutorial II
// Copyright (c) 2007 ClassiX Software GmbH, Hamburg (Germany)
// All rights reserved

// $Header: //iv/Freeware/AppsWh/Freeware/allpersons.mod#3 $ $DateTime: 2007/09/18 10:23:04 $ $Author: Bohl $


Module(AllPersons)
[
  Msg(SHOW_ALL_PERSONS)

  SHOW_ALL_PERSONS: OpenWindow(PersList, 1)
]


// show all persons in the database
Window(PersList, 580, 53, 360, 79, T("Alle Personen", "All persons"))
{
  Header(ListBoxHeader, HIDDEN, 0, 0, 0, 0, ListBox)
  {
    Prompt(name, 0, 0, T("Name", "Name"))
    Prompt(firstName, 0, 0, T("Vorname", "First name"))
    Prompt(nickName, 0, 0, T("Spitzname", "Nickname"))
    Prompt(dateOfBirth, 0, 0, T("Geburtstag", "Birthday"))
  }
  ObjectListView(ListBox, AUTO_POSITION, SELECT_MULTIPLE, 10, 5, 328, 40)
  [
    INITIALIZE: [ "CX_PERSON::name" COLOR LIGHTBLUE ] SetFormat
                [ "CX_PERSON::firstName" COLOR LIGHTRED ] SetFormat
                [ "CX_PERSON::nickName" COLOR CYAN ] SetFormat
                [ "CX_PERSON::dateOfBirth" COLOR GREEN ] SetFormat
                FindAll(CX_PERSON) FillObox
    DOUBLE_CLICK: GetObject SendMsg(PERSON_SELECTED)
    PERSON_CHANGED: UpdateObox
  ]
  Attach(ListBox, RIGHT, STRETCH, 10)
```

Copyright © ClassiX Software GmbH

ClassiX, CyberEnterprise, InstantView and AppsWarehouse are registered trademarks by the ClassiX Software GmbH
All rights are reserved (1993 – 2007). Subject to alterations.
All remaining hard- and software names are registered/copyrighted the rights of which rest exclusively with the Originator.

```
  Attach(ListBox, BOTTOM, STRETCH, 5)
}
```

Copyright © ClassiX Software GmbH
ClassiX, CyberEnterprise, InstantView and AppsWarehouse are registered trademarks by the ClassiX Software GmbH
All rights are reserved (1993 – 2007). Subject to alterations.
All remaining hard- and software names are registered/copyrighted the rights of which rest exclusively with the Originator.

### 5.2.3 relation.mod

```
// relation.mod - Tutorial II
// Copyright (c) 2007 ClassiX Software GmbH, Hamburg (Germany)
// All rights reserved

// $Header: //iv/Freeware/AppsWh/Freeware/Tutorial2/relation.mod#3 $ $DateTime: 2007/09/18 10:23:04 $ $Author: Bohl $


Module(Relation)
[
  Msg(PARENT_CHILD_RELATIONSHIP)

  PARENT_CHILD_RELATIONSHIP: OpenWindow(Relationships, 1)
]


// window to edit parent - child relationships
Window(Relationships, 50, 135, 361, 100, T("Verwandtschaftsbeziehungen", "Family relationships"))
{
  ObjectCombobox(Parent, 58, 5, 224, 37)
  [
    INITIALIZE: FindAll(CX_PERSON) FillObox
    SELECT:     GetObject FillWindow
  ]

  // button must be named to allow for overwriting in a derived module
  Button(RelBtn, 91, 20, 141, T("hat die Kinder", "has  children"))
  [
    SELECT: GetObject(, Parent) Copy(sexEnum)
            // set back reference according to parent gender
            if "mother" else "father" BackRefName(, children)
            GetObject(, Parent) DrainWindow
  ]

  ObjectList(CX_PERSON::children, ENTIRE, ACCEPT_DROP, AUTO_POSITION, 55, 37, 221, 45)
  [
    INITIALIZE: [ "CX_PERSON::name" COLOR BLUE ] SetFormat
                [ "CX_PERSON::firstName" COLOR BLUE ] SetFormat
    DROP:         FillObox
    DELETE:       OboxDel
  ]
}
```

Copyright © ClassiX Software GmbH
ClassiX, CyberEnterprise, InstantView and AppsWarehouse are registered trademarks by the ClassiX Software GmbH
All rights are reserved (1993 – 2007). Subject to alterations.
All remaining hard- and software names are registered/copyrighted the rights of which rest exclusively with the Originator.

## 5.2.4 ftree.mod

```
// ftree.mod - Tutorial II
// Copyright (c) 2002 ClassiX Software GmbH, Hamburg (Germany)
// All rights reserved


Module(FamilyTree)
[
  Msg(FAMILY_TREE_1, FAMILY_TREE_2)


  Var(maleSymbol, femaleSymbol)


  Define(Symbol)
     Copy(sexEnum) if femaleSymbol else maleSymbol;


  FAMILY_TREE_1: OpenWindow(Family, 1)
  FAMILY_TREE_2: OpenWindow(FTree, 1)
]


// shows grand parents and children of a person selected
Window(Family, 412, 135, 395, 100, T("3 Generationen", "3 Generations"))
{
   Prompt(20, 5, T("Vater:", "Father:"))
   String(CX_PERSON::father, 20, 13, 158)


   Prompt(220, 5, T("Mutter:", "Mother:"))
   String(CX_PERSON::mother, 220, 13, 157)


   Prompt(20, 28, T("Person:"))
   ObjectCombobox(Person, 112, 28, 175, 37)
   [
     INITIALIZE: FindAll(CX_PERSON) FillObox
     SELECT: GetObject FillWindow
   ]


   Prompt(20, 43, T("Kinder:", "Children:"))
   ObjectList(CX_PERSON::children, ENTIRE, 112, 43, 174, 37)
}



// display famly relationships as a tree
Window(FTree, 10, 10, 400, 200, T("Stammbaum ", "Family Tree"))
[
  INITIALIZE: CreateTransObject(CX_BITMAP) -> maleSymbol
              "CX_ROOTDIR\\BMP\\male.bmp" maleSymbol Put
              CreateTransObject(CX_BITMAP) -> femaleSymbol
              "CX_ROOTDIR\\BMP\\female.bmp" femaleSymbol Put


]
{
   ObjectCombobox(Person, 5, 5, 380, 37)
   [
     INITIALIZE: FindAll(CX_PERSON) FillObox
     SELECT: GetObject FillObox(, tree)
   ]


   ObjectTree(tree, 5, 15, 380, 170)
   [
     INITIALIZE: "CX_PERSON::call(Symbol)" SetFormat
                    "CX_PERSON::name" SetFormat
                 "CX_PERSON::firstName" SetFormat
                 [ "CX_PERSON::children" NODE ] SetFormat
   ]
}
```

Copyright © ClassiX Software GmbH
ClassiX, CyberEnterprise, InstantView and AppsWarehouse are registered trademarks by the ClassiX Software GmbH
All rights are reserved (1993 – 2007). Subject to alterations.
All remaining hard- and software names are registered/copyrighted the rights of which rest exclusively with the Originator.

### 5.2.5 controlledrel.mod

```
// controlledrel.mod - Tutorial II
// Copyright (c) 2002 ClassiX Software GmbH, Hamburg (Germany)
// All rights reserved



Module(ControlledRelation) : Relation
[
  Var(parent, tmp)


  Declare(ReportError)


  Define(CheckRelations)
     Get(children) -> tmp
     tmp if { tmp iterate { Dup parent @ if { ReportError cancel }
                             CheckRelations
                           }
            };


  Define(ReportError)
     T("Rekursion ...", "recursion ...") Attention;
]




Window(Relationships, 50, 135, 361, 100, T("Verwandtschaftsbeziehungen", "Family relationships"))
{


   Button(RelBtn, GREEN, 91, 20, 135, 8, T("hat die Kinder", "has  children"))
   [
     SELECT: SendMsg(, SUPER)
             GetObject(, Parent) -> parent, parent CheckRelations
   ]



}
```

### 5.2.6 Ableitung – advancedrel.mod (chapter 3.2)

```
// advancedrel.mod - Tutorial II
// Copyright (c) 2002 ClassiX Software GmbH, Hamburg (Germany)
// All rights reserved



Module(AdvancedRelation) : ControlledRelation
[
  Var(object, vector, reportString)


  Define(CheckRelations)
     -> object, object vector Insert
     object super::CheckRelations
     object vector Remove;


  Define(ReportError)
     "" -> reportString
     vector iterate { String Index if { "->" reportString + -> reportString }
                      reportString + -> reportString
                    }
     reportString PutValue(, report)
     super::ReportError
     "" PutValue(, report);
]
```

Copyright © ClassiX Software GmbH
ClassiX, CyberEnterprise, InstantView and AppsWarehouse are registered trademarks by the ClassiX Software GmbH
All rights are reserved (1993 – 2007). Subject to alterations.
All remaining hard- and software names are registered/copyrighted the rights of which rest exclusively with the Originator.

```
Window(Relationships, 50, 135, 361, 123, T("Verwandtschaftsbeziehungen", "Family relationships"))
{

   Button(RelBtn, CYAN, 91, 20, 135, 8, T("hat die Kinder", "has  children"))
   [
     SELECT: [] -> vector, SendMsg(, SUPER)
   ]

   Statusbar
   {
     String(report, 2, 1, 500)
     Attach(report, RIGHT, STRETCH, 2)
   }
}
```

# 5.3 Batch files

## 5.3.1 tutorial2-person.bat

```
REM tutorial2-person.bat - Tutorial II Start file for person.cxp
REM Copyright (c) 2007 ClassiX Software GmbH, Hamburg (Germany)
REM All rights reserved

@echo off
SET CX_ROOTDIR=\Classix\Freeware

SET CX_PATH=%CX_ROOTDIR%\AppsWH\Freeware\Tutorial2;%CX_ROOTDIR%\AppsWH\Freeware;%CX_ROOTDIR%\AppsWH
SET CX_PATH=%CX_PATH%;%CX_ROOTDIR%\System\Freeware;%CX_ROOTDIR%\System
SET CX_PATH=%CX_PATH%;%CX_ROOTDIR%\Projects\tutorial2

SET CX_SYSTEM=%CX_ROOTDIR%\System\Freeware;%CX_ROOTDIR%\System

start %CX_ROOTDIR%\Bin\cx_npr.exe person.cxp -I tutorial.ini
```

## 5.3.2 tutorial2.bat

```
REM tutorial2.bat - Tutorial II start file for tutorial2.cxp
REM Copyright (c) 2007 ClassiX Software GmbH, Hamburg (Germany)
REM All rights reserved

@echo off
SET CX_ROOTDIR=\Classix\Freeware

SET CX_SYSTEM=%CX_ROOTDIR%\System\Freeware;%CX_ROOTDIR%\System
SET CX_PATH=%CX_ROOTDIR%\AppsWH\Freeware\Tutorial2;%CX_ROOTDIR%\AppsWH\Freeware;%CX_ROOTDIR%\AppsWH
SET CX_PATH=%CX_PATH%;%CX_ROOTDIR%\Projects\tutorial2

start %CX_ROOTDIR%\Bin\cx_npr.exe tutorial2.cxp -I tutorial.ini
```

Copyright © ClassiX Software GmbH
ClassiX, CyberEnterprise, InstantView and AppsWarehouse are registered trademarks by the ClassiX Software GmbH
All rights are reserved (1993 – 2007). Subject to alterations.
All remaining hard- and software names are registered/copyrighted the rights of which rest exclusively with the Originator.