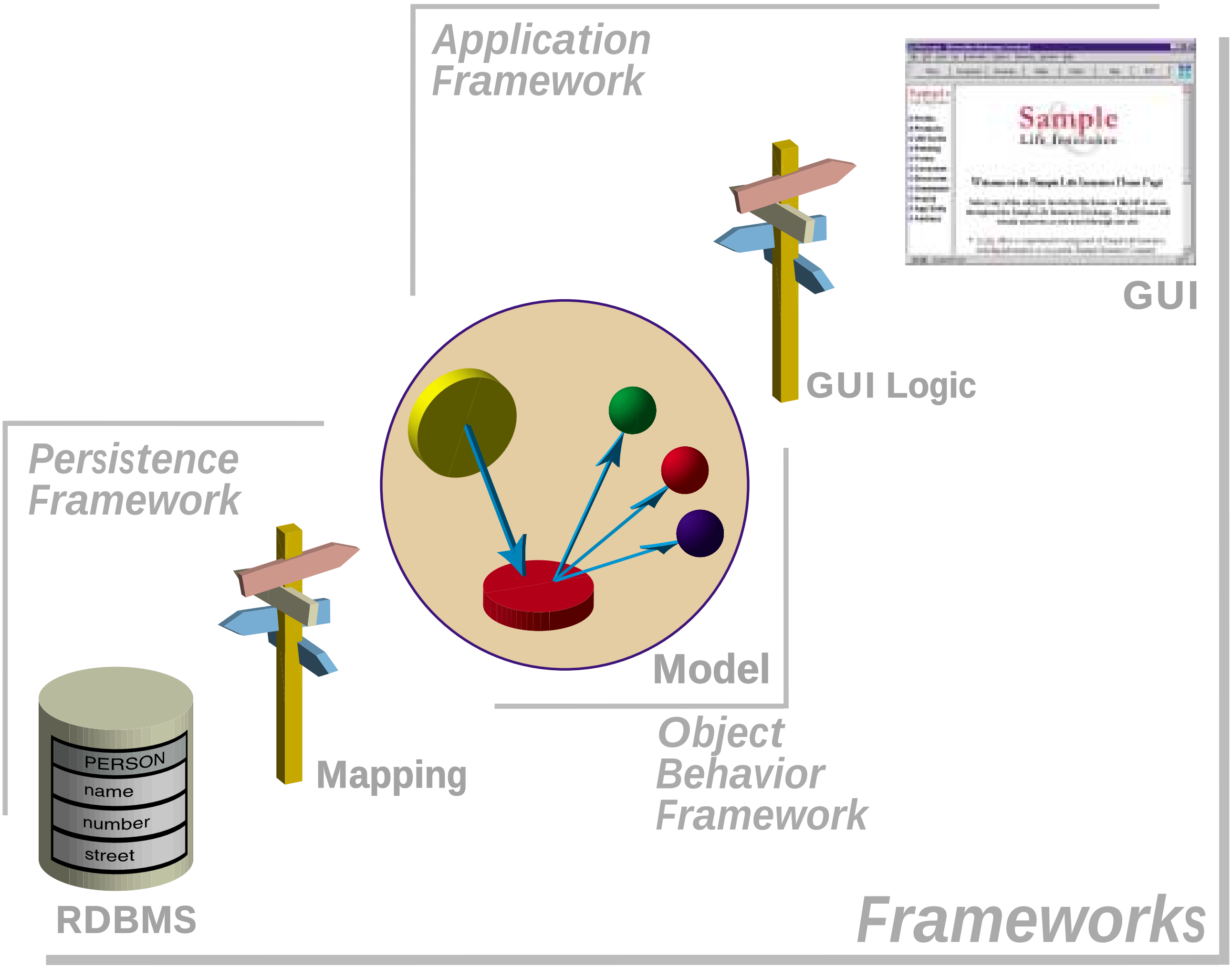
Frameworks

Getting Started

For Frameworks Version 5.0



Copyright and trademarks

Copyright

Mynd Frameworks V5.0.

Frameworks Getting Started Manual, July 2000

For more information about Mynd Frameworks, please contact:

Mynd SoftwareConsult GmbH
Taubenholzweg 1
D-51105 Köln (Cologne, Germany)

Tel. :	(+49) (0) 221 - 8029 - 0
FAX :	(+49) (0) 221 - 8029 - 999
Email:	info@mynd.de
Web:	www.mynd.com

Trademarks

Table of Contents

Appendix A

Documentation Overview

Congratulations

Congratulations on your purchase of Mynd Frameworks. Your purchase reflects your commitment to high-quality, cost-effective and rapid object-oriented software development. Extensive experience in software development for the banking and insurance industries has enabled PMS Micado to develop a suite of tools that can vastly shorten product development cycles. The documentation set from Mynd is designed to help you exploit the full potential of these tools in the shortest time possible.

Frameworks Getting Started (this manual)

Intended audience

The intended audience for this manual includes anyone who wishes to use Mynd Frameworks to raise the productivity and quality of software development with Visual Age for Smalltalk. This manual provides a very detailed step-by-step introduction that allows someone with even minimal Smalltalk, Visual Age or database experience to quickly master the basic capabilities of Mynd Frameworks.

Sections

This manual contains the following sections:

- **‘ Copyright and trademarks’ (page 3)**.
- **‘ Table of Contents’ (page 5)**.
- **‘ Documentation Overview’ (page 11)** (this section).
- **‘ Tutorial’ (page 17).** Provides a set of programming examples that demonstrate the major programming concepts and techniques of Frameworks.
- **‘ Glossary’ (page 121).** Includes any special terms used throughout this manual.
- **‘ List Of Figures’ (page 129)**.
- **‘ Index’ (page 133)**.

Recommendations for completing the step-by-step tutorial

For those with minimal IBM Visual Age Smalltalk or IBM DB2 experience

IBM Visual Age and DB2 for Smalltalk must be installed on your computer. Then this manual will guide you step-by-step in installing and using Mynd Frameworks. After completing the tutorial you will have a good overview of the basic capabilities of Frameworks.

For those with IBM Visual Age Smalltalk and IBM DB2 experience, but with no Frameworks experience

You will be able to quickly complete the step-by-step tutorial. When introducing a Frameworks concept, then manual references the following Frameworks manuals for more detailed information about Frameworks capabilities:

- Application Framework User’s Guide
- Persistence Framework User’s Guide
- Object Behavior Framework User’s Guide

For those with some previous Frameworks experience

You might want to glance at ‘1. Tutorial overview’ (page 19), which provides an overview of the content of the tutorial. And then go directly to those sections that cover topics that are new to you. However, note that each tutorial chapter assumes that the previous chapter was completed. Therefore, not having completed a chapter, you may have to do some extra work before completing the next chapter.

For those with advanced Frameworks experience

This Getting Started manual is intended as an introduction to Frameworks. Therefore, the tutorial may not be for you. However, your comments or requests for the content of the tutorial are appreciated (comments can be sent to **info.micado@notes.compuserve.com**).

Conventions used in this manual

The following conventions are used in this manual.

- A term being used for the first time is bold and italic. For example: ***Object Network***.
- Dialog names and menu entries are displayed in bold, with a forward slash between nested entries. For example: **System Transcript / Tools / Manage Applications**.

- Smalltalk code is fixed-width Courier. For example:
  `^self`

Reader comments

Any comments you have concerning this manual can be sent to:

info@mynd.de

Other manuals

From Mynd (and referenced throughout this manual):

- Application Framework User's Guide.
- Persistence Framework User's Guide.
- Object Behavior Framework User's Guide.

From IBM:

- IBM Visual Age for Smalltalk manuals.

Training from Mynd

This manual is provides complete information to help you start using Frameworks as soon as possible. However, it is also recommended to consider enrolling in a training course from Mynd that is customized to your specific needs. Mynd provides a wide-range of courses covering Smalltalk programming, object-oriented analysis, design, methodolgy, and project management.

For more information about training courses, please contact:

Mynd SoftwareConsult GmbH
Taubenholzweg 1
D-51105 Köln (Cologne, Germany)

Tel. :	(+49) (0) 221 - 8029 - 0
FAX :	(+49) (0) 221 - 8029 - 999
Email:	info@mynd.de
Web:	www.mynd.com

Installing Frameworks

System requirements

The tutorial assumes that you have the following system installation:

- Windows **NT**.
- IBM Visual for Smalltalk **Version 4.5**.
- Mynd Frameworks (Application Frameworks, Persistence Frameworks, Object Behavior Frameworks) **V5.0**.
- IBM DB2 Universal Database for Windows NT **Version 5**.

NOTE: Frameworks supports several different databases; however, the examples in the persistence chapters in this tutorial use IBM DB2 Version 5. If you dont have DB2, you can download a 60-day trial version of **DB2 Version 5.2 Single User for Windows NT** from IBM for free at **www.ibm.com**. If you decide to use a different database, the examples in the tutorial for DB2 demonstrate what tasks need to be accomplished.

Installation Overview

IBM Visual Age Smalltalk (version 4.5 recommended) must be already installed on your computer before installing Frameworks.

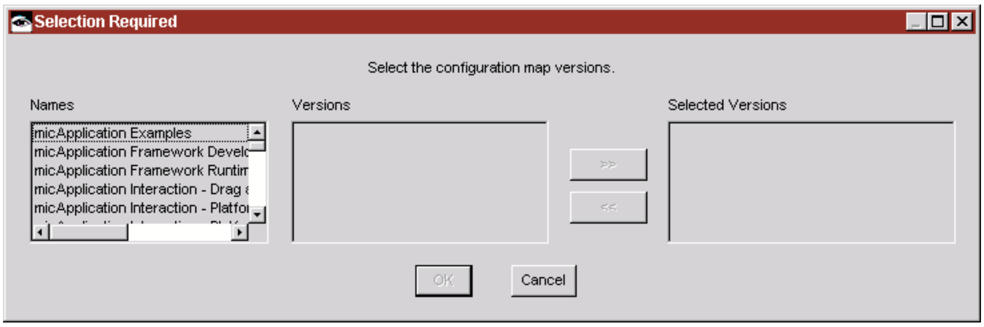
Installing Frameworks consists of 2 main steps:

- Importing the ***configuration maps*** from the ***library*** file on the CD ROM (\manager\V50.dat) to the library file on your computer (typically \vast\mgr50.dat).
- Loading the applications in the imported configuration maps (along with any other required maps) to your image file (typically \vast\abt.icx).

Importing configuration maps

Frameworks is installed in your Smalltalk environment by importing the required Frameworks applications using the Visual Age Configuration Maps Browser. The applications are imported by importing the configuration maps that contain the Frameworks applications (applications contain the classes and methods).

1. If not started: Start Visual Age for Smalltalk.
2. In the System Transcript: Select **Tools / Browse Configuration Maps**. The Configuration Maps Browser opens.
3. In the Configuration Maps Browser: Select **Names / Import** ("names" refers to the names of configuration maps).
4. In the dialog "Enter the full path name of the library" ("library" refers to the .dat file that contain the configuration maps): Double-click on the Frameworks library file **\Manager\V50.dat** (on the CD ROM). The "Selection Required" dialog appears.



Selection Required

Select the configuration map versions.

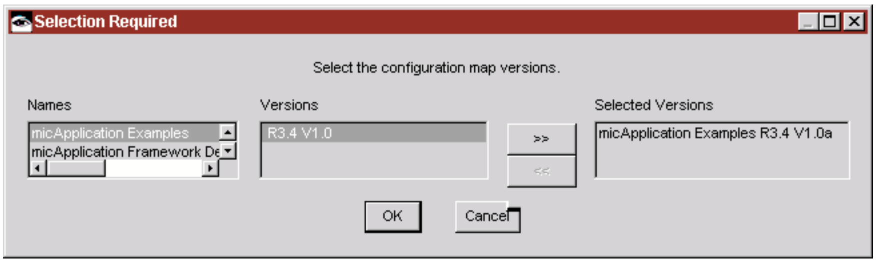
Names
micApplication Examples
micApplication Framework Develc
micApplication Framework Runtir
micApplication Interaction - Drag &
micApplication Interaction - Platfoı
Versions
>>
<<
Selected Versions
OK
Cancel

Figure 1. Dialog for selecting configuration map versions

Selecting the first configuration map for importing

5. In the **Names** box: Click on **micApplicationExamples** (a configuration map). The available versions appear in the "Versions" box.
6. In the **Versions** box: Click on **V5.0**.
7. Click on **>>**. micApplicationExamples V5.0 has been selected for importing and appears in the **Selec-**

ted Versions box.



Selection Required

Select the configuration map versions.

Names: micApplication Examples; micApplication Framework De

Versions: R3.4 V1.0

Selected Versions: micApplication Examples R3.4 V1.0a

Figure 2. micApplicationExamples R3.4 V1.0a in dialog for selecting configuration map versions

Selecting the remaining configuration maps for importing

8. Repeat the above procedure (selecting a version) for each entry in the **Names** column. Note: If a version "a" exists, then select only that version).

Importing the remaining configuration maps

9. Click **OK**:

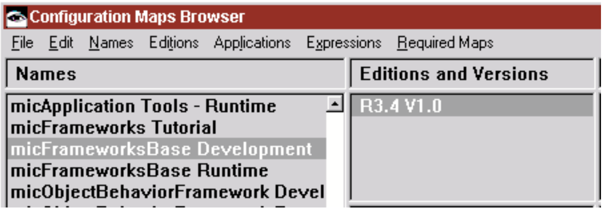
The selected configuration maps and their applications are imported to your library.The message "Finished importing from \vast\V50.dat" appears in the System Transcript. The configuraton maps are displayed and highlighted in the "Names" box of the Configuration Maps Browser.

Loading application maps (with required maps)

When a configuration map is loaded, all of its required configuration maps are also loaded. Therefore loading the configuration maps in the order described below will minimize the required effort to load all configuration maps.

Load micFrameworksBaseDevelopment

10. In the **Configuration Maps Browser**: Select **micFrameworksBaseDevelopment**. In the **Editions and Versions** box: The edition and version of the configuration map appear.



Configuration Maps Browser

Names:
micApplication Tools - Runtime
micFrameworks Tutorial
micFrameworksBase Development
micFrameworksBase Runtime
micObjectBehaviorFramework Devel

Editions and Versions: R3.4 V1.0

Figure 3. micFrameworksBaseDevelopment in configuration maps browser

11. Right click on the latest edition and version.
12. Select **Load with required maps**. The applications of the selected configuration map (and its required maps) are loaded (detailed information appears in the System Transcript). Note that an asterisk ("*") marks each listed application in the **Applications** box.

Load remaining configuration maps

13. Load the remaing configuration maps:
    - 13.1. **micObjectBehaviorFramework Development**. Applications that support Object Behavior functionality.
    - 13.2. **micApplicationFramework Development**.
    - 13.3. **micApplicationInteraction - Drag and Drop Runtime**.
    - 13.4. **micApplicationInteraction - Platform GUI Enhancements Development**. Required for support of visual programming in the Composition Editor.
    - 13.5. **micApplicationService - Authorization Runtime**. Required for authorization support.
    - 13.6. **micApplicationService - Validation Runtime**. Required for object validation.
    - 13.7. **micApplicationService - Multi Language Development**. Required for multi-language support.
    - 13.8. **micPersistenceOdbc Runtime**. Required for run-time support of ODBC.
    - 13.9. **micPersistenceOdbc Development**. Required for ODBC development tools (such as the POM Generator, etc.).
14. Select **File / Save Image**.
15. Close the Configuration Maps Browser.

Frameworks has now been loaded into your IBM Visual Age Smalltalk environment.

Frameworks tools available from micFrameworks menu

In the System Transcript the menu bar selection "micFrameworks" appears. Click on micFrameworks to display the submenu of Frameworks tools:

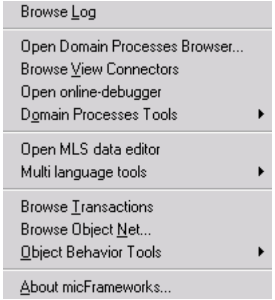


Figure 4. micFrameworks menu.

Framework Logger Tool

- Select **Browse Log** to open the **Framework Logger**. For more information about the Framework Logger, see ***Application Framework User's Guide*** chapter '5.4. Framework Logger' (page 233).

Application Framework Tools

- Select **Open Domain Processes Browser...** to open the DPB. For more information about the DPB, see ***Application Framework User's Guide*** chapter '5.2. Domain Processes Browser' (page 216).
- Select **Browse View Connectors** to open the **Connectors Browser**. For more information about the Connectors Browser, see ***Application Framework User's Guide*** chapter '5.3. View Connectors Browser' (page 232).
- Select **Open online-debugger** to open the **Application Framework Debugging Tool**. For more information about the Application Framework Debugging Tool, see ***Application Framework User's Guide*** chapter '5.5. Online Debugger' (page 235).
- Select **Domain Processes Tools** to open a submenu with the following selections:
  - Migrate Accessor Methods.
  - Reset Application Framework Caches

MLS Tools

- Select **Open MLS data editor** to open the **MLS data editor**. For more information about the MLS data editor, see ***Application Framework User's Guide*** chapter '7.3. The MLS Data Editor' (page 272).
- Select **Multilanguage tools** to open a submenu with special tools for the MLS subsystem.

Object Behavior Framework Tools

- Select **Browse Transactions** to open the **TransactionBrowser**. For more information about the TransactionBrowser, see ***Object Behavior Framework User's Guide*** chapter '3.6. Transaction Browser' (page 127).
- Select **Browse Object Net...** to open the **Object Net Browser**. For more information about the Object Net Browser, see ***Object Behavior Framework User's Guide*** chapter '3.3. Object Net Browser (NetBrowser)' (page 119).
- Select **Object Behavior Tools** to open a submenu with special tools for OBF.

Persistence Framework Tools

- Select **New Persistence Manager...** to open the **STOPF/ODBC** (for ODBC) tool. For more information about the **STOPF** tool, see ***Persistence Framework User's Guide*** chapter '10.2. The STOPF tool' (page 196).
- Select **Browse Persistence Manager...** to open the **STOPF/ODBC** (for ODBC) tool on an existing POM. For more information about the **STOPF** tool, see ***Persistence Framework User's Guide*** chapter '10.2. The STOPF tool' (page 196).
- Select **Persistence Tools** to open a submenu with special tools for PFW.

Frameworks version and Mynd contact information

- Select **About micFrameworks...** to display Frameworks version and Mynd contact information.

Tutorial

1. Tutorial overview

"Example is the school of mankind, and they will learn at no other." Edmund Burke.

This section presents a step-by-step tutorial that walks you through examples that demonstrate the major concepts in Frameworks. As such, the organization of the tutorial is dictated by what you need to do to get your job done as quickly as possible.

The following is an outline of the tutorial. It is highly recommended to work through each step in the tutorial.

Using the example code

- ‘2. Using the example code’ (page 21). Explains how to use the example code provided in the accompanying fw_gs_ex.txt (Frameworks Getting Started Examples) file.
- ‘3. Using the example applications’ (page 22). Explains how to import the versions of the example applications (from the file fw_gs_ex.dat) and how to load a version of the example applications.

Creating Application, Domain Object, Process, View

- ‘4. Create ZyxTutorial application’ (page 24). Explains how to create an application with the required prerequisites.
- ‘5. Create ZyxMember (Domain Object)’ (page 25). Shows how to create a simple domain object. Domain objects are objects such as Club, Member, Address, etc.
- ‘6. Create ZyxEditMember (Domain Process)’ (page 27).
- ‘7. Create ZyxEditMemberView (View)’ (page 29).

Analyzing the Domain Object, Process, View

- ‘8. View connections’ (page 32).

Transactions

- ‘9. Multiple non-transacted views’ (page 33).
- ‘10. Multiple views: transacted non-isolated’ (page 35).
- ‘11. Multiple views: transacted isolated’ (page 40).

Persistence

NOTE: Frameworks supports several different databases; however, the examples in the persistence chapters in this tutorial use IBM DB2 Version 5. If you dont have DB2, you can download a 60-day trial version of **DB2 Version 5.2 Single User for Windows NT** from IBM for free at **www.ibm.com**.

- ‘12. Create database’ (page 41).

Making application, Domain Object, Process persistent-capable

- ‘13. Making application ZyxTutorial persistent-capable’ (page 44).
- ‘14. Make DO ZyxMember persistent’ (page 45).
- ‘15. Make DP ZyxEditMember persistent’ (page 47).

POM Generators and POMs

- ‘16. Create POM’ (page 48).
- ‘17. Generate table ZYXMEMBER using POM ZyxPom’ (page 51).

Very basic persistence example

- ‘18. Creating , displaying, editting persistent DO’s in a table using Smalltalk’ (page 54).

Other basic examples

- ‘19. Add commit / abort buttons to ZyxEditMemberView’ (page 56).
- ‘20. Adding a variable to a persistent class’ (page 59).
- ‘21. Validate range of entered data in ZyxEditMemberView’ (page 63).

Parent processes

Creating parent process Domain Object, Process, View

- ‘22. Create DO ZyxClub’ (page 64).
- ‘23. Create DP ZyxEditClub / modify DP ZyxEditMember’ (page 66).
- ‘24. Create ZyxEditClubView (View)’ (page 68).

Transactions with parent/child processes

- ‘25. Transacted child process connection to ZyxEditMember’ (page 72).
- ‘26. Transacted parent process ZyxEditClub’ (page 75).

Adding and deleting to/from collections

- ‘27. Adding and deleting members’ (page 77).

Viewports

- ‘28. Implementing HoverHelp and F1 Help (using a ViewPort)’ (page 78).
- ‘29. Enabling/disabling buttons’ (page 80).
- ‘30. Using a ViewPort as a filter’ (page 82).

Static Group controls

- ‘31. Displaying / hiding a GroupControl’ (page 85).
- ‘32. Static ZyxEditClubView GroupBox containing ZyxEditMember as child process’ (page 88).
- ‘33. Static ZyxEditClubView Notebook containing ZyxEditMember as child process’ (page 91).

Authorization

- ‘34. Authorization: DO attribute accessors’ (page 94).
- ‘35. Authorization: DP methods’ (page 98).

Modality

- ‘36. Modality: MicFwPrimaryApplicationModal’ (page 100).
- ‘37. Modality: MicFwFullApplicationModal’ (page 102).
- ‘38. Modality: MicFwSystemModal’ (page 103).

Dynamic Group controls

- ‘39. Dynamic GroupBox’ (page 104).
- ‘40. Dynamic Notebook’ (page 107).
- ‘41. Container Icon Tree’ (page 112).

Drag&Drop

- ‘42. Drag & Drop’ (page 115) (also introduces the Multiple Selection List).

2. Using the example code

Example code in this User Guide

Example code is displayed throughout this document in courier fixed-width character format:

```
 FW CH 9. Working with transaction contexts."
[
  Smalltalk at: #ZyxContext put: MicFwTransactionManager newContext.
]
```

Executing example code in fw:_gs_ex.txt

If a block of code in this document begins with the text "**FW CH (#)**" (Frameworks Gettting Started Manual Chapter #), then the block is included in the file **fw_gs_ex.txt** (on the CD ROM). To execute this code in a Workspace window:

2.1. In the System Transcript: Select **File / Open**.

2.2. Double-click on **fw_gs_ex.txt** (on the CD ROM). A workspace dialog is opened with the example text.

If you are using a shared Smalltalk environment

The example code throughout this tutorial uses class names starting with the letters "**Zyx**". If you are doing this tutorial in a classroom environment where you will be sharing the Smalltalk development environment library (**mgr45.dat** on a server) with other students, then you should change "**Zyx**" to some combination of letters that no other student is using. This is necessary since you will be creating classs during the tutorial and no 2 classes are allowed to have the same name.

2.3. In the workspace: Select **Edit / Find/Replace** to replace all occurrences of "**Zyx**" with a unique combination of letters.

2.4. Select **File / Save**.

You can now execute the code as you read through this tutorial.

Note: Any Smalltalk code between square brackets ("[" and "]") should be selected and executed together. In a Visual Age workspace, the entire text between double brackets can be selected by simply placing the cursor immediately after the "[" (or immediately before the "]") and double-clicking.

3. Using the example applications

The examples in this tutorial are arranged in chapters. Each chapter builds on the work completed in the previous chapters. Therefore, in order to start doing the examples in a chapter, it is usually necessary that all the steps in the previous chapters were completed.

However, the file **fw_gs_ex.dat** (on the CD ROM) contains versions of the ZyxTutorial application for most of the chapters. Therefore, in order to start a chapter without having completed the previous chapters, simply load the application version for the previous chapter.

For example, to start immediately with Chapter 11, load edition FW GS 10 of the ZyxApplication.

NOTE: Ignore any alphabetic characters in the version number.

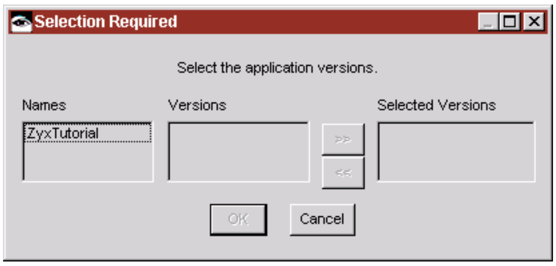
NOTE: Some chapters require a database (ZYX) for the examples. The database may need to be created to do the examples.

3.1. Import the ZyxTutorial application versions

3.1. In **Visual Age Organizer**: Select **Applications / Import/Export / Import applications...**. The dialog **Enter the full path name of the library** appears.

3.2. Select **fw_gs_ex.dat** on the CD ROM.

3.3. Click **Open**. The **Selection required** dialog appears.



Select the application versions.

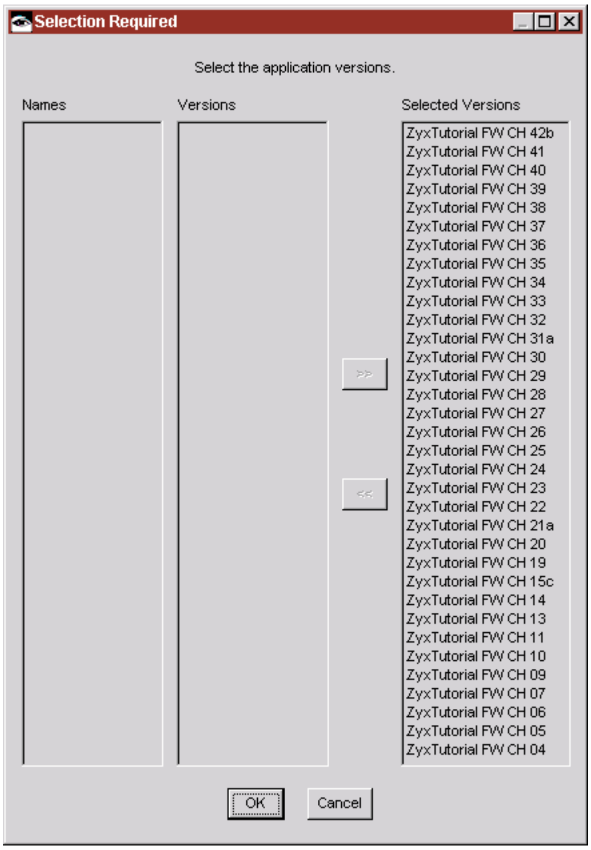
ZyxTutorial

Figure 3.1. Importing ZyxTutorial application versions

3.4. In the **Names** box: Select **ZyxTutorial**.

3.5. In the **Versions** box: Select all versions.

3.6. Click **>>**. The versions appear in the **Selected versions** box.



Select the application versions.

```
ZyxTutorial FW CH 42b
ZyxTutorial FW CH 41
ZyxTutorial FW CH 40
ZyxTutorial FW CH 39
ZyxTutorial FW CH 38
ZyxTutorial FW CH 37
ZyxTutorial FW CH 36
ZyxTutorial FW CH 35
ZyxTutorial FW CH 34
ZyxTutorial FW CH 33
ZyxTutorial FW CH 32
ZyxTutorial FW CH 31a
ZyxTutorial FW CH 30
ZyxTutorial FW CH 29
ZyxTutorial FW CH 28
ZyxTutorial FW CH 27
ZyxTutorial FW CH 26
ZyxTutorial FW CH 25
ZyxTutorial FW CH 24
ZyxTutorial FW CH 23
ZyxTutorial FW CH 22
ZyxTutorial FW CH 21a
ZyxTutorial FW CH 20
ZyxTutorial FW CH 19
ZyxTutorial FW CH 15c
ZyxTutorial FW CH 14
ZyxTutorial FW CH 13
ZyxTutorial FW CH 11
ZyxTutorial FW CH 10
ZyxTutorial FW CH 09
ZyxTutorial FW CH 07
ZyxTutorial FW CH 06
ZyxTutorial FW CH 05
ZyxTutorial FW CH 04
```

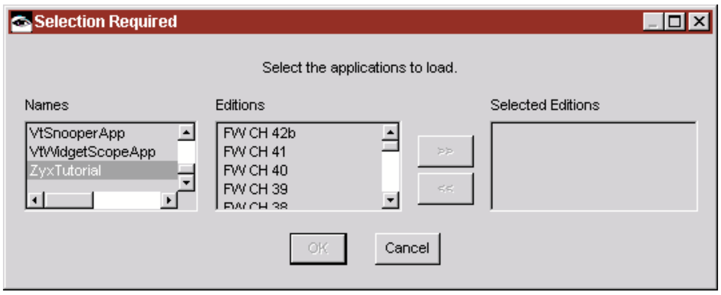
Figure 3.2. Selected versions of ZyxTutorial

3.7. Click **OK**. The application versions are imported.

3.2. Load the required ZyxTutorial application edition

3.8. In **Visual Age Organizer**: Select **Applications / Load / Available applications...**. The dialog **Selection required** appears.

3.9. In the **Names** box: Select **ZyxTutorial**.



Selection Required

Select the applications to load.

Names: VtSnooperApp, VtWidgetScopeApp, ZyxTutorial

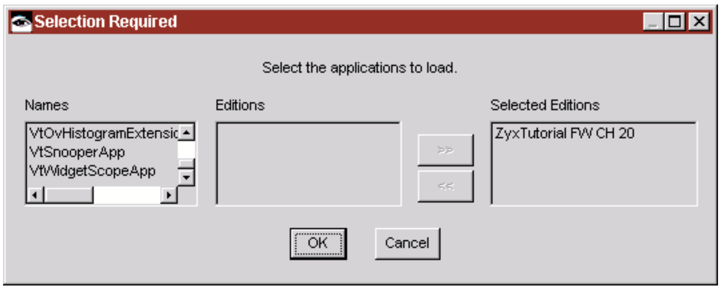
Editions: FW CH 42b, FW CH 41, FW CH 40, FW CH 39, FW CH 38

Selected Editions

Figure 3.3. ZyxTutorial selected as application to load

3.10. In the **Editions** box: Select the required edition (the edition number matches the chapter number).

3.11. Click **>>**. The edition appears in the **Selected editions** box (FW CH 20 (Frameworks Getting Started Chapter 20) chosen in the following diagram as an example).



Selection Required

Select the applications to load.

Names: VtOvHistogramExtensio, VtSnooperApp, VtWidgetScopeApp

Editions

Selected Editions: ZyxTutorial FW CH 20

Figure 3.4. Selected edition of ZyxTutorial

3.12. Click **OK**. The selected edition of ZyxTutorial is loaded.

3.3. Load a different ZyxTutorial application edition

If you need at some point to load a different edition, do the following:

3.13. In **Visual Age Organizer**: Select **ZyxTutorial**.

3.14. Right-click.

3.15. In the pop-up menu: Select **Load / Another edition...**. The **Selection required** dialog appears.

3.16. Double-click on the required edition. The selected edition is loaded.

4. Create ZyxTutorial application

In this chapter you will:

- Create application ZyxTutorial.

All of the classes created during this tutorial will be assigned to your ZyxTutorial application.

4.1. Create application ZyxTutorial.

4.1. In **Visual Age Organizer**: Select **Options / Full menus** to display the full menu set in VAST.

4.2. Select **Applications / New**. The **New Application** dialog is displayed.

4.3. In the **Name** field enter "**ZyxTutorial**".

4.4. Make sure that the checkbox **Subapplication of** is not checked.

4.5. Click **OK**. The application ZyxTutorial is selected in the list of applications.

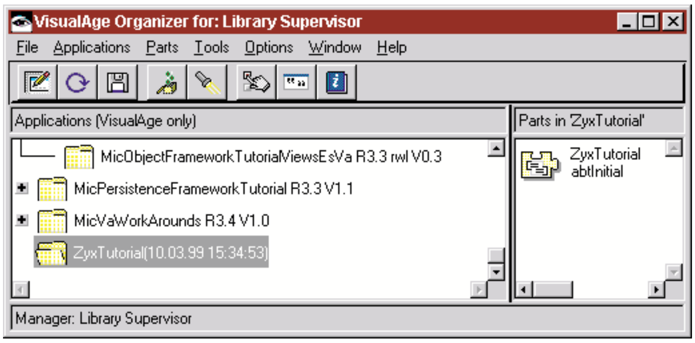

VisualAge Organizer for: Library Supervisor
File Applications Parts Tools Options Window Help
Applications (VisualAge only)
MicObjectFrameworkTutorialViewsEsVa R3.3 rwl V0.3
MicPersistenceFrameworkTutorial R3.3 V1.1
MicVaWorkArounds R3.4 V1.0
ZyxTutorial(10.03.99 15:34:53)
Parts in 'ZyxTutorial'
ZyxTutorial
abtInitial
Manager: Library Supervisor

Figure 4.1. ZyxTutorial in the application list.

5. Create ZyxMember (Domain Object)

In this chapter you will:

- Create ***Domain Object*** (DO) ZyxMember. ZyxMember attributes will reference objects such as a member name, weight, address, etc. For more information about DO's, consult the ***Application Framework User's Guide*** chapter '1.3.4. Domain Object' (page 38).
- Add attributes to the DO with the ***Object Net Browser*** (ONB) tool. The ONB is described in detail in the ***Object Behavior Framework User's Guide*** in chapter '3.3. Object Net Browser (NetBrowser)' (page 119).

5.1. Create MicFwDomainObject subclass ZyxMember

5.1. In **Visual Age Organizer**: Select **ZyxTutorial**.

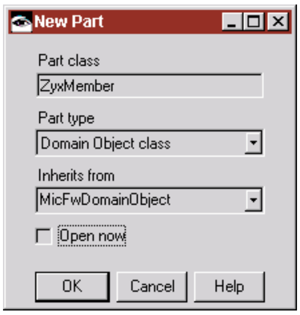
5.2. Select **Parts / New / Part**.

5.3. In the field **Part class**: Enter **ZyxMember**.

5.4. In the field **Part type**: Select **Domain Object Class**.

5.5. In the field **Inherits from**: Select **MicFwDomainObject**.

5.6. Uncheck the checkbox **Open Now**.



Part class: ZyxMember

Part type: Domain Object class

Inherits from: MicFwDomainObject

□ Open now

Figure 5.1. New part dialog settings for ZyxMember

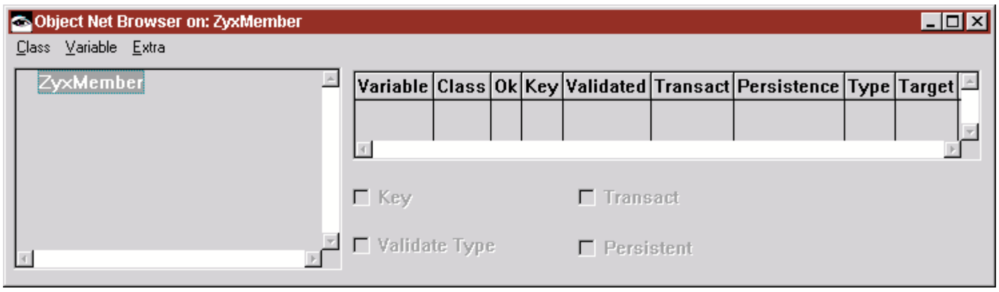
5.7. Select **OK**. Note that the new part is now displayed in the right window in the Organizer.

5.2. Add ZyxMember>>name (String)

5.8. In the **System Transcript**: Select **micFrameworks / Browse Object Net**.

5.9. In response to the question **Select root class**: Enter **ZyxMember**.

5.10. Select **OK**. The **Object Net Browser on: ZyxMember** is opened.



Variable	Class	Ok	Key	Validated	Transact	Persistence	Type	Target

□ Key □ Transact

□ Validate Type □ Persistent

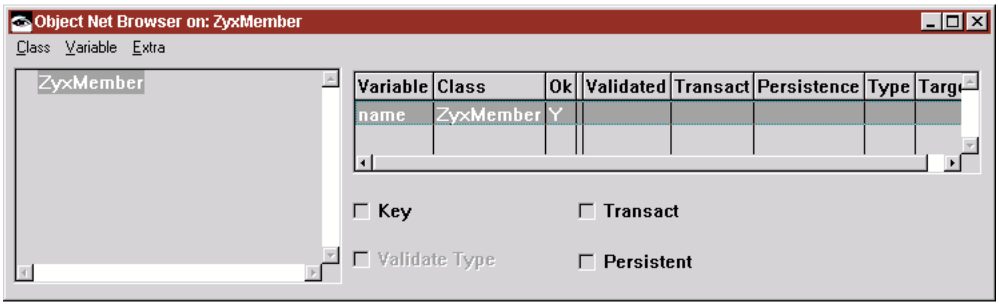
Figure 5.2. Object Net Browser on ZyxMember

5.11. Right-click in the box on the upper-right.

5.12. From the pop-up window: Select **Add Instance Variable**.

5.13. In response to **Enter the name for the instance variable**: Enter **name**.

5.14. Click **OK**. Note the new variable in the ONB.



ZyxMember

Variable	Class	Ok	Validated	Transact	Persistence	Type	Targ
name	ZyxMember	Y					

□ Key □ Transact

□ Validate Type □ Persistent

Figure 5.3. Variable "name" added to ZyxMember within the ONB

5.15. Select **Class / Save**. **NOTE**: Changes made in the ONB are not be actually implemented until the class has been saved.

5.16. Close the ONB.

5.17. Save the image.

6. Create ZyxEditMember (Domain Process)

In this chapter you will:

- Create ***Domain Process*** (DP) ZyxEditMember. ZyxEditMember methods implement the functionality required to edit a member's attributes. For more information about DP's, consult the ***Application Framework User's Guide*** chapter '1.4.3.2. Domain Process' (page 50)
- Create connections for the DP and DO in the ***Domain Processes Browser*** (DPB). The DPB is a Visual tool for setting a variety of parameters for the DP. For more information about DP, consult the ***Application Framework User's Guide*** chapter '5.2. Domain Processes Browser' (page 216).

6.1. Create MicFwDomainProcess subclass ZyxEditMember.

6.1. In **Visual Age Organizer**: Create a new part (do not open now) for **ZyxApplication** with:

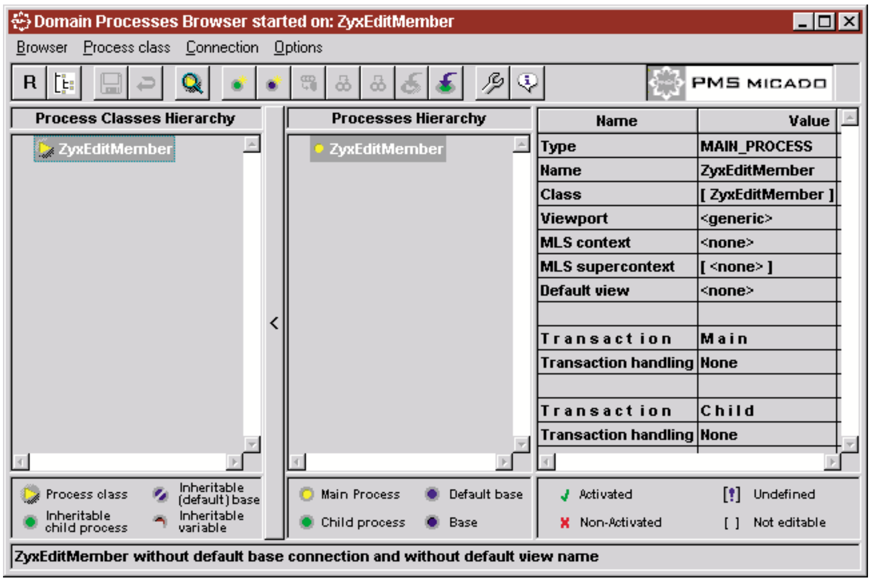
- **Part class**: Enter **ZyxEditMember**.
- **Part type**: Select **Domain Process Class**.
- **Inherits from**: Select **MicFwDomainProcess**.

6.2. Open Domain Process Browser (DPB) on ZyxEditMember

6.2. From **System Transcript**: Select **MicFrameworks / Open Domain Processes Browser...**.

6.3. In response to **Select a root class**: Enter **ZyxEditMember**. The **Selection Required** dialog appears.

6.4. In response to "Choose a class": Double-click on **ZyxEditMember**. The DPB appears:



Name	Value
Type	MAIN_PROCESS
Name	ZyxEditMember
Class	[ZyxEditMember]
Viewport	<generic>
MLS context	<none>
MLS supercontext	[<none>]
Default view	<none>
Transaction	Main
Transaction handling	None
Transaction	Child
Transaction handling	None

ZyxEditMember without default base connection and without default view name

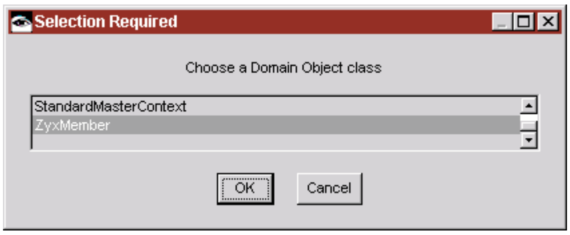
Figure 6.1. DPB on ZyxEditMember

6.3. Adding a Base Connection to a DP

A ***Base Connection*** refers to the connection between the DP class and a DO class (the DO is the "base"; the origins of the term "base" are historical in nature).

6.5. In the **Processes Hierarchy** box: Right-click on **ZyxEditMember**.

6.6. Select **Add Base Connection**. The **Selection required** dialog appears.

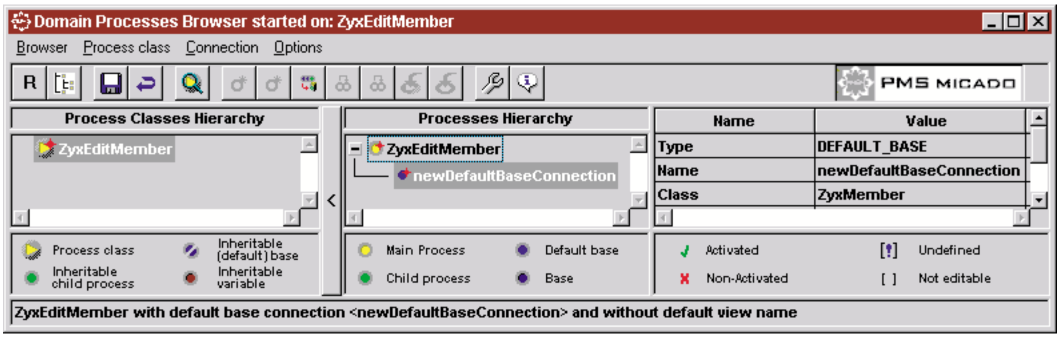


Choose a Domain Object class

StandardMasterContext
ZyxMember

Figure 6.2. Dialog for selecting ZyxMember as the domain object class for ZyxEditMember

6.7. In response to **Choose a domain object class**: Double-click on **ZyxMember**. Note that in the DP ZyxEditMember now has a connection to ZyxMember and that the connection is named **newDefaultBaseConnection**:



Name	Value
Type	DEFAULT_BASE
Name	newDefaultBaseConnection
Class	ZyxMember

ZyxEditMember with default base connection <newDefaultBaseConnection> and without default view name

Figure 6.3. Connection from ZyxEditMember to ZyxMember in DPB

Note: The red arrows indicate that unsaved changes have been made in the DPB.

6.3.1. Change the names of the connections

The names in the column **Processes Hierarchy** are the names of ***connections***. These names will be changed now to make it easier to distinguish the connection names during the course of the tutorial.

6.3.1.1. Rename the DP connection

6.8. In the **Processes Hierarchy** column: Click on **ZyxEditMember**.

6.9. In the **Value** column: In the row **Name**: Click on **ZyxEditMember**.

6.10. Change the name of the connection to **eMPConn** (edit Member Process Connection). This is the name of the connection to the DP ZyxEditMember.

6.11. Click in a different area of the Domain Processes Browser in order to reflect the change in the DPB.

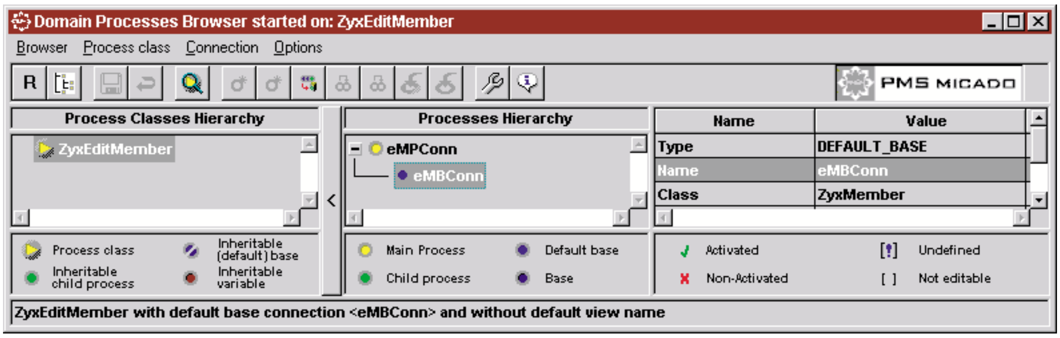
6.3.1.2. Rename the DO connection

6.12. In the **Processes Hierarchy** column: Click on **newDefaultBaseConnection**.

6.13. In the **Value** column: In the row **Name**: Click on **newDefaultBaseConnection**.

6.14. Change the name of the connection to **eMBConn** (edit Member Base Connection). This is the name of the connection to the DO ZyxMember.

6.15. Select **Browser / Save all changes**.



Name	Value
Type	DEFAULT_BASE
Name	eMBConn
Class	ZyxMember

ZyxEditMember with default base connection <eMBConn> and without default view name

Figure 6.4. DP dialog after ZyxEditMember / ZyxMember connections renamed

Note that the red arrows have disappeared (the changes have been saved).

6.16. Close the DPB.

6.17. Save the image.

7. Create ZyxEditMemberView (View)

In this chapter you will:

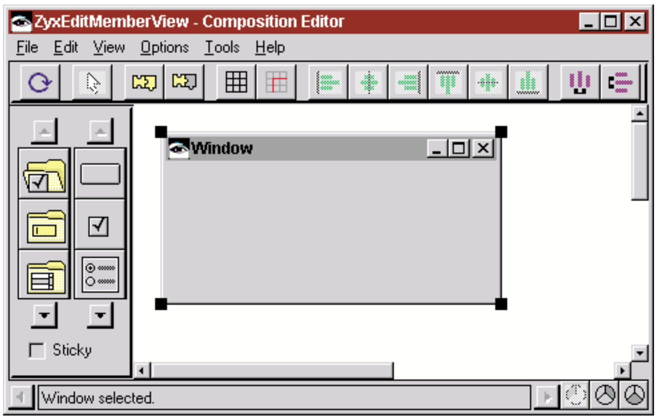
- Create ***View*** ZyxEditMemberView. ZyxEditMemberView will provide the user interface to the DP ZyxEditMember and DO ZyxMember. For more information about views, see the ***Application Framework User's Guide*** chapter '1.3.9. Connecting Objects to Views' (page 43).
- Add a text field to the view that connects to the DO variable ZyxMember>>name via the ***naming convention***. The naming convention is a method of specify connections to a part with the **partName**. The naming convention is described in the ***Application Framework User's Guide*** chapter '3.5. View Parts' (page 113)
- Assign the view ZyxEditMemberView to the DP ZyxEditMember using the DPB.
- Test the view.

7.1. Create AbtAppBldrView subclass ZyxEditMemberView.

7.1. In **Visual Age Organizer**: Create and open (make sure the **Open now** checkbox is checked) a new part for **ZyxApplication** with:

- **Part class**: Enter **ZyxEditMemberView**.
- **Part type**: Select **Visual Part**.
- **Inherits from**: Select **AbtAppBldrView**.

The following dialog appears:


ZyxEditMemberView - Composition Editor
File Edit View Options Tools Help
Window
Sticky

Window selected.

Figure 7.1. ZyxEditMemberView initial window in the Composition Editor

7.1.1. Rename window

7.2. Double-click on the window on the free-form surface. The Properties dialog appears.

7.3. Change property **Title** to **ZyxEditMember**.

7.4. Click **OK**.

7.2. Add label and text field to View

7.5. Add a label to the view (click on the label icon in the parts palette; the cursor becomes a crosshair; click in the view window).

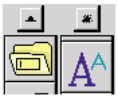


Figure 7.2. Label part in the parts palette

7.6. Change the **label** property **object** to **Name**.

7.7. Add a text field to the view.

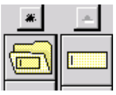


Figure 7.3. Text field part in the parts palette

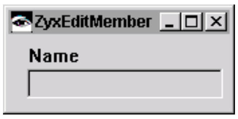


Figure 7.4. ZyxEditMemberView with label and text field

7.8. Change the **text field** property **partName** to **name_**. This specifies that the content of the text field is referenced by the variable "name" in the base connection for the view.

7.9. Select **File / Save Part**.

7.10. Close the Composition Editor.

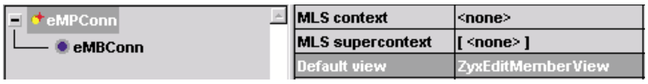
7.3. Assign View to Process

7.11. Open the **DPB** on **ZyxEditMember**.

7.12. In the **Processes Hierarchy** box: Select process connection **eMPConn**.

7.13. In column **Name** row **Default View**: Click on column **Value**. A drop-down list appears.

7.14. Select from the drop-down list **ZyxEditMemberView**.



MLS context	<none>
MLS supercontext	[<none>]
Default view	ZyxEditMemberView

Figure 7.5. Assigning ZyxEditMemberView to ZyxEditMember in the DPB

7.15. Select **Browser / Save all changes**. NOTE: If this option is not available, click anywhere in the dialog to register the changes and then the option will be available.

7.16. Close the DPB.

7.17. Save the image.

7.4. Test

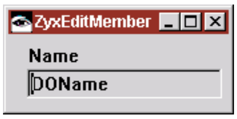
7.4.1. Display DO attributes (ZyxMember>>name) in View

NOTE: If when executing the following code an exception is thrown (stating that a view is not defined): Reopen the DPB and reassign ZyxEditMemberView to ZyxEditMember.

7.18. In the workspace execute the following code (NOTE: If you get the message "invalid character sequence": replace the single apostrophe (') characters from the text file with the characters generated via your computer’s keyboard):

```
FW CH 7. 1. Test simple view."
[
  | aDO aDP |
  aDO := ZyxMember new.
  aDO name: 'DOName'.
  aDP := ZyxEditMember new.
  aDP eMBConn: aDO.
  aDP openView.
]
```

The following dialog appears:



Name

DOName

Figure 7.6. Display of DO ZyxMember attributes in View ZyxEditMemberView

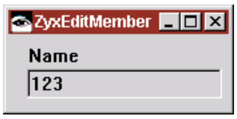
7.19. Close the dialog.

7.4.2. Display DO attribute of invalid type (not String)

7.20. In the workspace execute the following code:

```
FW CH 7. 2. Invalid type of object referenced by attribute."
[
  | aDO aDP |
  aDO := ZyxMember new.
  aDO name: 123.
  aDP := ZyxEditMember new.
  aDP eMBConn: aDO.
  aDP openView.
]
```

Note that no exception is thrown. The Integer object is automatically converted into a string object for display.



Name

123

Figure 7.7. Improper object type (Integer) auto-converted and displayed in text field (String required)

7.21. Close the dialog.

7.4.3. Attempt to display DO of invalid type (exception)

7.22. In the workspace execute the following code:

```
FW CH 7. 3. Invalid type of domain object."
[
  | aDO aDP |
  aDO :=  'Invalid DO'.
  aDP := ZyxEditMember new.
  aDP eMBConn: aDO.
]
```

"Invalid model" exception is thrown, since only a ZyxMember instance can be assigned (as the Base "model") to the DP Base Connection.

8. View connections

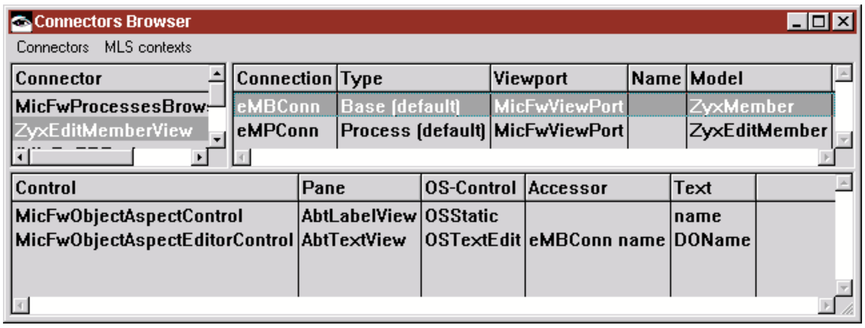
In this chapter you will:

- Use the ***Connections Browser*** (CB) to analyze the status of connections between the Views, DP’s, and DO’s. For more information about the Connections Browser, see the ***Application Framework User’s Guide*** chapter ‘5.3. View Connectors Browser’ (page 232).

8.1. Open Connections Browser

8.1. From **System Transcript**: Select **micFrameworks / Browse View Connectors**. The Connectors Browser opens.

8.2. In the **Connector** list: Select **ZyxEditMemberView**:



Connector
MicFwProcessesBrow
ZyxEditMemberView

Connection	Type	Viewport	Name	Model
eMBConn	Base (default)	MicFwViewPort		ZyxMember
eMPConn	Process (default)	MicFwViewPort		ZyxEditMember

Control	Pane	OS-Control	Accessor	Text
MicFwObjectAspectControl	AbtLabelView	OSStatic		name
MicFwObjectAspectEditorControl	AbtTextView	OSTextEdit	eMBConn name	DOName

Figure 8.1. Connections Browser for ZyxEditMemberView

Note the following:

- There connections for the view are eMBConn and eMPConn.
- The connection types are base and process.
- The connection models are ZyxMember and ZyxEditMember.
- eMBConn has a control for the label and text field.
- eMPConn has a control for the window.

8.3. Close the Connections Browser.

9. Multiple non-transacted views

In this chapter you will:

- Open 2 views of the object referenced by an attribute. An object can be assigned to the attribute from either window.
- Test non-transacted changes.

9.1. Create ZyxEditMember>>openOn: class and instance methods

9.1. Create the following methods:

```
ZyxEditMember class>>openOn: aMember
  self new openOn: aMember.
ZyxEditMember>>openOn: aMember
  self eMBConn: aMember.
  self openView.
```

9.2. Save the image.

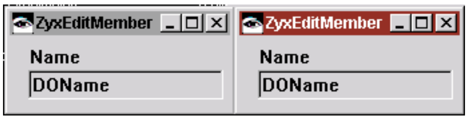
9.2. Test

9.2.1. Open 2 views

9.3. In the workspace execute the following code:

```
FW CH 9. Test 2 views of same object."
[
  | aDO |
  aDO := ZyxMember new.
  aDO name: 'DOName'.
  ZyxEditMember openOn: aDO.
  ZyxEditMember openOn: aDO.
]
```

The following 2 dialogs appear (if only 1 dialog is visible: move the visible dialog to the right to make the second dialog visible):



Name
DOName

Name
DOName

Figure 9.1. 2 views of same object.

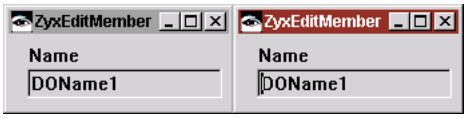
The LEFT dialog is the first dialog that was opened (with the first "ZyxEditMember openOn: aDO"). The RIGHT dialog (the dialog with the focus) is the second dialog that was opened (with the second "ZyxEditMember openOn: aDO").

9.2.2. Test non-transacted changes

Both views are of the same object. Changes to the objects made in either dialog are not transacted (transacted changes will be discussed in the next section).

9.4. In the **LEFT** dialog: Change the name to **DOName1.** The new String object will be the new object referenced by the ZyxMember object attribute name.

9.5. Click in the **RIGHT** dialog.



Name
DOName1

Name
DOName1

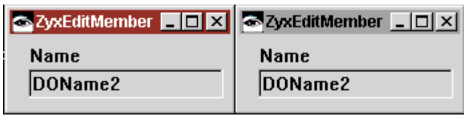
Figure 9.2. Non-transacted changes in a view are reflected in other views

Clicking in the RIGHT dialog changed the focus, causing the following:

- The String object entered in the text field in the left dialog was assigned to the ZyxMember object attribute **name**.
- The right view (ZyxEditMemberView) is updated (and thus the current ZyxMember object attributes are displayed).

9.6. In the **RIGHT** dialog: Change the name to **DOName2**.

9.7. Click in the **LEFT** dialog.



DOName2

DOName2

Figure 9.3. Non-transacted changes to the same object can be made in multiple dialogs

Note that the attributes of the ZyxEditMember object can be changed in either dialog.

9.8. Close both views.

10. Multiple views: transacted non-isolated

The problem with the situation in the previous chapter is that an object can be changed from any view, overwriting the changes made in any other views. This undesireable situation can be avoided by transacting changes.

This chapter will explore non-isolated transactions (uncommitedRead transactions). uncommitedRead is the default setting for transaction contexts. If a context is uncommitedRead, then the dialogs belonging to that context will display the transacted changes from other dialogs.

For a complete overview and a detailed description of transaction concepts, see the ***Object Behavior Framework User's Guide*** chapter '1.3. Transactions' (page 27).

In this chapter you will:

- Define DP ZyxEditMember as a transacted process using the DPB.
- Define DO variable ZyxMember>>name as transacted using the ONB.
- Open the **Transaction Browser** to monitor the transacted changes. For more information about the Transaction Browser, see the ***Object Behavior Framework User's Guide*** chapter '3.6. Transaction Browser' (page 127).
- Test transacted changes:
    - View transaction information in the TB, including ***TrLevel1*** and ***object version*** information for the ***active*** and ***inactive context***.
    - Attempt to assign an object to a transacted variable that is locked by a transaction context that is not active.
    - Abort a transaction using the TB.

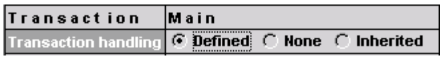
10.1. Define ZyxEditMember as transacted process

10.1. Open the **DPB** for **ZyxEditMember**.

10.2. In the **Processes Hierarchy** box: Select **eMPConn**.

10.3. For **Transaction Main**: In column **Value** row **Transaction handling**: Click on **None**. 3 radio buttons appear.

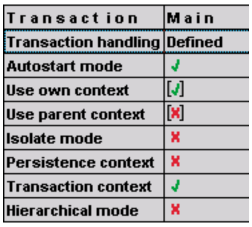
10.4. Select **Defined**:



Transaction	Main
Transaction handling	● Defined ○ None ○ Inherited

Figure 10.1. Defining transaction handling for ZyxEditMember (Transaction Main)

10.5. Click elsewhere in the DPB to enter the change. Note the default settings for the transacted process (the transaction context is non-isolation mode and is enabled):



Transaction	Main
Transaction handling	Defined
Autostart mode	✓
Use own context	[✓]
Use parent context	[✗]
Isolate mode	✗
Persistence context	✗
Transaction context	✓
Hierarchical mode	✗

Figure 10.2. Default settings in the DPB for Transaction Main

10.6. **Save all changes**.

10.7. Close the DPB.

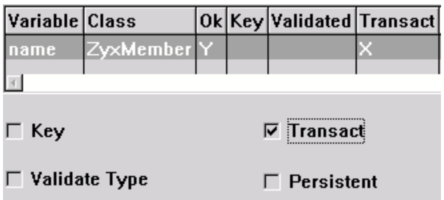
10.2. Specify ZyxMember>>name as transacted

The DP is now specified as being transacted. However, changes made to DO attribute ZyxMember>>name will not be transacted unless the actual attribute is defined as being transacted with the ONB (this creates the necessary methods for supporting transaction behavior for the attribute).

10.8. Open the **ONB** on **ZyxMember**.

10.9. Select variable **name**.

10.10. Check the checkbox **Transacted**:



Variable	Class	Ok	Key	Validated	Transact
name	ZyxMember	Y			X

☐ Key ☑ Transact

☐ Validate Type ☐ Persistent

Figure 10.3. Specifying ZyxMember>>name as transacted

10.11. **Save the changes**.

10.12. Save the image.

10.3. Transaction Browser

10.13. From the **System Transcript**: Select **micFrameworks / Browse Transactions**. The Transaction Browser (TB) is opened:

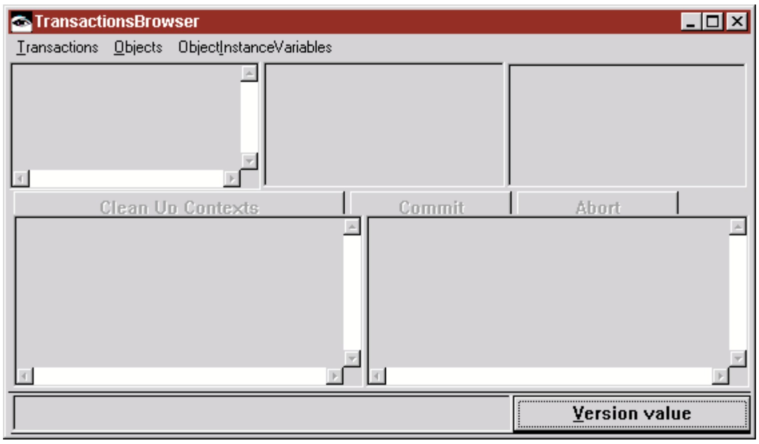


Figure 10.4. Transaction Browser dialog

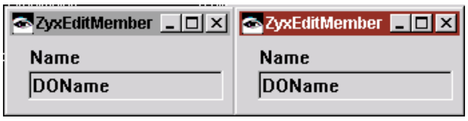
Note: ***MicFwTechnicalTransactionContext***'s may be present and displayed in the transaction browser. These contexts used by various Frameworks tools, and therefore are of no concern at the moment. Note that there are no other contexts open at the moment.

10.4. Test

10.14. In the workspace execute the following code:

```
FW CH 10. Multiple views: Transacted non-isolated."
[
  | aDO |
  aDO := ZyxMember new.
  aDO name: 'DOName'.
  ZyxEditMember openOn: aDO.
  ZyxEditMember openOn: aDO.
]
```

The following 2 dialogs appear (if only 1 dialog is visible: move the visible dialog to the right to make the second dialog visible):



Name: DOName

Name: DOName

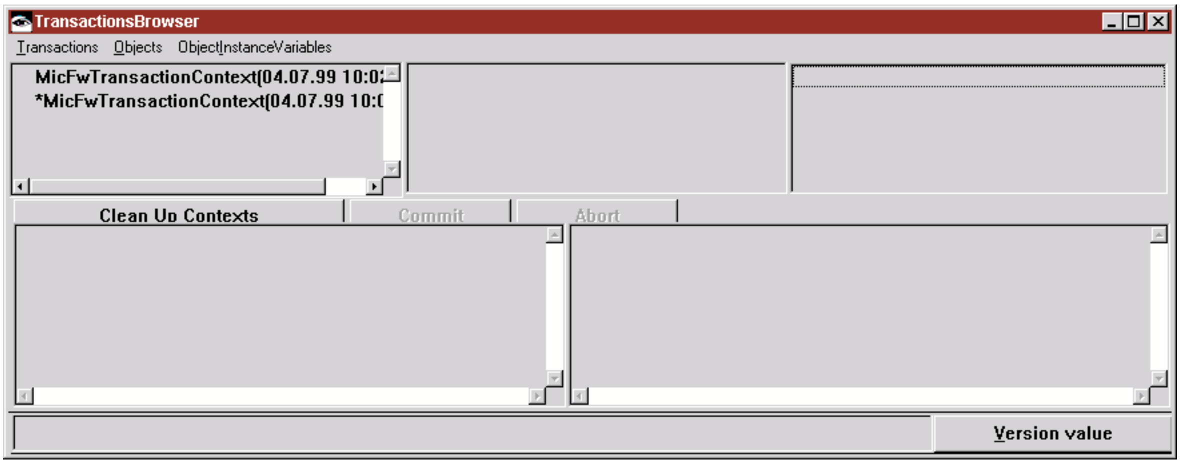
Figure 10.5. Open 2 views of transacted object attribute

The LEFT dialog is the first dialog that was opened (with the first "ZyxEditMember openOn: aDO"). The RIGHT dialog (the dialog with the focus) is the second dialog that was opened (with the second "ZyxEditMember openOn: aDO").

10.4.1. No transacted changes

10.4.1.1. View transaction info in TB

10.15. In the TB: Select **Transactions / Update**. Note that 2 ***MicFwTransactionContext***'s were created when the dialogs were opened:



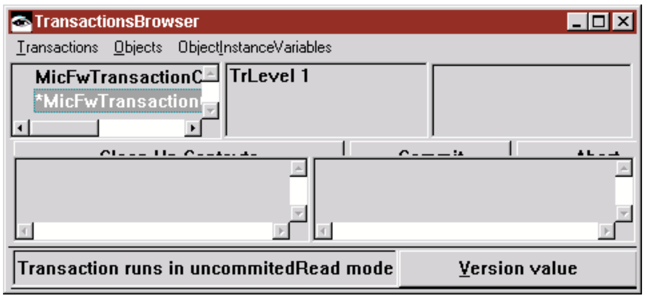
MicFwTransactionContext(04.07.99 10:0:
*MicFwTransactionContext(04.07.99 10:(

Figure 10.6. Transaction contexts created when 2 views of object opened

The ***inactive context*** (the context without the asterisk ("*") in front of it) is assigned to the LEFT dialog that is currently not selected (does not have the focus). The ***active context*** (the context with the asterisk ("*") in front of it) is assigned to the RIGHT dialog that is currently selected (has the focus).

10.4.1.1.1. Display TrLevels (only TrLevel1) of contexts

10.16. Select the **active context**. The **TrLevels** for the context are displayed in the upper-middle box (only TrLevel1 exists).



MicFwTransactionC
*MicFwTransaction
TrLevel 1
Transaction runs in uncommitedRead mode

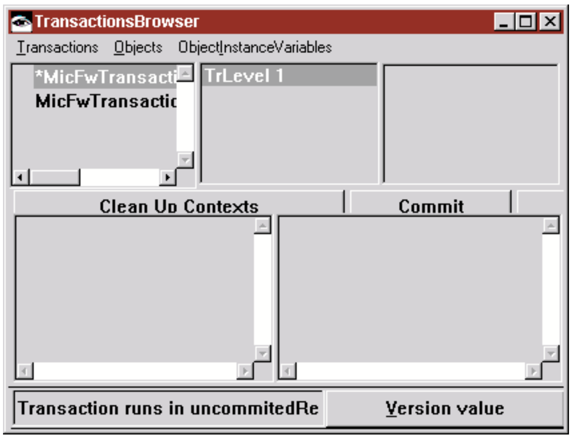
Figure 10.7. No transacted changes for the object whose attributes havent been changed in the view

10.17. Select the **inactive context**. Note that only TrLevel1 exists.

10.4.1.1.2. Display contents (empty) of TrLevel1 for contexts

10.18. Select the **active context**.

10.19. Select ***TrLevel1***. The upper-right box displays the ***object versions*** (transacted changes for variables while the transaction context was active) for the selected TrLevel (empty, since no transacted changes were made while this context was active).



*MicFwTransacti
MicFwTransactic
TrLevel 1
Transaction runs in uncommitedRe

Figure 10.8. No transacted changes for the object whose attributes havent been changed in the view

10.20. Select the **inactive context**.

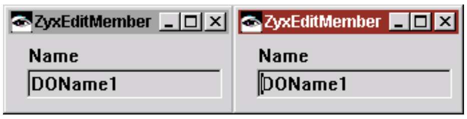
10.21. Select TrLevel1. Note that this TrLevel1 also contains no object versions.

10.4.2. Make a transacted change in the LEFT dialog

10.22. Select the **LEFT** dialog. This causes the transaction context assigned to the LEFT dialog to be activated (and the context for the RIGHT dialog is deactivated).

10.23. In the LEFT dialog: Change the name to **DOName1**. This change is transacted because the DP instance for the dialog is transacted AND ZyxMember>>name is specified as a transacted variable.

10.24. Click in the **RIGHT** dialog. Note that the change in focus updates both views.



Name: DOName1

Name: DOName1

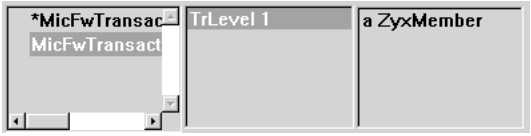
Figure 10.9. The second view of an object is updated after transacted changes in the first view

10.4.2.1. View transaction info in TB

10.4.2.1.1. For the inactive context / TrLevel1: Display objects with transacted changes to variables

10.25. In the TB: Select the inactive (for the LEFT dialog) **MicFwTransactionContext**.

10.26. Select **TrLevel1**. Note the contents of the upper-right box in the TB (Note: If the ZyxMember object is not displayed, refresh the TB view by clicking on the other Transaction Context).



*MicFwTransac
MicFwTransact

TrLevel 1

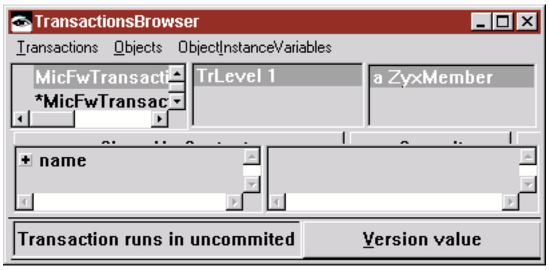
a ZyxMember

Figure 10.10. In the TB: Objects with transacted changes for selected context / TrLevel1

a ZyxMember is an object whose variable(s) reference an object version (ie, transacted changes have been made to the variable) that is in the selected TrLevel (TrLevel1) of the selected transaction context (the inactive context).

10.4.2.1.2. For the object: Display variables with transacted changes

10.27. In the TB: Select **a ZyxMember**. Note that in the lower left box the variable **name** appears. The lower left box contains for the selected object (in the upper right box) all variables with transacted changes.



MicFwTransacti
*MicFwTransac

TrLevel 1

a ZyxMember

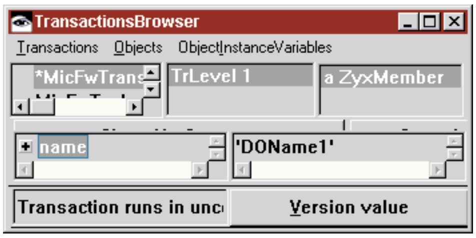
+ name

Figure 10.11. In the TB: Variables with transacted changes for selected object

10.4.2.1.3. For the variable: Display version value (uncommitted target) of object version

10.28. If the **Variable value** button is displayed: Click on the button to display the **Version value** button.

10.29. Select **name**. Note that the ***version value*** **'DOName1'** is displayed.



*MicFwTrans

TrLevel 1

a ZyxMember

+ name

'DOName1'

Figure 10.12. In the TB: Version value (uncommitted target) for selected variable

The "version value" is also referred to as the "uncommitted target". The version value is the last object that was assigned to the variable while the selected context was active and the selected TrLevel was the highest TrLevel.

10.4.2.1.4. For the variable: Display variable value (committed target) of object version

10.30. Click on ***Version value***. The button becomes ***Variable value***. Note that the variable value

’DOName’ is displayed.

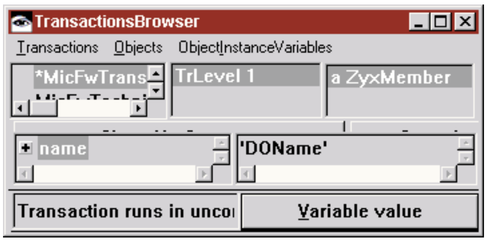


Figure 10.13. In the TB: Variable value (committed target) for selected variable

10.4.2.2. What occurred when ’DOName1’ was entered in the LEFT dialog

When a new object was "assigned" to the object attribute in the LEFT dialog, the following occurred:

- An version object was created for and assigned to the attribute. This version object references 2 objects:
  - The **variable value** (**committed target**): The object (’DOName’) that was reference by the variable before any transacted changes
  - The **version value** (**uncommitted target**): The last object (’DOName1’) assigned to the variable while the transaction context was active and the selected TrLevel was the highest TrLevel.
- The version value was displayed in the LEFT dialog.

The version value was also displayed in the RIGHT dialog when the dialog was updated (obtained the focus).

10.4.3. Attempt to assign object to variable locked by another transaction

When a new object is assigned to a transacted variable while a context is active, then the variable is ***locked*** by that context. If an attempt is made to assign an object to the variable while a different context is active, a ***transaction write conflict exception*** is thrown.

10.31. In the **RIGHT** dialog: Change the name to **DOName2**.

10.32. Click in the **LEFT** dialog. A **transaction write conflict exception** is thrown.

10.33. Close the debugger window.

10.4.4. Abort a transaction context using TB

10.34. In the TB: Make sure that TrLevel1 of the the transaction context that contains the transacted changes is selected.

10.35. Click **Abort**. This aborts TrLevel1, which also aborts the context (a context is always aborted if TrLevel1 is aborted), which also aborts the changes that were made in the dialog. Note that the transaction context no longer exists in the TB. Note also that the dialog content has been updated:

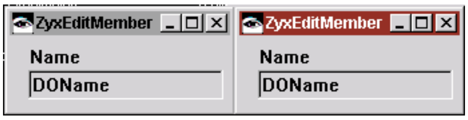


Figure 10.14. Aborted changes as reflected in the views

Note: Normally transacted changes are aborted or committed by clicking on a dialog button such as "Cancel" or "Accept changes". Adding such buttons to the dialog will be demonstrarted later in this tutorial.

10.4.5. Assign new object to transacted variable (in RIGHT dialog)

10.36. In the RIGHT dialog: Change the name to **’DOName2’**.

10.37. Click on the LEFT dialog. Note that the dialog was updated. The variable is now locked by the context for the RIGHT dialog (changing the variable in the LEFT dialog will throw a transaction write conflict exception).

10.4.6. Close the dialogs

10.38. Close the RIGHT **ZyxEditMember** dialog. Note in the **TB** that the transaction context no longer exists.

10.39. Attempt to close the LEFT **ZyxEditMember** dialog. Note that a ***"MicFwTransactionContext is not running"*** exception is thrown. This is because the context was aborted from the TB, not by clicking on a button in the dialog. Caution should be exerted when aborting or committing contexts or TrLevels within the TB.

10.40. Close the debugger window. The ZyxEditMember dialog cannot be closed.

11. Multiple views: transacted isolated

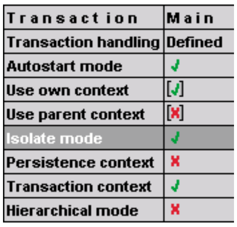
Sometimes you may not want the versioned objects from other dialogs to be displayed in a dialog (ie, you want only the version values to be displayed). This can be accomplished by specifying the Transaction Main for the DP to be in ***isolatedMode***. This chapter will explore isolated transactions.

In this chapter you will:

- Define DP ZyxEditMember as a transacted ISOLATED process using the DPB.
- Test transacted changes for isolated contexts.

11.1. Define ZyxEditMember as transacted ISOLATED process

11.1. In the **DPB**: For **eMPConn Main Transaction**: Set **Isolate** mode to **True** (click in column **Value** in the row **Isolate** to display radio buttons; select **True**):



Transaction	Main
Transaction handling	Defined
Autostart mode	✓
Use own context	[✓]
Use parent context	[✗]
Isolate mode	✓
Persistence context	✗
Transaction context	✓
Hierarchical mode	✗

Figure 11.1. Defining ZyxEditMember as an ISOLATED transacted process in the DPB

11.2. **Save all changes**.

11.3. Close the DPB.

11.4. Save the image.

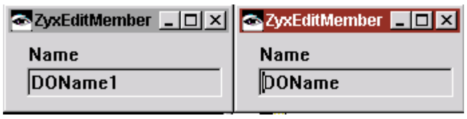
11.2. Test

11.5. In the workspace execute the following code:

```
FW CH  11. Multiple views: Transacted isolated."
[
  | aDO |
  aDO := ZyxMember new.
  aDO name: 'DOName'.
  ZyxEditMember openOn: aDO.
  ZyxEditMember openOn: aDO.
]
```

11.6. In the **LEFT** dialog: Change **name** to **DOName1**.

11.7. Click in the **RIGHT** dialog. The new string object is NOT displayed.



ZyxEditMember

Name: DOName1

ZyxEditMember

Name: DOName

Figure 11.2. Isolated transacted changes in multiple views

11.8. Close both **ZyxEditMember** dialogs.

12. Create database

In this chapter you will:

- Create a database using IBM DB Universal Database Version 5. This database will be used to create persistent objects.

NOTE: If you dont have DB2, you can download a 60-day trial version of **DB2 Version 5.2 Single User for Windows NT** from IBM for free at **www.ibm.com**. If you decide to use different database software, be sure to make appropriate changes throughout the rest of this tutorial.

Note: The actual steps for creating the database on your computer may differ, even if you are using Windows NT and DB2 Version 5.

12.1. Create database

12.1. From the **Start Menu**: Select **Programs / DB2 for Windows NT / Administration Tools / Control Center**. The following dialog appears:

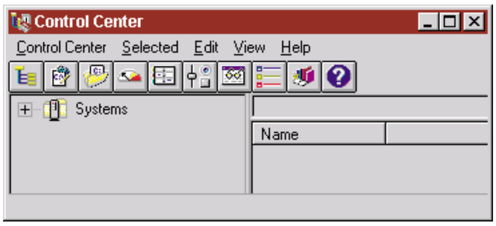


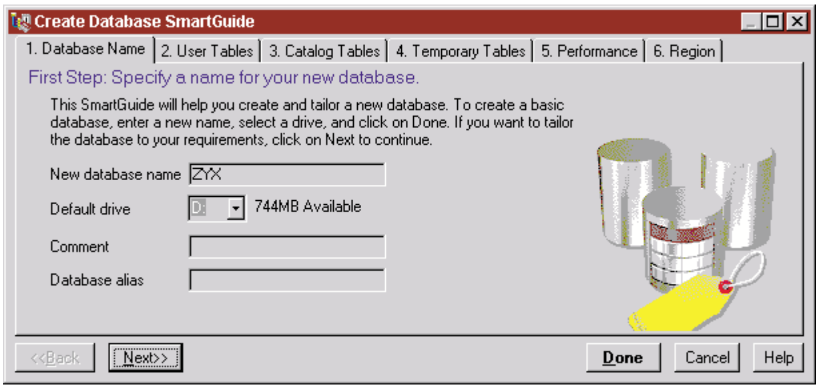
Figure 12.1. DB Control Center dialog

12.2. Expand the **Systems** tree until **Databases** appears.

12.3. Right-click on **Databases**.

12.4. Select **Create / New...**. The **Create database SmartGuide** dialog appears.

12.5. In the text field **New database name**: Enter **ZYX**:



First Step: Specify a name for your new database.

This SmartGuide will help you create and tailor a new database. To create a basic database, enter a new name, select a drive, and click on Done. If you want to tailor the database to your requirements, click on Next to continue.

New database name: ZYX

Default drive: D: 744MB Available

Comment:

Database alias:

Figure 12.2. Defining the new database name

12.6. Click **Done**. The database is created (creating the database may take several minutes):

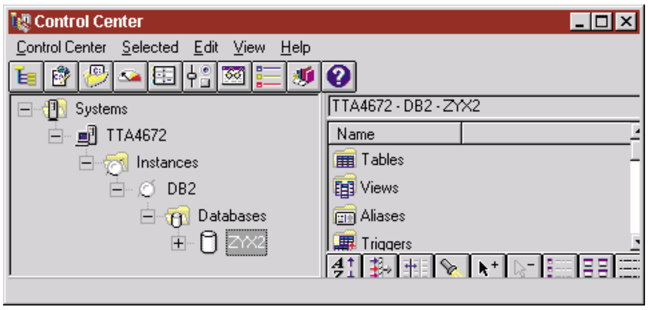
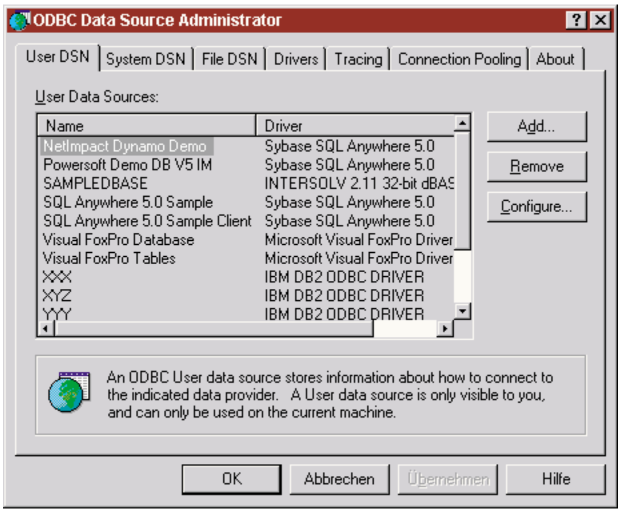


Figure 12.3. New DB2 database in the Control Center

12.2. Add database as machine datasource

12.7. From the **Start** menu: Select **Settings / System Controls**. The **System Controls** dialog appears.

12.8. Double-click on **ODBC Data Sources**. The **ODBC Data Source Administrator** dialog appears.



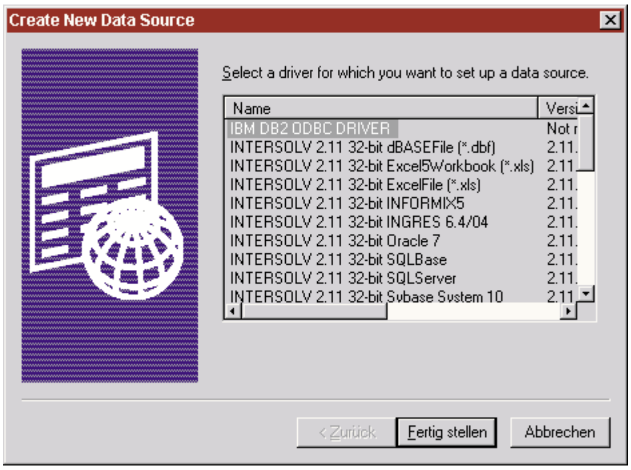
User Data Sources:

Name	Driver
NetImpact Dynamo Demo	Sybase SQL Anywhere 5.0
Powersoft Demo DB V5 IM	Sybase SQL Anywhere 5.0
SAMPLEDBASE	INTERSOLV 2.11 32-bit dBAS
SQL Anywhere 5.0 Sample	Sybase SQL Anywhere 5.0
SQL Anywhere 5.0 Sample Client	Sybase SQL Anywhere 5.0
Visual FoxPro Database	Microsoft Visual FoxPro Driver
Visual FoxPro Tables	Microsoft Visual FoxPro Driver
XXX	IBM DB2 ODBC DRIVER
XYZ	IBM DB2 ODBC DRIVER
YYY	IBM DB2 ODBC DRIVER

An ODBC User data source stores information about how to connect to the indicated data provider. A User data source is only visible to you, and can only be used on the current machine.

Figure 12.4. ODBC Data Source Administrator dialog

12.9. In the **User DSN** tab: Click **Add...**. The **Create New Datasource** dialog appears:



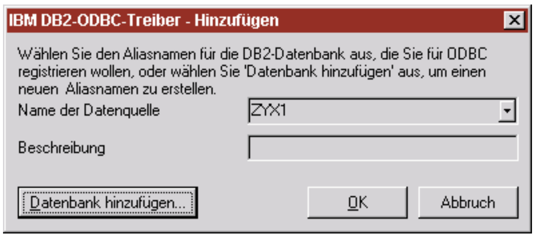
Select a driver for which you want to set up a data source.

Name	Versi
IBM DB2 ODBC DRIVER	Not r
INTERSOLV 2.11 32-bit dBASEFile (*.dbf)	2.11.
INTERSOLV 2.11 32-bit Excel5Workbook (*.xls)	2.11.
INTERSOLV 2.11 32-bit ExcelFile (*.xls)	2.11.
INTERSOLV 2.11 32-bit INFORMIX5	2.11.
INTERSOLV 2.11 32-bit INGRES 6.4/04	2.11.
INTERSOLV 2.11 32-bit Oracle 7	2.11.
INTERSOLV 2.11 32-bit SQLBase	2.11.
INTERSOLV 2.11 32-bit SQLServer	2.11.
INTERSOLV 2.11 32-bit Sybase System 10	2.11

Figure 12.5. Create New Datasource dialog

12.10. Click on **IBM DB2 ODBC DRIVER**.

12.11. Click **Continue**. The **IBM DB2-ODBC-Driver - Add** dialog appears.



Wählen Sie den Aliasnamen für die DB2-Datenbank aus, die Sie für ODBC registrieren wollen, oder wählen Sie 'Datenbank hinzufügen' aus, um einen neuen Aliasnamen zu erstellen.

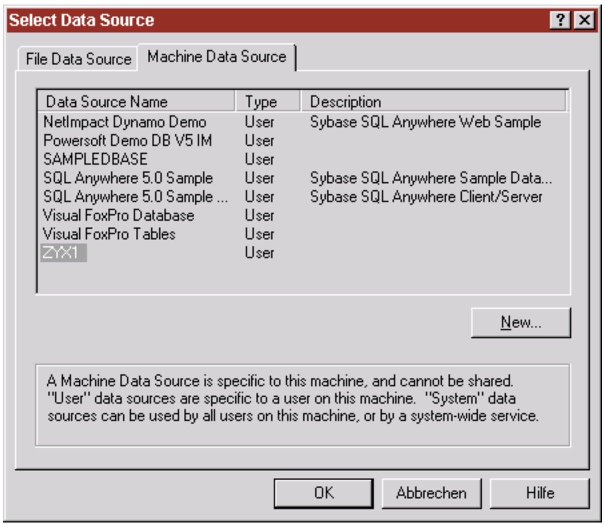
Name der Datenquelle: ZYX1

Beschreibung:

Figure 12.6. IBM DB2-ODBC-Driver add dialog

12.12. From the drop-down list **Datasource name**: Select **ZYX**.

12.13. Click **OK**.The **ODBC Data Source Administrator** dialog appears.



Data Source Name	Type	Description
NetImpact Dynamo Demo	User	Sybase SQL Anywhere Web Sample
Powersoft Demo DB V5 IM	User	
SAMPLEDBASE	User	
SQL Anywhere 5.0 Sample	User	Sybase SQL Anywhere Sample Data...
SQL Anywhere 5.0 Sample ...	User	Sybase SQL Anywhere Client/Server
Visual FoxPro Database	User	
Visual FoxPro Tables	User	
ZYX1	User	

A Machine Data Source is specific to this machine, and cannot be shared. "User" data sources are specific to a user on this machine. "System" data sources can be used by all users on this machine, or by a system-wide service.

Figure 12.7. ODBC Data Source Administrator dialog after datasource ZYX added

12.14. Click **OK**.

13. Making application ZyxTutorial persistent-capable

In this chapter you will:

- Add the prerequisite for application ZyxTutorial for support of object persistence with ODBC

13.1. In the **Visual Age Organizer**: Select the application **ZyxTutorial**.

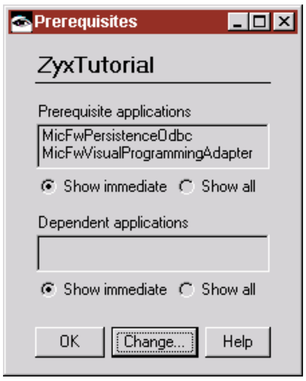
13.2. Select **Applications / Prerequisites**.

13.3. Click **Change**.

13.4. In the **Available** list: Select **MicFwPersistenceOdbc**.

13.5. Click **>>**.

13.6. Click **OK**.



ZyxTutorial

Prerequisite applications

MicFwPersistenceOdbc
MicFwVisualProgrammingAdapter

● Show immediate ○ Show all

Dependent applications

● Show immediate ○ Show all

Figure 13.1. Adding MicFwPersistenceOdbc as a ZyxTutorial prerequisite

13.7. Click **OK**.

14. Make DO ZyxMember persistent

In this chapter you will:

- In the ONB: Specify a variable type for ZyxMember>>name. A type is required for persistence-support (ie, the object referenced by ZyxMember>>name will be stored in a database, and a database requires a specific type). The type is specified with the ***Type Editor***. For more information about the Type Editor, see the ***Object Behavior Framework User's Guide*** chapter '3.4. Type Editor' (page 124)
- In the ONB: Specify ZyxMember>>name as persistent. This is required in order to create the necessary accessors for ZyxMember>>name that support persistence.
- In the ONB: Add variable ZyxMember>>id (type Integer, persistent). Each persistent ZyxMember instance must have at least 1 variable that serves as a unique identifier of the instance. This variable is know as the ***key variable*** for ZyxMember and will be the key for the table that contains persistent ZyxMember objects.

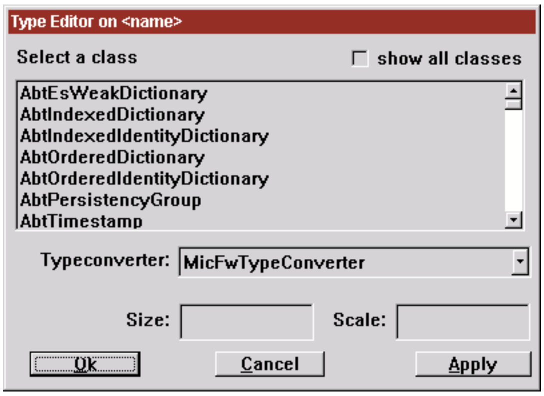
14.1. For ZyxMember>>name: Specify Type, Persistent (Transacted)

14.1. Open the **ONB** on **ZyxMember**.

14.2. Select **name**.

14.3. Right-click.

14.4. Select **Set type**. The **Type Editor on <name>** appears.

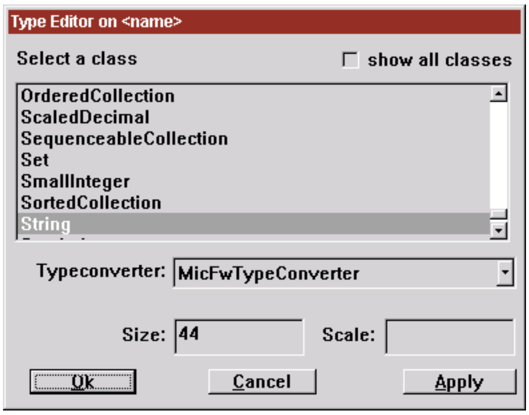


Typeconverter: MicFwTypeConverter

Figure 14.1. Type Editor dialog

14.5. Select class **String**.

14.6. In the field **Size**: Enter **44** (this value should be greater than the anticipated maximum length of a string entered for ZyxMember>>name).

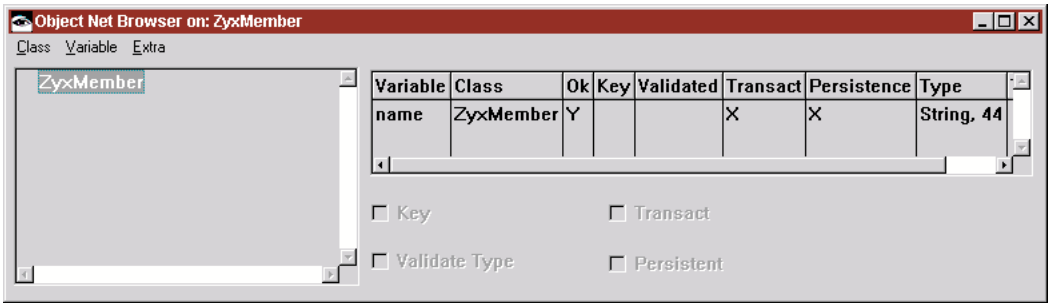


String
Typeconverter: MicFwTypeConverter
Size: 44

Figure 14.2. Type Editor class settings for ZyxMember>>name

IMPORTANT: The size of the string must be specified to avoid an error in the database.

14.7. Click **OK**. Note that the type of the variable is now defined in the ONB.



Variable	Class	Ok	Key	Validated	Transact	Persistence	Type
name	ZyxMember	Y			X	X	String, 44

□ Key □ Transact

□ Validate Type □ Persistent

Figure 14.3. ZyxMember>>name type information as displayed in ONB

14.8. Select **name**.

14.9. Check the checkbox **Persistent**. This specifies that name references persistent objects (String objects).

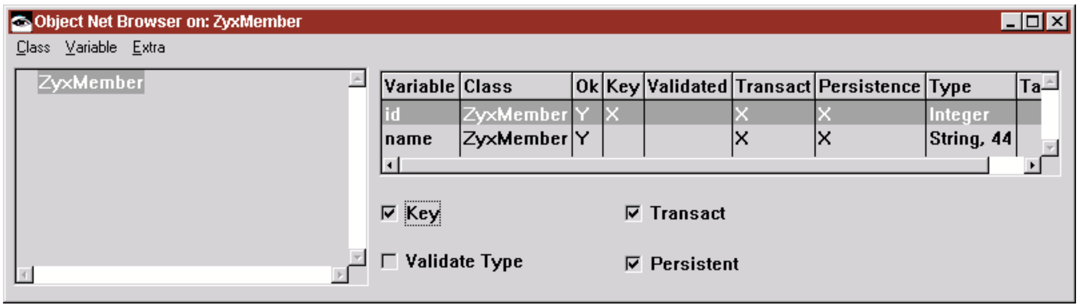
14.2. Add ZyxMember>>id, specify Type, Persistent (Transacted)

14.10. Add the variable **id** to ZyxMember.

14.11. For **id**: Check the checkbox **Key**. This specifies that id is the unique identifier for the ZyxMember instances. Note: A class must have at least 1 key variable in order to be made persistent.

14.12. Type **id** as **Integer** (leave **Size** and **Scale** blank).

14.13. For **id**: Check the checkbox **Persistent** (**Transacted** will be checked automatically). This specifies that id references persistent objects (Integer objects). The ONB dialog should be similar to the following:



Variable	Class	Ok	Key	Validated	Transact	Persistence	Type	Ta
id	ZyxMember	Y	X		X	X	Integer	
name	ZyxMember	Y			X	X	String, 44	

☑ Key ☑ Transact

□ Validate Type ☑ Persistent

Figure 14.4. ZyxMember>>id settings in the ONB

14.14. **Save the changes**.

14.15. Close the ONB.

14.16. Save the image.

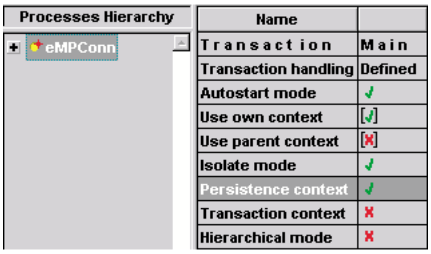
15. Make DP ZyxEditMember persistent

In this chapter you will:

- In the DP: Specify that DP ZyxEditMember supports ***persistent contexts***.
- Create method ZyxEditMember>>persistenceObjectManager (this method must be created if persistent transaction handling is selected for the process in the DPB). The significance of this method will be explained later.

15.1. Enable ZyxEditMember persistent contexts

15.1. In the **DPB** for **ZyxEditMember**: For **eMPConn Transaction Main**: Set **Persistence Context** to **True**.



Name	
Transaction	Main
Transaction handling	Defined
Autostart mode	✓
Use own context	[✓]
Use parent context	[✗]
Isolate mode	✓
Persistence context	✓
Transaction context	✗
Hierarchical mode	✗

Figure 15.1. Setting ZyxEditMember Transaction Main as persistent

15.2. Save the changes.

15.3. Close the DPB.

15.2. Create ZyxEditMember>>persistentObjectManager

15.4. Create the following method:

```
ZyxEditMember>>persistenceObjectManager
  | pom |
  pom := MicFwPersistenceManagerOdbc named: 'ZyxPom'.
  pom isConnected ifFalse: [ pom connectWithDriverComplete ].
  ^pom.
```

15.5. Save the image.

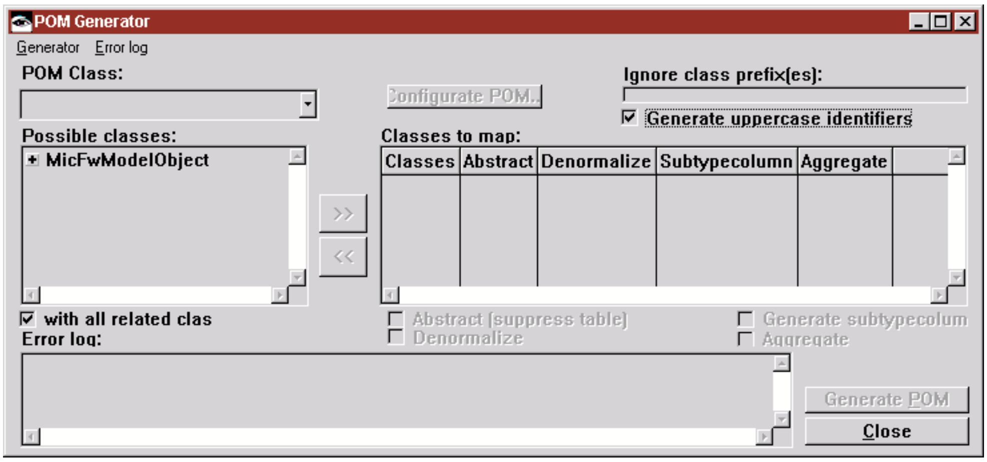
16. Create POM

In this chapter you will:

- Configure the ***POM Generator***: The POM Generator is used to create a ***POM*** (Persistent Object Manager) for specified persistent classes and the specified type of database. A POM manages the interface between the instantiations of a persistent class and the database. For more information about the POM Generator, see the ***Persistence Framework User's Guide*** chapter '10.3. The POM Generator Tool' (page 221).
- Store the POM Generator settings globally.
- Generate the POM "ZyxPom".
- View the contents of the POM (mappings between the persistent objects and the database).
- Store the POM.

16.1. Configure POM Generator settings

16.1. From the **System Transcript**: Select **micFrameworks / Persistence Tools / Object Net -> POM Generator**. The POM Generator dialog is opened.



Possible classes: + MicFwModelObject

☑ Generate uppercase identifiers

☑ with all related clas

Figure 16.1. POM Generator dialog

16.1.1. Add ZyxMember to the POM possible classes list

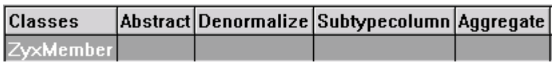
The classes that will be managed by the POM must specified.

16.2. In the **Possible classes:** box: Double-click on **MicFwModelObject**.

16.3. In the **Possible classes:** box: Double-click on **MicFwDomainObject**.

16.4. In the **Possible classes:** box: Select on **ZyxMember**.

16.5. Click **>>**.



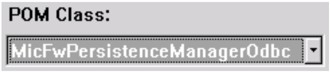
Classes	Abstract	Denormalize	Subtypecolumn	Aggregate
ZyxMember				

Figure 16.2. Selecting ZyxMember in the POM possible classes list

16.1.2. Select the POM class

The ***POM class*** determines what type of database the POM will interface with.

16.6. From the **POM Class:** drop-down list: Select **MicFwPersistenceManagerOdbc**.



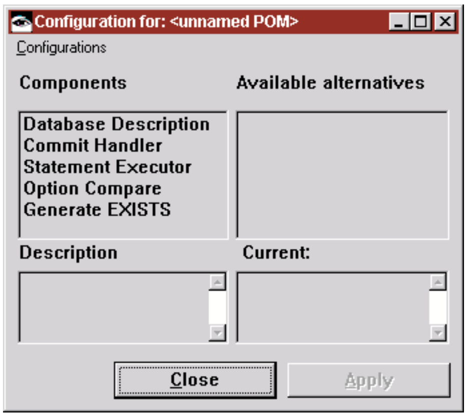
POM Class: MicFwPersistenceManagerOdbc

Figure 16.3. Selecting the POM class

16.1.3. Configure the POM

The POM must be configured for the particular version of the database.

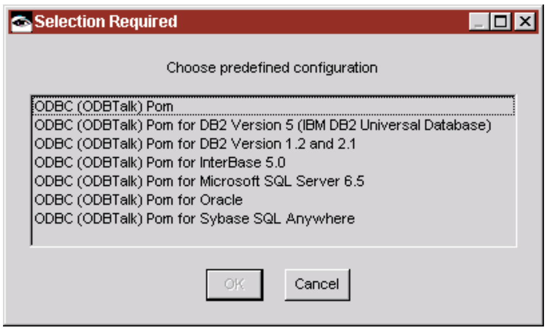
16.7. Click on **Configurate POM...**. The **Configuration for: <unnamed POM>** dialog appears:



Database Description
Commit Handler
Statement Executor
Option Compare
Generate EXISTS

Figure 16.4. Configurate POM dialog

16.8. From the menu: Select **Configurations / Select...**. The following dialog appears:



Choose predefined configuration

ODBC (ODBTalk) Pom
ODBC (ODBTalk) Pom for DB2 Version 5 (IBM DB2 Universal Database)
ODBC (ODBTalk) Pom for DB2 Version 1.2 and 2.1
ODBC (ODBTalk) Pom for InterBase 5.0
ODBC (ODBTalk) Pom for Microsoft SQL Server 6.5
ODBC (ODBTalk) Pom for Oracle
ODBC (ODBTalk) Pom for Sybase SQL Anywhere

Figure 16.5. Dialog for selecting a POM configuration

16.9. Select **ODBC (ODBTalk) Pom for DB2 Version 5 (IBM DB2 Universal Database)**.

16.10. Click **OK**.

16.11. Click **Close**.

16.1.4. Store the POM Generator settings globally as ZyxPomGenerator

The settings for the POM Generator can be stored and then used again later.

16.12. In the **POM Generator** dialog: Select **Generator / Store globally as...**.

16.13. In the **Enter name** dialog: Enter **ZyxPomGenerator**.

16.2. Generate POM

When the POM is generated the ***STOPF / ODBC*** tool is opened. The meaning of STOPF (Smalltalk Object Persistence Framework) is historic in nature.

16.14. Click on **Generate POM**. The **STOPF / ODBC** dialog appears:

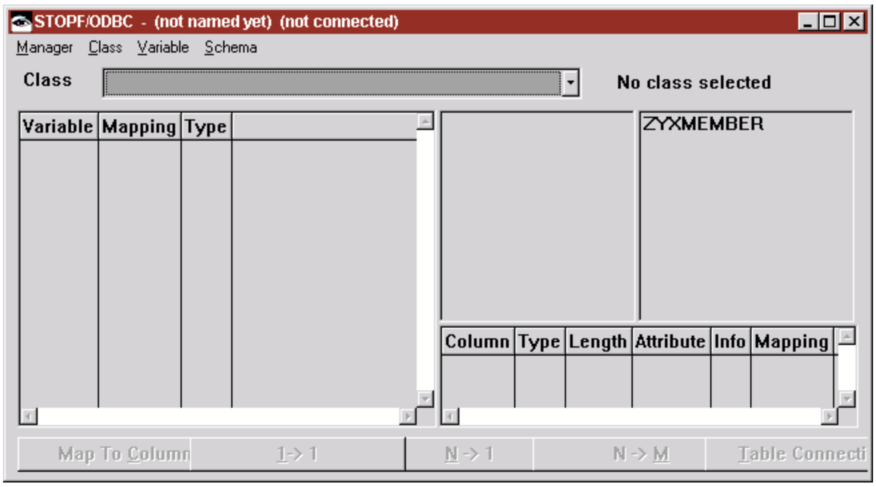


Figure 16.6. STOPF / ODBC dialog

16.2.1. Close the POM Generator

16.15. In the POM Generator dialog: Click **Close**.

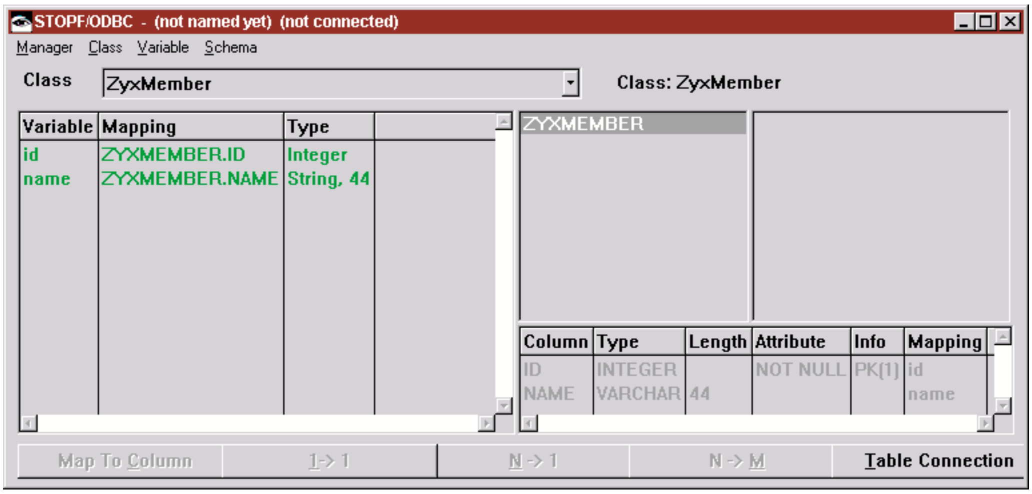
16.16. In response to **Store Generator settings**: Click **No** (the settings were just saved above).

16.2.2. View tables defined within POM

The ***object mapping*** between the persistent classes and the database tables can be viewed in the STOPF.

16.17. From the **Class:** drop-down list: Select **ZyxMember**. Information about the class variables is displayed.

16.18. Click on table **ZYXMEMBER** (in the upper-right box). Information about the tables is displayed.



Class: ZyxMember

Class: ZyxMember

Variable	Mapping	Type
id	ZYXMEMBER.ID	Integer
name	ZYXMEMBER.NAME	String, 44

ZYXMEMBER

Column	Type	Length	Attribute	Info	Mapping
ID	INTEGER		NOT NULL	PK[1]	id
NAME	VARCHAR	44			name

Figure 16.7. Viewing table information for tables defined within POM

16.3. Store the POM

16.19. From the **STOPF / ODBC** menu: Select **Manager / Store Globally As...**.

16.20. In response to **Enter global name for persistence manager**: Enter **ZyxPom**.

16.21. Click **OK**. Notice the name of the POM in the STOPF / ODBC dialog title bar.

16.22. Save the image.

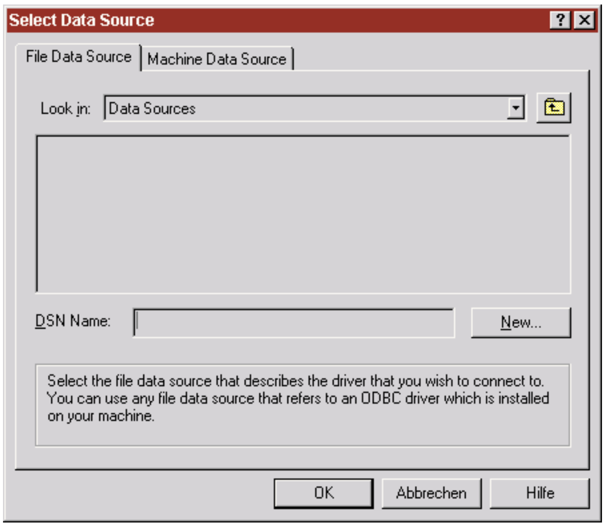
17. Generate table ZYXMEMBER using POM ZyxPom

In this chapter you will:

- Connect the POM to database ZYX using the ***STOPF / ODBC*** tool. For more information about the STOPF / ODBC tool, see the ***Persistence Framework User's Guide*** chapter '10.2. The STOPF tool' (page 196).
- Generate ***DDL*** (Data Definition Language) code for database ZYX using the POM. The DDL defines the structure of the database for storing persistent objects in machine- and human-readable form.
- Create a table in the ZYX database by executing the DDL code using the ***ISQL*** tool. For more information about the ISQL tool, see the ***Persistence Framework User's Guide*** chapter '10.4. Interactive SQL' (page 225).

17.1. Connect POM to database

17.1. From the **STOPF/ODBC** dialog menu: Select **Manager / Connect to database**. The **Select Data Source** dialog appears:



Look in: Data Sources

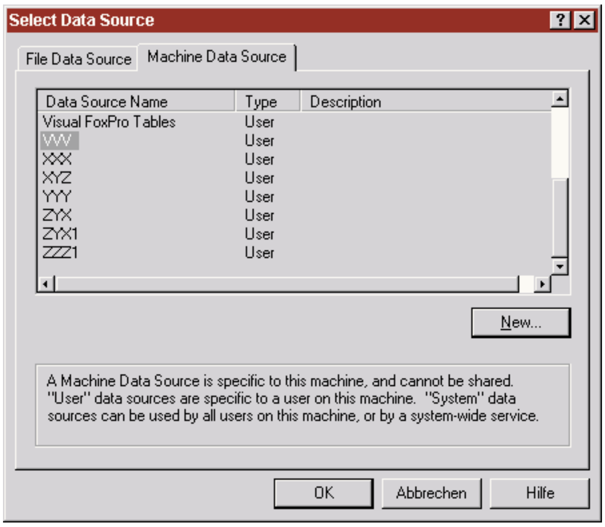
DSN Name:

Select the file data source that describes the driver that you wish to connect to. You can use any file data source that refers to an ODBC driver which is installed on your machine.

Figure 17.1. Connecting POM to a database: Select Data Source dialog.

17.2. Select the **Machine Data Source** tab.

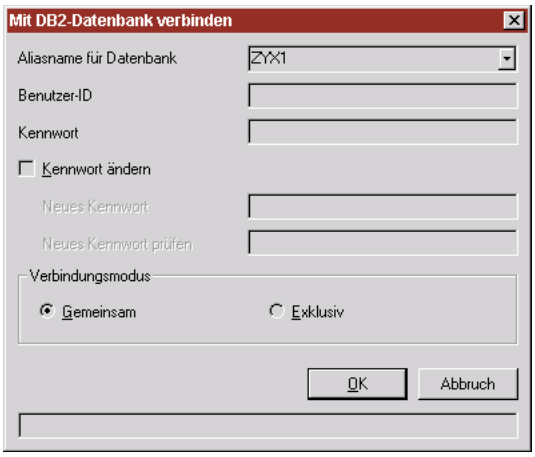
17.3. Select **ZYX**.



Data Source Name	Type	Description
Visual FoxPro Tables	User	
VVV	User	
XXX	User	
XYZ	User	
YYY	User	
ZYX	User	
ZYX1	User	
ZZZ1	User	

A Machine Data Source is specific to this machine, and cannot be shared. "User" data sources are specific to a user on this machine. "System" data sources can be used by all users on this machine, or by a system-wide service.

17.4. Click **OK**. The **Connect to DB2 Database** dialog appears.



Mit DB2-Datenbank verbinden

Aliasname für Datenbank: ZYX1

Benutzer-ID:

Kennwort:

□ Kennwort ändern

Neues Kennwort:

Neues Kennwort prüfen:

Verbindungsmodus: ● Gemeinsam ○ Exklusiv

Figure 17.2. Connect to DB2 Database dialog

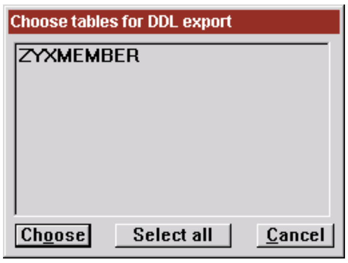
17.5. Enter your **User-ID** (typically db2admin).

17.6. Enter your **Password** (typically db2admin).

17.7. Click **OK**. "connected" appears at the top of the STOPF/ODBC dialog.

17.2. Generate DDL

17.8. From the **STOPF / ODBC** menu: Select **Schema / Export**. The following dialog appears:



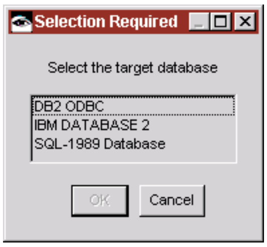
Choose tables for DDL export

ZYXMEMBER

Figure 17.3. Dialog for choosing tables for DDL export

17.9. Select (click on) **ZyxMember**.

17.10. Click **Choose**. The following dialog appears:



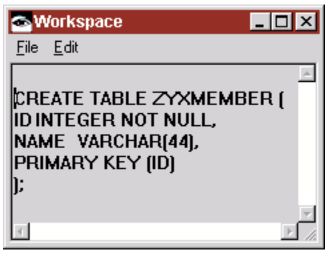
Selection Required

Select the target database

DB2 ODBC
IBM DATABASE 2
SQL-1989 Database

Figure 17.4. Dialog for selecting target datbase for DDL export

17.11. Double-click on **IBM DATABASE 2**. A Workspace window is opened which contains the DDL code (plus a ";" semicolon character at the end) required to create the needed tables:



```
CREATE TABLE ZYXMEMBER (
ID INTEGER NOT NULL,
NAME  VARCHAR(44),
PRIMARY KEY (ID)
);
```

Figure 17.5. DDL code for creating tables

17.3. Execute DDL

17.12. From the **System Transcript** menu: Select **micFrameworks / Persistence Tools / Interactive SQL**. The Interactive SQL dialog is opened.

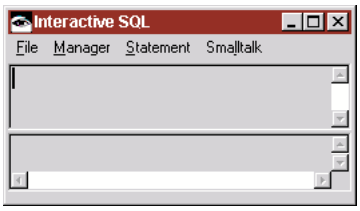
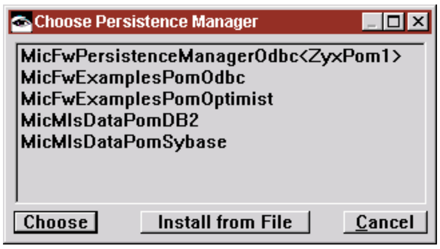


Figure 17.6. Interactive SQL dialog

17.3.1. Select POM

17.13. Select **Manager / Choose...**. The following dialog appears:



MicFwPersistenceManagerOdbc<ZyxPom1>
MicFwExamplesPomOdbc
MicFwExamplesPomOptimist
MicMlsDataPomDB2
MicMlsDataPomSybase

Figure 17.7. Dialog for selecting POM for ISQL dialog

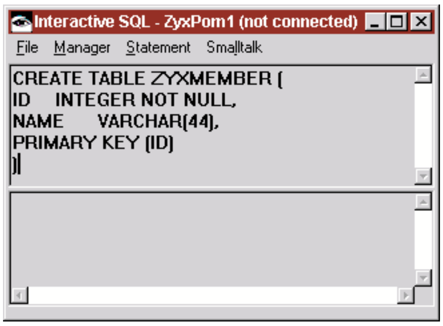
17.14. Select **MicFwPersistenceManagerOdbc<ZyxPom>**.

17.15. Click **Choose**.

17.3.2. Execute DDL against database to create ZYX:ZYXMEMBER

17.16. Copy the **DDL code** from the **Workspace** to the **Interactive SQL** dialog **upper** box.

17.17. Delete the **last semicolon (";")** from the DDL code:



```
CREATE TABLE ZYXMEMBER (
ID    INTEGER NOT NULL,
NAME      VARCHAR(44),
PRIMARY KEY (ID)
)
```

Figure 17.8. DDL code in the ISQL dialog (with trailing colon deleted)

17.18. In the **Interactive SQL** dialog: Select **Statement / Execute once**. The message "*** ready ***" appears.

17.19. Select **Statement / Commit**. The message "*** commit ***" appears. The table ZYXMEMBER has been created.

17.20. Close the Workspace that contains the SQL commands.

17.21. Close the ISQL dialog.

17.22. In response to "Disconnect the POM?": Click **No**.

17.23. Close the STOPF dialog.

17.24. In response to "Disconnect the POM?": Click **No**.

17.25. Save the image.

18. Creating , displaying, editting persistent DO’s in a table using Smalltalk

In this chapter you will:

- Create persistent objects in table ZYXMEMBER (using Smalltalk). The table ZYXMEMBER currently contains no rows, since no objects have been stored to the table.
- Display the data in the table (using DB2).
- Display the data in the table (using Smalltalk) with the ZyxEditMemberView.
- Edit the persistent objects in the table (using Smalltalk) using the ZyxEditMemberView.

18.1. Creating persistent objects in table ZYXMEMBER (using Smalltalk)

18.1. Execute the following code to store 2 instances to the table (note: you may have to reconnect to the database).

```
FW CH 18. Create 2 ZyxMember instances in table ZYXMEMBER."
[
  | pom con mem |
  pom := MicFwPersistenceManagerOdbc named: 'ZyxPom'.
  pom isConnected ifFalse: [ pom connectWithDriverComplete ].
  con := pom newPersistenceContext.
  con beginTransaction.
  mem := ZyxMember newPersistent.
  mem id: 1.
  mem name: 'memberWithId1'.
  mem := ZyxMember newPersistent.
  mem id: 2.
  mem name: 'memberWithId2'.
  con commitTransaction.
]
```

18.2. View data in table (using DB2)

18.2.1. Disconnect the POM manager from the database

IMPORTANT: The POM must be disconnected from the database. If not disconnected from the database, the POM will have a lock on the database and the Sample Contents will not be displayed.

18.2. From the **System Transcript**: Select **micFrameworks / Browse Persistence Manager...**.

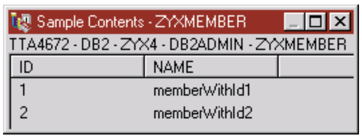
18.3. In the **Choose Persistence Manager** dialog: Double-click on **MicFwPersistenceManagerOdbc<ZyxPom>**. The **STOPF/ODBC ZyxPom (connected)** dialog appears.

18.4. From the STOPF/ODBC dialog: Selected **Manager / Disconnect from database**.

18.2.2. Display Sample contents for ZYXMEMBER.

18.5. In the **Control Center**: Right-click on table **ZYXMEMBER**.

18.6. Select **Sample contents**. Note the contents of the database:



Sample Contents - ZYXMEMBER

TTA4672 - DB2 - ZYX4 - DB2ADMIN - ZYXMEMBER

ID	NAME
1	memberWithId1
2	memberWithId2

Figure 18.1. Sample contents of ZyxMember in database

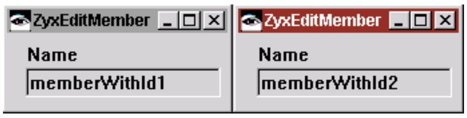
18.3. Load ZyxMember instances from table ZYXMEMBER (using Smalltalk)

18.7. Execute the following code to load the 2 ZyxMember instances from the database and display in a ZyxEditMemberView dialog (you will have to connect to the database).

```
FW CH 18. Load 2 ZyxMember instances from table ZYXMEMBER."
[
  | pom con all mem |
  pom := MicFwPersistenceManagerOdbc named: 'ZyxPom'.
  pom isConnected ifFalse: [ pom connectWithDriverComplete ].
  con := pom newPersistenceContext.
```

```
    con beginTransaction.
    all := pom getAllInstancesOf: ZyxMember.
    con commitTransaction.
    ZyxEditMember openOn: all first.
    ZyxEditMember openOn: all last.
]
```

The following dialogs are opened:



Name
memberWithId1

Name
memberWithId2

Figure 18.2. ZyxEditMemberView views of 2 objects

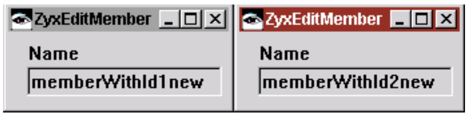
Note that each view is of a different object. Making changes in one dialog does not affect the contents of the other dialog.

18.4. Edit the persistent objects in the table (using Smalltalk)

18.8. Change the names in the dialogs to **memberWithId1new** and **memberWithId2new**.

18.9. Close both dialogs.

18.10. Execute the above code again. Note that the previous changes were saved to the database:

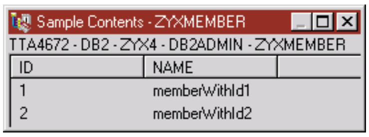


Name
memberWithId1new

Name
memberWithId2new

Figure 18.3. ZyxEditMemberView's show that changes were saved to the database

18.11. View the Sample Contents for table ZYX:ZYXMEMBER. Note that the changes have been saved to the database.



TTA4672 - DB2 - ZYX4 - DB2ADMIN - ZYXMEMBER

ID	NAME
1	memberWithId1
2	memberWithId2

Figure 18.4. Changes made in dialogs are saved to the database

19. Add commit / abort buttons to ZyxEditMemberView

ZyxEditMember is a subclass of MicFwDomainProcess, which provides standard buttons for aborting and committing transacted changes with (...AndCloseView) or without (...AndBegin) closing the view. These buttons can be added to the view with the Quick Form option.

In this chapter you will:

- Add ***commitAndBegin*** and ***abortAndBegin*** buttons to ZyxEditMemberView. These buttons will commit / abort the transacted changes that have been entered in the view without commiting / aborting the transaction context for the view (and without closing the view).
- Add ***commitAndCloseView*** and ***abortAndCloseView*** buttons to ZyxEditMemberView. These buttons will commit / abort the transacted changes that have been entered in the view AND commit / abort the transaction context for the view AND close the view.
- Test the buttons on the persistent objects in table ZYXMEMBER.

Note: To copy parts in the CE:

- Press and hold down the CTRL key.
- Left-click and hold the mouse on the part to be copied.
- Move the mouse pointer to the location that the copied part should be copied to.
- Release the mouse button and CTRL key.

19.1. Add commit / abort buttons

19.1.1. Add commitAndBegin button

19.1. Open the **Composition Editor** on **ZyxEditMemberView** (double-click on ZyxEditMemberView in the Visual Age Organizer).

19.2. Add a **Push Button** to **ZyxEditMemberView**.

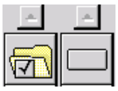


Figure 19.1. Push button part in the parts palette

19.3. Change the **Push Button** property **object** to **commitAndBegin**.

19.4. Change the **Push Button** property **partName** to **commitAndBegin_**.

19.1.2. Add abortAndBegin button

19.5. Add a **Push Button** to **ZyxEditMemberView**.

19.6. Change the **Push Button** property **object** to **abortAndBegin**.

19.7. Change the **Push Button** property **partName** to **abortAndBegin_**.

19.1.3. Add commitAndCloseView button

19.8. Add a **Push Button** to **ZyxEditMemberView**.

19.9. Change the **Push Button** property **object** to **commitAndCloseView**.

19.10. Change the **Push Button** property **partName** to **commitAndCloseView_**.

19.1.4. Add abortAndCloseView button

19.11. Add a **Push Button** to **ZyxEditMemberView**.

19.12. Change the **Push Button** property **object** to **abortAndCloseView**.

19.13. Change the **Push Button** property **partName** to **abortAndCloseView_**.

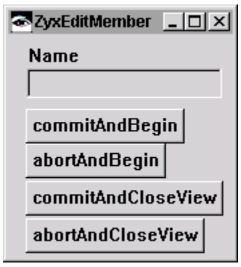


Figure 19.2. ZyxEditMemberView with commit and aborts buttons (4)

19.14. From the **Composition Editor menu**: Select **File / Save Part**.

19.15. Close the **Composition Editor**.

19.16. Save the image.

19.2. Test

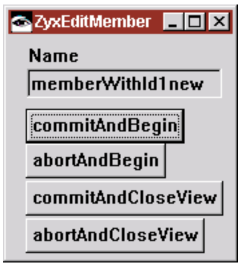
19.2.1. Commiting / aborting changes without closing the view

19.2.1.1. Open view of persistent object

19.17. Execute the following code to load the first instance of ZyxMember from the database and display in a dialog.

```
FW CH 19. Load 1 ZyxMember instance from table ZYXMEMBER."
[
  | pom con all mem |
  pom := MicFwPersistenceManagerOdbc named: 'ZyxPom'.
  pom isConnected ifFalse: [ pom connectWithDriverComplete ].
  con := pom newPersistenceContext.
  con beginTransaction.
  all := pom getAllInstancesOf: ZyxMember.
  con commitTransaction.
  ZyxEditMember openOn: all first.
]
```

The following dialog is opened:



Name: memberWithId1new

Figure 19.3. ZyxEditMemberView with commit / abort buttons

19.2.1.2. Make transacted change to persistent object in view

19.18. Change the name to **memberWithId1v2**.

19.19. Open the **TB**. Note the transaction contents:

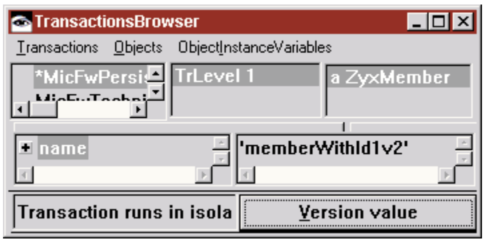


Figure 19.4. Version value (uncommitted target) for ZyxMember>>name in the TB

19.20. View the databank **Sample contents**. Note that the change has NOT been stored to the database.

19.2.1.3. Commit transacted change without closing the view

19.21. Click **commitAndBegin**.

19.22. Click in the **TB**. Note that the transacted changes have been implemented:

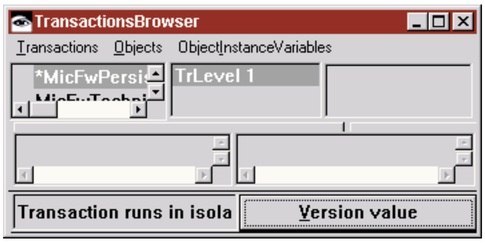

TransactionsBrowser
Transactions Objects ObjectInstanceVariables
*MicFwPersi
TrLevel 1
Transaction runs in isola
Version value

Figure 19.5. Version value (none) after change committed in view

19.23. View the databank **Sample contents**. Note that the change has been stored to the database.

19.2.1.4. Make transacted change to persistent object in view

19.24. Change the name to **memberWithId1v3**.

19.25. Open the **TB**.

19.26. Select **Transactions / Update**. Note the transaction contents.

19.2.1.5. Abort transacted changes without closing the view

19.27. Click **abortAndBegin**. Note that **memberWithId1v2** is displayed.

19.28. Click in the **TB**. Note that the transacted changes have been aborted.

19.29. View the databank **Sample contents**. Note that the change has NOT been stored to the database.

19.2.2. Commiting / aborting changes and closing the view

19.30. Perform the tests as above, except use the **commitAndCloseView** and the **abortAndCloseView** buttons.

20. Adding a variable to a persistent class

Adding a variable to a persistent class involves:

- Adding the variable to the class in the ONB.
- Modifying the views that will display the variable contents.
- Modifying the POM.
- Modifying the database (regenerating the database would require first deleting the database, resulting in the loss of persistent objects stored in the database).

In this chapter you will:

- In the ONB: Add ZyxMember>>weight, which represents the weight of a member.
- In the CE: Add field "weight" to ZyxEditMemberView
- Regenerate POM ZyxPom.
- In DB2: Add column WEIGHT to database ZYX table ZYXMEMBER.
- Test.

20.1. Add ZyxMember variable weight

20.1. In the **ONB**: Add the variable **ZyxMember>>weight** with:

- Type **Integer** (leave **Size** and **Scale** blank).
- Check **Persistent** (**Transaction**).

20.2. Save the changes.

20.3. Close the ONB.

20.2. Add text field for weight to ZyxEditMemberView

20.4. Add a **label** to **ZyxEditMemberView**.

20.5. Change the **label** property **object** to **weight**.

20.6. Add a **text field** to the view.

20.7. Change the **text field** property **partName** to **weight_**.

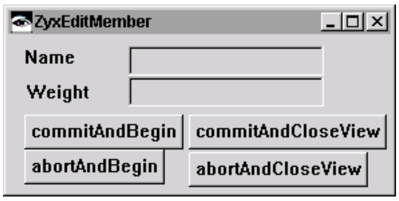


Figure 20.1. Adding weight field to ZyxEditMemberView

Note: The parts in a view may be moved within the view several times throughout this tutorial.

20.8. Select **File / Save Part**.

20.9. Close the CE.

20.3. Regenerate POM

20.3.1. Reload POM Generator

The POM Generator settings were saved earlier under the global name ZyxPomGenerator.

20.10. From the **System Transcript**: Select **micFrameworks / Persistence Tools / Object Net -> POM Generator**. The POM Generator dialog appears.

20.11. Select **Generator / Choose**.

20.12. In response to **Choose a stored POM Generator**: Double-click on **ZyxPomGenerator**.

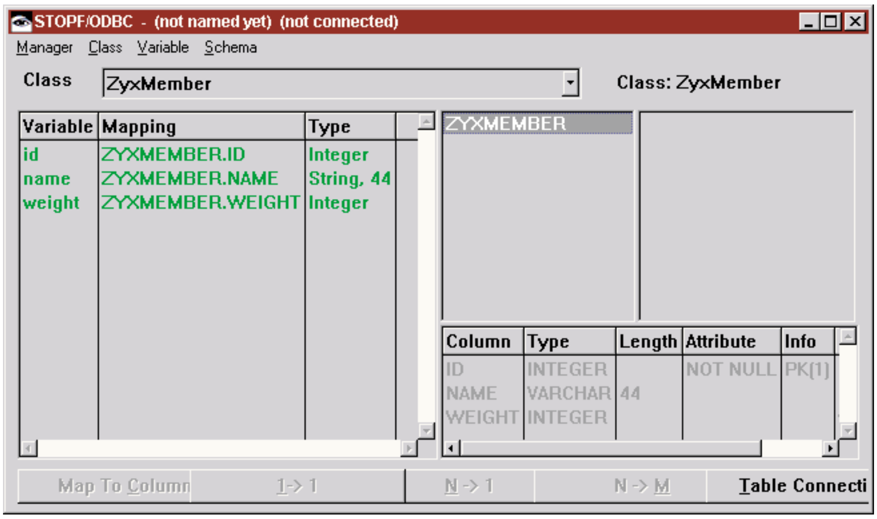
20.3.2. Generate POM

20.13. Click on **Generate POM**. The **STOPF/ODBC** tool opens.

20.3.2.1. View the new mapping

20.14. In the **STOPF/ODBC**: Select **ZyxMember** from the **Class** drop-down list.

20.15. Click on **ZYXMEMBER** in the upper right-window. Note that the POM mapping has been updated for the new column in the table.


STOPF/ODBC - (not named yet) (not connected)
Manager Class Variable Schema
Map To Column
1-> 1
N -> 1
N -> M
Table Connecti

Class ZyxMember

Class: ZyxMember

Variable	Mapping	Type
id	ZYXMEMBER.ID	Integer
name	ZYXMEMBER.NAME	String, 44
weight	ZYXMEMBER.WEIGHT	Integer

ZYXMEMBER

Column	Type	Length	Attribute	Info
ID	INTEGER		NOT NULL	PK(1)
NAME	VARCHAR	44		
WEIGHT	INTEGER			

Figure 20.2. New mapping in the STOPF / ODBC dialog for ZyxMember

20.3.3. Store the POM

20.16. From the **STOPF / ODBC** menu: Select **Manager / Store Globally As...**.

20.17. In response to **Enter global name for persistence manager**: Enter **ZyxPom**.

20.18. Click **OK**. Notice the name of the POM in the STOPF / ODBC dialog title bar.

IMPORTANT: The previous POM was overwritten with NO warning. Be careful not to inadvertantly overwrite a POM.

20.19. Close the **POM Generator** and **STOPF/ODBC** dialogs.

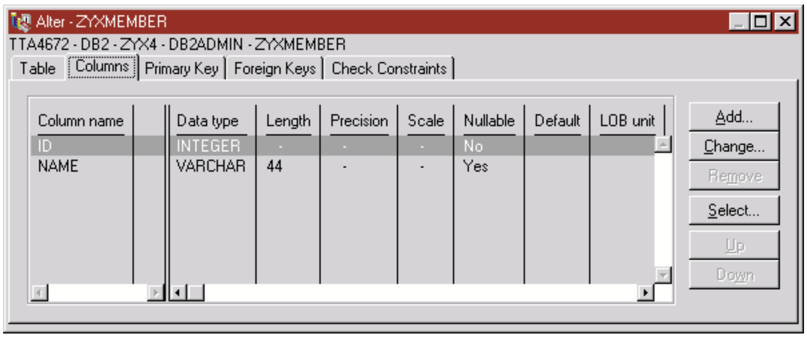
20.20. Save the image.

20.4. Add column WEIGHT to ZYX:ZYXMEMBER

Note: The POM should not be connected to database ZYX.

20.21. In the **DB2 Control Center**: Double-click on table **ZYXMEMBER** (in database **ZYX**). The **Alter - ZYXMEMBER** dialog is opened.

20.22. Select tab **Columns**.



TTA4672 - DB2 - ZYX4 - DB2ADMIN - ZYXMEMBER

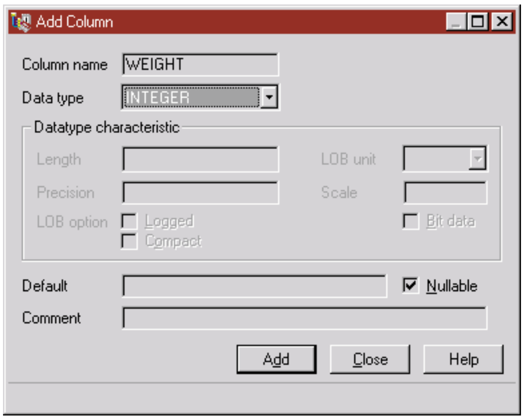
Column name	Data type	Length	Precision	Scale	Nullable	Default	LOB unit
ID	INTEGER	-	-	-	No		
NAME	VARCHAR	44	-	-	Yes		

Figure 20.3. Tab Columns in the Alter ZYXMEMBER dialog

20.23. Click **Add**. The dialog **Add Column** opens.

20.24. In the **Column name** field: Enter **WEIGHT**.

20.25. From the **Data type** Drop-down list: Select **INTEGER**.



Column name WEIGHT

Data type INTEGER

☑ Nullable

Figure 20.4. Selecting the data type for the added column

20.26. Click **Add**.

20.27. Click **Close**.

20.28. In the **Alter - ZYXMEMBER** dialog: Click **OK**. The column is added to the table.

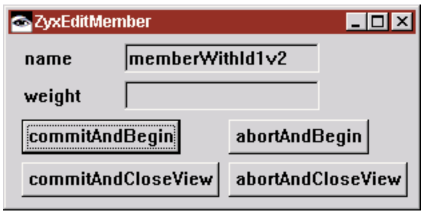
20.5. Test

20.5.1. Open view of persistent object

20.29. Execute the following code to load the first instance of ZyxMember from the database and display in a dialog.

```
FW CH 20. Load 1 ZyxMember instance from table ZYXMEMBER."
[
  | pom con all mem |
  pom := MicFwPersistenceManagerOdbc named: 'ZyxPom'.
  pom isConnected ifFalse: [ pom connectWithDriverComplete ].
  con := pom newPersistenceContext.
  con beginTransaction.
  all := pom getAllInstancesOf: ZyxMember.
  con commitTransaction.
  ZyxEditMember openOn: all first.
]
```

The following dialog is opened:



name memberWithId1v2

weight

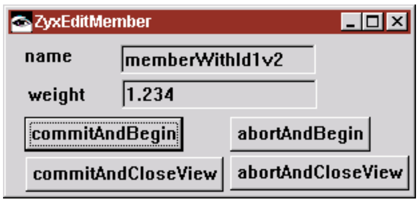
Figure 20.5. ZyxEditMember view with weight label / text field

20.5.2. Enter object of valid type (Integer) for weight

Entering a weight of valid type (such as 1234) and clicking commitAndBegin will result in the entered data being stored to the database.

20.30. Change **weight** to **1234**.

20.31. Click on **commitAndBegin**.

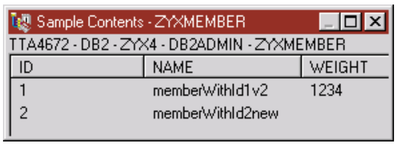


name memberWithId1v2

weight 1.234

Figure 20.6. Changing weight to a valid value in ZyxEditMemberView

20.32. View the databank **Sample contents**. Note that the change has been stored to the database.



TTA4672 - DB2 - ZYX4 - DB2ADMIN - ZYXMEMBER

ID	NAME	WEIGHT
1	memberWithId1v2	1234
2	memberWithId2new	

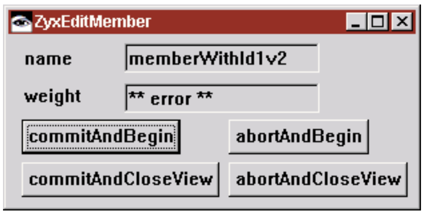
Figure 20.7. New weight stored to database

20.5.3. Enter object of invalid type (String) for weight

Entering a weight of invalid type (such as ’abc’) and clicking commitAndBegin will result in the entered data not being stored to the database.

20.33. Change **weight** to **abc**.

20.34. Click on **commitAndBegin**.



name: memberWithId1v2

weight: ** error **

Figure 20.8. Attempting to change weight to an invalid value in ZyxEditMemberView

20.35. View the databank **Sample contents**. Note that **weight** = **1234**.

21. Validate range of entered data in ZyxEditMemberView

A value of the proper type entered in a view field can be rejected if it is not within a range specified in the DO class method ***initializeValidation***. For more information about validation, see the ***Application Framework User's Guide*** chapter '4.7. Validation' (page 197).

In this chapter you will:

- Create method ZyxMember class>>initializeValidation
- Change the transaction main context for ZyxEditMember to non-persistent (so that the database is not used).
- Test.

21.1. Add ZyxMember class>>initializeValidation

21.1. Create the following method:

```
ZyxMember class>>initializeValidation
  self validateWrite: #weight
    using: [ :value | (value notNil and: [ value > 20 ] )
    and: [ value < 500 ] ]
```

21.2. Save the class.

21.2. Change ZyxEditMember Transaction main to non-persistent context

21.3. In the **DPB** for **ZyxEditMember**: For **Transaction Main**: Deselect **Persistence context**.

21.4. Save all changes.

21.5. Close the DPB.

21.6. Save the image.

21.3. Test

21.7. Execute the following code in the workspace (note the line for initializeValidation):

```
FW CH 21. Validate range of ZyxMember>>weight."
[
  | aDO |
  ZyxMember initializeValidation.
  aDO := ZyxMember new.
  aDO name: 'DOName'.
  aDO weight: 200.
  ZyxEditMember openOn: aDO.
]
```

Note that **weight** = 200 (within the valid range).

21.8. Enter **20** for the weight (min = 21).

21.9. Click in the "name" text field. Note that weight reverts to 200.

21.10. Enter **21**.

21.11. Click in the "name" text field. Note that weight stays at 21.

21.12. Enter **500** (max = 499).

21.13. Click in the "name" text field. Note that weight stays at 21.

21.14. Enter **499**.

21.15. Click in the "name" text field. Note that weight stays at 499.

21.16. Close the view.

22. Create DO ZyxClub

In this chapter you will:

- Create DO ZyxClub.
- Create ZyxClub>>members (transacted, persistent) with variable "type" as a collection of any number of ZyxMember instances using the ***Relationship Editor*** (RE) (for more information about the Relationship Editor, see the ***Object Behavior Framework User's Guide*** chapter '3.5. Relationship Editor' (page 125)) to:
  - Establish a ***relationship*** between ZyxClub>>members and ZyxMember. The relationship will be specified as being a ***primitive relationship***, which means that the members attribute will reference a collection of ZyxMember objects, but the ZyxMember objects will have no attribute that reference the ZyxClub instance.
  - Specify the ***cardinality*** of the relationship as ***0..N***. The cardinality specifies how many ZyxMember instances are allowed in the collection referenced by ZyxClub>>members. The cardinality 0..N means that 0 <= (number of ZyxMember instances in collection) <= infinity.
- Create ZyxClub>>currentMember. currentMember will reference the ZyxMember instance that is currently selected in ZyxEditClubView (to be created later).

22.1. Create MicFwDomainObject subclass ZyxClub.

22.1. In **Visual Age Organizer**: For **ZyxApplication**: Create a new part with:

- **Part class**: **ZyxClub**.
- **Part type**: **Domain Object Class**.
- **Inherits from**: **MicFwDomainObject**.

22.2. Add ZyxClub>>members and specify relationship / cardinality to ZyxMember

22.2. In the **ONB** on **ZyxClub**: Add instance variable **ZyxClub>>members**.

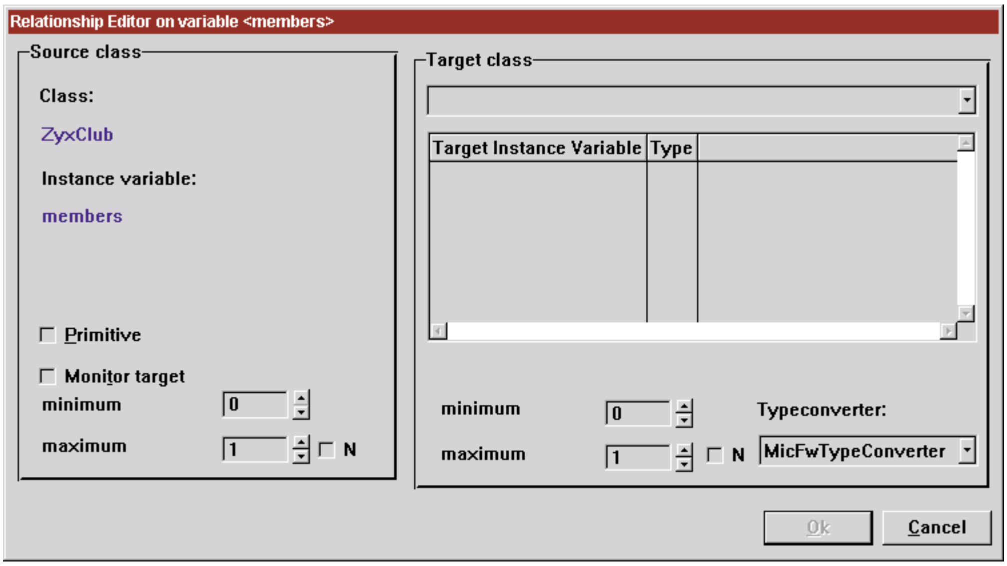
22.3. For **members**: Check **Persistent (Transact)**.

22.2.1. Specify relationship

22.4. Select **members**.

22.5. Right-click on **members**.

22.6. Select **Set Relationship**. The **Relationship Editor on variable <members>** dialog is opened:



Relationship Editor on variable <members>

Source class

Class: ZyxClub

Instance variable: members

□ Primitive

□ Monitor target

minimum 0

maximum 1 □ N

Target class

Target Instance Variable	Type

minimum 0

maximum 1 □ N

Typeconverter: MicFwTypeConverter

Figure 22.1. Relationship Editor dialog

Note that the source class / instance variable = ZyxClub>>members.

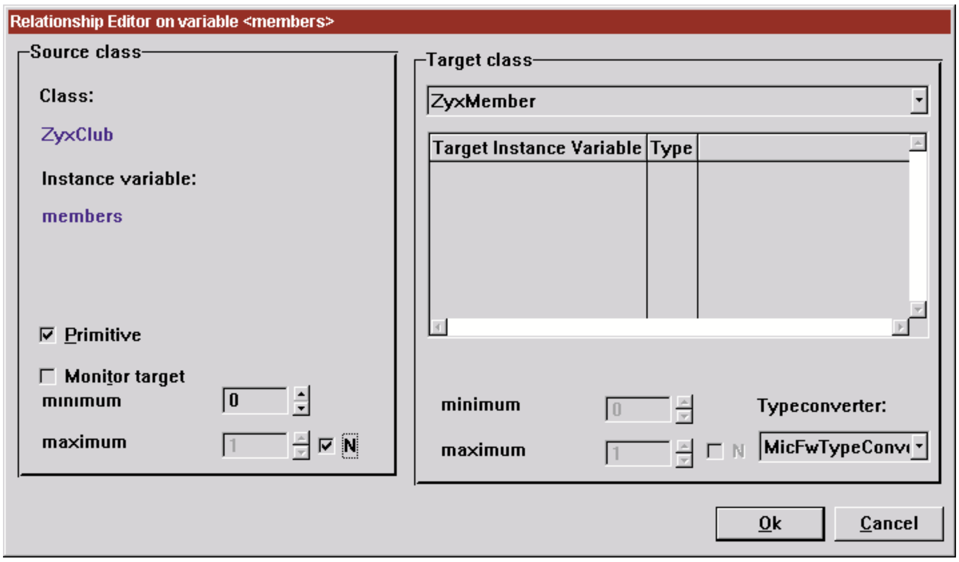
22.7. Check the checkbox **Primitive**.

22.8. In the **Target Class** drop-down list: Select **ZyxMember**.

22.2.2. Specify cardinality

22.9. In the box **Source class**: Check the checkbox **N**. The ***cardinality*** of the relationship is now ***0..N***.

The cardinality of this relationship specifies that ZyxClub>>members can reference from 0 to an unlimited number of ZyxMember instances.



Relationship Editor on variable <members>

Source class

Class: ZyxClub

Instance variable: members

☑ Primitive

□ Monitor target

minimum 0

maximum 1 ☑ N

Target class

ZyxMember

Target Instance Variable	Type

minimum 0

maximum 1 □ N

Typeconverter: MicFwTypeConv

Figure 22.2. Defining cardinality in the Relationship Editor

22.10. Click **OK**. The RE dialog closes and the ONB regains the focus. Note the relationship and target info displayed for ZyxClub>>members. Note that ZyxMember is also displayed in the ONB, since it is now a part of the ***object net*** of ZyxClub. An object net is the complete set of all objects that are connected to each other through relationships defined by the object attributes.

22.3. Add ZyxClub>>currentMember

22.11. In the **ONB**: Add instance variable **ZyxClub>>currentMember**.

22.12. For **currentMember**: Check **Persistent (Transact)**.

22.13. **Save the class**.

22.14. Close the ONB.

22.15. Save the image.

23. Create DP ZyxEditClub / modify DP ZyxEditMember

In this chapter you will:

- Create **DP ZyxEditClub**. ZyxEditClub will be the DP for editting the attributes of the ZyxClub and ZyxMember instances.
- In the DPB: Create **connection eCPConn** to ZyxEditClub.
- Specify DP ZyxEditMember as a ***child process*** of ZyxEditClub. For more information about child processes, see the ***Object Behavior Framework User's Guide*** chapter '1.3.4.7.3. Hierarchical contexts' (page 36).
- In the DPB: Create **base connection eCBConn** to ZyxClub.
- In the DPB: Add **child process connection eMPChConn** from ZyxEditClub to ZyxEditMember.
- Create **ZyxEditClub>>selectedMember** accessors for the member that is currently selected in the proccess.
- Create **ZyxEditClub>>openOn:**. openOn: establishes the base connection to ZyxClub and then sends the openView message.
- Create **ZyxEditMember>>edit**. The message <edit> will be sent to the selected ZyxEditMember instance when the Edit button (to be created later) is clicked in ZyxEditClubView.

23.1. Create MicFwDomainProcess subclass ZyxEditClub.

23.1. In **Visual Age Organizer**: Create a new part (do not open now) for **ZyxApplication** with:

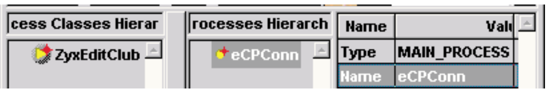
- **Part class**: **ZyxEditClub**.
- **Part type**: **Domain Process Class**.
- **Inherits from**: **MicFwDomainProcess**.

23.2. Create connection eCPConn to ZyxEditClub

23.2. Open the **DPB** on **ZyxEditClub**.

23.3. In the **Processes Hierarchy** column: Click on **ZyxEditClub**.

23.4. In the **Name/Value** columns: In row **Name**: Change **zyxEditClub** to **eCPConn** (edit Club Process Connection). This is the name of the connection to the process ZyxEditClub.



Name	Valu
Type	MAIN_PROCESS
Name	eCPConn

Figure 23.1. Changing the name of the connection to ZyxEditClub in the DPB

23.3. Create base connection eCBConn to ZyxClub

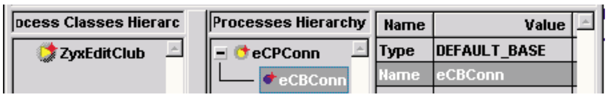
23.5. In the **Processes Hierarchy** box: Right-click on **eCPConn**.

23.6. Select **Add Base Connection**.

23.7. In response to **Choose a domain object class**: Double-click on **ZyxClub**. Note that the domain process ZyxEditClub now has a connection to ZyxClub and that the connection is named **newDefaultBaseConnection**.

23.8. Click on **newDefaultBaseConnection** in the Processes Hierarchy column.

23.9. In the **Name/Value** columns: In row **Name**: Change **newDefaultBaseConnection** to **eCBConn** (edit Club Base Connection).



Name	Value
Type	DEFAULT_BASE
Name	eCBConn

Figure 23.2. Changing the name of the connection to ZyxClub in the DPB

23.4. Add child process connection eMPChConn from ZyxEditClub to ZyxEditMember

23.10. In the **Processes Hierarchy** box: Click on **eCPConn**.

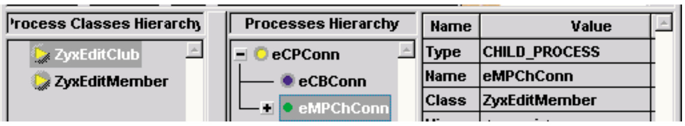
23.11. Right-click.

23.12. Select **Add child connection**.

23.13. In response to **Choose a class**: Double-click on **ZyxEditMember**. ZyxEditMember is added as a child process with a connection named newChildProcessConnection.

23.14. Click on **newChildProcessConnection** in the Processes Hierarchy column.

23.15. Change the name of connection **newChildProcessConnection** to **eMPChConn** (edit Member Process Child Connection).



Name	Value
Type	CHILD_PROCESS
Name	eMPChConn
Class	ZyxEditMember

Figure 23.3. Adding a child process connection to ZyxEditMember in the DPB

23.16. Click anywhere in the DPB to register the changes.

23.17. Select **Browser / Save all changes**.

23.18. Close the **DPB**.

23.5. Create ZyxEditClub>>selectedMember, selectedMember:

23.19. Create the following method:

```
ZyxEditClub>>selectedMember
^self eCBConn currentMember
```

23.20. Create the following method:

```
ZyxEditClub>>selectedMember: aMember
   self eCBConn currentMember: aMember
```

23.6. Create ZyxEditClub>>openOn:

23.21. Create the following method:

```
ZyxEditClub>>openOn: aClub
   self eCBConn: aClub.
   self openView.
```

23.22. Save the class.

23.7. Create ZyxEditMember>>edit

23.23. Create the following method:

```
ZyxEditMember>>edit
   self openOn: self parent selectedMember.
```

<parent> returns the parent process (ZyxEditClub). <selectedMember> returns the member currently selected in the list.

23.24. Save the class.

23.25. Save the image.

24. Create ZyxEditClubView (View)

ZyxEditClubView will be the view for ZyxEditClub.

In this chapter you will:

- Create view ZyxEditClubView.
- Add a drop-down list that displays the club members and allows the selection of the currentMember.
- Create ZyxMember>>asListEntry. The **asListEntry** method must be implemented by a DO when instances of the DO may be displayed in a view with a drop-down list. The object returned by the method is the object that is displayed in the list (typically a string).
- Add a push button that is connected to ZyxEditMember>>edit.
- Add commitAndCloseView / abortAndCloseView buttons.
- In the DPB: Assign ZyxEditClubView to ZyxEditClub.
- Test.

24.1. Create AbtAppBldrView subclass ZyxEditClubView

24.1. In **Visual Age Organizer**: Create a new part for **ZyxApplication** with:

- **Part class**: Enter **ZyxEditClubView**.
- **Part type**: Select **Visual Part**.
- **Inherits from**: Select **AbtAppBldrView**.

24.2. Open the **Composition Editor** on **ZyxEditClubView**.

24.3. Double-click on the **window**. The properties dialog appears.

24.4. Change the **partName** to **ZyxEditClub**.

24.5. Click **OK**.

24.2. Add drop-down list to ZyxEditClubView

24.6. Add a **drop-down list** to the view.

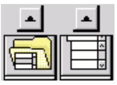


Figure 24.1. Drop-down list in the parts palette

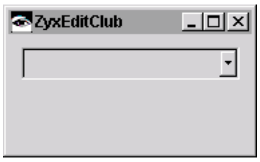
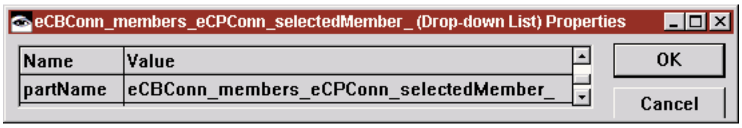


Figure 24.2. Drop-down list in ZyxEditClubView

24.7. Set **drop-down list** property **partName** to **eCBConn_members_eCPConn_selectedMember_**.



eCBConn_members_eCPConn_selectedMember_ (Drop-down List) Properties

Name	Value
partName	eCBConn_members_eCPConn_selectedMember_

Figure 24.3. Naming convention settings for drop-down list

The above partName has the following meaning:

- The "_" (underline) characters separate ***name parts*** (this is not to be confused with the partName).
- **eCBConn** specifies the base connection to the ZyxClub object.
- **members** specifies the accessors (getter and setter) of the DO (ZyxClub) than reference the Collection of references to the ZyxMember objects displayed in the drop-down list.
- **eCPConn** specifies the DP which implements the method specified in the fourth part of the name.
- **selectedMember** specifies the accessors (getter and setter) of the DO (ZyxMember) that is currently selected in the drop-down list.

24.8. Select **File / Save Part**.

24.3. Create ZyxMember>>asListEntry

24.9. Create the following method:

```
ZyxMember>>asListEntry
   ^self name asString
```

24.4. Add push button with connection to ZyxEditMember>>edit

24.10. Add a **Push Button**.

24.11. Change the **Push Button** property **object** to **edit**.

24.12. Change the **Push Button** property **partName** to **eMPChConn_edit_**.

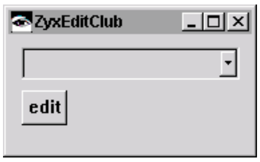


Figure 24.4. ZyxEditClubView with edit button

24.5. Add commitAndCloseView / abortAndCloseView buttons

24.13. Add a **Push Button**.

24.14. Change the **Push Button** property **object** to **commitAndCloseView**.

24.15. Change the **Push Button** property **partName** to **commitAndCloseView_**.

24.16. Add a **Push Button**.

24.17. Change the **Push Button** property **object** to **abortAndCloseView**.

24.18. Change the **Push Button** property **partName** to **abortAndCloseView_**.

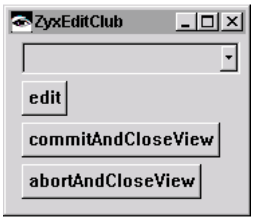


Figure 24.5. ZyxEditClubView with commit and abort buttons

24.19. Save the part.

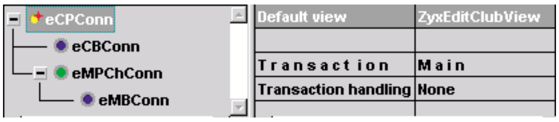
24.20. Close the CE.

24.6. Assign View to Process

24.21. Open the **DPB** on **ZyxEditClub**.

24.22. In the **Processes Hierarchy** box: Select process connection **eCPConn**.

24.23. In the **Name/Value** columns: In row **Default View**: Select from the drop-down list **ZyxEditClubView**.



Default view	ZyxEditClubView
Transaction	Main
Transaction handling	None

Figure 24.6. Assigning ZxyEditClubView to ZyxEditClub in DPB

24.24. **Save all changes**.

24.25. Close the DPB.

24.26. Save the image.

24.7. Test

24.27. In the workspace execute the following code:

```
FW CH 24. Test ZyxEditClubView (and ZyxEditMemberView)."
[
    | club member1 member2 |
    ZyxMember initializeValidation.
    member1 := ZyxMember new.
    member1 name: 'member1'.
    member1 weight: 201.
    member2 := ZyxMember new.
    member2 name: 'member2'.
    member2 weight: 202.
    club := ZyxClub new.
    club members add: member1.
    club members add: member2.
    ZyxEditClub new openOn: club.
    ZyxEditClub new openOn: club.
]
```

2 views of the same club object are opened (if only 1 view is visible: move the view to uncover the other view):

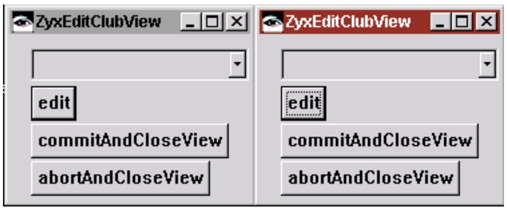


Figure 24.7. ZyxEditClubView dialogs

24.28. In the **LEFT** dialog: Select a **member**. Note that the member selection is also displayed in the RIGHT dialog.

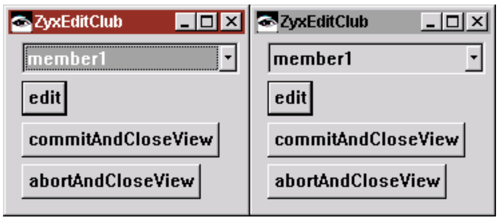
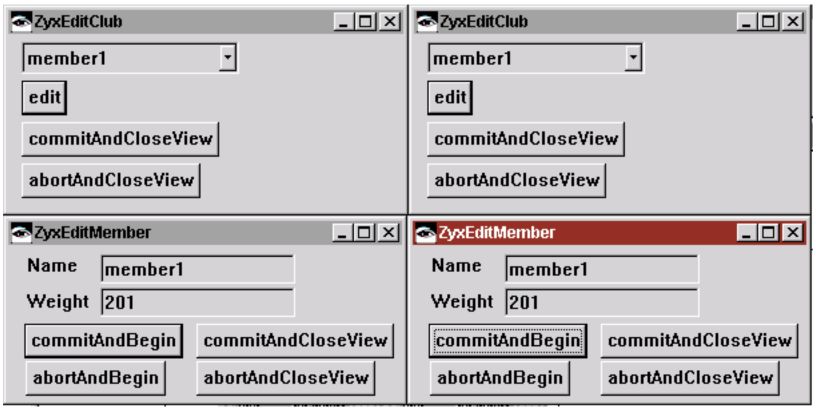


Figure 24.8. Selecting member from drop-down list selects member in other drop-down list

24.29. In the **LEFT** dialog: Click **edit**. A ZyxEditMember dialog is opened.

24.30. In the **RIGHT** dialog: Click **edit**. A second ZyxEditMember dialog is opened.



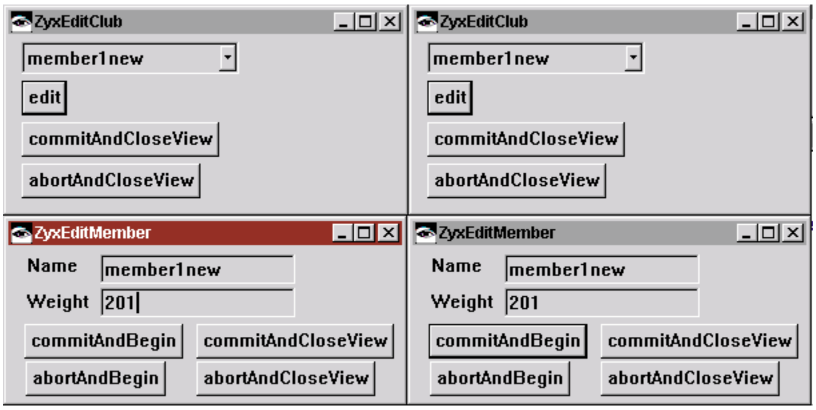
Name member1
Weight 201

Name member1
Weight 201

Figure 24.9. 2 ZyxEditClub dialogs each with its own ZyxEditMember child dialog

24.31. Change the **name** attribute in the **LEFT ZyxEditMember** dialog to **member1new**.

24.32. Click in a different field. Note that the new name is displayed in ALL dialogs:



Note that all changes are reflected in the dialogs regardless of which button (commit or abort) is pressed. This is becuause no transaction handling has been defined for the processes. This will be explored in the next chapter.

24.33. Click an abort button in a ZyxEditClubView dialog. Note that the corresponding ZyxEditMember dialog also closes, since the dialog is a child of the ZyxEditClubView dialog.

24.34. Close the remaining views.

25. Transacted child process connection to ZyxEditMember

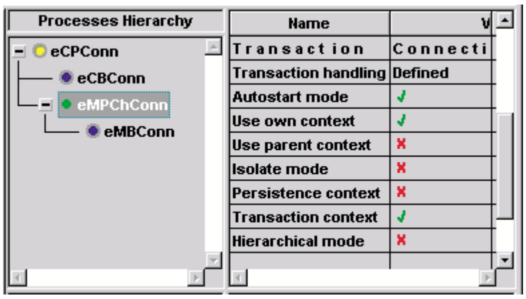
In this chapter you will:

- In the DPB: Define the child process connection to ZyxEditMember as transacted. This will cause the changes made in the child view (ZyxEditMemberView) will be transacted. However, there will still be no transaction settings for the parent process ZyxEditClub.
- Test (the transacted changes in the child view will not be shown in the parent view since the parent process is not transacted):
  - Abort changes made in a child view.
  - Commit changes made in a child view.

25.1. Define default transaction settings for child process connection to ZyxEditMember (eMPChConn)

25.1. In the **DPB** for **ZyxEditClub**: Define **Transaction Connection Transaction Handling** for **eMPChConn** with the following (default) options enabled:

- **Autostart mode**.
- **Use own context**.
- **Transaction context**.



Name	V
Transaction	Connecti
Transaction handling	Defined
Autostart mode	✓
Use own context	✓
Use parent context	✗
Isolate mode	✗
Persistence context	✗
Transaction context	✓
Hierarchical mode	✗

Figure 25.1. Defining transaction settings for child connection to ZyxEditMember

25.2. **Save all changes**.

25.3. Close the DPB.

25.4. Save the image.

25.2. Test

25.5. In the workspace execute the following code:

```
FW CH 25. Test ZyxEditClubView (with transacted ZyxEditMember)."
[
  | club member1 member2 |
  ZyxMember initializeValidation.
  member1 := ZyxMember new.
  member1 name: 'member1'.
  member1 weight: 201.
  member2 := ZyxMember new.
  member2 name: 'member2'.
  member2 weight: 202.
  club := ZyxClub new.
  club members add: member1.
  club members add: member2.
  ZyxEditClub new openOn: club.
  ZyxEditClub new openOn: club.
]
```

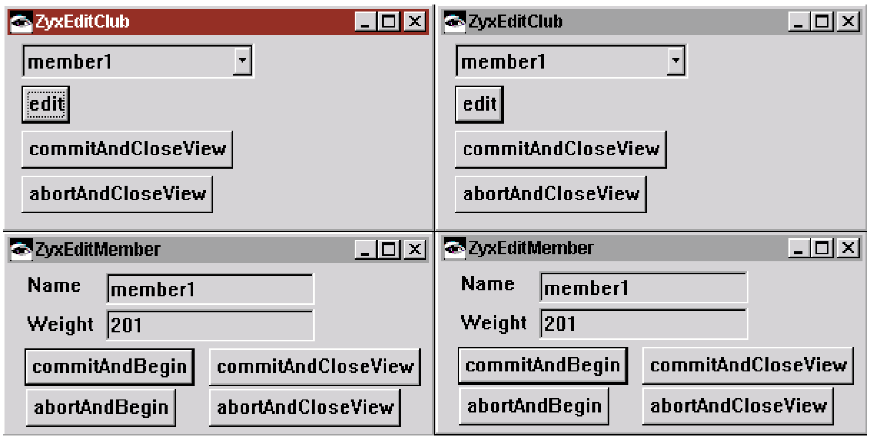
2 **ZyxEditClubView** dialogs appear.

25.2.1. Abort changes to member name in child view

25.6. From the **drop-down list** in the **LEFT ZyxEditClub** dialog: Select **member1**.

25.7. Click **edit**. The **LEFT ZyxEditMemberView** dialog appears.

25.8. In the **RIGHT ZyxEditClub** dialog: Click **edit**. The **RIGHT ZyxEditMemberView** dialog appears.



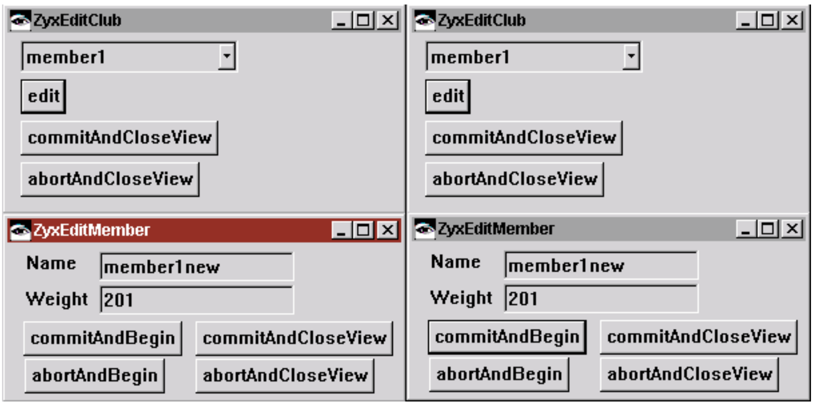
Name member1
Weight 201

Name member1
Weight 201

Figure 25.2. 2 ZyxEditClub dialogs with ZyxEditMember child dialogs

25.9. Change **name** in the **LEFT ZyxEditMember** dialog to **member1new**.

25.10. Click anywhere to change the focus. Note that the **ZyxEditClub dialogs** are NOT updated with the new name, since transaction handling is not defined for ZyxEditClub:



Name member1new
Weight 201

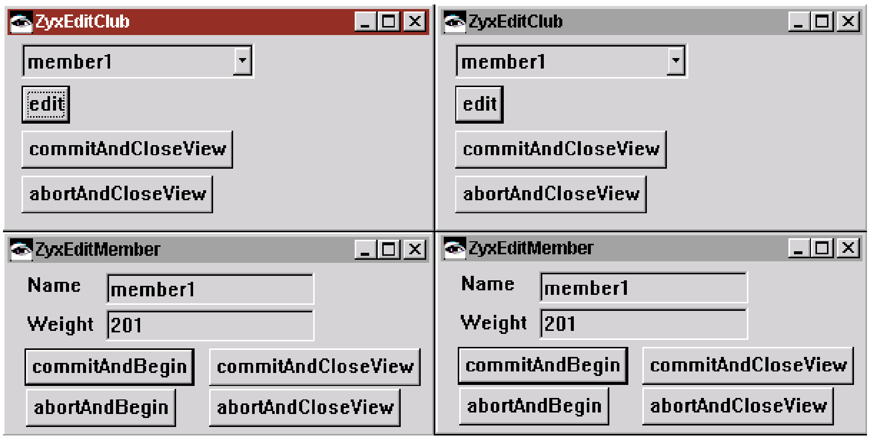
Name member1new
Weight 201

Figure 25.3. Transacted changes in child are not reflected in parent

25.11. Attempt to change the name in the **RIGHT ZyxEditMember** dialog. A transaction conflict exception is thrown.

25.12. Close the debugger window.

25.13. Click **abortAndBegin** in **LEFT ZyxEditMember** dialog. Note that the changes are aborted in the RIGHT ZyxEditMember dialog also.



Name member1
Weight 201

Name member1
Weight 201

Figure 25.4. Changes abort in LEFT child dialog are aborted in RIGHT child dialog also.

25.2.2. Commit changes to member name in child view

25.14. Change **name** in the **LEFT ZyxEditMember** dialog to **member1new**.

25.15. Click **commitAndBegin**. Note that the changes are show in all dialogs:

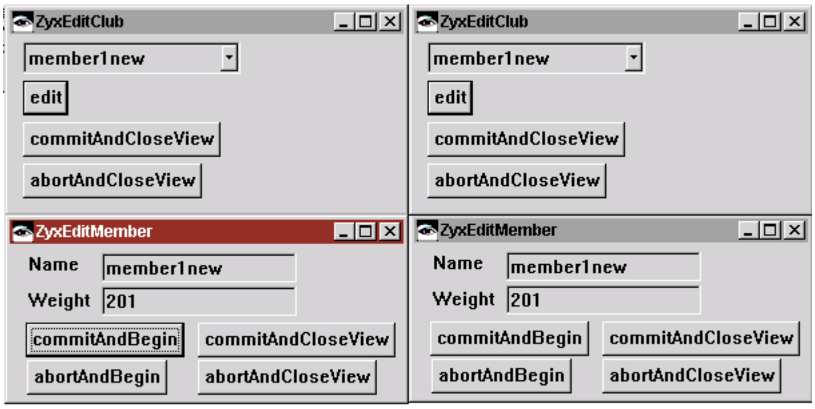


Figure 25.5. Committed changes in the child are immediately reflected in parent view

25.16. In the **ZyxEditClub** dialogs: Click **commitAndCloseView** to close the parent and child dialogs.

26. Transacted parent process ZyxEditClub

In this chapter you will:

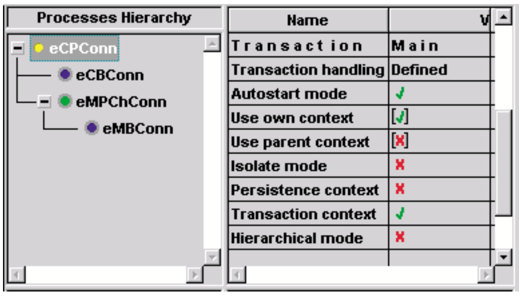
- In the DPB: Define the parent process ZyxEditClub as transacted (non-isolated). This will cause the transacted changes made in the child view (ZyxEditMemberView) to be displayed in the parent view.
- Test.

Note: If the transactions for the parent process are specified as isolated, then the changes in the child will not be shown in the parent until they are committed.

26.1. Define default transaction settings for parent process connection to ZyxEditClub (eCPConn)

26.1. In the **DPB**: Define **Transaction Main Transaction Handling** for **eCPConn** with the following (default) options enabled:

- **Autostart mode**.
- **Use own context**.
- **Transaction context**.



Name	V
Transaction	Main
Transaction handling	Defined
Autostart mode	✓
Use own context	[✓]
Use parent context	[✗]
Isolate mode	✗
Persistence context	✗
Transaction context	✓
Hierarchical mode	✗

Figure 26.1. Defining transaction settings for ZyxEditClub in DPB

26.2. **Save all changes**.

26.3. Close the DPB.

26.4. Save the image.

26.2. Test

26.5. In the workspace execute the following code:

```
FW CH 26. Test transacted ZyxEditClub (& transacted ZyxEditMember)."
[
  | club member1 member2 |
  ZyxMember initializeValidation.
  member1 := ZyxMember new.
  member1 name: 'member1'.
  member1 weight: 201.
  member2 := ZyxMember new.
  member2 name: 'member2'.
  member2 weight: 202.
  club := ZyxClub new.
  club members add: member1.
  club members add: member2.
  ZyxEditClub new openOn: club.
  ZyxEditClub new openOn: club.
]
```

2 **ZyxEditClubView** dialogs appear.

26.2.1. Display uncommitted changes in child view in parent view

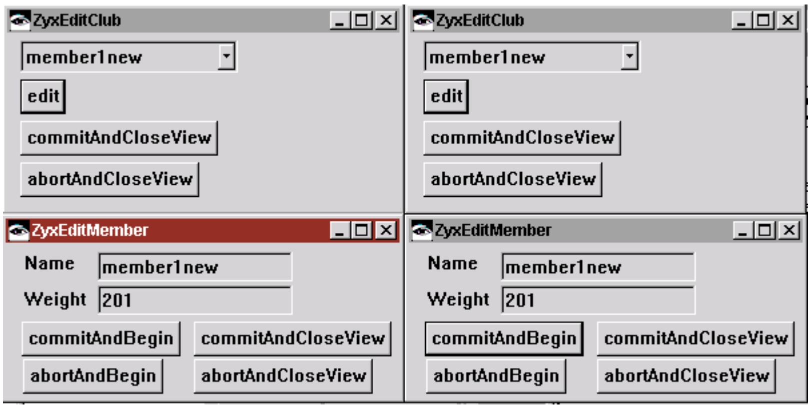
26.6. From the **drop-down list** in the **LEFT ZyxEditClub** dialog: Select **member1**.

26.7. Click **edit**. The **LEFT ZyxEditMemberView** dialog appears.

26.8. In the **RIGHT ZyxEditClub** dialog: Click **edit**. The **RIGHT ZyxEditMemberView** dialog appears.

26.9. Change **name** in the **LEFT ZyxEditMember** dialog to **member1new**.

26.10. Click anywhere to change the focus. Note that the **ZyxEditClubView**’s are updated with the new name, since transaction handling is not defined for ZyxEditClub:



Name member1new
Weight 201

Name member1new
Weight 201

Figure 26.2. Uncommitted changes in child are displayed in parent

26.11. Close the dialogs.

27. Adding and deleting members

In this chapter you will:

- Create ZyxEditClub methods for adding and deleting ZyxMember's.
- Add push buttons to ZyxEditClubView with connections to the ZyxEditClub add and delete methods.
- Test.

27.1. Create ZyxEditClub>> addMember, deleteMember

27.1. Create the following method:

```
ZyxEditClub>>addMember
   self eCBConn members add: ( ZyxMember new name: ’new’ ).
   self commitAndBegin.
```

27.2. Create the following method:

```
ZyxEditClub>>deleteMember
   self eCBConn members remove: self eCBConn currentMember.
   self commitAndBegin.
```

27.2. Add push buttons addMember / deleteMember to ZyxEditClubView

27.3. Add a **Push Button** to **ZyxEditClubView**.

27.4. Change the **Push Button** property **object** to **addMember**.

27.5. Change the **Push Button** property **partName** to **addMember_**.

27.6. Add a **Push Button** to **ZyxEditClubView**.

27.7. Change the **Push Button** property **object** to **deleteMember**.

27.8. Change the **Push Button** property **partName** to **deleteMember_**.

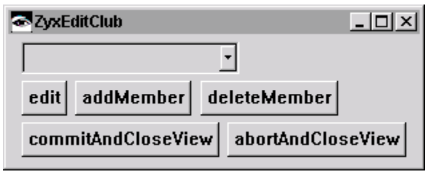


Figure 27.1. addMember and deleteMember buttons in ZyxEditClubView

27.9. **Save all changes**.

27.10. Save the image.

27.3. Test

27.11. In the workspace execute the following code:

```
FW CH 27. Add and delete ZyxMember’s to ZyxClub."
[
   | club member1 member2 |
   member1 := ZyxMember new
      name: 'member1';
      weight: 111.
   club := ZyxClub new.
   club members add: member1.
   ZyxEditClub new openOn: club.
]
```

27.12. Use the addMember and deleteMember buttons to add and delete members.

28. Implementing HoverHelp and F1 Help (using a ViewPort)

Implementing ***Hover Help*** and ***F1 Help*** for a view requires that a ***ViewPort*** is created for the DO whose attributes are displayed in the view. For detailed information about ViewPorts, see the ***Application Framework User's Guide*** chapter '1.3.11. Viewports' (page 45).

Caution: Viewports can also be created for DP's. Thus, the viewports ZyxMemberViewPort and ZyxEditMemberViewPort could exist. Therefore, always pay special attention to ViewPort names.

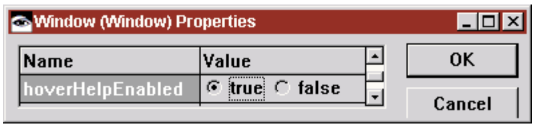
In this chapter you will:

- Enable HoverHelp in ZyxEditMemberView.
- Create viewport ZyxMemberViewPort.
- Create ZyxMemberViewPort>>nameHoverHelpText. This method returns the String object to be displayed in the HoverHelp for attribute ZyxMember>>name. The method name consists of 2 parts:
  - The partName for the part in the view (without the "_" character) that displays the attribute. The partName is "name_"; therefore the first part of the method name must be "name".
  - "HoverHelpText".
- Create ZyxMemberViewPort>>nameHelpText. This method returns the String object to be displayed in the F1 Help for attribute ZyxMember>>name. The method name is similar to that for HoverHelp, exept that the second part of the name is simply "HelpText".
- Assign ZyxMemberViewPort as the ViewPort for eMBConn (a ***Generic viewport*** is assigned to the eMBConn by default).
- Test.

28.1. Enable HoverHelp in ZyxEditMemberView

28.1. In the **Composition Editor** for **ZyxEditMemberView**: Double-click on the **ZyxEditMemberView window**. The **Window properties** dialog is opened.

28.2. Set **hoverHelpEnabled** to **true**.



Window (Window) Properties

Name	Value
hoverHelpEnabled	● true ○ false

Figure 28.1. Enabling Hover Help for a view

28.3. Click **OK**.

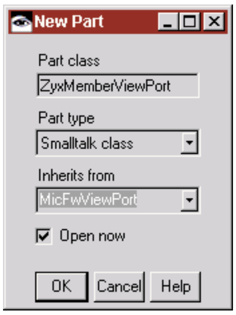
28.4. Close the CE.

28.5. **Save the part**.

28.2. Create MicFwViewPort subclass ZyxMemberViewPort

28.6. In **Visual Age Organizer**: Create and open (make sure the **Open now** checkbox is checked) a new part for **ZyxApplication** with:

- **Part class**: **ZyxMemberViewPort**.
- **Part type**: **Viewport**.
- **Inherits from**: **MicFwViewPort**.



New Part

Part class: ZyxMemberViewPort

Part type: Smalltalk class

Inherits from: MicFwViewPort

☑ Open now

Figure 28.2. Settings for ZyxApplication part ZyxMemberViewPort

28.3. Create ZyxMemberViewPort>>nameHoverHelpText

28.7. Create the following method:

```
ZyxMemberViewPort>>nameHoverHelpText
   ^'HoverHelp text for name'
```

28.4. Create ZyxMemberViewPort>>nameHelpText

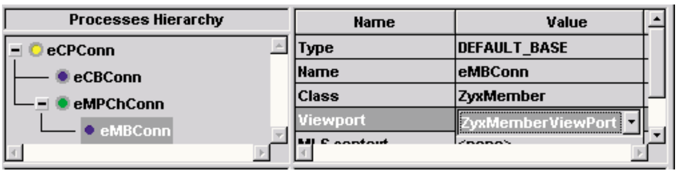
28.8. Create the following method:

```
ZyxMemberViewPort>>nameHelpText
   ^'F1 help text for name'
```

28.9. **Save the class**.

28.5. Assign ZyxMemberViewPort as the ViewPort for eMBConn

28.10. In the **DPB** on **ZyxEditClub**: Select **ZyxMemberViewPort** as the **ViewPort** for **eMBConn**.



Processes Hierarchy

- eCPConn
  - eCBConn
  - eMPChConn
    - eMBConn

Name	Value
Type	DEFAULT_BASE
Name	eMBConn
Class	ZyxMember
Viewport	ZyxMemberViewPort
MLS context	<none>

Figure 28.3. Assigning ZyxMemberViewPort as the ViewPort for the child connection to ZyxMember

28.11. **Save all changes**.

28.12. Close the DPB.

28.13. Save the image.

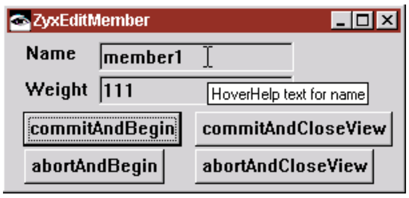
28.6. Test

28.14. In the workspace execute the following code:

```
FW CH 28. HoverHelp and F1 Help for ZyxMember."
[
   | member1 |
   member1 := ZyxMember new
      name: 'member1';
      weight: 111.
   ZyxEditMember new openOn: member1.
]
```

The **ZyxEditMember** dialog appears.

28.15. Move the cursor into the **name text field**. The hover help text appears:



ZyxEditMember

Name: member1

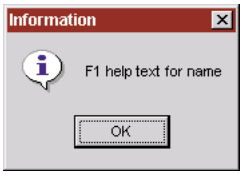
Weight: 111

HoverHelp text for name

Figure 28.4. HoverHelp for name in ZyxEditMember dialog

28.16. Click on the **name text field**.

28.17. Press **F1**. An **Information** dialog appears:



Information

F1 help text for name

Figure 28.5. F1 Help for name in ZyxEditMember dialog

28.18. Close the dialog.

29. Enabling/disabling buttons

Controls in a view can be enabled/disabled depending on the boolean value returned by a method for the DP of the view.

In this chapter you will:

- Create ViewPort ZyxEditMemberViewPort.
- Create ZyxEditMemberViewPort>>editEnabled, which returns a boolean object to control whether or not the button is enabled. The method name consists of 2 parts:
  - The partName for the control in the view (without the "_" character). The partName is "edit_"; therefore the first part of the method name must be "edit".
  - "Enabled".
- Assign ZyxEditMemberViewPort as the ViewPort for the connection eMPChConn.
- Test.

29.1. Create MicFwViewPort subclass ZyxEditMemberViewPort

29.1. In **Visual Age Organizer**: Create and open (make sure the **Open now** checkbox is checked) a new part for **ZyxApplication** with:

- **Part class**: **ZyxEditMemberViewPort**.
- **Part type**: **Viewport**.
- **Inherits from**: **MicFwViewPort**.

29.2. Create ZyxEditMemberViewPort>>editEnabled

29.2. Create the following method:

```
ZyxEditMemberViewPort>>editEnabled
^(self model parent selectedMember notNil).
```

29.3. Assign ZyxEditMemberViewPort to eMPChConn

29.3. In the **DPB** on **ZyxEditClub**: Select **ZyxEditMemberViewPort** as the **ViewPort** for **eMPChConn**.

29.4. **Save all changes**.

29.5. Close the DPB.

29.6. Save the image.

29.4. Test

29.7. In the workspace execute the following code:

```
FW CH 29. Enable/disable edit button."
[
  | club member1 |
  member1 := ZyxMember new
    name: 'member1';
    weight: 111.
  club := ZyxClub new.
  club members add: member1.
  ZyxEditClub new openOn: club.
]
```

29.8. The **ZyxEditClubView** dialog appears with no member selected. Note that the **edit** button is disabled:

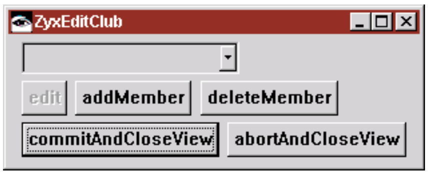


Figure 29.1. ZyxEditClubView with disabled edit button

29.9. Select **member1**. Note that the **edit** button is now enabled.

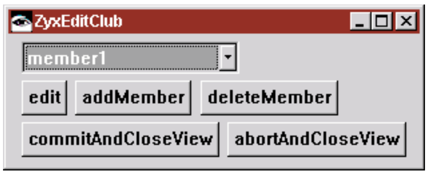


Figure 29.2. ZyxEditClubView with enabled edit button

29.10. Close the dialog.

30. Using a ViewPort as a filter

Controls within a View can be changed. For example, a drop-down list that displays a single selected object from a list of objects can be replaced with a list that displays multiple objects.

The problem is that the interface to one control may differ from the interface of another. For example:

- When an object is selected in a drop-down list, the drop-down list returns the object.
- When an object is selected in a List, the List returns a Collection that contains the selected object.

Thus, changing a control in a view could require changing the methods in the DP for the view. This is, however, a very undesireable solution.

This problem is solved by using a ViewPort as a ***filter***. A ViewPort is created for the view with the new controls. The Viewport will "intercept" and "translate" any messages between the ViewPort and the DP.

In this chapter you will:

- Create ZyxEditClubView2, which has a List instead of a drop-down list.
- Create ZyxEditClubViewPort2 (ZyxEditClubViewPort does not exist (the default ViewPort was used previously)).
- Create ZyxEditClubViewPort2 filter accessors for selectedMember
- Specify ZyxEditClubView2/ZyxEditClubViewPort2 as the view/ViewPort for ZyxEditClub.
- Test.

30.1. Create ZyxEditClubView2 (with List instead of Drop-down list)

30.1.1. Copy ZyxEditClubView to ZyxEditClubView2

30.1. In the **Visual Age Organizer**: Right-click on **ZyxEditClubView**.

30.2. Select **Copy**.

30.3. In response to "**New name for copy of class?**": Enter **ZyxEditClubView2**.

30.4. Click **OK**.

30.5. In response to "**Do you wish to browse methods that reference the class(es) to be copied?**": Click **No** (do not browse classes).

30.1.2. In ZyxEditClubView2: Delete Drop-down list and add a List

30.6. In the **Composition Editor** for **ZyxEditClubView2**: Delete **drop-down list**.

30.7. Add a **List** to the view.

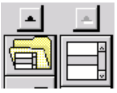


Figure 30.1. The List part in the parts palette

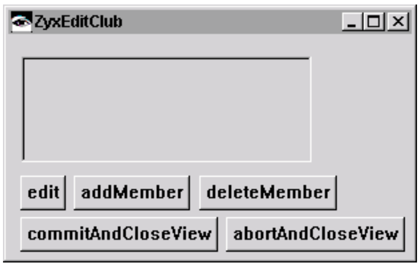


Figure 30.2. ZyxEditClubView2 with List

30.8. Change the **List partName** to **eCBConn_members_eCPConn_selectedMember_name_**.

30.9. **Save the part**.

30.10. Close the CE.

30.2. Create MicFwViewPort subclass ZyxEditClubViewPort2

30.11. In **Visual Age Organizer**: Create and open (make sure the **Open now** checkbox is checked) a new part for **ZyxApplication** with:

- **Part class**: **ZyxEditClubViewPort2**.
- **Part type**: **Viewport**.

- **Inherits from**: **MicFwViewPort**.

30.3. Create ZyxEditClubViewPort2 filter accessors for selectedMember

30.12. Create the following method:

```
ZyxEditClubViewPort2>>selectedMember
   ^OrderedCollection with: self model selectedMember.
```

30.13. Create the following method:

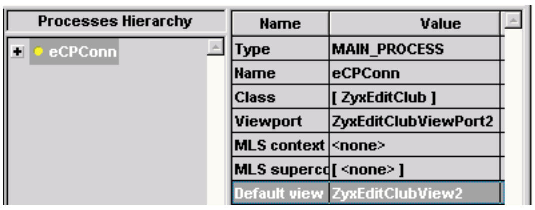
```
ZyxEditClubViewPort2>>selectedMember: aCollection
   aCollection notEmpty
      ifTrue: [ self model selectedMember: aCollection first ].
```

30.14. Save the class.

30.4. Specify ZyxEditClubView2/ZyxEditClubViewPort2 as the view/ViewPort for ZyxEditClub connection eCPConn

30.15. In the **DPB** on **ZyxEditClub**: Change the **eCPConn Default view** from ZyxEditClubView to **ZyxEditClubView2.**

30.16. Set the **eCPConn ViewPort** to **ZyxEditClubViewPort2.**



Processes Hierarchy

+ eCPConn

Name	Value
Type	MAIN_PROCESS
Name	eCPConn
Class	[ZyxEditClub]
Viewport	ZyxEditClubViewPort2
MLS context	<none>
MLS superco	[<none>]
Default view	ZyxEditClubView2

Figure 30.3. Assign ZyxEditClubView2/ZyxEditClubViewPort2 to ZyxEditClub in the DPB

30.17. Save all changes.

30.18. Close the DPB.

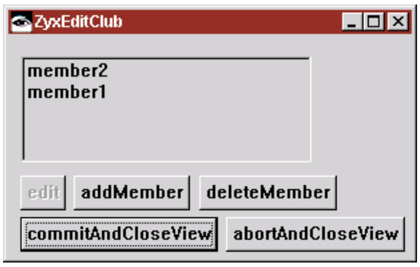
30.19. Save the image.

30.5. Test

30.20. In the workspace execute the following code:

```
FW CH 30. ViewPort as a filter(ZyxEditClubView drop-down list replaced with
list)."
[
   | club member1 member2 |
   member1 := ZyxMember new
      name: 'member1';
      weight: 111.
   member2 := ZyxMember new
      name: 'member2';
      weight: 222.
   club := ZyxClub new.
   club members add: member1.
   club members add: member2.
   ZyxEditClub new openOn: club.
]
```

30.21. In the workspace: Select the code from the previous example and **execute with Execute**. The **ZyxEditClubView** dialog appears with the new List.



ZyxEditClub

member2
member1

Figure 30.4. ZyxEditClubView with List (instead of drop-down list)

30.22. Test the dialogs as in the previous chapters.

31. Displaying / hiding a GroupControl

Controls can be displayed / hidden. The implementation of this functionality is similar to that for enabling / disabling controls (as with the push buttons earlier).

In this chapter you will:

- Add a **GroupBox** to the ZyxEditMemberView whose contents are only displayed if the weight of the selected member meets certain criteria ("notExcessive", in this case < 100). The GroupBox partName consists of 2 parts:
  - The child connection (**eMPChConn_**) to the process (ZyxEditMember).
  - **personalInfo_**.
- Add a **label** and **text field** for ZyxMember>>weight to the GroupBox (the original weight label and field are not deleted from ZyxEditMemberView (otherwise you would not be able to change the weight)).
- Create ZyxEditMemberViewPort>>**personalInfoVisible**, which returns a boolean value that determines whether or not the GroupBox should be displayed.
- Specify ZyxEditMemberViewPort as the ViewPort for ZyxEditMember connection eMPConn. This is required since ZyxEditMemberViewPort is only assigned to connection eMPChConn, and in the test examples you will be opening ZyxEditMemberView directly (not from the ZyxEditClub process).
- Creating ZyxMember>>**weightNotExcessive**, which determines whether the weight meets the criteria for being displayed.
- Test

31.1. Add GroupBox to ZyxEditMemberView

31.1. In the **Composition Editor** for **ZyxEditMemberView**: Add a **GroupBox**.

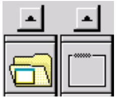


Figure 31.1. Group Box in the parts palette

31.2. Double-click on the **GroupBox** to open the properties list.

31.3. Change **label** to **Personal info**.

31.4. Change **partName** to **personalInfo_** (do not forget the trailing underscore character).

31.5. Click **OK**.

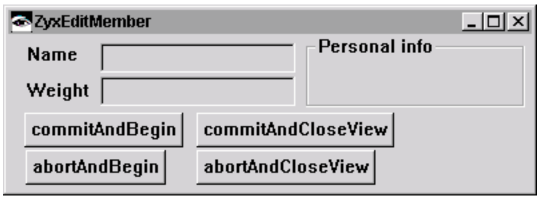


Figure 31.2. ZyxEditMemberView with Personal info group box

31.2. Add label and text field for ZyxMember>>weight to GroupBox

31.6. Add a **Label** to the GroupBox.

31.7. Change the **Label** property **object** to **Weight**.

31.8. Add a **Text field** to the GroupBox.

31.9. Change the **Text field** property **partName** to **PI_weight_** (Note: The prefix "PI_" is added because the partName must be unique (the part weight_ already exists on the view)):

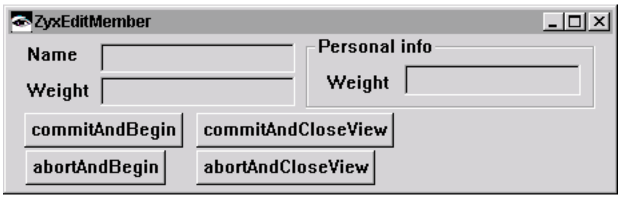


Figure 31.3. ZyxMember>>weight in the Personal info group box

31.10. **Save the part**.

31.11. Close the CE.

31.3. Create ZyxEditMemberViewPort>>personalInfoVisible

31.12. Create the following method:

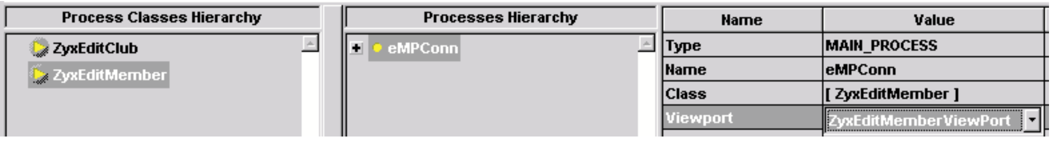
```
ZyxEditMemberViewPort>>personalInfoVisible
  ^self model weightNotExcessive.
```

31.13. **Save the class**.

31.4. Specify ZyxEditMemberViewPort as the ViewPort for ZyxEditMember connection eMPConn (not eMPChConn)

31.14. In the **DPB** on **ZyxEditClub**: In the column **Process Classes Hierarchy**: Select **ZyxEditMember**.

31.15. Set the **eMPConn ViewPort** to **ZyxEditMemberViewPort.**



Process Classes Hierarchy

- ZyxEditClub
- ZyxEditMember

Processes Hierarchy

- eMPConn

Name	Value
Type	MAIN_PROCESS
Name	eMPConn
Class	[ZyxEditMember]
Viewport	ZyxEditMemberViewPort

Figure 31.4. Assign ZyxEditMemberViewPort to ZyxEditMember connection eMPConn in the DPB

31.16. Save all changes.

31.17. Close the DPB.

31.5. Create ZyxEditMember>>weightNotExcessive

31.18. Create the following method:

```
ZyxEditMember>>weightNotExcessive
  self eMBConn weight isNil ifTrue: [ ^true ].
  ^self eMBConn weight < 100 .
```

31.19. **Save the class**.

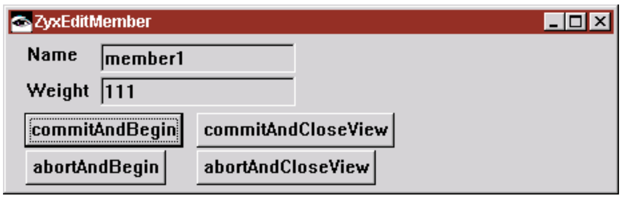
31.20. Save the image.

31.6. Test

31.21. In the workspace execute the following code:

```
FW CH 31. Displaying / hiding a Group Control (GroupBox)."
[
  | member1 |
  ZyxMember initializeValidation.
  member1 := ZyxMember new
    name: 'member1';
    weight: 111.
  ZyxEditMember new openOn: member1.
]
```

Note that the weight is **111**; therfore, the **GroupBox** is **not displayed**.



ZyxEditMember

Name: member1

Weight: 111

Figure 31.5. ZyxMemberView Personal info GroupBox not displayed if weight excessive

31.22. Change the weight to **99**.

31.23. Click to change the focus. Note that the **GroupBox** is **displayed**.

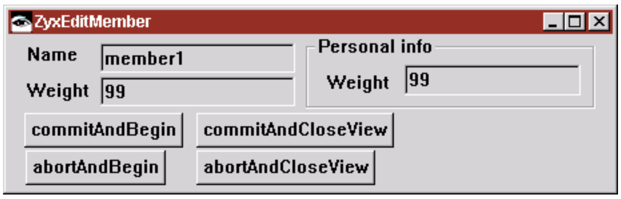


Figure 31.6. ZyxMemberView Personal info GroupBox displayed if weight not excessive

31.24. Click **abortAndBegin**. Note that the **GroupBox** is **not displayed**.

31.25. Close the dialog.

32. Static ZyxEditClubView GroupBox containing ZyxEditMember as child process

A GroupControl can be a quasi-view in that it can contain a child process.

In this chapter you will:

- Copy the GroupBox from ZyxEditMemberView to ZyxEditClubView2. The label "Weight" in the GroupBox will be renamed "Initial weight".
- Add commitAndBegin and abortAndBegin buttons to the GroupBox.
- Modify ZyxEditClub>>selectedMember: so that the eMBConn for the eMPChConn is set.
- Create ZyxEditClubViewPort2>>personalInfoVisible so that the GroupBox is only visible if a member has been selected from the list.
- Modify ZyxMember>>initializeValidation so that a valid value for ZyxMember>>weight is -1.
- Create ZyxEditClubViewPort2>>personalInfoEnabled that enables the GroupBox in ZyxEditClubView only if ZyxMember>>weight == -1. Thus the weight in ZyxEditClubView can only be modified if the weight == -1.
- Test.

32.1. Copy GroupBox from ZyxEditMemberView to ZyxEditClubView2

32.1. In the **Composition Editor** on **ZyxEditMemberView**: Select the **Personal info** groupbox.

32.2. Press **Ctrl-Ins**.

32.3. Close the CE.

32.4. Open the **Composition Editor** on **ZyxEditClubView2**.

32.5. Press **Shift-Ins**. The cursor turns into a cross-hair.

32.6. Click on **ZyxEditClubView2** window to add the group box.

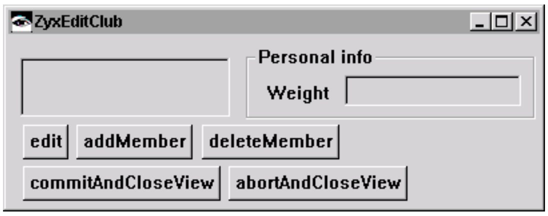


Figure 32.1. ZyxEditClubView with static GroupBox for ZyxEditMember

32.7. Change the **Personal info GroupBox** property **partName** to **eCPConn_personalInfo_eMBConn_eMPChConn_**.

32.8. Change the label **Weight** to **Initial weight**.

32.2. Add commitAndBegin and abortAndBegin buttons to GroupBox

32.9. Add a **Push Button** to **GroupBox**.

32.10. Change **Push Button** property **object** to **commitAndBegin**.

32.11. Change **Push Button** property **partName** to **commitAndBegin_**.

32.12. Add a **Push Button** to **GroupBox**.

32.13. Change **Push Button** property **object** to **abortAndBegin**.

32.14. Change **Push Button** property **partName** to **abortAndBegin_**.

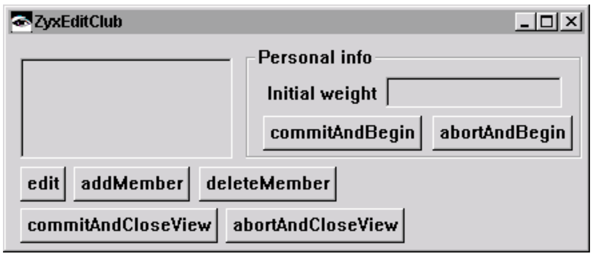


Figure 32.2. ZyxEditClub static GroupBox for ZyxEditMember with commit/abort buttons

32.15. Save the part.

32.16. Closet the CE.

32.3. Modify ZyxEditClub>>selectedMember:

32.17. Modify the method as shown below:

```
ZyxEditClub>>selectedMember: aMember
   self eCBConn currentMember: aMember.
   self eMPChConn eMBConn: aMember. "added line"
```

32.18. Save the class.

32.4. Create ZyxEditClubViewPort2>>personalInfoVisible

32.19. Create the following method:

```
ZyxEditClub>>personalInfoVisible
   ^self model selectedMember notNil.
```

32.5. Modify ZyxMember>>initializeValidation

32.20. Create the following method:

```
initializeValidation
   self validateWrite: #weight
      using: [ :value | ( (value notNil and: [ value > 20 ] )
      and: [ value < 500 ] ) or: [ value == -1 ] ]
```

32.21. Save the class.

32.6. Create ZyxEditClubViewPort2>>personalInfoEnabled

32.22. Create the following method:

```
ZyxEditClub>>personalInfoEnabled
   ^self model selectedMember weight == -1.
```

32.23. Save the class.

32.24. Save the image.

32.7. Test

32.25. In the workspace: Execute the following code:

```
FW CH 32. ZyxEditClubView static groupbox containing ZyxEditMember as child
process."
[
   | club member1 member2 |
   ZyxMember initializeValidation.
   member1 := ZyxMember new
      name: 'member1';
      weight: -1.
   member2 := ZyxMember new
      name: 'member2';
      weight: -1.
   club := ZyxClub new.
   club members add: member1.
   club members add: member2.
   ZyxEditClub new openOn: club.
]
```

The following dialog appears (the GroupBox is hidden because no member has been selected):

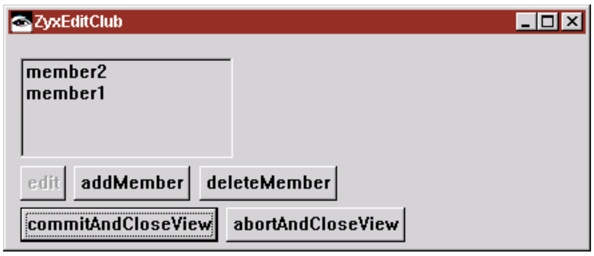
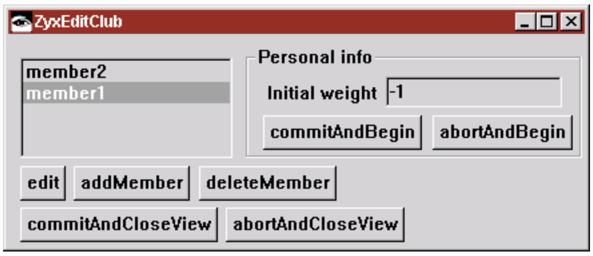


Figure 32.3. ZyxEditClub dialog with hidden ZyxEditMember GroupBox

32.26. Select **member1**. Note that the GroupBox **Personal info** is enabled:



ZyxEditClub

member2
member1

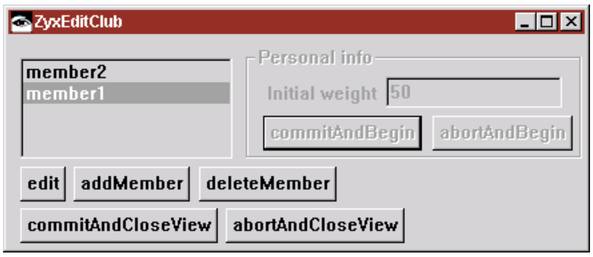
Personal info

Initial weight -1

Figure 32.4. ZyxEditClub dialog after ZyxMember selected from list

32.27. Change the **weight** to **50**.

32.28. Click **commitAndBegin**. Note that the group box is displayed but disabled:



ZyxEditClub

member2
member1

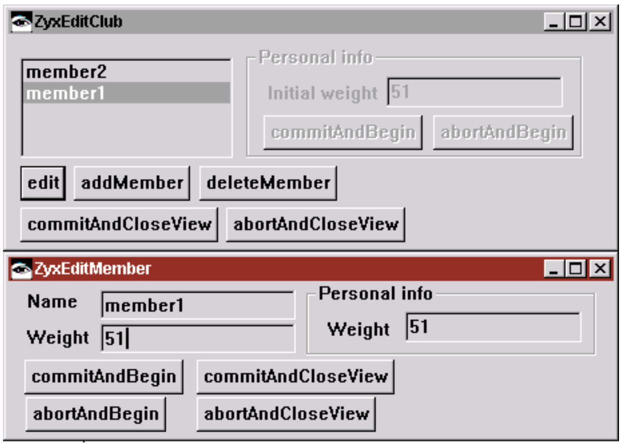
Personal info

Initial weight 50

Figure 32.5. ZyxEditClub dialog with disabled ZyxEditMember groupbox

32.29. Click on **edit**. The ZyxEditMember dialog is opened.

32.30. In the **ZyxEditMember** dialog: Change **weight** to **51**. Note that the change is also displayed in the disabled ZyxEditClub groupbox:



ZyxEditClub

member2
member1

Personal info

Initial weight 51

ZyxEditMember

Name member1

Weight 51

Personal info

Weight 51

Figure 32.6. ZyxEditClub / ZyxEditMember dialogs

32.31. Close the dialogs.

33. Static ZyxEditClubView Notebook containing ZyxEditMember as child process

There are several types of GroupControls, including GroupBoxes and Notebooks. In this chapter you will add a notebook to ZyxEditClubView whose functions are analogous to those of the GroupBox in the previous chapter.

In this chapter you will:

- Add a Notebook to ZyxEditClubView2.
- Add a Notebook Page to the Notebook.
- Add a label and text field to the Notebook page for weight.
- Add commitAndBegin / abortAndBegin buttons to the Notebook page.
- Test.

33.1. Add Notebook to ZyxEditClubView2

33.1. Add a **PM Notebook** to **ZyxEditClubView2**.

Figure 33.1. PM Notebook in the parts palette

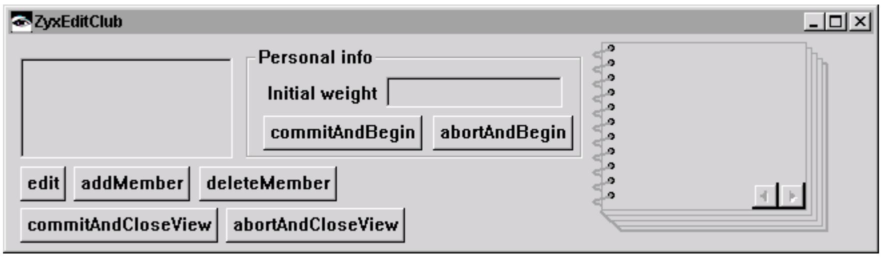


Figure 33.2. ZyxEditClubView with PM Notebook

33.2. Change the **Notebook** property **partName** to **eCPConn_personalInfo_**.

33.2. Add Notebook page to Notebook

33.3. Add a **Notebook page** to the **Notebook**.

Figure 33.3. Notebook page in the parts palette

33.4. Change the **Notebook page** property **partName** to **X_X_X_eMPChConn_**.

33.5. Change the **Notebook page** property **tabLabel** to **Weight** (in the **Notebook properties** dialog click on the button on the **tabLabel** drop-down list to open the **Tab label** dialog, where you can enter the text **Weight**).

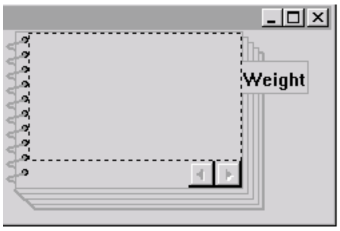


Figure 33.4. ZyxEditClubView Notebook with page Weight

33.3. Add <weight> label and text field to Notebook page

33.6. Add a **Label** to the **Notebook page**.

33.7. Change **Label** property **object** to **Initial weight**.

33.8. Add a **Text Field** to the **Notebook page**.

33.9. Change **Text Field** property **partName** to **Y_weight_** ("Y" for a unique partName).

33.4. Add commitAndBegin and abortAndBegin buttons to Notebook page

33.10. Add a **Button** to **GroupBox**.

33.11. Change **Button** property **object** to **commitAndBegin**.

33.12. Change **Button** property **partName** to **Y_commitAndBegin_** ("Y" for a unique partName).

33.13. Add a **Button** to **GroupBox**.

33.14. Change **Button** property **object** to **abortAndBegin**.

33.15. Change **Button** property **partName** to **Y_abortAndBegin_** ("Y" for a unique partName).

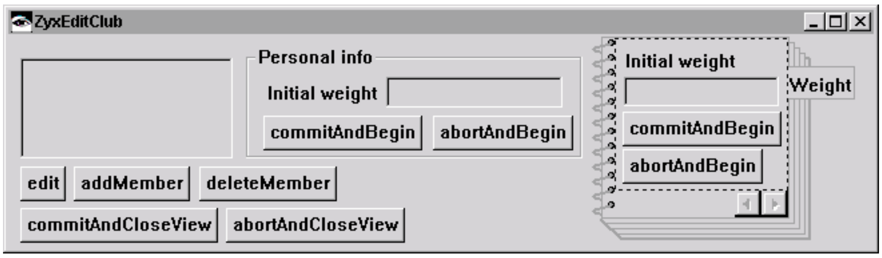


Figure 33.5. ZyxEditClubView with finished notebook page

33.16. Save the part.

33.17. Close the CE.

33.18. Save the image.

33.5. Test

33.19. In the workspace: Execute the following code:

```
FW CH 33. ZyxEditClubView static PM Notebook containing ZyxEditMember as child
process."
[
  | club member1 member2 |
  ZyxMember initializeValidation.
  member1 := ZyxMember new
    name: 'member1';
    weight: -1.
  member2 := ZyxMember new
    name: 'member2';
    weight: -1.
  club := ZyxClub new.
  club members add: member1.
  club members add: member2.
  ZyxEditClub new openOn: club.
]
```

The following dialog appears:

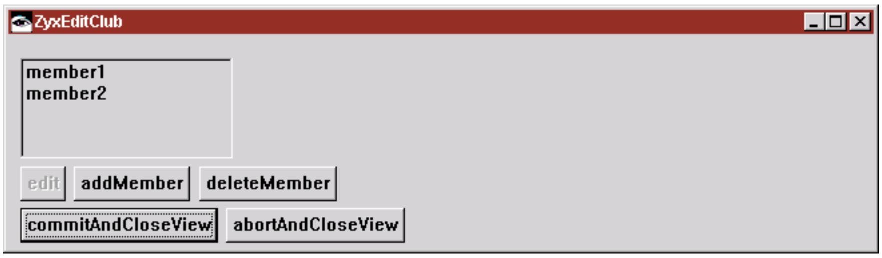


Figure 33.6. ZyxEditClub dialog with hidden GroupBox and PM notebook

33.20. Select **member1**. Note that the the GroupBox **Personal info** and the **Notebook** are visible:

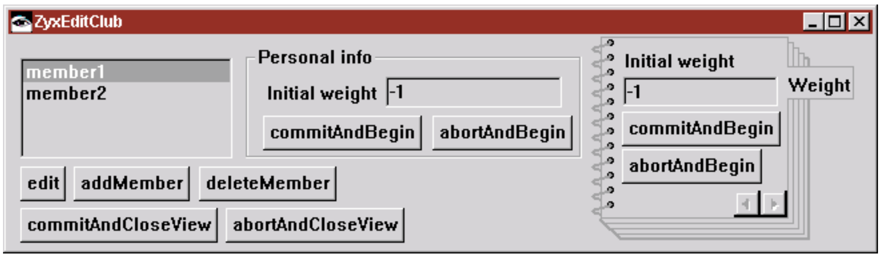


Figure 33.7. ZyxEditClub dialog with visible GroupBox and PM Notebook

33.21. Test the Notebook as you tested the GroupBox in the previous chapter. Note that the notebook page functions exactly as the groupbox, and that any changes in the notebook are reflected in the GroupBox (and vice versa).

34. Authorization: DO attribute accessors

Authorization refers to the ability to enable or disable the reading / writing of a DO attribute using methods of the following format:

- DO>>authorizeRead #variable using: [(expression that evaluates to boolean)].
- DO>>authorizeWrite #variable using: [(expression that evaluates to boolean)].

For detailed information about Authorization of DO attribute accessors, see the ***Application Framework User's Guide*** chapter '4.6. Authorization' (page 195).

In this chapter you will:

- Create MicFwDomainObject subclass ZyxUser. ZyxUser will be the object that is authorized to read / write attributes.
- Create ZyxUser class>>current. This method returns a reference to the single instance of ZyxUser.
- In the ONB: Add variables ZyxUser>>authorizeReadWeight, authorizeWriteWeight. These variables store the authorization condition for the ZyxUser instance.
- Add authorizeReadWeight / authorizeWriteWeight toggle buttons to ZyxEditClubView2. These buttons can be used to easily change the authorization for the user.
- Create ZyxClub>>authorizeReadWeight / authorizeWriteWeight accessors. These methods store the input from the buttons to the ZyxUser.
- Create ZyxMember class>>initializeAuthorization. This is the method that initializes authorization for ZyxMember attributes.
- Modify ZyxEditClubViewPort2>>personalInfoEnabled so that the GroupBoxes in ZyxEditClubView are always enabled.
- Test.

34.1. Create MicFwDomainObject subclass ZyxUser

34.1. In **Visual Age Organizer**: Create and open (make sure the **Open now** checkbox is checked) a new part for **ZyxApplication** with:

- **Part class**: **ZyxUser**.
- **Part type**: **Domain object class**.
- **Inherits from**: **MicFwDomainObject**.

34.2. Create ZyxUser class>>current

34.2. Add **ZyxUser class** variable **Current**.

```
MicFwDomainObject subclass: #ZyxUser
  instanceVariableNames: ''
  classVariableNames: 'Current'
  poolDictionaries: ''
```

34.3. Create the following method:

```
ZyxUser class>>current
  Current isNil ifTrue: [ Current := self new initialize ].
  ^Current.
```

34.4. Save the class.

34.3. Add variables ZyxUser>>authorizeReadWeight, authorizeWriteWeight (in ONB)

34.5. In the **ONB** on **ZyxUser**: Add **ZyxUser>>authorizeReadWeight** (no typing).

34.6. In the **ONB** on **ZyxUser**: Add **ZyxUser>>authorizeWriteWeight** (no typing).

34.7. **Save the class**.

34.8. Close the ONB.

34.4. Add authorizeReadWeight, authorizeWriteWeight toggle buttons to ZyxEditClubView2

34.9. In the **Composition Editor** on **ZyxEditClubView2**: Add a **Toggle button** to **ZyxEditClubView**.

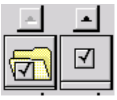


Figure 34.1. Toggle button (checkbox) in the parts palette

34.10. Change the **toggle button** property **object** to **authorizeReadWeight**.

34.11. Change the **toggle button** property **partName** to **authorizeReadWeight_**.

34.12. Add **Toggle button** to **ZyxEditClubView**.

34.13. Change the **toggle button** property **object** to **authorizeWriteWeight**.

34.14. Change the **toggle button** property **partName** to **authorizeWriteWeight_**.

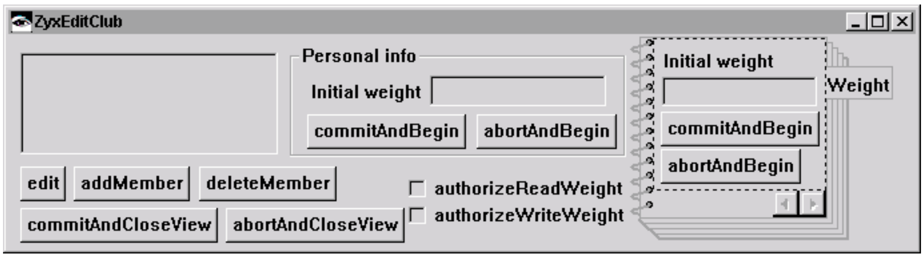


Figure 34.2. ZyxEditClubView2 with the authorizeReadWeight, authorizeWriteWeight toggle buttons

34.15. **Save the part**.

34.16. Close the CE.

34.5. Create ZyxClub>>authorizeReadWeight, authorizeWriteWeight accessors

34.17. Create the following method:

```
ZyxClub>>authorizeReadWeight
  ^ZyxUser current authorizeReadWeight
```

34.18. Create the following method:

```
ZyxClub>>authorizeReadWeight: aBoolean
  ZyxUser current authorizeReadWeight: aBoolean.
```

34.19. Create the following method:

```
ZyxClub>>authorizeWriteWeight
  ^ZyxUser current authorizeWriteWeight
```

34.20. Create the following method:

```
ZyxClub>>authorizeWriteWeight: aBoolean
  ZyxUser current authorizeWriteWeight: aBoolean.
```

34.21. Save the class.

34.6. Create ZyxMember class>>initializeAuthorization

34.22. Create the following method:

```
ZyxMember class>>initializeAuthorization
  ZyxUser current authorizeReadWeight
      ifNil:  [ ZyxUser current authorizeReadWeight: true ].
  self authorizeRead: #weight
       using: [ ZyxUser current authorizeReadWeight ].
  ZyxUser current authorizeWriteWeight
      ifNil:  [ ZyxUser current authorizeWriteWeight: true ].
  self authorizeWrite: #weight
       using: [ ZyxUser current authorizeWriteWeight ].
```

34.23. Save the class.

34.7. Modify ZyxEditClubViewPort2>>personalInfoEnabled

34.24. Modify the following method:

```
ZyxEditClub>>personalInfoEnabled
  ^true.
```

34.25. Save the class.

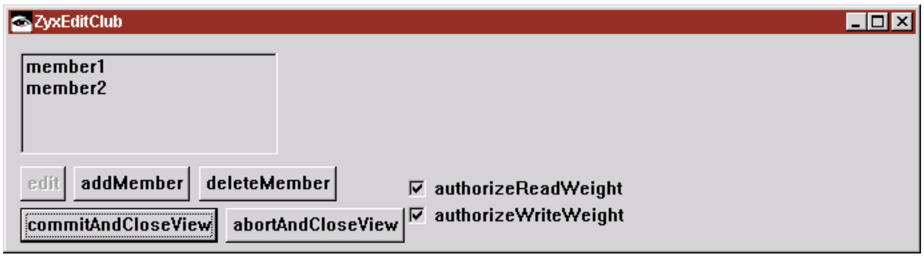
34.26. Save the image.

34.8. Test

34.27. In the workspace: Execute the following code (note that a line has been added to initializeAuthorization for ZyxMember attributes):

```
FW CH 34. ZyxMember>>weight read and write authorization."
[
   | club member1 member2 |
   ZyxMember initializeValidation.
   ZyxMember initializeAuthorization. "added line"
   member1 := ZyxMember new
      name: 'member1';
      weight: 111.
   member2 := ZyxMember new
      name: 'member2';
      weight: 222.
   club := ZyxClub new.
   club members add: member1.
   club members add: member2.
   ZyxEditClub new openOn: club.
]
```

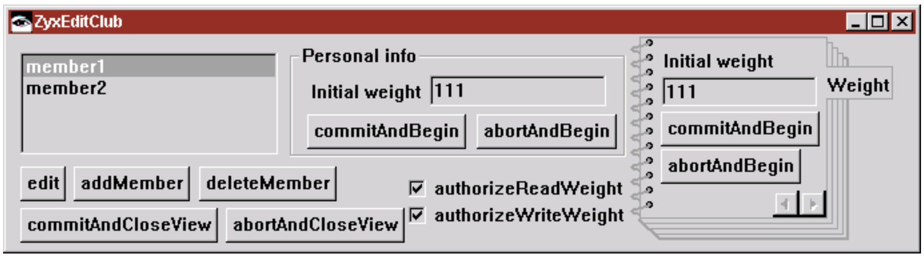
The following dialog opens:



☑ authorizeReadWeight
☑ authorizeWriteWeight

Figure 34.3. ZyxEditClub dialog with authorizeReadWeight / authorizeWriteWeight toggle buttons

34.28. Select **member1**. The **weight** attribute for member1 is displayed.

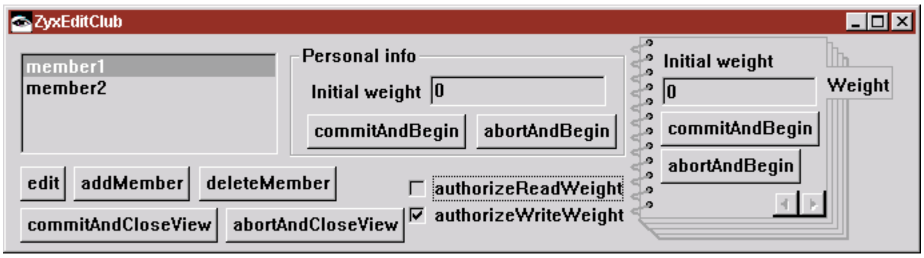


Initial weight 111

☑ authorizeReadWeight
☑ authorizeWriteWeight

Figure 34.4. ZyxEditClub dialog after member selected

34.29. **Uncheck** the toggle button **authorizeReadWeight**. Note that a value of 0 is displayed for the weight:

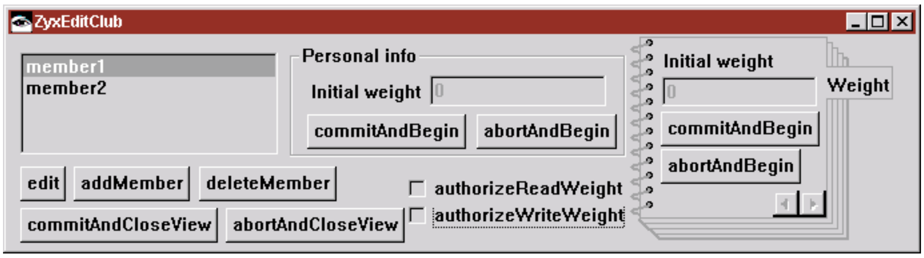


Initial weight 0

☐ authorizeReadWeight
☑ authorizeWriteWeight

Figure 34.5. ZyxEditClub dialog after reading of ZyxMember attribute weight unauthorized

34.30. **Uncheck** the toggle button **authorizeWriteWeight**. Note that the entry fields for write are disabled:



Initial weight 0

☐ authorizeReadWeight
☐ authorizeWriteWeight

Figure 34.6. ZyxEditClub dialog after writing of ZyxMember attribute weight unauthorized

34.31. Close the dialog.

35. Authorization: DP methods

Authorization refers to the ability to enable or disable the execution of DP method using methods of the following format:

- DP>>authorizePerform #method using: [(expression that evaluates to boolean)].

For detailed information about Authorization of DP methods, see the ***Application Framework User's Guide*** chapter '4.6. Authorization' (page 195).

In this chapter you will:

- In the ONB: Add variable ZyxUser>>authorizeDeleteMember. This variable stores the authorization for ZyxEditClub>>deleteMember for the ZyxUser instance.
- Add authorizeDeleteMember toggle button to ZyxEditClubView2. This button can be used to easily change the authorization for the user.
- Create ZyxClub>>authorizeDeleteMember accessors. These methods store the input from the check-box to the ZyxUser.
- Create ZyxEditClub class>>initializeAuthorization. This is the method that initializes authorization for ZyxEditClub DP's.
- Test.

35.1. Add variable ZyxUser>>authorizeDeleteMember (in ONB)

35.1. In the **ONB**: Add **ZyxUser>>authorizeDeleteMember** (not typed).

35.2. **Save the class**.

35.3. Close the ONB.

35.2. Add authorizeDeleteMember toggle button to ZyxEditClubView2

35.4. In the **Composition Editor** on **ZyxEditClubView2**: Add a **Toggle button** to **ZyxEditClubView**.

35.5. Change the **toggle button** property **object** to **authorizeDeleteMember**.

35.6. Change the **toggle button** property **partName** to **authorizeDeleteMember_**.

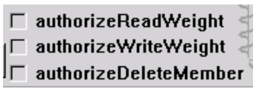


Figure 35.1. authorizeDeleteMember toggle button in ZyxEditClubView2

35.7. **Save the part**.

35.8. Close the CE.

35.3. Create ZyxClub>>authorizeDeleteMember accessors

35.9. Create the following method:

```
ZyxClub>>authorizeDeleteMember
  ^ZyxUser current authorizeDeleteMember
```

35.10. Create the following method:

```
ZyxClub>>authorizeDeleteMember: aBoolean
  ZyxUser current authorizeDeleteMember: aBoolean.
```

35.4. Create ZyxEditClub class>>initializeAuthorization

35.11. Create the following method (note the code in bold):

```
ZyxEditClub class>>initializeAuthorization
  ZyxUser current authorizeDeleteMember
     ifNil:  [ ZyxUser current authorizeDeleteMember: true ].
  self authorizePerform: #deleteMember
      using: [ ZyxUser current authorizeDeleteMember ].
```

35.12. Save the class.

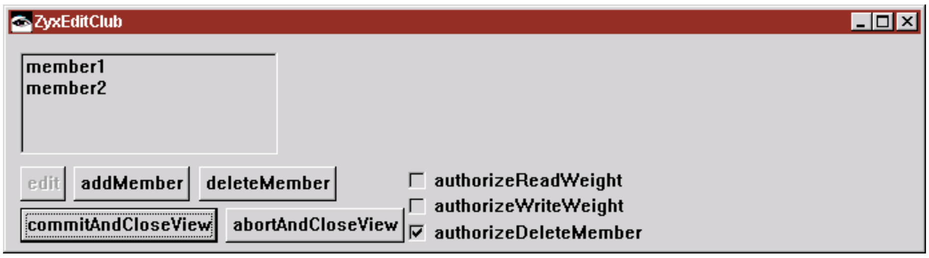
35.13. Save the image.

35.5. Test

35.14. In the workspace: Execute the following code (note that a line has been added to initializeAuthorization for ZyxEditClub methods):

```
FW CH 35. ZyxEditMember>>deleteMember authorization."
[
  | club member1 member2 |
  ZyxMember initializeValidation.
  ZyxMember initializeAuthorization.
  ZyxEditClub initializeAuthorization.  "added line"
  member1 := ZyxMember new
    name: 'member1';
    weight: 111.
  member2 := ZyxMember new
    name: 'member2';
    weight: 222.
  club := ZyxClub new.
  club members add: member1.
  club members add: member2.
  ZyxEditClub new openOn: club.
]
```

The following dialog opens:

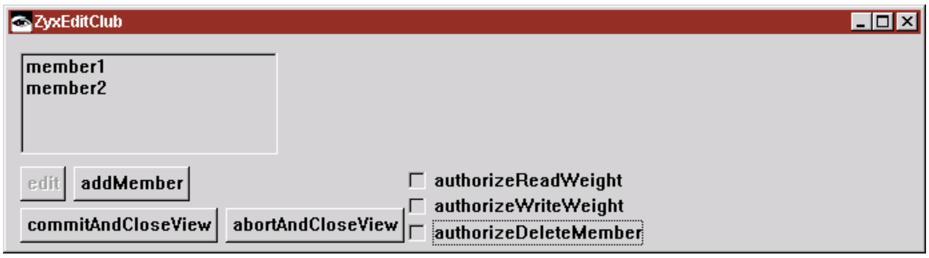


member1
member2

□ authorizeReadWeight
□ authorizeWriteWeight
☑ authorizeDeleteMember

Figure 35.2. ZyxEditClub dialog with authorizeDeleteMember toggle button

35.15. Uncheck the toggle button **authorizeDeleteMember**. Note that the deleteMember button disappears:



member1
member2

□ authorizeReadWeight
□ authorizeWriteWeight
□ authorizeDeleteMember

Figure 35.3. ZyxEditClub dialog after ZyxEditClub method deleteMember unauthorized

35.16. Close the dialog.

36. Modality: MicFwPrimaryApplicationModal

Setting the ***modality*** of a process to ***MicFwPrimaryApplicationModal*** disables the controls of the parent dialog when the process view is opened as a child dialog.

In this chapter you will:

- Create ZyxEditMember class>>modality. This method sets the modality for ZyxEditMember to MicFwPrimaryApplicationModal. Thus, if a ZyxEditMember dialog is opened by pressing the edit key in a ZyxEditClub dialog, then the ZyxEditClub dialog is disabled.
- Test.

36.1. Create ZyxEditMember class>>modality

36.1. Create the following method:

```
ZyxEditMember class>>modality
  ^#MicFwPrimaryApplicationModal
```

36.2. Save the class.

36.3. Save the image.

36.2. Test

36.4. In the workspace: Execute the following code:

```
FW CH 36. ZyxEditMember modality = MicFwPrimaryApplicationModal."
[
  | club member1 |
  member1 := ZyxMember new
    name: 'member1';
    weight: 99.
  club := ZyxClub new.
  club members add: member1.
  ZyxEditClub new openOn: club.
  ZyxEditClub new openOn: club.
]
```

The following 2 ZyxEditClub dialogs appear (the dialogs have been resized in the diagram below):

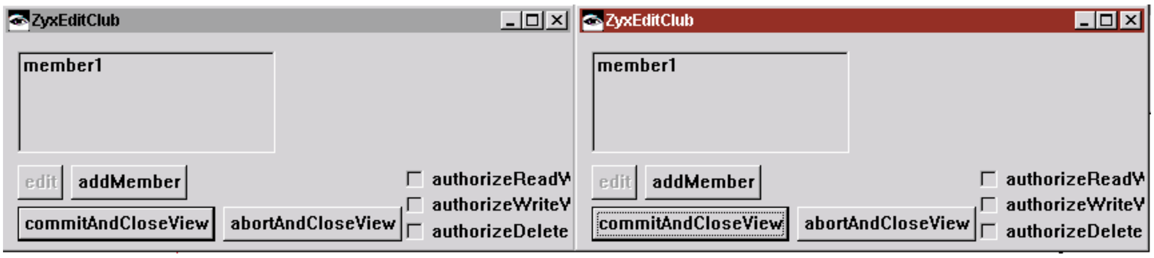


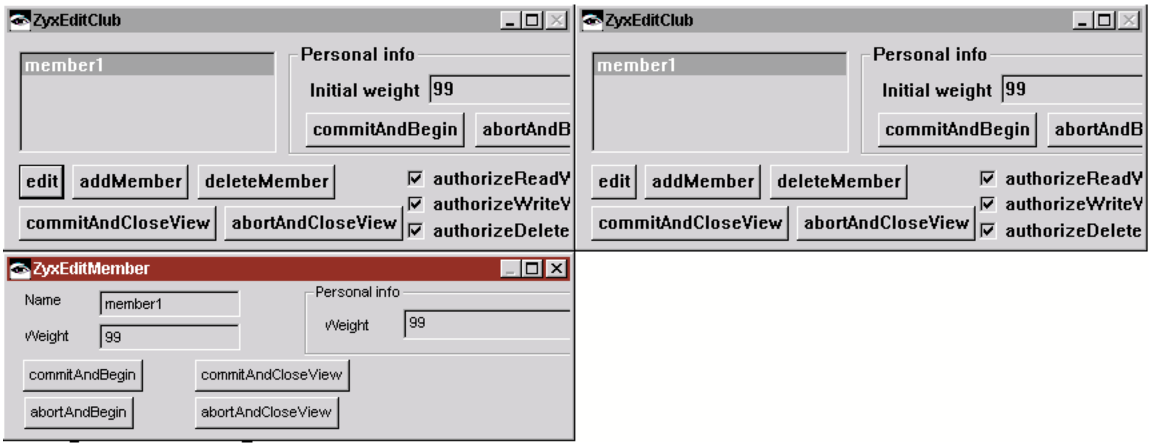
Figure 36.1. 2 ZyxEditClub dialogs

2 ZyxEditClub dialogs appear.

36.5. Check all authorization checkboxes.

36.6. In the **RIGHT ZyxEditClub** dialog: Select **member1**.

36.7. In the **LEFT ZyxEditClub** dialog: Click on **edit**. The LEFT ZyxEditMember dialog is opened.:



Initial weight 99

Initial weight 99

Name member1

Weight 99

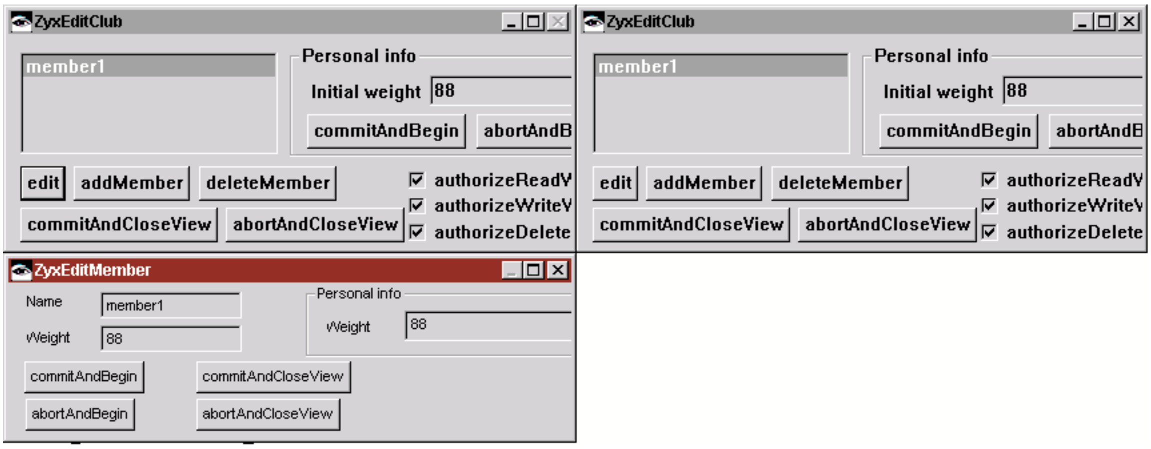
Weight 99

Figure 36.2. 2 ZyxEditClub and LEFT ZyxEditMember dialogs

36.8. Attempt to use a control in the **LEFT ZyxEditClub** dialog. The controls are **NOT** enabled because the dialog has a subdialog.

36.9. In the **RIGHT ZyxEditClub** dialog: Change the **weight** to **88**. Note that the controls **ARE** enabled because the dialog has NO subdialog.

36.10. Click in another dialog to change the focus. Note that the new weight is shown in the other dialogs:



Initial weight 88

Initial weight 88

Name member1

Weight 88

Weight 88

Figure 36.3. Changes in RIGHT ZyxEditClub dialog are reflected in all dialogs

36.11. Note that other **VAST dialogs** (not the dialogs or subdialogs generated from the workspace code) and the dialogs of **non-VAST** applications are **available**.

36.12. Close the dialogs.

37. Modality: MicFwFullApplicationModal

Setting the ***modality*** of a process to ***MicFwFullApplicationModal*** disables the controls in **all** other application dialogs when the process view is opened.

In this chapter you will:

- Modify ZyxEditMember class>>modality. This method sets the modality for ZyxEditMember to MicFwFullApplicationModal. Thus, if a ZyxEditMember dialog is opened, then all other application dialogs are disabled.
- Test.

37.1. Modify ZyxEditMember class>>modality

37.1. Edit the method as shown:

```
ZyxEditMember class>>modality
  ^#MicFwFullApplicationModal
```

37.2. Save the class.

37.3. Save the image.

37.2. Test

37.4. In the workspace: Execute the following code:

```
FW CH 37. ZyxEditMember modality = MicFwFullApplicationModal."
[
  | club member1 |
  member1 := ZyxMember new
    name: 'member1';
    weight: 99.
  club := ZyxClub new.
  club members add: member1.
  ZyxEditClub new openOn: club.
  ZyxEditClub new openOn: club.
]
```

2 ZyxEditClub dialogs appear.

37.5. Check all authorization checkboxes.

37.6. In the **RIGHT ZyxEditClub** dialog: Select **member1**.

37.7. In the **LEFT ZyxEditClub** dialog: Click on **edit**. The LEFT ZyxEditMember dialog is opened.

37.8. Attempt to use a control in the **LEFT ZyxEditClub** dialog. The controls are **NOT** enabled because the dialog has a subdialog.

37.9. In the **RIGHT ZyxEditClub** dialog: The controls are **NOT** enabled because the other ZyxEditClub dialog has a subdialog.

37.10. Note that other **VAST dialogs** (not the dialogs or subdialogs generated from the workspace code) and the dialogs of **non-VAST** applications are **available**.

37.11. Close the dialogs.

38. Modality: MicFwSystemModal

Setting the ***modality*** of a process to ***MicFwSystemModal*** disables the controls in **all** other VAST dialogs when the process view is opened.

In this chapter you will:

- Modify ZyxEditMember class>>modality. This method sets the modality for ZyxEditMember to MicFwSystemModal. Thus, if a ZyxEditMember dialog is opened, then all other VAST dialogs are disabled.
- Test.

38.1. Edit ZyxEditMember class>>modality

38.1. Edit the method as shown:

```
ZyxEditMember class>>modality
   ^#MicFwSystemModal
```

38.2. Save the class.

38.3. Save the image.

38.2. Test

38.4. In the workspace: Execute the following code:

```
FW CH 38. ZyxEditMember modality = MicFwSystemModal."
[
   | club member1 |
   member1 := ZyxMember new
      name: 'member1';
      weight: 99.
   club := ZyxClub new.
   club members add: member1.
   ZyxEditClub new openOn: club.
]
```

1 ZyxEditClub dialog appears.

38.5. In **ZyxEditClub** dialog: Select **member1**.

38.6. Click on **edit**. The ZyxEditMember dialog is opened.

38.7. Note that all other **VAST dialogs** (the System Transcript, Visual Age Organizer, etc.) are **NOT available**.

38.8. Close the ZyxEditMember dialog. Note that all controls in all dialogs are again available.

38.9. Close the ZyxEditClub dialog.

39. Dynamic GroupBox

The contents of a GroupBox can be dynamic, which means that the content of the GroupBox can change dynamically during program execution.

In this chapter you will:

- Disable modality (from the previous chapter).
- Create ZyxMember subclasses ZyxStudent, ZyxTeacher. In order to simplify the implementation, it is assumed that Teachers and Students are all Club Members. The terms Teacher and Student are used rather than Trainer and Trainee because they are more easily distinguished.
- Modify the ZyxEditClubView2 GroupBox:
  - Change the name. The name will be part of the name of the methods for implementing the dynamic nature of the GroupBox.
  - Delete the parts inside the GroupBox. The only contents of the GroupBox will be the selected form.
- Create ZyxEditClubViewPort2>>memberFormLabel. <aspectName>Label is the method that returns the String object that will be used as the label for the dynamic group box.
- Create ZyxEditClubViewPort2>>memberForm. memberForm returns a MicFwMetaControlList with a MicFwMetaPart. The MicFwMetaPart contains an accessor string and part class name for the form that in the dynamic group box. The type of form returned depends on whether or not the selected member is a ZyxStudent or ZyxTeacher.
- Create ZyxStudentForm. ZyxStudentForm is the form in the dynamic group box when a ZyxStudent is selected.
- Create ZyxTeacherForm. ZyxTeacherForm is the form in the dynamic group box when a ZyxTeacher is selected.
- Test.

39.1. Disable modality (edit ZyxEditMember class>>modality)

39.1. Edit the method as shown to disable modality (for the same result: delete the method):

```
ZyxEditMember class>>modality
  ^#MicFwNoModality
```

39.2. Save the class.

39.2. Create ZyxMember subclasses ZyxStudent, ZyxTeacher

39.3. In **Visual Age Organizer**: Create a new part for **ZyxApplication** with:

- **Part class**: **ZyxStudent**.
- **Part type**: **Domain object class**.
- **Inherits from**: **ZyxMember**.

39.4. In **Visual Age Organizer**: Create a new part for **ZyxApplication** with:

- **Part class**: **ZyxTeacher**.
- **Part type**: **Domain object class**.
- **Inherits from**: **ZyxMember**.

39.3. Modify GroupBox in ZyxEditClubView2

39.5. In **ZyxEditClubView2**: Change **GroupBox partName** to **X_memberForm_**.

39.6. Delete the parts inside the GroupBox (**text label, text field, 2 push buttons**).

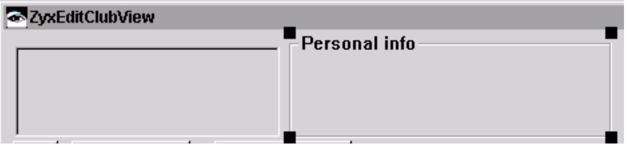


Figure 39.1. Dynamic group box in ZyxEditClubView2

39.7. **Save the part**.

39.4. Create ZyxEditClubViewPort2>>memberFormLabel

<aspectName>Label is the method that returns the String object that will be used as the label for the dynamic group box.

39.8. Create the following method:

```
ZyxEditClubViewPort2>>memberFormLabel
  self model selectedMember isNil
    ifTrue: [ ^'Selected is not teacher nor student' ].
  ( self model selectedMember isKindOf: ZyxStudent )
    ifTrue: [ ^'Student info' ].
  ( self model selectedMember isKindOf: ZyxTeacher )
    ifTrue: [ ^'Teacher info' ].
```

39.5. Create ZyxEditClubViewPort2>>memberForm

memberForm returns a MicFwMetaControlList with a MicFwMetaPart. The MicFwMetaPart contains an accessor string and part class name for the form that in the dynamic group box.

39.9. Create the following method:

```
ZyxEditClubViewPort2>>memberForm
  | mpc |
  mpc := self newMetaControlListForAspect: #memberForm.
  self model selectedMember isNil ifTrue: [ ^mpc ].
  ( self model selectedMember isKindOf: ZyxTeacher ) ifTrue:
    [ mpc add: ( MicFwMetaPart new
        accessorString: 'VOID VOID VOID eMPChConn'
        partClassName: #ZyxTeacherForm ) ].
  ( self model selectedMember isKindOf: ZyxStudent ) ifTrue:
    [ mpc add: ( MicFwMetaPart new
        accessorString: 'VOID VOID VOID eMPChConn'
        partClassName: #ZyxStudentForm ) ].
  ^mpc.
```

39.6. Create ZyxStudentForm

39.10. Create Visual Part **ZyxStudentForm** (open now).

39.11. Delete the **window**.

39.12. Add a **Form**.

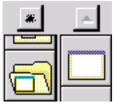


Figure 39.2. Form part in the parts palette

39.13. Add a **label** with property **object** as **Student weight**.

39.14. Add a **Text field** with property **partName** as **weight_**.

39.15. Add a **button** with **object CommitAndBegin** and **partName commitAndBegin_**.

39.16. Add a **button** with **object AbortAndBegin** and **partName abortAndBegin_**.

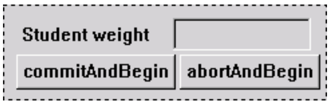


Figure 39.3. ZyxStudentForm

39.17. Save the part.

39.7. Create ZyxTeacherForm

39.18. Copy Visual Part **ZyxStudentForm** to **ZyxTeacherForm**.

39.19. Change the **label** property **object** to **Teacher weight**.

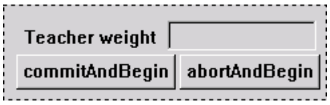


Figure 39.4. ZyxTeacherForm

39.20. Save the part.

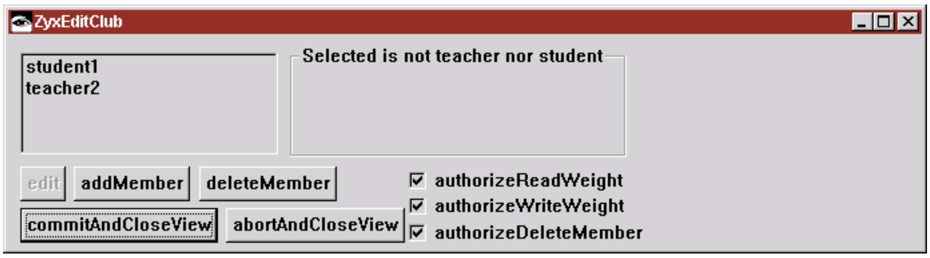
39.21. Save the image.

39.8. Test

39.22. Execute the following code in the workspace (note the lines for ZyxTeacher, ZyxStudent initializeAuthorization: initializing authorization for the parent class does not initialize authorization for subclasses):

```
FW CH 39. Dynamic GroupBox."
[
  | club student teacher  |
  ZyxMember initializeValidation.
  ZyxMember initializeAuthorization.
  ZyxTeacher initializeAuthorization. "added line"
  ZyxStudent initializeAuthorization. "added line"
  ZyxEditClub initializeAuthorization.
  student := ZyxStudent new
    name: 'student1';
    weight: 111.
  teacher := ZyxTeacher new
    name: 'teacher2';
    weight: 222.
  club := ZyxClub new.
  club members add: student.
  club members add: teacher.
  ZyxEditClub new openOn: club.
]
```

The following dialog is opened:



ZyxEditClub

student1
teacher2

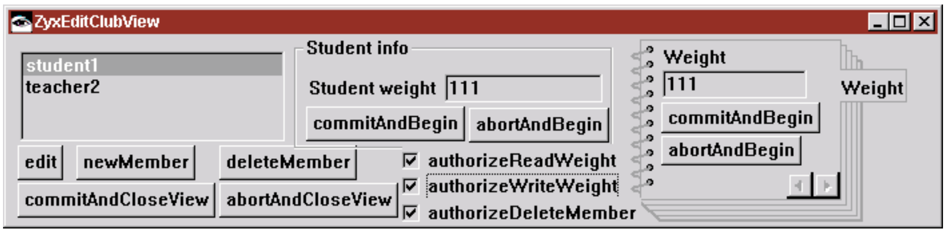
Selected is not teacher nor student

☑ authorizeReadWeight
☑ authorizeWriteWeight
☑ authorizeDeleteMember

Figure 39.5. ZyxEditClub with dynamic group box

39.23. Enable **read**, **write**, and **delete** authorization (check the checkboxes).

39.24. Select **student1**.



ZyxEditClubView

student1
teacher2

Student info

Student weight 111

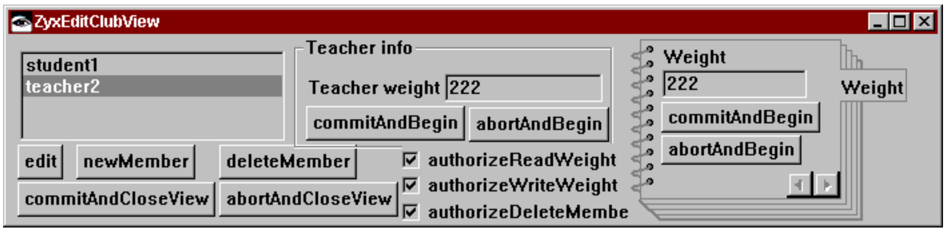
Weight

111

☑ authorizeReadWeight
☑ authorizeWriteWeight
☑ authorizeDeleteMember

Figure 39.6. ZyxStudentForm in ZyxEditClub dialog

39.25. Select **teacher2**.



ZyxEditClubView

student1
teacher2

Teacher info

Teacher weight 222

Weight

222

☑ authorizeReadWeight
☑ authorizeWriteWeight
☑ authorizeDeleteMembe

Figure 39.7. ZyxTeacherForm in ZyxEditClub dialog

39.26. Close the dialog.

40. Dynamic Notebook

The contents of a Notebook can be dynamic.

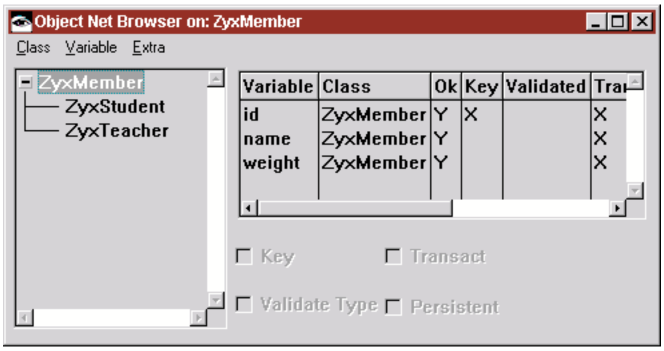
In this chapter you will:

- Create a relationship betwen ZyxTeacher and ZyxStudent. A Teacher can have any number of Students (cardinality 0..N), and a Student can have only 1 Teacher (cardinality 1..1).
- Change the name of the ZyxEditClubView2 Notebook. The name will be part of the name of the methods for implementing the dynamic nature of the GroupBox.
- Create ZyxEditClubViewPort2>>pages. Method pages returns a MicFwMetaControlList with a MicFwMetaPart (or no MicFwMetaPart if no member is selected). The MicFwMetaPart contains an accessor string, part class name and label for the dynamic notebook page.
- Create ZyxAssignedStudentsForm. ZyxAssignedStudentsForm is a notebook page if a ZyxTeacher is selected.
- Create ZyxAssignedTeacherForm. ZyxAssignedTeacherForm is a notebook page if a ZyxStudent is selected.
- Create ZyxEditMemberViewPort>>teacherName. teacherName returns the name of the teacher of the selected student.
- Create ZyxEditMemberViewPort>>teacher. teacher returns the teacher assigned to a student.
- Create ZyxEditMemberViewPort>>students. students returns the collection of students assigned to the teacher.
- Test.

40.1. Establish 1<->N relationship between ZyxStudent, ZyxTeacher

A ZyxTeacher can have any number of ZyxStudents. A ZyxStudent can have only 1 ZyxTeacher.

40.1. Open the **ONB** on **ZyxMember**. Note that ZyxStudent and ZyxTeacher are included in the Object Net:



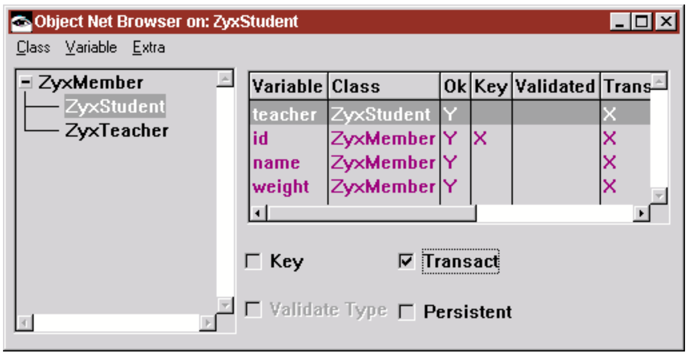
Variable	Class	Ok	Key	Validated	Tra
id	ZyxMember	Y	X		X
name	ZyxMember	Y			X
weight	ZyxMember	Y			X

□ Key □ Transact

□ Validate Type □ Persistent

Figure 40.1. ZyxMember and subclasses in the ONB

40.2. Add instance variable **teacher** to **ZyxStudent** (**transacted**). Note that the variables inherited from ZyxMember are displayed in purple:



Variable	Class	Ok	Key	Validated	Trans
teacher	ZyxStudent	Y			X
id	ZyxMember	Y	X		X
name	ZyxMember	Y			X
weight	ZyxMember	Y			X

□ Key ☑ Transact

□ Validate Type □ Persistent

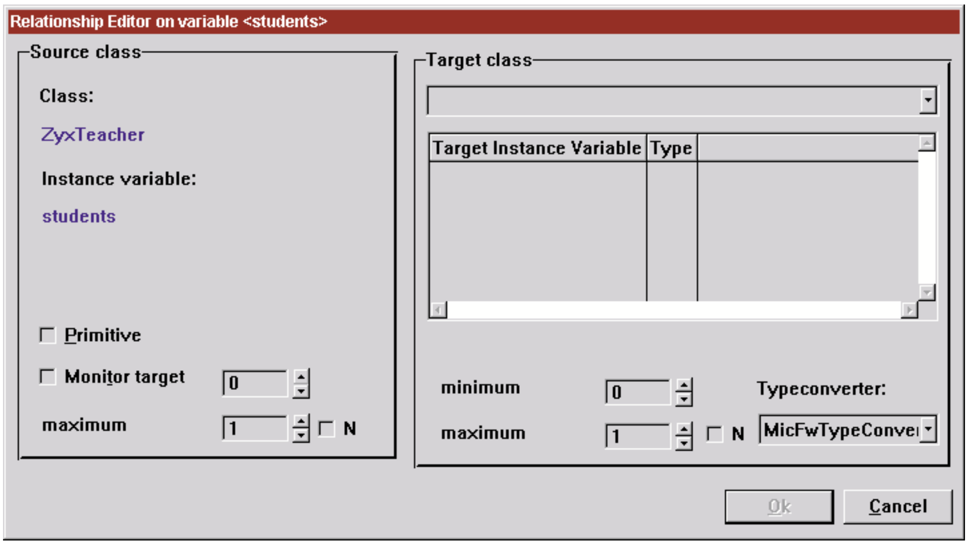
Figure 40.2. ZyxStudent variables in the ONB

40.3. Add instance variable **students** to **ZyxTeacher** (**transacted**).

40.4. Select **ZyxTeacher>>students** in the ONB.

40.5. Right-click.

40.6. Select **Set Relationship**. The Relationship Editor on ZyxTeacher>>students appears:



Class: ZyxTeacher

Instance variable: students

□ Primitive

□ Monitor target 0

maximum 1 □ N

Target Instance Variable	Type

minimum 0

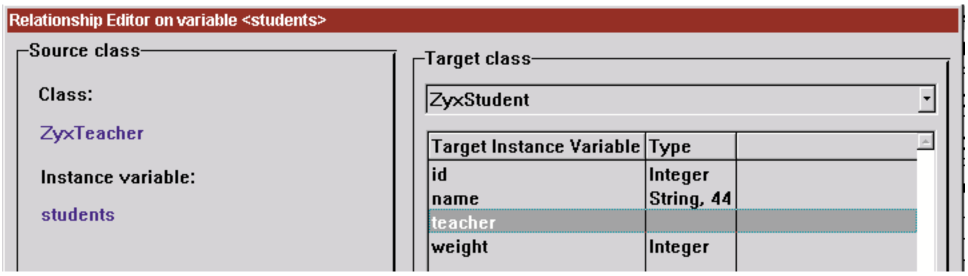
maximum 1 □ N

Typeconverter: MicFwTypeConve

Figure 40.3. Relationship Editor on ZyxTeacher>>students

40.7. From the **Target Class** drop-down list: Select **ZyxStudent**.

40.8. In the **Target Instance Variable** box: Select **teacher**.



Class: ZyxTeacher

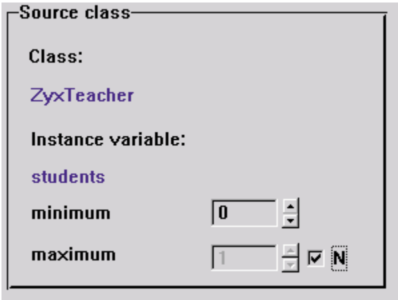
Instance variable: students

ZyxStudent

Target Instance Variable	Type
id	Integer
name	String, 44
teacher	
weight	Integer

Figure 40.4. Selected Target class / instance variable ZyxStudent>>teacher in Relationship Editor

40.9. In the **Source class** box: Check the **N checkbox** for **maximum**.



Class: ZyxTeacher

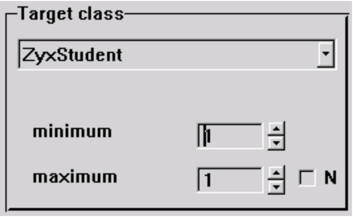
Instance variable: students

minimum 0

maximum 1 ☑ N

Figure 40.5. Setting ZyxTeacher>>students cardinality to 0...N

40.10. In the **Target class** box: Select **1** as the **minimum**.



ZyxStudent

minimum 1

maximum 1 □ N

Figure 40.6. Setting ZyxStudent>>teacher cardinality to 1...1

40.11. Click **OK**. Note that the type is now defined in the ONB for ZyxTeacher>>students.

40.12. Save all changes.

40.13. Close the ONB.

40.2. Rename notebook in ZyxEditClubView2

40.14. In **ZyxEditClubView2**: Change the **notebook** (**not the notebook page**) **partName** to **X_pages_X_X_**.

40.15. **Save the part**.

40.3. Create ZyxEditClubViewPort2>>pages

Method pages returns a MicFwMetaControlList with a MicFwMetaPart (or no MicFwMetaPart if no member is selected). The MicFwMetaPart contains an accessor string, part class name and label for the dynamic notebook page.

40.16. Create the following method:

```
ZyxEditClubViewPort2>>pages
  | mpc |
  mpc := self newMetaControlListForAspect: #pages.
  self model selectedMember isNil ifTrue: [ ].
  ( self model selectedMember isKindOf: ZyxTeacher ) ifTrue:
    [ self model selectedMember students isNil ifFalse:
      [ mpc add: ( MicFwMetaPart new
        accessorString: 'VOID VOID VOID VOID'
        partClassName: #ZyxAssignedStudentsForm
        label: ’Students’ ) ] ].
  ( self model selectedMember isKindOf: ZyxStudent ) ifTrue:
    [ self model selectedMember teacher isNil ifFalse:
      [ mpc add: ( MicFwMetaPart new
        accessorString: 'VOID VOID VOID VOID'
        partClassName: #ZyxAssignedTeacherForm
        label: ’Teacher’ ) ] ].
  ^mpc.
```

40.4. Create ZyxAssignedStudentsForm

ZyxAssignedStudentsForm is a notebook page if a ZyxTeacher is selected.

40.17. Create Visual Part **ZyxAssignedStudentsForm**:

40.18. Delete the **window**.

40.19. Add a **Form**.

40.20. Add a **label** with property **object** as **Assigned students**.

40.21. Add a **List** with property **partName** as **eMPChConn_students_X_X_name_**.

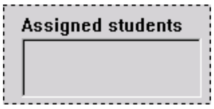


Figure 40.7. ZyxAssignedStudentsForm

40.22. **Save the part**.

40.5. Create ZyxAssignedTeacherForm

ZyxAssignedTeacherForm is a notebook page if a ZyxStudent is selected.

40.23. Create Visual Part **ZyxAssignedTeacherForm**:

40.24. Delete the **window**.

40.25. Add a **Form**.

40.26. Add a **label** with property **object** as **Assigned teacher**.

40.27. Add a **Text field** with property **partName** as **eMPChConn_teacherName_**.

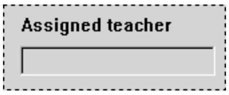


Figure 40.8. ZyxAssignedTeacherForm

40.28. **Save the part**.

40.6. Create ZyxEditMemberViewPort>>teacherName

teacherName returns the name of the teacher of the selected student.

40.29. Create the following method:

```
ZyxEditMemberViewPort>>teacherName
  ^self model eMBConn teacher name
```

40.7. Create ZyxEditMemberViewPort>>teacher

teacher returns the teacher assigned to a student.

40.30. Create the following method:

```
ZyxEditMemberViewPort>>teacher
  ^self model eMBConn teacher
```

40.8. Create ZyxEditMemberViewPort>>students

students returns the collection of students assigned to the teacher.

40.31. Create the following method:

```
ZyxEditMemberViewPort>>students
  ^self model eMBConn students
```

40.32. Save the method.

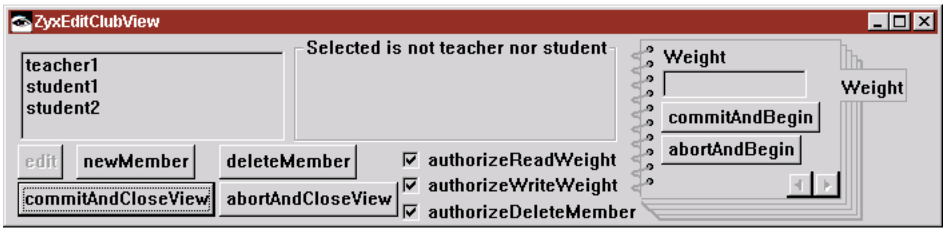
40.33. Save the image.

40.9. Test

40.34. Execute the following code in the workspace:

```
FW CH 40. Dynamic Notebook."
[
  | club student1 student2 teacher1  |
  ZyxMember initializeValidation.
  ZyxMember initializeAuthorization.
  ZyxTeacher initializeAuthorization.
  ZyxStudent initializeAuthorization.
  ZyxEditClub initializeAuthorization.
  student1 := ZyxStudent new
    name: 'student1';
    weight: 111.
  student2 := ZyxStudent new
    name: 'student2';
    weight: 112.
  teacher1 := ZyxTeacher new
    name: 'teacher1';
    weight: 222.
  student1 teacher: teacher1.
  student2 teacher: teacher1.
  teacher1 students add: student1.
  teacher1 students add: student2.
  club := ZyxClub new.
  club members add: student1.
  club members add: student2.
  club members add: teacher1.
  ZyxEditClub new openOn: club.
]
```

The following dialog is opened:



teacher1
student1
student2

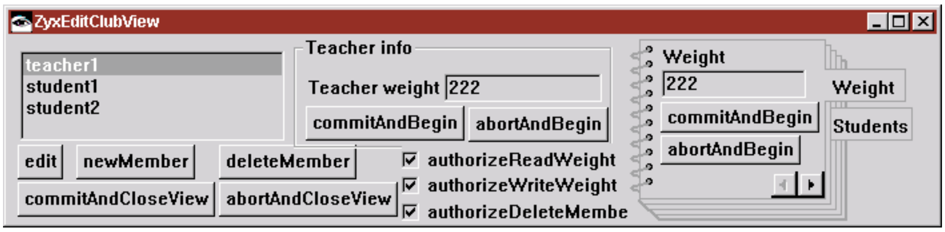
Selected is not teacher nor student

☑ authorizeReadWeight
☑ authorizeWriteWeight
☑ authorizeDeleteMember

Figure 40.9. ZyxEditClub dialog with dynamic notebook

Note that the notebook contains only the single static page.

40.35. Select **teacher1**. Note that the notebook contains the static page and a dynamic **Students** page.


ZyxEditClubView

teacher1
student1
student2

Teacher info

Teacher weight 222

☑ authorizeReadWeight
☑ authorizeWriteWeight
☑ authorizeDeleteMembe

Weight
222

Figure 40.10. ZyxEditClub dynamic notebook with teacher selected

40.36. Select the **Students** page.

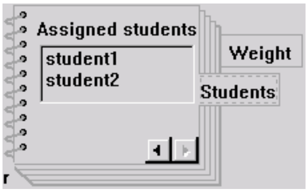
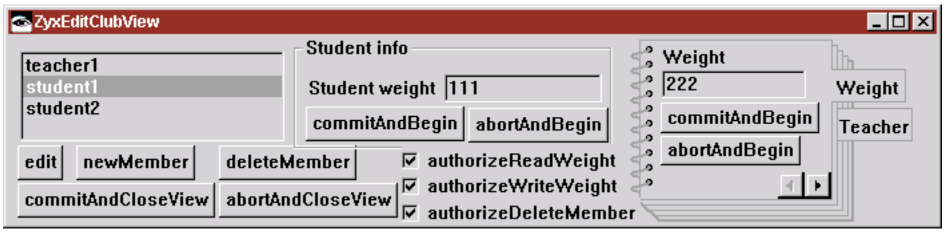

Assigned students
student1
student2

Figure 40.11. Dynamic notebook tab Students

40.37. Select **student1**. Note that the notebook contains the static page and a dynamic **Teacher** page.


ZyxEditClubView

teacher1
student1
student2

Student info

Student weight 111

☑ authorizeReadWeight
☑ authorizeWriteWeight
☑ authorizeDeleteMember

Weight
222

Figure 40.12. ZyxEditClub dynamic notebook with student selected

40.38. Select the **Teacher** page.

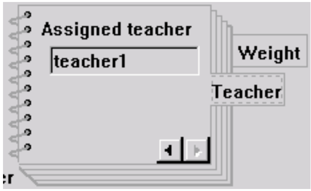

Assigned teacher
teacher1

Figure 40.13. Dynamic notebook tab Students

40.39. Close the dialog.

41. Container Icon Tree

In this chapter you will:

- Add a ***Container Icon Tree*** that displays the Members in a tree format (and add a heading for the existing List).
- Create ViewPorts ZyxTeacherViewPort, ZyxStudentViewPort.
- Create class method portName for ZyxTeacherViewPort, ZyxStudentViewPort. These methods specify the port name for a class when the port name is not specified in the DPB.
- Create methods membersHasChildren, membersChildren for ZyxTeacherViewPort, ZyxStudentView-Port.
- Create ZyxMember>>asListEntry.
- Test.

41.1. Add Container Icon tree to ZyxEditClubView2

41.1. Add a **Container Icon tree** to **ZyxEditClubView2**.

Figure 41.1. Container Icon Tree in the parts palette

41.2. Change the **Container Icon tree** property **partName** to **eCBConn_members_eCPConn_selectedMember_**.

41.3. Change the **Container Icon tree** property **showIcons** to **false**.

41.4. Add a **Label** part above the **Container Icon Tree**.

41.5. Change the **Label** property **object** to **Member relationships**.

41.6. Add a **Label** part above the **List**.

41.7. Change the **Label** property **object** to **Members**.

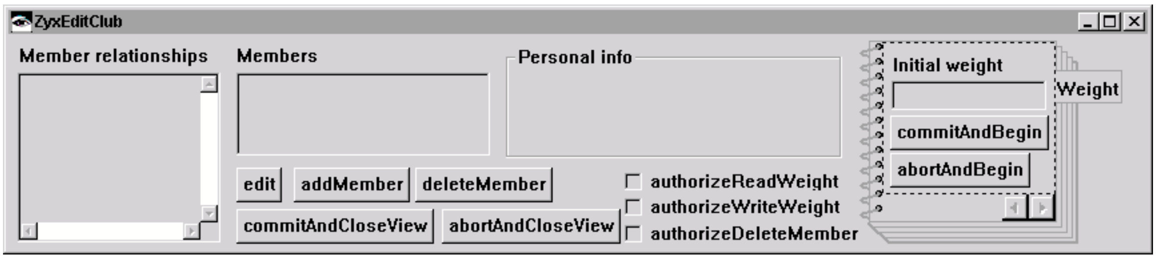


Figure 41.2. ZyxEditClubView2 with Container Icon Tree

41.8. **Save the part**.

41.2. Create MicFwViewPort subclasses ZyxTeacherViewPort, ZyxStudentViewPort

41.9. In **Visual Age Organizer**: Create a new part for **ZyxApplication** with:

- **Part class**: **ZyxStudentViewPort**.
- **Part type**: **ViewPort**.
- **Inherits from**: **MicFwViewPort**.

41.10. In **Visual Age Organizer**: Create a new part for **ZyxApplication** with:

- **Part class**: **ZyxTeacherViewPort**.
- **Part type**: **ViewPort**.
- **Inherits from**: **MicFwViewPort**.

41.3. Create portName methods

41.11. Create the following method:

```
ZyxTeacherViewPort class>>portName
^'ZyxTeacher'
```

41.12. Create the following method:

```
ZyxStudentViewPort class>>portName
^'ZyxStudent'
```

41.4. Create membersHasChildren, membersChildren methods

41.13. Create the following method:

```
ZyxTeacherViewPort>>membersHasChildren
  ^self model students notEmpty
```

41.14. Create the following method:

```
ZyxTeacherViewPort>>membersChildren
  ^self model students asArray
```

41.15. Create the following method:

```
ZyxStudentViewPort>>membersHasChildren
  ^self model teacher notNil
```

41.16. Create the following method:

```
ZyxStudentViewPort>>membersChildren
  ^OrderedCollection with: self model teacher
```

41.5. Create ZyxMember>>asListEntry

41.17. Create the following method:

```
ZyxMember>>asListEntry
  ^name
```

41.18. Save the image.

41.6. Test

41.19. Execute the following code in the workspace:

```
FW CH 41. Container Icon Tree."
[
  | club student1 student2 teacher1  |
  ZyxMember initializeValidation.
  ZyxMember initializeAuthorization.
  ZyxTeacher initializeAuthorization.
  ZyxStudent initializeAuthorization.
  ZyxEditClub initializeAuthorization.
  student1 := ZyxStudent new
    name: 'student1';
    weight: 111.
  student2 := ZyxStudent new
    name: 'student2';
    weight: 112.
  teacher1 := ZyxTeacher new
    name: 'teacher1';
    weight: 222.
  student1 teacher: teacher1.
  student2 teacher: teacher1.
  teacher1 students add: student1.
  teacher1 students add: student2.
  club := ZyxClub new.
  club members add: student1.
  club members add: student2.
  club members add: teacher1.
  ZyxEditClub new openOn: club.
]
```

The following dialog is opened:

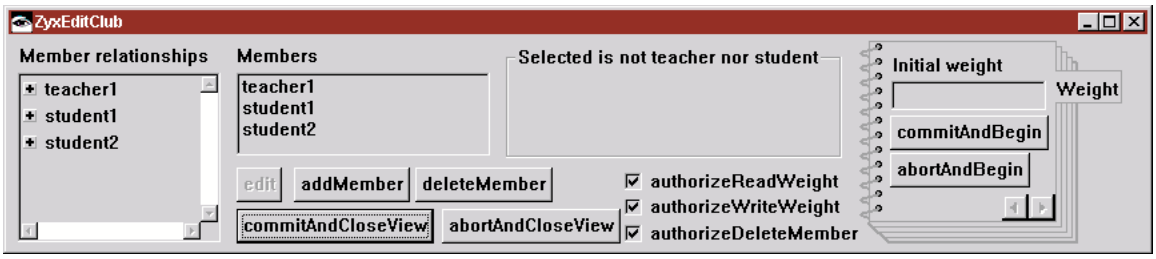
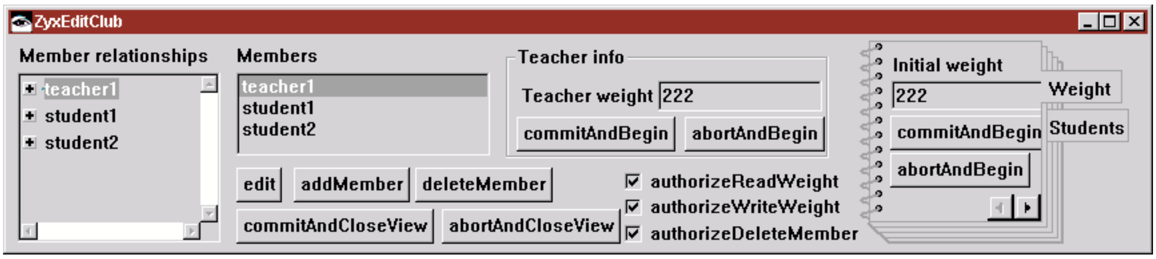


Figure 41.3. ZyxEditClub dialog with Container Icon Tree

41.20. Select **teacher1** in the **Container Icon Tree**. Note that teacher1 is also selected in the List.



Teacher weight 222

Initial weight 222

Figure 41.4. teacher1 selected in the Container Icon Tree

41.21. Expand the tree for **teacher1**.

41.22. Select **student1** in the teacher1 tree. Note that student1 is also selected in the List.

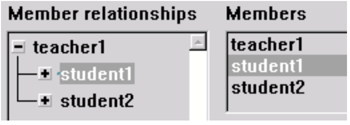


Figure 41.5. teacher1 -> student1 selected in the Container Icon Tree

41.23. Expand the tree for the selected **student1**.

41.24. Select **teacher1** in the student1 tree. Note that teacher1 is also selected in the List.

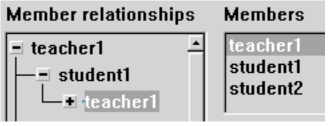


Figure 41.6. teacher1 -> student1 -> teacher1 selected in the Container Icon Tree

41.25. Close the dialog.

42. Drag & Drop

In this chapter you will

- Add a ***Multiple Selection List*** that contains prospective ZyxStudents.
- Create primtive 0..N relationship between ZyxClub>>prospects and ZyxStudent.
- In the ONB: Add variable selection to ZyxClub typed as OrderedCollection.
- Create accessors for ZyxClub>>selectedProspects. selectedProspects are the multiple prospects selected in the Prospects multiple selection list.
- Create MicFwViewPort subclass ZyxClubViewPort.
- Specify ZyxClubViewPort as ViewPort for eCBConn (in DPB).
- Create ZyxClubViewPort>>prospectsProvidedOperations. This method returns the Drag&Drop operations supported by the baseConnection prospects of the multiple selection list.
- Create ZyxClubViewPort>>membersAcceptedOperations. This method returns the Drag&Drop operations supported by the baseConnection members of the container icon tree and the List.
- Create ZyxClubViewPort>>prospectsCanMoveMembers:. This method enables the moving (dragging) of prospects from the multiple selection list only if the currentMember is a ZyxTeacher (prospects cannot be assigned to a ZyxStudent).
- Create ZyxClubViewPort>>prospectsMoveMembers:onto:. This is the method that does the actual dragging of member from the multiple selection list to the other lists.
- Test.

42.1. Add Multiple Selection List to ZyxEditClubView2

42.1. Add a **Multiple Selection List** to **ZyxEditClubView2**.

Figure 42.1. Multiple Selection List in the parts palette

42.2. Change the **Multiple Selection List** property **partName** to **eCBConn_prospects_eCBConn_selectedProspects_name_**.

42.3. Add a **Label** part above the **Multiple Selection List**.

42.4. Change the **Label** property **object** to **Prospects**.

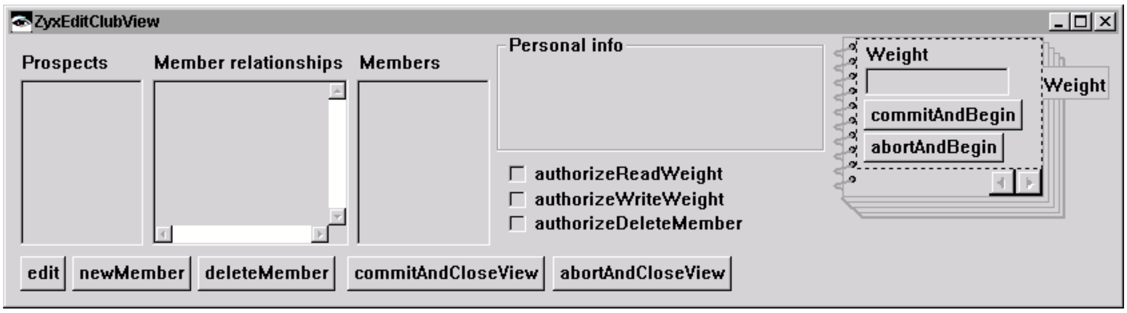


Figure 42.2. ZyxEditClubView2 with Multiple Selection List

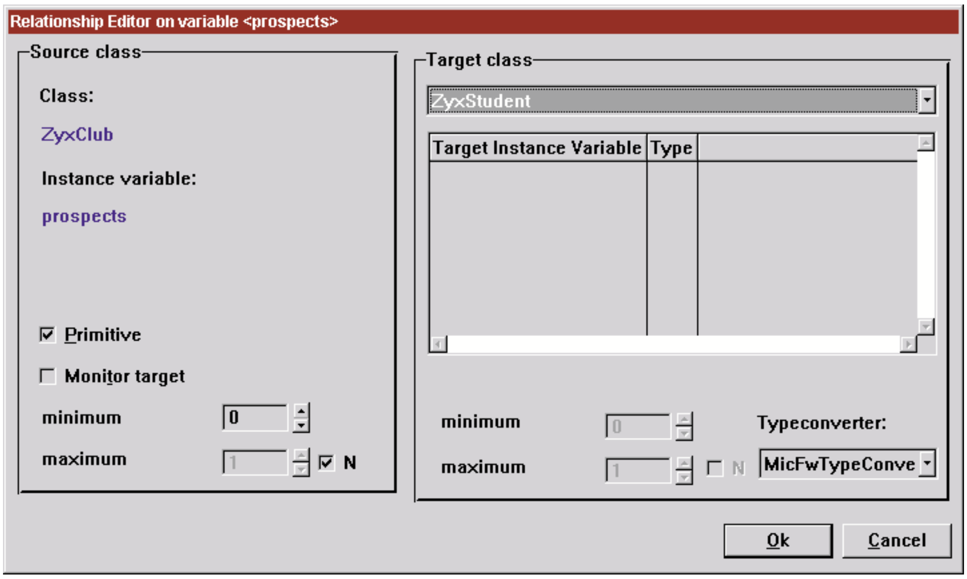
42.5. **Save the part**.

42.6. Close the CE.

42.2. Create relationship (primitive) between ZyxClub>>prospects and ZyxStudent

42.7. In the **ONB**: Add variable **prospects (transacted)** to **ZyxClub**.

42.8. Type as **primitive ->N relationship** with **cardinality 0..N** with target **ZyxStudent**.



Relationship Editor on variable <prospects>

Source class

Class: ZyxClub

Instance variable: prospects

☑ Primitive

□ Monitor target

minimum 0

maximum 1 ☑ N

Target class

ZyxStudent

Target Instance Variable	Type

minimum 0

maximum 1 □ N

Typeconverter: MicFwTypeConve

Figure 42.3. ZyxClub>>prospects primitive ->N relationship to ZyxStudent in RE

42.3. Create ZyxClub>>selection, selection: (in ONB)

42.9. In the **ONB**: Add variable **selection** (**transacted**)to **ZyxClub**.

42.10. Type as **OrderedCollection**.

42.11. **Save the changes**.

42.12. Close the ONB.

42.4. Create ZyxClub>>selectedProspects, selectedProspects:

42.13. Create the following method:

```
ZyxClub>>selectedProspects: selectedObjects
  ^self selection: (selectedObjects isCollection
    ifTrue:   [selectedObjects]
    ifFalse: [Array with: selectedObjects])
```

42.14. Create the following method:

```
ZyxClub>>selectedProspects
  ^self selection
```

42.15. **Save the class**.

42.5. Create MicFwViewPort subclass ZyxClubViewPort

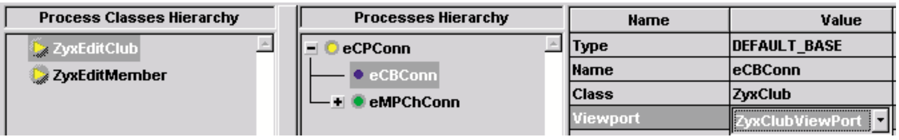
42.16. In **Visual Age Organizer**: Create a new part (open Now) for **ZyxApplication** with:

- **Part class**: **ZyxClubViewPort**.
- **Part type**: **Viewport**.
- **Inherits from**: **MicFwViewPort**.

42.6. Specify ZyxClubViewPort as ViewPort for eCBConn (in DPB)

42.17. Open the DPB on ZyxEditClub.

42.18. Specify ZyxClubViewPort as the ViewPort for eCBConn.



Process Classes Hierarchy

- ZyxEditClub
- ZyxEditMember

Processes Hierarchy

- eCPConn
  - eCBConn
  - eMPChConn

Name	Value
Type	DEFAULT_BASE
Name	eCBConn
Class	ZyxClub
Viewport	ZyxClubViewPort

Figure 42.4. ZyxClubViewPort specified as viewport for eCBConn in DPB

42.19. Save the changes.

42.20. Close the DPB.

42.7. Create ZyxClubViewPort>>prospectsProvidedOperations

42.21. Create the following method:

```
ZyxClubViewPort>>prospectsProvidedOperations
^MicFwOperations enableMove
```

42.8. Create ZyxClubViewPort>>membersAcceptedOperations

42.22. Create the following method:

```
ZyxClubViewPort>>membersAcceptedOperations
^MicFwOperations enableMove
```

42.9. Create ZyxClubViewPort>>prospectsCanMoveMembers:

42.23. Create the following method:

```
ZyxClubViewPort>>prospectsCanMoveMembers: aLOVP
  ^self model currentMember isKindOf: ZyxTeacher
```

42.10. Create ZyxClubViewPort>>prospectsMoveMembers:onto:

42.24. Create the following method:

```
ZyxClubViewPort>>prospectsMoveMembers: aList onto: aVPOrNil
  | mListSelectedItems |
  mListSelectedItems := aList first model selectedProspects.
  self model members addAll: mListSelectedItems.
  self model prospects removeAll:  mListSelectedItems.
  self model currentMember students addAll: mListSelectedItems.
  ^true
```

42.25. **Save the class**.

42.26. Save the image.

42.11. Test

42.27. Execute the following code in the workspace:

```
FW CH 42. Drag&Drop."
[
  | club prospect1 prospect2 student1 teacher1  |
  ZyxMember initializeValidation.
  ZyxMember initializeAuthorization.
  ZyxTeacher initializeAuthorization.
  ZyxStudent initializeAuthorization.
  ZyxEditClub initializeAuthorization.
  prospect1 := ZyxStudent new
    name: 'prospect1';
    weight: 111.
  prospect2 := ZyxStudent new
    name: 'prospect2';
    weight: 222.
  student1 := ZyxStudent new
    name: 'student1';
    weight: 123.
  teacher1 := ZyxTeacher new
    name: 'teacher1';
    weight: 135.
  student1 teacher: teacher1.
  club := ZyxClub new.
  club members add: teacher1.
  club members add: student1.
  club prospects add: prospect1.
  club prospects add: prospect2.
  ZyxEditClub new openOn: club.
]
```

The following dialog is opened:

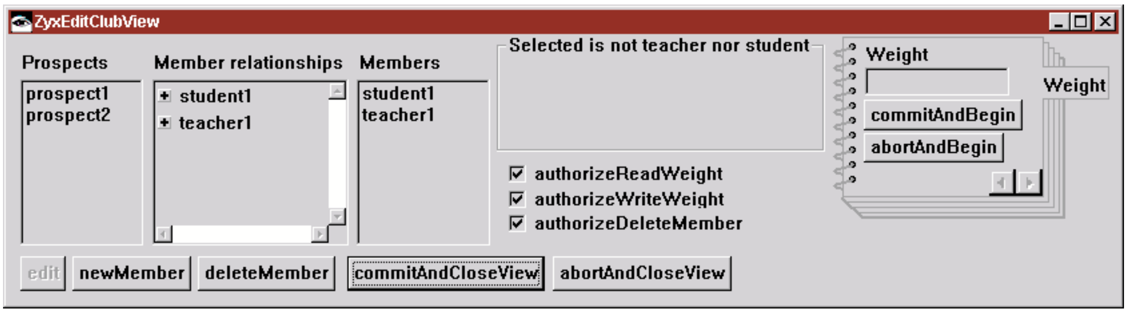


Figure 42.5. ZyxEditClub dialog with Multiple Selection List (for Drag&Drop)

42.28. Select **prospect1** in the **Prospects List**.

42.29. Attempt to **drag prospect1** to **Member Relationships List** (not allowed, since no ZyxTeacher has been selected in the Member Relationships List).

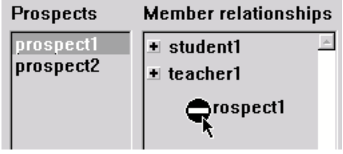


Figure 42.6. prospect1 cannot be dropped into Member relationships because no ZyxTeacher selected

42.30. Select **student1** in the **Member Relationships List**.

42.31. Attempt to **drag prospect1** to **Member Relationships List** (again not allowed, since no Zyx-Teacher has been selected in the Member Relationships List).

42.32. Select **teacher1** in the **Member Relationships List**.

42.33. **Drag prospect1** to **Member Relationships List**.

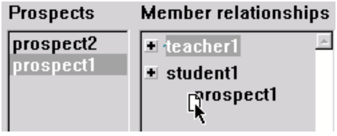


Figure 42.7. Drag&Dropping prospect1 into Member relationships (ZyxTeacher selected)

42.34. Expand tree for **teacher1** in **Member Relationships List**. Note that prospect1 was assigned to teacher1.

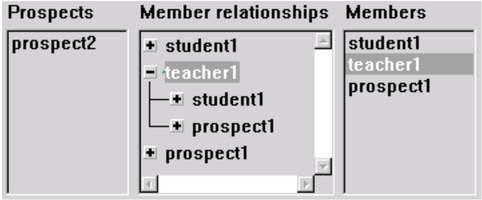


Figure 42.8. prospect1 after being Drag&Dropped to teacher1

42.35. Close **ZyxEditClubView**.

42.36. Execute the code in the workspace again.

42.37. Select **teacher1**.

42.38. Select **prospect1** and **prospect2** (click on both while holding down the **Ctrl** key).

42.39. **Drag prospect1 and prospect2** to **Member Relationships List**.

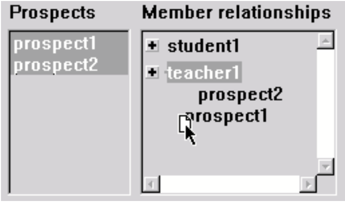


Figure 42.9. Drag&Dropping prospect1 and prospect2 into Member relationships (ZyxTeacher selected)

42.40. Expand tree for **teacher1** in **Member Relationships List**. Note that prospect1 and prospect2 were assigned to teacher1.

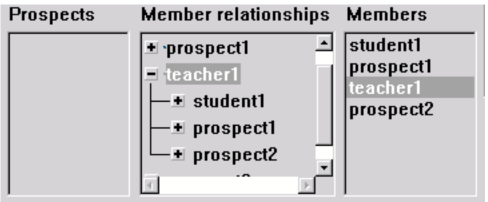


Figure 42.10. prospect1 and prospect2 after being Drag&Dropped to teacher1

42.41. Close the dialog.

Appendix A

Glossary

This glossary defines terms that are presented throughout this manual.

->1 relationship: In ->1 relationships, a source instance variable references 0..1 target objects (nil or 1 object). No target instance variable refers to the source object.

An example of a ->1 relationship is the relationship between Person (source) and PersonName (target). Person instance variable name references 1 PersonName object (a person has only 1 name). No PersonName instance variable references the Person object (a PersonName object never needs to know which Person object is referencing it).

->N relationship: In ->N relationships, a source instance variable references min...max target objects (where min...max is specified by the cardinality).

An example of a ->N relationship would be the relationship between Person (source) and PersonName (target), with the assumption that a person can have more than 1 name. Person instance variable name references min...max PersonName objects, where min = 1 and max is unspecified. No PersonName instance variable references the Person object.

1<->1 relationship: In 1<->1 relationships, a source instance variable references 0..1 (signified by the "1" on the RIGHT of "1<->1") target object. The target instance variable references the same 1 (signified by the "1" on the LEFT of "1<->1") source object or nil. The relationship is not primitive, because the target object should have a reference to the source object.

An example of a 1<->1 relationship would be the relationship between Customer (source) and Portfolio (target). Customer instance variable portfolio references 1 Portfolio object. Portfolio instance variable customer references the 1 Customer object. The relationship is not primitive, because a Portfolio object should know which object is its Customer.

1<->N relationship: In 1<->N relationships, a source instance variable references N (where N is any number with the range max...min specified by the cardinality of the relationship) target objects. There is an instance variable in all of the referenced target objects that reference the 1 source object.

An example of a 1<->N relationship would be the relationship between Employee (source) and Customer (target). Employee instance variable ownedCustomers would reference N (cardinality min <= N <= cardinality max) Customer objects. The instance variable ownerEmployee in each referenced Customer object would reference the 1 Employee object.

Abort a context: A context is aborted when all version objects within the context are dereferenced (ie, none of the changes that were transacted while the context was active will actually be implemented) and the context ceases to exist. See: abortcontext.

Abstract Control:: An Abstract Control simplifies external access to the Framework, as such access can only take place via real Control. There are only a small number of genuinely different Abstract Controls (see command example in the MVC chapter). The actual access attempts are handled with real Control via an appropriate Adapter.

Focus change, issuing commands and transfer of data - including a range of state information like validation or authorization - will be done in this abstract layer. The real Controls are completely decoupled from all Domain Process actions and Domain Model data; all technical details of external interfaces (GUI, DDE, ...) are completely hidden in the Abstract Control implementation.

Abstract Event: Abstract Window Events are the objects that really perform the communication between the view system and the model world, whereas Abstract Windows are merely containers for Abstract Events which additionally may provide some services for them. Abstract events can be divided into two functional types: Abstract events which propagate changes in the model world to the view and Abstract Events which propagate changes or requests from the user interface to the model world.

Abstract Value: An Abstract Value is a container object that is used by Viewports to keep and propagate information about a Viewport Aspect to and from the view. It does not only contain a value for the content of a control, but also different kinds of state information like whether or not the control should be enabled, what the (background) color is, which context help text is displayed, and so on.

Abstract View: When a real View is created, an Abstract View begins to exist in its shadow. The lifetime of the Abstract View is exactly determined by the lifetime of the real View. The Abstract View's purpose is to manage the Abstract Controls corresponding one-to-one to the real Controls on the view. Moreover, the Abstract View performs coordination between abstract and platform layer when opening, activating and closing the view. It communicates with the real Platform View using a Platform Adapter.

Accessor Generator: The accessor generator is the object that creates the OBF accessors for a variable.

Active Context: Only 1 context can be active at anytime. While a context is active, any changes to any transacted variables will be recorded in object versions. This variable will be locked by the context, which means that no changes may be made to the variable while a sibling context or parent context is active as long as this context exists.

Adapter: A Platform Adapter system is introduced which does the translation of protocols and overcomes the architectural differences between the host Smalltalk system architecture and the Application Framework architecture. This makes the core part of the framework itself portable.

Archiver: See: Code Archiver.

Authorization: The access to individual objects controlled on the model level. Authorization works both for accessing attributes of Domain Objects and for executing Aspects of business processes.

Broker: To allow subsystems or specific requests to them to be exchanged with an own implementation, Application Framework uses Broker classes that offer a thin public interface with the internal knowledge how to delegate the call to the subsystem. A common Broker concept enables the developers to modify request algorithms or subsystem behavior quite easy.

Cardinality: The cardinality of a relationship determines the required min and max number of target objects referenced by the source variable. In 2-way relationships, there are 2 cardinalities. The second cardinality determines the requried min and max number of source objects referenced by the target variable.

CB: Connections Browser.

Child context: A child context can change any variable locked by its parent. A variable, having been changed by the child context, is now locked by the child context. The parent context cannot change a variable locked by the child. See: parent Context.

Code Generator: See Accessor Generator.

Commit a context: Committing a context has the same effect as committing all transaction levels (TrLevels) in the context.

Committed target: In a VersionObject: A getter message to a source object will return the committed target object referenced by the variable if no context is active OR if [the active context read mode is isolate AND the source object’s variable is not locked by the active context AND the active context is not a child context of the context with the variable lock]. See: transacted Target.

Concurrent contexts: 2 contexts are concurrent if there is no parent-child relationship between them. Such contexts can also be referred to as "sibling" contexts. A sibling context may not change a variable locked by another sibling context.

Concurrent transactions: See Concurrent Contexts.

Connector: A Connector is used to connect objects (Domain Objects and Domain Processes) to an external view. The Connector decouples view and model objects and provides access to connected Domain Objects and Domain Processes on a low level. In this respect, it forms a bridge between the objects, decoupling also Domain Objects from Domain Processes and thus making them more combinable and exchangeable.

Context: See Transaction Context.

DDL: A language enabling the structure and instances of a database to be defined in a human- and machine-readable form.

Default Base Connection: The Default Base Connection will be used to hold the Domain Object if no explicit Base Connection is defined for this process.

Delegation Model: An concept used to decouple subsystems from each other. Application programmers implement subclasses of Viewport to isolate the model from the interaction subsystem.

Domain Model: A Domain Model is a design phrase for real world concepts like Customer, Policy or Address. Its instances are called Domain Objects.

Domain Object: Domain objects describe that part of the MVC architecture which corresponds to business terms. The behavior and structure are primarily defined in this respect. The appropriate tool (object net browser) from the Object Behavior Framework is used for this purpose, and mapping to the database is described with STOPF from Persistence Framework . The Domain Object also has open interfaces for connecting authorization and validation.

Domain Process: A Domain Process is a design phrase for processing and workflow-oriented tasks and control flow. Its instances are also called Domain Processes. Due to their controlling-oriented nature, they take also responsibility for managing a Transaction Context if appropriate.

Domain Processes Browser: A tool, supplied with the Application Framework to browse the exiting Domain Processes

DPB: Domain Processes Browser.

Extended Description (MicFwExtendedDescription): The object that completely describes the typing of all variables in a class. The class method createExtendedDescription creates the MicFwExtendedDescription object.

Framework Logger: The tracing facility of the Framework's. Events and messages will be routed through this logger.

Framework: A Framework is a software architecture for certain tasks, which components can be easily reused by application developers. It provides the system with a basic structure in being a collection of cooperating and conceptual concise classes and methods, which are designed to support a task-oriented work progress in application development.

Inactive Context: A non-active context. See: Active Context.

InstanceVariableDescription (MicFwInstanceVariableDescription): The object that completely describes the typing of a single instance variable in a class. The class method createExtendedDescription creates the MicFwInstanceVariableDescription object (which is referenced within the MicFwExtendedDescription object).

Isolated Context: A variable getter sent to a source object that is locked by another context while an isolated context is active will return not the committed target of the transaction object in the other context, but rather the uncommitted target. See: uncommitedread context.

M<->N relationship: In M<->N relationships, a source instance variable references M (where M is any number with the range max...min specified by the cardinality LEFT of the relationship) target objects. There is an instance variable in all of the referenced target objects that reference the N (where N is any number with the range max...min specified by the cardinality RIGHT of the relationship) objects of the source object class, including the current source object.

An example of an M<->N relationship would be the relationship between Person (source) and Address (target). Person instance variable addresses would reference M (cardinality min <= M <= cardinality max) Address objects. Address instance variable persons would reference N (cardinality min <= N <= cardinality max) Person objects, including the current Person object.

Mapper: To maintain registration of loosely coupled elements within the Framework, Mappers are used to set, get and remove associations between unique, constant names and their corresponding classes, which themselves may change or may be reimplemented in your project. Mappers are implemented as singletons and can be reached through an easy to use interface as they extend Object with one method per Mapper.

MicFwTransactionContext:

Model View Connector: Is the object that holds the child-process- and Base-Connections for one process.

Model View Controller: The MVC (concept of a Model View Controller) defines an architecture intended to yield a strict decoupling of Domain Model Aspects, flow control and external views - mostly displayed graphically for user interaction.

Nested transaction levels: The transaction levels in a context are nested if more than 1 level exists.

OBF (Object Behavior Framework): OBF is a library of classes that when added to your Smalltalk development environment provides a Framework for defining the requirements and restrictions for the behavior of objects.

Object Net Browser: A visual OBF tool that dispayes both the object net of a class and information about the instance variables of the class (key, validated, transacted, persistent, type, relationship). The visual OBF tools Type Editor and Relationship Editor are opened from the Object Net Browser.

Object Net: An Object Net consists of objects and its relationships between them.

ObjectVersion (version object): An ObjectVersion is created if a transaction context is active and a target object is assigned to a transacted variable of a source object. The ObjectVersion is assigned to the highest TrLevel in the active context. If an ObjectVersion already exists in this TrLevel, it is replaced. The ObjectVersion references the committed target object (committed) and the uncommitted target object (uncommitted) referenced by the variable.

ONB: Object Net Browser.

Packaging: The process of creating a runtime executable.

Parent context: A parent context is a context object that created its child context when it responded to the newTransactionContext message. The parent context can have any number of child contexts. A parent context can also be a child context. If parent context B has a parent context A and B also has a child context C, then A and C also have a parent child relationship. See: Child Context.

POM: Persistent Object Manager.

Primitive relationship: A 1-way relationship. See: 1-way relationship.

Relationship Editor: A visual OBF tool for establishing relationships between 2 classes. See: object Net Browser.

Relationship: In Smalltalk, relationships between objects are simply represented as object pointers. No distinction is made between complex and simple (scalar) data types since all are full-scale objects and there is conceptually no difference between a relationship and an "embedded" value.

Beyond this, the Mynd Object Behavior Framework provides an elaborated relationship concept maintaining referential integrity between objects. Those relationships may even be mapped to (relational) databases with the help of the Mynd Persistence Framework

RelationshipDescription (MicFw~): Object which contains complete information about a relationship.

Running context: A context that has a TrLevel1 (it may have higher levels also).

Sibling contexts: See Concurrent Contexts.

STOPF / ODBC: Smalltalk Object Persistence Framework

Transacted variable: A variable that has been marked as transacted in the Object Net Browser. NOTE:Changes to a variable will not be transacted even if a transaction context is active if the variable is not marked as transacted.

Transaction Browser: A Visual OBF tool for displaying and manipulating (aborting / committing) transactions.

Transaction Context: A logical unit that can contain transaction levels and that can be related to other contexts as a parent, sibling, or child.

Transaction Level: A subunit of a transaction context. A transaction level has a single object version for each variable of each object that was assigned a new target object while the the transaction level was the highest level in its context and its context was active.

Transaction Manager: The transaction manager is a single instance (singleton) of MicFwTransactionManager and manages all running contexts.

Transaction: A defined state of an object that runs under transactional control will be stored. If the interaction on this object fails by some reason, this object can be restored to this state, if the interaction succeeds, the persistent state of this object will be modified to this new state and the old state will be dropped.

TrLevel1, 2, ...: Transaction level 1, 2, ...

Type Converter: A type converter is assigned to a typed variable (including variables in a relationship). When an object is assigned to the variable that is not of the type specified for the variable, the type converter will be used to attempt to create the correct type of object containing the information in the original object and assign this correct type of object to the variable.

Type Editor: A visual tool for setting a variable type as a simple type (Integer, Date, etc.). The type converter and size / scale for the variable type can also be selected. The Type Editor is opened from the Object Net Browser.

TypeDescription (MicFw~): Object which contains complete information about the typing of an instance variable.

Typing: The term typing defines the assignment of types (normally basic classes like Integer or String) to instance variables. Since Smalltalk is an untyped language, the Mynd Object Behavior and Persistence Frameworks introduce typing in order to support storing of objects into (typed) relational databases.

UML (Unified Markup Language): Graphical notation for OO analysis and design.

uncommitedRead context: If an ObjectVersion exists for aSourceObject>>aSOVariable (ie, aSOVariable is transacted and anUncommittedTargetObject was assigned to aSOVariable while aContext1 was active): If aContext2 is an active uncommitedRead context and getter message sOVariable is sent to aSourceObject, the object returned will be anUncommittedTargetObject. See: Isolated context.

Note: The message is spelled "uncommitedRead".

Uncommitted target: In a VersionObject: A getter message to a source object will return the uncommitted target object referenced by the variable if [the active context read mode is uncommitedReadisolate AND the active context does not have a lock on the source object variable AND the active context is not a child context of the context that has the lock] OR if the source object’s variable is locked by the active context. See: Committed Target.

Validation of ExtendedDescription: Validation of an ExtendedDescription involves checking the Object Net for errors.

Validation: Like the authorization service, validation ties in at the model layer and is used for checking value-based access to and from attributes based on certain rules.

VariableDescription (MicFw~): Object which contains complete information about an instance variable.

Viewport: In order to separate view-related state handling from Domain Objects and Domain Processes, Application Framework implements a Viewport as a Delegation Model concept. Functionality for transportation and the filtering of data and states concerning an object net from a certain object's or view's direction is delegated from the Domain Models to them, leading to a more lightweight and view independent kind of Domain Object.

Viewports "learn" the dependencies between their Aspects and the corresponding Model Aspects while running the application. This read trace frees the programmer from dealing with changed-events and allows Application Framework to update both sides automatically and in a generic way.

A Dispatcher is called Viewport if Domain Objects are concerned.

List Of Figures

Index