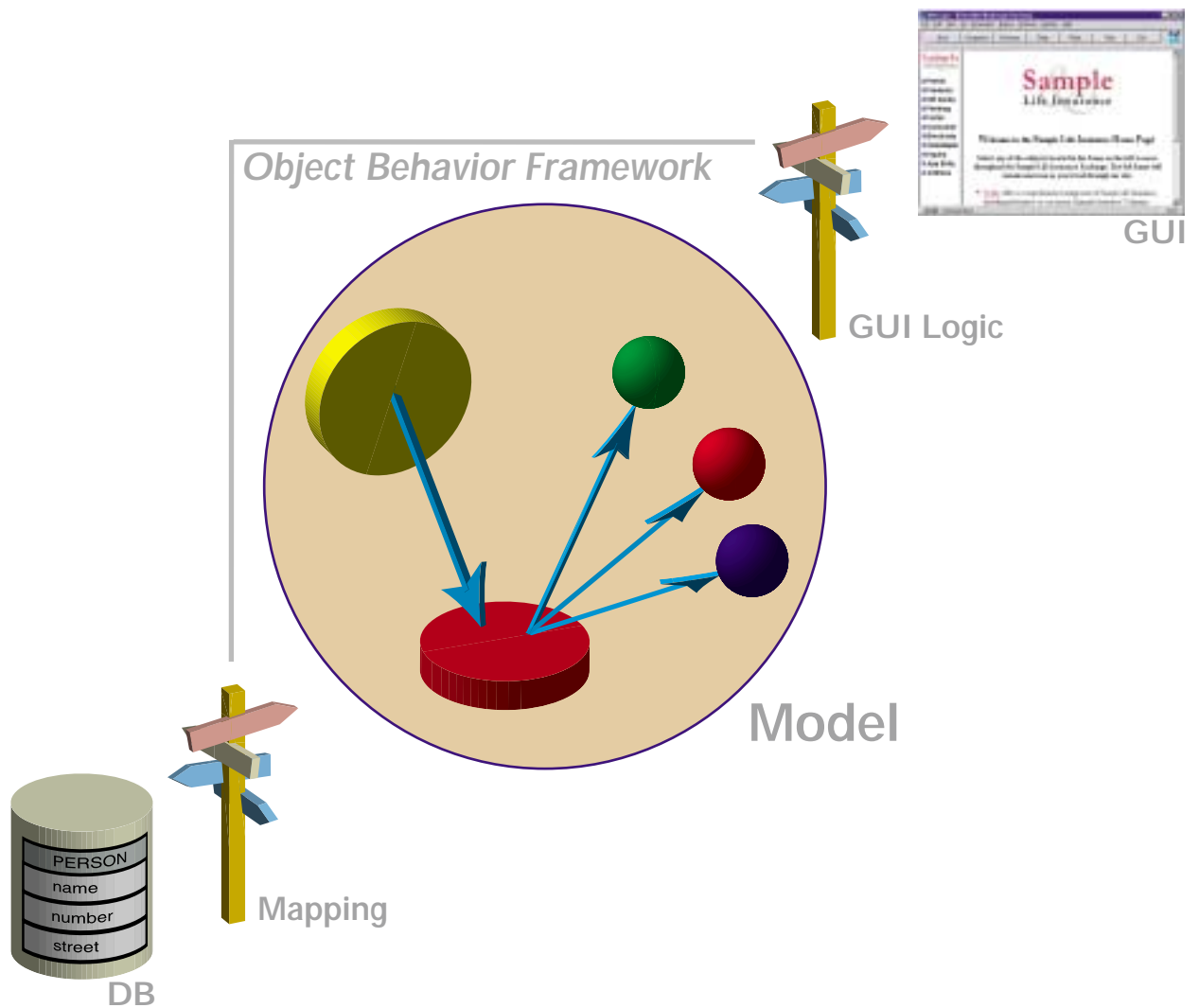


## Object Behavior Framework

# User's Guide

For Frameworks Version 5.0





---

# Copyright and trademarks

## Copyright

Copyright 2000 Mynd. All rights reserved.

Mynd Frameworks V5.0.

Object Behavior Framework User's Guide, 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

*ENVY* is a registered trademark of the Object Technology International corporation.

*Visual Age for Smalltalk* and *OS/2* are registered trademarks of the International Business Machines Corp.

*Windows* and *Windows/NT* are registered trademarks of the Microsoft Corp.



---

# Table of Contents

|   |           |
|---|-----------|
| <b>Copyright and trademarks .....</b>   | <b>3</b>  |
| <b>Table of Contents .....</b>  | <b>5</b>  |
| <b>Documentation Overview .....</b>   | <b>9</b>  |
| Congratulations.....  | 9         |
| Object Behavior Framework User's Guide (this manual).....                     | 9         |
| Other manuals .....   | 10        |
| Training from PMS Micado .....  | 10        |
| <b>Release Notes .....</b>  | <b>11</b> |
| Version 5.0.....  | 11        |
| Version 4.0.....  | 11        |
| Version 3.5.....  | 11        |
| Release 3.4 Version 1.0.....  | 12        |
| Release 3.3 Version 1.3.....  | 13        |
| <b>Installing OBF .....</b>   | <b>15</b> |
| System requirements.....  | 15        |
| Installation Overview.....  | 15        |
| Importing OBF configuration maps .....  | 15        |
| Loading OBF configuration maps .....  | 15        |
| OBF tools available from micFrameworks menu .....                             | 16        |
| <br>  |           |
| <b>1</b>  |           |
| <b>Concepts .....</b>   | <b>17</b> |
| 1.1. What is OBF? .....   | 19        |
| 1.1.1. Object behavior.....   | 19        |
| 1.1.2. Framework.....   | 20        |
| 1.2. Variable types / relationships .....                                     | 21        |
| 1.2.1. What are variable types / relationships.....                           | 21        |
| 1.2.2. Why variable types / relationships are not supported by Smalltalk..... | 21        |
| 1.2.3. Why variable types / relationships are supported by OBF .....          | 22        |
| 1.2.4. How variable types / relationships are implemented by OBF .....        | 22        |
| 1.3. Transactions .....   | 27        |
| 1.3.1. What are transactions .....  | 27        |
| 1.3.2. Why are transactions not supported in Smalltalk.....                   | 27        |
| 1.3.3. Why are transactions supported by OBF .....                            | 27        |
| 1.3.4. How transactions are implemented by OBF .....                          | 28        |
| <br>  |           |
| <b>2</b>  |           |
| <b>Tutorial .....</b>   | <b>47</b> |
| 2.1. Tutorial overview .....  | 49        |
| 2.2. Using the example code (obf_ex.txt) .....                                | 50        |
| 2.2.1. Example code in this User Guide.....                                   | 50        |
| 2.2.2. Executing example code in obf_ex.txt .....                             | 50        |
| 2.3. Create ZyxTutorial application .....                                     | 51        |
| 2.3.1. Create application ZyxTutorial .....                                   | 51        |
| 2.3.2. Specify Prerequisites .....  | 51        |
| 2.4. Create new model / class (OMB) .....                                     | 52        |
| 2.4.1. Create new model.....  | 52        |
| 2.4.2. Create a class within the model (not in VA).....                       | 53        |
| 2.4.3. Save the model.....  | 53        |

|  |    |
|--|----|
| 2.4.4. Verify that class ZyxClass1 does not exist in VA .....                                      | 53 |
| 2.5. Reopen / modify model .....   | 54 |
| 2.5.1. Reopen (overwrite / reopen) the model.....  | 54 |
| 2.5.2. Modify model / Discard all changes .....  | 54 |
| 2.6. Export model .....  | 55 |
| 2.6.1. Export to .st files (.st file for each class).....  | 55 |
| 2.6.2. Export to an .xml file (single .xml file for all classes).....                              | 55 |
| 2.7. Save model to VisualAge .....   | 56 |
| 2.7.1. Saving to VA from within OMB.....   | 56 |
| 2.7.2. Using "File in" to save contents of .st file to VA .....                                    | 56 |
| 2.8. Classes .....   | 57 |
| 2.8.1. Default application for new classes.....  | 57 |
| 2.8.2. Create ZyxClass1 subclass ZyxClass11 .....  | 57 |
| 2.8.3. Create MicFwDomainObject subclass ZyxClass2 .....   | 57 |
| 2.8.4. Remove classes from model.....  | 58 |
| 2.8.5. Add MicFwDomainObject subclass ZyxClass1 from VA to model .....                             | 58 |
| 2.8.6. Add ZyxClass1 subclass ZyxClass11 from VA to model .....                                    | 58 |
| 2.8.7. Finding a class in the model.....   | 58 |
| 2.9. Variables / Framework accessors .....   | 59 |
| 2.9.1. Add variable var1, var2 to ZyxClass1 .....  | 59 |
| 2.9.2. Framework accessors generated when model variables saved to VA.....                         | 59 |
| 2.9.3. Modify accessors for ZyxClass1>>var1 .....  | 60 |
| 2.9.4. Modify accessor prefix defaults for all classes .....                                       | 60 |
| 2.9.5. Remove var2.....  | 61 |
| 2.9.6. Restore var2; Make a virtual variable.....  | 61 |
| 2.9.7. Make var2 a non-virtual variable; Redefine var2 in subclass ZyxClass11 .....                | 61 |
| 2.9.8. Remove redefine of var2 in subclass ZyxClass11 .....  | 61 |
| 2.10. Variable typing (Integer) (with manual type validation, manual/automatic type conversion)    | 63 |
| 2.10.1. Delete classes / Create ZyxPerson, ZyxPerson>>id.....                                      | 63 |
| 2.10.2. Type ZyxPerson>>id as Integer .....  | 63 |
| 2.10.3. Framework accessors for typed variable .....   | 63 |
| 2.10.4. Test with no validation.....   | 63 |
| 2.10.5. Test manual validation (message isTypeValid).....  | 64 |
| 2.10.6. Test manual conversion (message convertValues) using MicFwTypeConverter .....              | 64 |
| 2.10.7. Test manual conversion (message convertValues) using MicFwTypeConverter .....              | 65 |
| 2.10.8. Specify automatic conversion (checkbox validateType) .....                                 | 66 |
| 2.10.9. Framework accessors for automatic conversion .....   | 66 |
| 2.10.10. Test automatic conversion using MicFwTypeConverter .....                                  | 66 |
| 2.10.11. Test automatic conversion using MicFwEmptyTypeConverter .....                             | 66 |
| 2.11. Variable typing (Date) .....   | 67 |
| 2.11.1. Create ZyxPerson>>dateOfBirth with type Date .....   | 67 |
| 2.11.2. Test .....   | 67 |
| 2.12. Persistent / Key variables .....   | 69 |
| 2.12.1. Specify ZyxPerson>>id as persistent.....   | 69 |
| 2.12.2. Framework accessors for Persistent variable .....  | 69 |
| 2.12.3. Specify ZyxPerson>>id as a key variable .....  | 69 |
| 2.12.4. Specify ZyxPerson>>dateOfBirth as a persistent variable.....                               | 69 |
| 2.12.5. Application: Key variable for persistence object.....                                      | 69 |
| 2.12.6. Application: Date variable for persistence object .....                                    | 70 |
| 2.13. ->1 relationship .....   | 71 |
| 2.13.1. Create MicFwDomainObject subclass ZyxName with variables firstName, lastName .....         | 71 |
| 2.13.2. Add variable ZyxPerson>>name.....  | 71 |
| 2.13.3. Establish a ->1 (primitive) relationship from ZyxPerson>>name to 1..1 ZyxName objects..... | 71 |
| 2.13.4. Test .....   | 72 |
| 2.13.5. Application: ->1 relationship .....  | 73 |
| 2.14. Adding class nets to the model .....   | 75 |

|   |            |
|---|------------|
| 2.14.1. Create ZyxName subclass ZyxNameSubclass; Save to VA .....   | 75         |
| 2.14.2. Add ZyxNameSubclass class net to model.....   | 75         |
| 2.14.3. Add ZyxName class net to model .....  | 75         |
| 2.14.4. Add ZyxPerson class net to model .....  | 75         |
| <b>2.15. -&gt;N relationship .....</b>  | <b>77</b>  |
| 2.15.1. Create MicFwDomainObject subclass ZyxAddress with variable cityName .....                         | 77         |
| 2.15.2. Add variable ZyxPerson>>addresses .....   | 77         |
| 2.15.3. Establish a ->N (primitive) relationship from ZyxPerson>>addresses to 1..3 ZyxAddress objects.... | 77         |
| 2.15.4. Test.....   | 78         |
| 2.15.5. Review how the OBF classes and methods you created can be used by PFW. ....                       | 79         |
| <b>2.16. 1&lt;-&gt;1 relationship .....</b>   | <b>81</b>  |
| 2.16.1. Create classes and variables.....   | 81         |
| 2.16.2. Set ZyxCustomer>>portfolio type as 1<->1 relationship.....  | 81         |
| 2.16.3. Review how the OBF classes and methods you created can be used by PFW. ....                       | 82         |
| <b>2.17. 1&lt;-&gt;N relationship .....</b>   | <b>83</b>  |
| 2.17.1. Create classes and variables.....   | 83         |
| 2.17.2. Set ZyxEmployee>>ownedCustomers type as 1<->N relationship.....                                   | 83         |
| 2.17.3. Review how the OBF classes and methods you created can be used by PFW. ....                       | 84         |
| <b>2.18. M&lt;-&gt;N relationship .....</b>   | <b>85</b>  |
| 2.18.1. Create classes and variables.....   | 85         |
| 2.18.2. Set ZyxPerson>>addresses type as M<->N relationship.....  | 85         |
| 2.18.3. Review how the OBF classes and methods you created can be used by PFW. ....                       | 86         |
| <b>2.19. 1 context: 1 (non-nested) TrLevel .....</b>  | <b>88</b>  |
| 2.19.1. Cleaning up.....  | 88         |
| 2.19.2. Create non-running, non-active context.....   | 88         |
| 2.19.3. Specify ZyxName>>firstName as transacted.....   | 89         |
| 2.19.4. Assign an object to a variable (with no active context) .....                                     | 89         |
| 2.19.5. Assign an object to a variable (with active context) .....  | 90         |
| 2.19.6. TrLevel1: Committing.....   | 91         |
| 2.19.7. TrLevel1: Aborting.....   | 92         |
| 2.19.8. Committing a single-level context .....   | 93         |
| 2.19.9. Aborting a single-level context.....  | 93         |
| <b>2.20. 1 context: Multiple (nested) TrLevels .....</b>  | <b>94</b>  |
| 2.20.1. Create context, TrLevel1, TrLevel2.....   | 94         |
| 2.20.2. Abort highest level (TrLevel2).....   | 95         |
| 2.20.3. 2-level transaction: Abort all TrLevels.....  | 95         |
| 2.20.4. Commit highest level (TrLevel2) .....   | 96         |
| 2.20.5. 2-level transaction: Commit all TrLevels.....   | 97         |
| 2.20.6. Aborting a multi-level context.....   | 98         |
| 2.20.7. Committing a multi-level context.....   | 98         |
| <b>2.21. Multiple contexts: Concurrent .....</b>  | <b>99</b>  |
| 2.21.1. Create 2 (concurrent) contexts; mode uncommittedRead (default) .....                              | 99         |
| 2.21.2. 2 concurrent contexts; mode isolate .....   | 100        |
| 2.21.3. Attempt to change variable in context that is locked by other context .....                       | 101        |
| <b>2.22. Multiple contexts: Parent / Child .....</b>  | <b>102</b> |
| 2.22.1. Context and 2 child contexts (child and grandchild) .....   | 102        |
| 2.22.2. Test in uncommittedRead and isolate modes for a parent / child.....                               | 104        |
| 2.22.3. Analyze how a variable lock can be transferred .....  | 105        |

### 3

|                                       |            |
|---------------------------------------|------------|
| <b>Tools .....</b>                    | <b>107</b> |
| 3.1. Introduction .....               | 109        |
| 3.2. Object Model Browser (OMB) ..... | 110        |
| 3.2.1. Primary functions .....        | 110        |
| 3.2.2. Opening OMB .....              | 110        |
| 3.2.3. OMB dialog .....               | 111        |

|  |     |
|--|-----|
| 3.3. Object Net Browser (NetBrowser) .....               | 119 |
| 3.3.1. Primary functions .....                           | 119 |
| 3.3.2. Opening the Net Browser:.....                     | 119 |
| 3.3.3. Classes window .....                              | 119 |
| 3.3.4. Net Browser submenu Class.....                    | 119 |
| 3.3.5. Net Browser submenu Variable .....                | 121 |
| 3.3.6. Variable parameters .....                         | 123 |
| 3.4. Type Editor .....                                   | 124 |
| 3.4.1. Primary functions .....                           | 124 |
| 3.4.2. Opening the Type Editor .....                     | 124 |
| 3.4.3. Fields / checkboxes in the type editor .....      | 124 |
| 3.5. Relationship Editor .....                           | 125 |
| 3.5.1. Primary functions .....                           | 125 |
| 3.5.2. Opening the Relationship Editor .....             | 125 |
| 3.5.3. Fields in the Relationship Editor.....            | 125 |
| 3.6. Transaction Browser .....                           | 127 |
| 3.6.1. Primary functions .....                           | 127 |
| 3.6.2. Opening the Transaction Browser.....              | 127 |
| 3.6.3. Submenu Transactions .....                        | 128 |
| 3.6.4. Submenu Objects.....                              | 129 |
| 3.6.5. Submenu ObjectInstanceVariables.....              | 129 |
| 3.6.6. Context / TransactionLevel fields / buttons ..... | 129 |
| 3.6.7. Versioned objects fields / buttons .....          | 130 |

## Appendix A

|           |     |
|-----------|-----|
| API ..... | 131 |
|-----------|-----|

## Appendix B

|                              |            |
|------------------------------|------------|
| Glossary .....               | 141        |
| <i>List Of Tables</i> .....  | <b>149</b> |
| <i>List Of Figures</i> ..... | <b>151</b> |
| <i>Index</i> .....           | <b>155</b> |



---

# Documentation Overview

---

## Congratulations

Congratulations on your purchase of PMS Micado Object Behavior Framework (OBF), just one of the powerful performance-enhancing tools from PMS Micado. 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. This User's Guide is designed to help you exploit the full potential of one of these tools in the shortest time possible.

---

## Object Behavior Framework User's Guide (this manual)

### Intended audience

The intended audience for this manual includes Smalltalk application developers, database developers, and project management. The reader should have basic familiarity with the Smalltalk language and programming techniques. Basic knowledge of UseCase and Unified Modelling Language is highly recommended.

### Sections

The content of this manual is presented in the following sections:

- **'Copyright and trademarks' (page 3).**
- **'Table of Contents' (page 5).**
- **'Documentation Overview' (page 9).** This section.
- **'Release Notes' (page 11).** Provides important information regarding this release of OBF.
- **'1. Concepts' (page 17).** Describes OBF concepts. This section is recommended as a good review for experienced OBF users. Those with limited OBF experience might first do the tutorial in 'Tutorial' (page 47), since the tutorial introduces the major concepts with step-by-step programming.
- **'2. Tutorial' (page 47).** Provides a set of programming examples that demonstrate the major programming concepts and techniques of OBF. The example Smalltalk code in the tutorial is also provided in a separate file (obf\_ex.txt) that can be run from a workspace dialog.
- **'3. Tools' (page 107).** Describes the OBF development environment tools. A review of the tools used in the tutorial.
- **'Appendix A. API' (page 131).** Describes API methods for OBF.
- **'Appendix B. Glossary' (page 141).** Includes any special terms used throughout this manual.
- **'List Of Tables' (page 149).**
- **'List Of Figures' (page 151).**
- **'Index' (page 155).**

### Recommended sections

#### For readers with no previous OBF experience

Get first-hand experience with OBF by starting with 'Tutorial' (page 47). Refer to 'Concepts' (page 17), 'Tools' (page 107), and 'Glossary' (page 141) as required.

#### For experienced OBF users

Discover what is new in this version by reading 'Release Notes' (page 11).

It is also recommended to read about the new Object Model Browser (OME), which replaces the Object Net Explorer in this version of the Frameworks (see 'Object Net Browser (NetBrowser)' (page 119) for more details).

### 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.micado@notes.compuserve.com](mailto:info.micado@notes.compuserve.com)

---

## Other manuals

From Mynd (and referenced throughout this manual):

- Frameworks Getting Started Manual (from Mynd).
- Application Framework User's Guide (from Mynd).
- Persistence Framework User's Guide (from Mynd).

From IBM:

- IBM Visual Age for Smalltalk manuals.
- 

## Training from PMS Micado

This manual is an excellent source of information about OBF. However, if you need to start using OBF immediately, it is highly recommended to enroll in any one of the many specialized PMS Micado training courses. PMS Micado provides a wide-range of courses covering Smalltalk programming, object-oriented analysis, design, methodology, and project management.

For more information about training courses, please contact:

b 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](mailto:info@mynd.de)

Web: [www.mynd.de](http://www.mynd.de)

---

# Release Notes

---

## Version 5.0

### 1. VisualAge 5.01 compatibility

Frameworks (including Object Behavior Framework) now run under VisualAge 5.01.

---

## Version 4.0

### 1. Object Model Explorer

Version 4.0 includes the Object Model Browser (OMB). With the OMB, classes and variables can be created, variables typed, and relationships specified and saved in a model (outside of VA). The model, when ready, can then be saved to VA.

For more information, see:

- 'Tutorial' (page 47).
  - '3.2. Object Model Browser (OMB)' (page 110).
- 

## Version 3.5

### 1. Dynamic variable properties definition in the Object Model Explorer

You can dynamically define properties for variables of your classes in the Object Model Explorer. Properties are class-based and will be inherited by subclasses. The Object Model Explorer is now the default model explorer; however, the Object Net Browser is still available.

### 2. Image size reduction of Runtime Extended Description

The class method `#initExtendedDescription:` contains all information that the Object Behavior Framework holds about the class and its typed attributes. (You can enter and modify this information with the Object Model Explorer or Object Net Browser.)

Some of the information stored in the `#initExtendedDescription:` method is necessary during runtime others is only used by the framework tools during development time.

The `#initExtendedDescription:` method is used to lazily initialize the extended description of a class. The class's extended description is then kept in a class instance variable named `#extendedDescription`. When the `#extendedDescription` is `<nil>`, the `#initExtendedDescription:` method will initialize the extended description the next time it is accessed. (lazy initialization)

It is possible to instantiate all information stored in the `#initExtendedDescription:` method or to instantiate only the information that is relevant for the FW's runtime components. The latter will reduce the size of your Smalltalk image, but the FW tools cannot operate on extended description's with runtime information only.

When you have loaded FW tools (e.g. Object Net Browser or Object Model Explorer) into your image, the `#initExtendedDescription:` method will instantiate complete information by default. When you have loaded FW runtime components (no tools, no development configuration maps), then by default the `#initExtendedDescription:` method will instantiate the runtime information only.

To reduce the size of your packaged runtime image, you have several possibilities:

1. You do not care about the contents of the `#extendedDescription` class variables in the image.
2. You want to minimize the image as good as possible.
3. You want to minimize the image but you also want to have the best performance.

Here is what you can do and what the consequences are:

1. You do not care about the contents of the `#extendedDescription` class variables in the image.

You just package your runtime image somewhere. The VA-Packager will most likely package the contents of the `#extendedDescription` class variables into the runtime image. For those classes, where the variable is already initialized before packaging, the extended description will also be initialized in the packaged image. The extended description of other classes will be initialized lazy the first time it is accessed during runtime of your application.

When you start the VA-Packager from an image, in which FW development components have been loaded, you might package completely instantiated extended descriptions (including development information also) into the runtime application, which increases the size of the runtime more than necessary.

---

2. You want to minimize the image as good as possible.

The runtime image is smallest, when all #extendedDescriptions are <nil> before packaging.

To archive this:

a. Open the Extended-Description-Conversion tool from your Transcript window:

Menu micFrameworks->Object Behavior Tools->Convert Extended Descriptions...

b. Select the radio button "Remove Extended Descriptions" and press "Start".

c. Close the tool afterwards and package your application.

3. You want to minimize the runtime image but you also want to have the best performance.

You can preinitialize all #extendedDescriptions with the runtime extended description information for all classes in your image before packaging. The image size relations are:

```
{all #extendedDescriptions = <nil>}
<
{all #extendedDescriptions = runtime extended descriptions}
<
{ all #extendedDescriptions = complete extended descriptions }
```

### 3. Changed API methods

#createExtendedDescription methods must be migrated before using the classes.

- These methods are now development methods (former runtime methods):

```
MicFwApplicationObject class>>#allKeyAspects
MicFwApplicationObject class>>#setTypeConverter:for:
```

- Removed methods:

```
MicFwApplicationObject class>>#hasKey (micInternal, R)
MicFwPersistentObject class>>#removePersistence: (micInternal, D)
MicFwAppliationObject class>>#hasNTolRelationship (micAPI, D)
```

### 4. Tool for migrating #createExtendedDescription methods (required for this release)

Before you can use your classes with the new release, you must migrate (regenerate) all #createExtendedDescription methods with a migration tool.

This migration tool generates a new #initExtendedDescription: class method and deletes the former #createExtendedDescription class method.

Usage:

Open the migration tool with

Transcript->micFrameworks->Migrate #createExtendedDescription methods

The tool will initialize the extended descriptions from the old method (or from the new method when already available).

The tool then asks whether you want to migrate the generated method or not.

If you press "no", then no methods will be modified and the extended description objects are initialized so that you can work with your old classes without the need to migrate the methods before testing your code.

If you press "yes", the tool prompts for applications in which methods should be regenerated.

After selecting the adequate applications and pressing "Choose" all classes for which the #createExtendedDescription method needs to be regenerated in the selected applications will be migrated.

Note: Make sure, that you have privileges to modify code in the applications you selected.

---

## Release 3.4 Version 1.0

### 1. System requirements

System configuration requirements:

- Operating system: IBM OS/2 , Windows 95, Windows 98, Windows NT, AIX.
- Smalltalk development environment: IBM Visual Age for Smalltalk Version 4.x.

### 2. Hierarchical transaction contexts

The concept of transactions has been extended in this release. A context can now be created as a child of another (parent) context. A parent context can have any number of child contexts. Any context can be a parent context.

For details about this new feature refer to:

- '1.3.4.7.5. Parent-child contexts' (page 37). Describes the concept of hierarchical transaction contexts.
- '2.21. Multiple contexts: Concurrent' (page 99). Examples that demonstrate how hierarchical transac-

tion contexts work.

### 3. Find Redefined Variable Typings

A new menu item appears in this version of OBF. From the System Transcript select "micFrameworks / Object Behavior Tools / Find Redefined Variable Typings". This will print a list of variables whose types were redefined in a subclass of the class that originally defined the variable.

### 4. "Lazy initialize" for relationship accessors

Normally the relationship variables are initialized by the class method initializeRelationshipsFor:.

This can, however, require a significant amount of system resources for objects with many relationships, especially when these objects are created at one time (such as when objects are loaded from a database). Therefore, the generation of the initializeRelationshipsFor: method is now optional.

If no initializeRelationshipsFor: method exists, then the relationships are initialized by the generated accessor methods. This type of relationship initialization is referred to as **lazy initialization**.

#### Enabling lazy initialization (disabling generation of initializeRelationshipsFor:)

- From the Object Net Browser select **Extra / Options**. The following dialog appears:

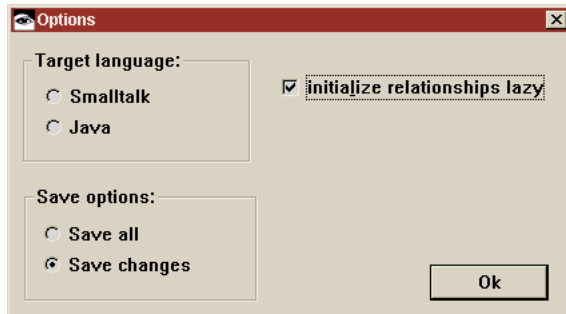


Figure 1: Initialize relationships lazy checkbox

- Check the checkbox **initialize relationships lazy**.
- Click **OK**.

#### Enabling lazy initialization for classes created in previous OBF versions

There are 2 methods for enabling lazy initialization of classes created in previous OBF versions.

Method 1:

- Load classes into the Object Net Browser.
- From the Object Net Browser select **Extra / Options**.
- Check the checkbox **initialize relationships lazy**.
- Select **Save all**.
- Click **OK**.

Method 2:

- From the Object Net Browser select **Extra / Options**.
- Check the checkbox **initialize relationships lazy**.
- Click **OK**.
- Call the following methods for the classes that required lazy initialization:
  - generateCreateExtendedDescriptionMethod
  - generateAccessorMethods

---

## Release 3.3 Version 1.3

### 1. System requirements

System configuration requirements:

- Operating system: IBM OS/2 , Windows 95, Windows NT, AIX.
- Smalltalk development environment: IBM Visual Age for Smalltalk Version 4.x.

### 2. Regeneration of Basic Accessors

Accessor and #createExtendedDescription methods for typed and persistent classes created with previous releases (R3.2 V1.3 or earlier) must be regenerated. To regenerate the methods:

- 1.1. In the Transcript window: Select **micFrameworks / Object Behavior Tools / Migrate generated accessors**. OBF scans your image for methods that need to be regenerated. Classes containing methods that need to be regenerated are displayed.

- 1.2. Select all methods (recommended).
- 1.3. Click OK. The methods are regenerated.

**Note:** The accessor methods created during regeneration are for internal use within OBF. Do not use these methods in applications.

---

# Installing OBF

---

## System requirements

This tutorial assumes that you have the following:

- Windows NT.
  - IBM Visual for Smalltalk **Version 4.5**.
  - Mynd Object Behavior Framework **V5.0**.
- 

## Installation Overview

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

Installing Frameworks consists of 2 main steps:

- Importing the **configuration maps** from the **library** file on the CD ROM (**lmanagerV50.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 OBF configuration maps

OBF is installed in your Smalltalk environment by importing the required 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 OBF library file **pfw2232.dat** (on the CD ROM).
5. In the "Selection required" dialog in the "Name" box: Click on **micFrameworksBaseDevelopment** (a configuration map).
6. A version number ("**V5.0**") appears in the "Versions" box. Click on the version.
7. Click on the ">>" button. "**micFrameWorksBase Development V5.0**" has now been selected as a configuration map to import and appears in the "Selected Versions" box.
8. Import the configuration map "**micFrameworksBase Runtime**".
9. Import the configuration map "**micObjectBehaviorFramework Development**".
10. Import the configuration map "**micObjectBehaviorFramework Runtime**".
11. Select "OK".

The above 4 configuration maps and their applications are imported to your library. The message "Finished importing from (OBF library file)" appears in the System Transcript. The configuraton maps are displayed and highlighted in the "Names" box of the Configuration Maps Browser.

---

## Loading OBF configuration maps

A configuration map can sometimes only be loaded after other configuration maps that the first configuration map requires are loaded. Visual Age will automatically load any required maps. In this case **micObjectBehaviorFrameworkDevelopment** requires **micObjectBehaviorFrameworkRuntime** and **micFrameworksBase Development**, each of which requires **micFrameworksBase Runtime**. Therefore, loading **micObjectBehaviorFrameworkDevelopment** will cause all required maps to be loaded.

12. In the Configuration Maps Browser: Select "**micObjectBehaviorFramework Development**".
  13. In the "Editions and Versions" box: The edition and version codes appear. The applications and requires maps for the selected map are also displayed. Right click on the latest edition and version.
  14. Select "Load with required maps". All of the required maps are loaded (detailed information appears in the System Transcript).
  15. Close the Configuration Maps Browser.
-

---

## OBF tools available from micFrameworks menu

OBF has now been loaded into your IBM Visual Age Smalltalk environment. In the System Transcript the menu bar selection "micFrameworks" appears. The submenu selections include the following OBF functions:

- Browse Log (open Framework Logger).
- Browse Transactions (open Transaction Browser).
- Browse Object Net... (open Object Net Browser).
- Object Behavior Tools (utilities for OBF).
- About micFrameworks... (version and contact information)



# 1

# Concepts



---

## 1.1. What is OBF?

**OBF** stands for "Object Behavior Framework". OBF is a library of classes that when added to your Smalltalk development that provide the ability to define the requirements and restrictions for **object behavior** using a **framework**.

OBF functionality is required by the 2 other frameworks:

- **Persistence Framework (PFW)**, which provides a framework for integrating objects and databases.
- **Application Framework (AFW)**, which provides a framework for integrating objects and applications.

---

### 1.1.1. Object behavior

#### 1.1.1.1. Implementation of object behavior in Frameworks

OBF provides the ability to control object behavior by creating special accessor methods for an object's variables.

##### 1.1.1.1.1. Standard accessor methods

When a standard accessor method (a getter or setter) is sent to an object (the **source object**), the following occurs:

- In response to a getter message: The source object will return a reference to the **target object** referenced by the source object variable.
- In response to a setter message: The source object variable will be changed to reference the target object referenced in the setter message.

##### 1.1.1.1.2. Special accessor methods

The special accessor methods created by OBF provide the following functionality.

###### 1.1.1.1.2.1. Instance variable typing

**Instance variable typing** restricts the class of the target object to a standard Smalltalk class (or one of its subclasses).

Variable typing is described in detail in '1.2. Variable types / relationships' (page 21).

###### 1.1.1.1.2.2. Object Relationships

An **Object relationship** is to some extent similar to variable typing, in that it restricts the class of the target object. However, the target object class is not a standard Smalltalk class, but rather a user-defined class. The user-defined class can be modified, which allows relationships between the source and target objects to be defined.

Relationships are described in detail in '1.2. Variable types / relationships' (page 21).

###### 1.1.1.1.2.3. Transactions

**Transactions** provide a method of transacting changes to a variable. A source variable always references 2 target objects: a **committed target** and an **uncommitted target**. When a new target object is assigned to a transacted variable (while an **active context** exists), the target object becomes the uncommitted target of the variable. Thus, the process of assigning a new object to a variable is transacted. This is implemented in the setter method for the variable. The getter method for a variable can return either the committed or uncommitted target object, depending on the context in which the getter message was sent. Transactions are described in detail in '1.3. Transactions' (page 27).

#### 1.1.1.2. Applications of object behavior

The ability to control object behavior has the following applications.

##### 1.1.1.2.1. Instance variable typing

By restricting the type of object referenced by a variable, the object can easily be made **persistent** (stored in a database).

This and other applications of variable typing are described in detail in '1.2. Variable types / relationships' (page 21).

##### 1.1.1.2.2. Object relationships

Object relationships can be used to model the relationships between the real-world objects in a systematic and controlled fashion. The interrelationships between objects (objects reference each other through variables) creates an **object net**. The object net can be mapped to database tables, providing a persistent representation of the object net.

This and other applications of object relationships are described in detail in '1.2. Variable types / relationships' (page 21).

### 1.1.1.2.3. Transactions

Transactions can be used to control input from a user dialog. For example, all changes in the dialog can be transacted. If the "OK" button in the dialog is clicked, then the transacted changes can be **committed**. If the "CANCEL" button in the dialog is click, then the transacted changes can be **aborted**.

Transactions can also reduce the number of changes to databases and can handle such problems as concurrent access in the Smalltalk environment, freeing the database from such tasks.

This and other applications of transactions are described in detail in '1.3. Transactions' (page 27).

---

## 1.1.2. Framework

OBF provides a framework that automates much of the work involved in implementing the OBF functionality described above.

### 1.1.2.1. OBF Tools

OBF provides a variety of tools for specifying and monitoring OBF functionality. These tools are described in 'Tools' (page 107) and include the following.

#### 1.1.2.1.1. Object Model Browser

The **Object Model Browser (OMB)** is used to create an **object model**. An object model is a specification of object nets / classes and the variables for these nets / classes. An object model can be created within the OMB without actually creating anything in the Smalltalk image. Thus incomplete object nets can be created without creating errors in the Smalltalk image. The Framework aspects (variable typing, relationships) of the contents of the object model, when considered complete, can be tested by the OMB without having been saved to the Smalltalk environment. If no errors are detected, then the model contents can be saved to the Smalltalk image.

OMB provides the complete functionality previously provided by the Object Net Browser / Type Editor / Relationship Editor (these tools are still available).

OMB is described in detail in '3.2. Object Model Browser (OMB)' (page 110).

#### 1.1.2.1.2. Object Net Browser

The **Object Net Browser (ONB)** is used to create an object net.

Note: It is recommended to use the Object Model Browser instead of ONB.

ONB is described in detail in '3.3. Object Net Browser (NetBrowser)' (page 119).

#### 1.1.2.1.3. Type Editor

The **Type Editor (TE)** is used to specify the type of an instance variable.

Note: It is recommended to use the Object Model Browser instead of TE.

TE is described in detail in '3.4. Type Editor' (page 124).

#### 1.1.2.1.4. Relationship Editor

A **Relationship Editor (RE)** is used to specify the relationships between objects.

Note: It is recommended to use the Object Model Browser instead of RE.

RE is described in detail in '3.5. Relationship Editor' (page 125).

#### 1.1.2.1.5. Transaction Browser

The **Transaction Browser (TB)** is used to view or modify transactions.

TB is described in detail in '3.6. Transaction Browser' (page 127).

---

## 1.2. Variable types / relationships

This chapter describes the concepts related to variable types and relationships.

---

### 1.2.1. What are variable types / relationships

#### 1.2.1.1. Variable types

A variable type defines the allowable class (or subclasses) that can be assigned to an object. Available types include standard types (Integer, Date, String, etc.) and user-defined classes.

#### 1.2.1.2. Relationships

A **Relationship** is established between a **Source** object and a **Target** object when a Source object variable type is the Target object and the Target object is a modifiable object (ie, not standard object types such as Integer, Date, etc.).

##### 1.2.1.2.1. 1-way relationships

A relationship is a **1-way relationship** when the 1-way relationship between a source object and 1 or more target objects is specified in entirety by a single source object variable.

##### Example

A 1-way relationship is specified between a Person object (source) and a PersonName object (target) by variable `Person>>name`, which can reference a single PersonName object.

##### Example

A 1-way relationship is specified between a Person object (source) and an Address object (target) by variable `Person>>addresses`, which can reference multiple Address objects.

##### 1.2.1.2.2. 2-way relationships

A relationship is a **2-way relationship** when the 2-way relationship between a source object and a target object is specified by a single source object variable and a single target object variable.

##### Example

A 2-way relationship is specified between a Employee object (source) and a Customer object (target) by:

- Variable `Employee>>ownedCustomers`, which can reference multiple Customer object.
- Variable `Customer>>ownerEmployee`, which can reference a single Employee object.

---

### 1.2.2. Why variable types / relationships are not supported by Smalltalk

#### 1.2.2.1. Variable types

In typed languages such as Java, every variable must be defined as having a single type. For example:

```
int integerVariable;
float floatingPointVariable;
```

Type checking is carried out during compilation.

Smalltalk does not provide variable typing. A variable can reference any object. This supports 2 important concepts in OOP:

- Run-time binding of messages to methods.
- Polymorphism.

##### 1.2.2.1.1. Run-time binding of messages to methods

**Run-time binding** refers to the way in which the method referenced by a message is determined during run-time. This is a fundamental difference between OOP and procedural languages. Procedural languages use **static binding**, which binds a procedure and to a procedure call compilation.

##### 1.2.2.1.2. Polymorphism

**Polymorphism** means that the same message can be sent to more than 1 class of object, and thus the method implemented in response to the message can be different for each class of object that the message is sent to. The message is polymorphic, in that this same message means different things to different classes of objects.

With polymorphism and no variable typing, a single statement can send the same message to different objects.

##### Example

```
aPerson personalInformation.
```

The class of object returned in response to the message `personalInformation` depends on the object that

the message was sent to. For example, a Customer object (Customer is a subclass of Person) in response to the personAllInformation message might return a different class of object than an Employee object (Employee is a subclass of Person). The Employee object might return an object with much more detailed personal information than the object returned by the Customer object.

### 1.2.2.2. Relationships

Relationships provide the ability to work with objects stored in databases as linked tables. Smalltalk provides no support for relationships.

---

## 1.2.3. Why variable types / relationships are supported by OBF

### 1.2.3.1. Variable types

OBF typing provides the following:

- Predictable object behavior
- Unrecognized message avoidance
- External software component integration
- Database interface

#### 1.2.3.1.1. Predictable Object Behavior

A Smalltalk program consists of objects sending messages to other objects and receiving back a reference to an object. Therefore, controlling the behavior of an object means controlling what kind of objects the object's variables can reference.

Smalltalk variables are not directly accessible. An accessor message must be sent to an object that specifies a new object that an object's variable should reference.

#### Example

Assume that `PersonName>>firstName` should reference a string object (the person's first name). Other objects can change the object referenced by `PersonName>>firstName` with the following message:

```
aPersonNameObject firstName: 'John'.
```

Typically `PersonName>>firstName` would be similar to the following:

```
PersonName>>firstName: aString  
firstName := aString.
```

The unpredictability in object behavior arises since `aString` could reference any type of object.

#### 1.2.3.1.2. Unrecognized message avoidance

A message should normally be sent only to a certain class and its subclasses. For example, the message `getPersonAllInformation` should only be sent to an instance of class `Person` or its subclasses. OBF provides a *design-time framework* and *run-time type verification* to avoid unrecognized messages.

#### 1.2.3.1.3. External software component integration

Typing in software components integrated with a Smalltalk application must be maintained. For example, many kinds of drivers (for example, C or C++ libraries) are written in a language that supports static type checking. And Java is a strongly typed language.

#### 1.2.3.1.4. Database interface

Databases require stringent data typing. A Smalltalk program that accesses a database must ensure that the proper type of object is written to a database. The wrong type of object could cause a low-level fatal error.

#### Example

```
aPersonNameObject firstName: 123.  
aPersonNameObject firstName writeToDatabase.
```

If the database is expecting a string object, an error would be generated.

### 1.2.3.2. Relationships

OBF relationships allow rows in linked tables in a database to be loaded as objects. The table relationships are represented by object relationships. The object relationships verified and maintained by OBF allow the objects to be loaded and stored to the database seamlessly, avoiding such problems as *invalid column types* and *concurrent access violations*.

This OBF functionality is used by **PFW**, an advanced tool that provides extensive support of *object persistence* (storage of objects in relational databases).

---

## 1.2.4. How variable types / relationships are implemented by OBF

### 1.2.4.1. Tools

Simple types and relationships are established using the the Relationship Editor (described in detail in ‘3.5. Relationship Editor’ (page 125)).

### 1.2.4.2. Accessors

Types and relationships are verified during program execution with the accessor classes and methods generated by the Relationship Editor.

#### Example

The following accessors are typical for a variable with a simple type:

- Getter:

```
PersonName>>firstName
  ^self readAccessTo: #firstName currentValue: firstName.
```

- Setter:

```
PersonName>>firstName: anObject
  ^self writeAccessTo: #firstName newValue: anObject currentValue: first-
Name.
```

Thus, sending a getter or setter message to an object does not immediate change the object referenced by the variable. The setter `writeAccessTo:newValue:currentValue:` message is sent to the object. The implementor of the method for that message has been created automatically by OBF when specifying the instance variable type using OBF tools.

The accessor methods for simple types and all relationships are described in more detail in ‘Tutorial’ (page 47).

### 1.2.4.3. Description objects

OBF creates for every class that has a typed variable the class method ***createdExtendedDescription***. This method is used to create an ***MicFwExtendedDescription*** object that describes the typed variables in a class. This object has the following structure:

```
MicFwExtendedDescription
  MicFwInstanceVariableDescription
    MicFwTypeDescription
    MicFwRelationshipDescription
```

The ***MicFwExtendedDescription*** object references a ***MicFwInstanceVariableDescription*** object for each typed variable in the class. This ***MicFwInstanceVariableDescription*** references either a ***MicFwTypeDescription*** (simple types) or a ***MicFwRelationshipDescription*** (relationship type). The ***MicFwRelationshipDescription*** object contains complete information about the relationship between 2 classes (classes, variables, cardinality, etc.).

Description objects are described via examples in ‘Tutorial’ (page 47).

### 1.2.4.4. Type: Standard

Standard types are used when the allowed objects referenced by a variable should be limited to a standard Smalltalk type (ie, Integer, Date, String, etc.).

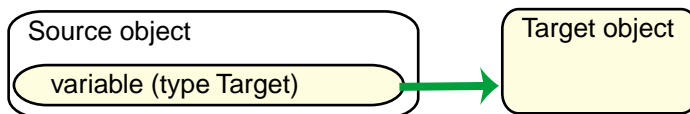


Figure 1.2.1. Standard type

#### Example

A standard type is demonstrated in the tutorial by specifying the type of `Person>>dateOfBirth` as a Date object.

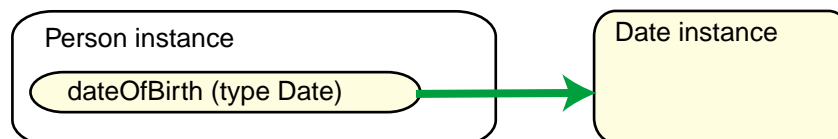


Figure 1.2.2. Standard type example

### 1.2.4.5. Relationships: ->1 (primitive 1-way)

->1 relationships are used when the allowed objects referenced by a variable should be limited to single

instances of a class (or subclasses) that was created within the Framework.

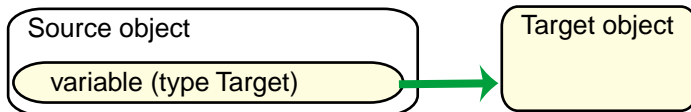


Figure 1.2.3. ->1 (primitive) relationship

#### 1.2.4.6. Cardinality

The **cardinality** of a relationship refer to the allowed number of target objects that each variable can reference. For a ->1 relationship the cardinality is always 0..1 ("0" indicates that the variable can point to nil).

##### Example

A ->1 relationship is demonstrated in the tutorial by specifying the type of Person>>name as a PersonName instance.

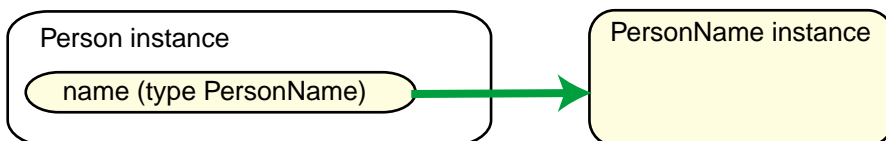


Figure 1.2.4. ->1 (primitive) relationship example

A reference to a PersonName object (or nil) would be returned in response to the "name" getter message to a Person object.

#### 1.2.4.7. Relationships: ->N (primitive 1-way)

->N relationships are used when the allowed object referenced by a variable should be limited to a collection of N instances of a class (and it's subclasses) that was created within the Framework, where N specifies the allowed number of target objects in the set and min<N<max (the cardinality of the relationship is min..max).

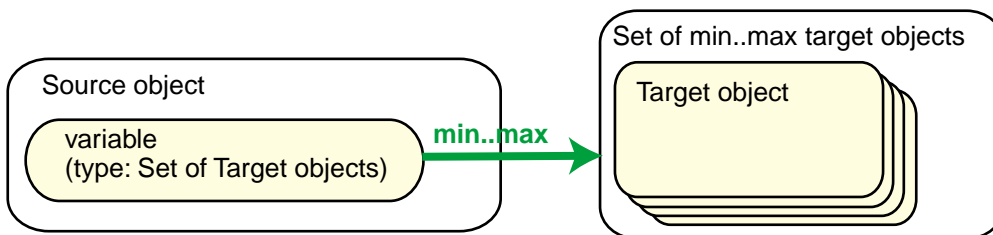


Figure 1.2.5. ->N (primitive) relationship

##### Example

A ->N relationship is demonstrated in the tutorial by specifying the type of Person>>addresses as a set of 1..3 Address objects.

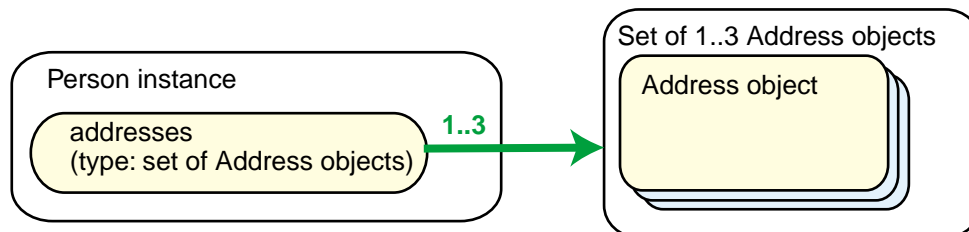


Figure 1.2.6. ->N (primitive) relationship example

A reference to a collection of Address objects would be returned in response to the "address" getter message to a Person object.

#### 1.2.4.8. Relationships: 1<->1

1<->1 relationships are used when:

- The allowed objects referenced by the Source variable should be limited to single instances of the Target class (or it's subclasses) that was created within the Framework
- The allowed objects referenced by the Target variable should be limited to single instances of the



Source class (and it's subclasses) that was created within the Framework.

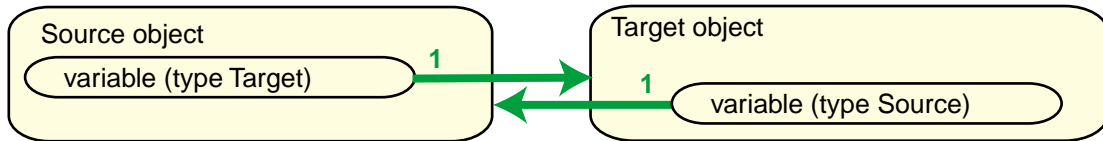


Figure 1.2.7. 1<->1 relationship

**Example**

A 1<->1 relationship is demonstrated in the tutorial by specifying:

- The type of Customer>>portfolio as a Portfolio instance.
- The type of Portfolio>>customer as a Customer instance.

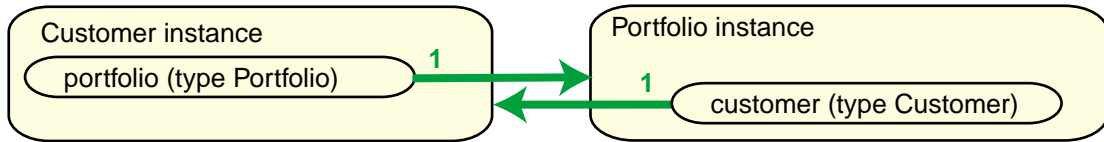


Figure 1.2.8. 1<->1 relationship example

**1.2.4.9. Relationships: 1<->N**

1<->N relationships are used when:

- The allowed objects referenced by a Source variable should be limited to a collection of instances of a Target class (or it's subclasses) that was created within the Framework, where N specifies the allowable number of target object in the set and min<N<max (the cardinality of the relationship is min..max).
- The allowed objects referenced by a Target variable should be limited to single instances of a Source class (or it's subclasses) that was created within the Framework.

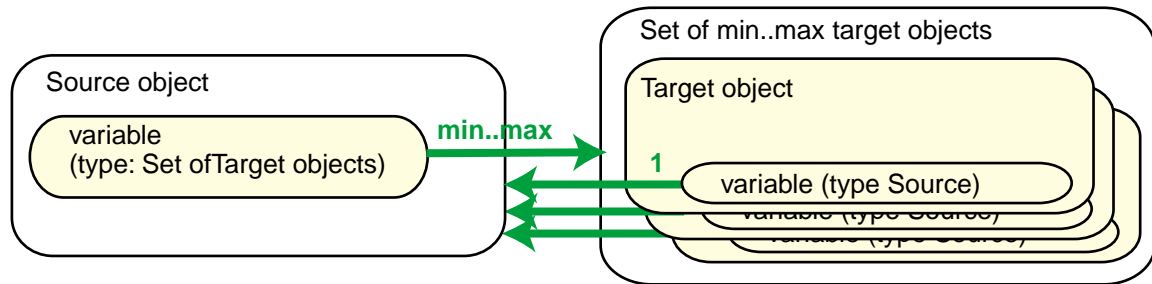


Figure 1.2.9. 1<->N relationship

**Example**

A 1<->N relationship is demonstrated in the tutorial by specifying:

- The type of Employee>>ownedCustomers as a set of 1..3 Customer objects.
- The type of Customer>>ownerEmployee as a single Employee instance.

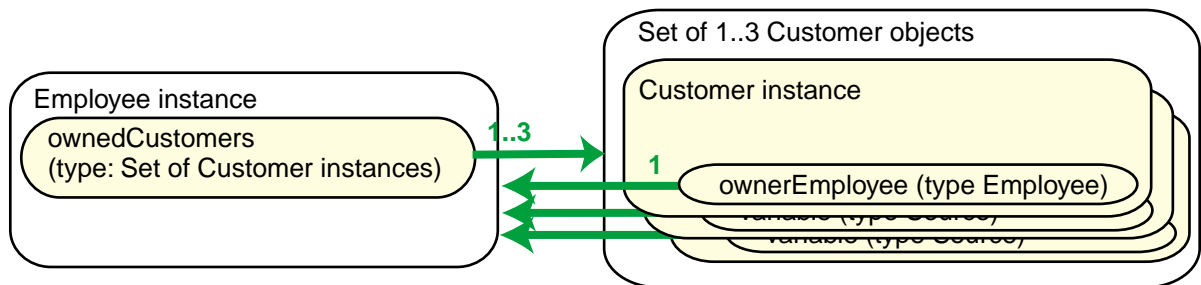


Figure 1.2.10. 1<->N relationship example

**1.2.4.10. Relationships: M<->N**

M<->N relationships are used when:

- The allowed objects referenced by a Source variable should be limited to a collection of instances of a Target class (or it's subclasses) that was created within the Framework, where N specifies the allowable number of target objects in the collection and min<N<max (the N cardinality of the relationship is min..max).

- The allowed objects referenced by a Target variable should be limited to a collection of instances of the Source class (or its subclasses) that was created within the Framework, where M specifies the allowable number of source objects in the collection and min<M<max (the M cardinality of the relationship is min..max).

Note: All of the Target objects in the collection referenced by a Source variable are unique (not the same objects). All of the Source objects in the collection referenced by a Target variable are unique (not the same objects).

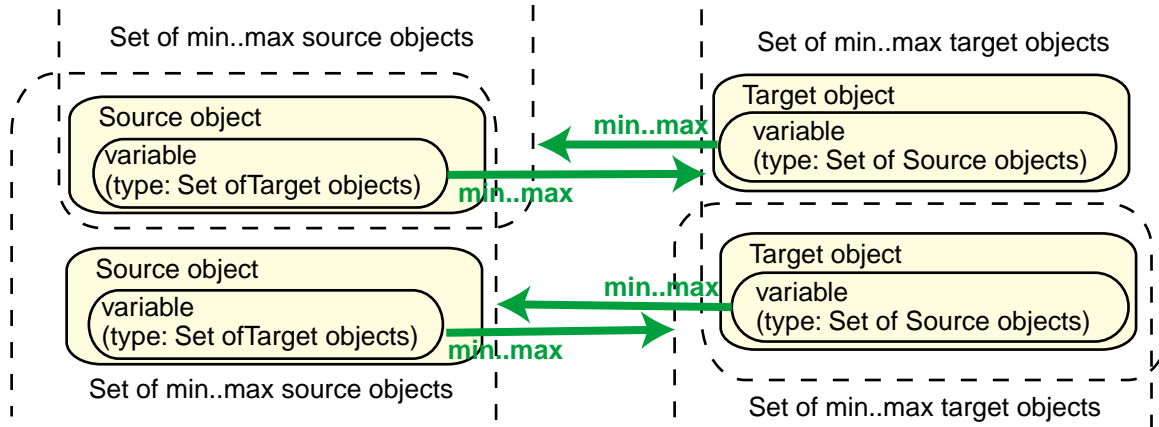


Figure 1.2.11. M<->N relationship

### Example

An M<->N relationship is demonstrated in the tutorial by specifying:

- The type of Person>>addresses is a collection of 1..2 Address objects.
- The type of Address>>persons is a collection of 1..2 Person objects.

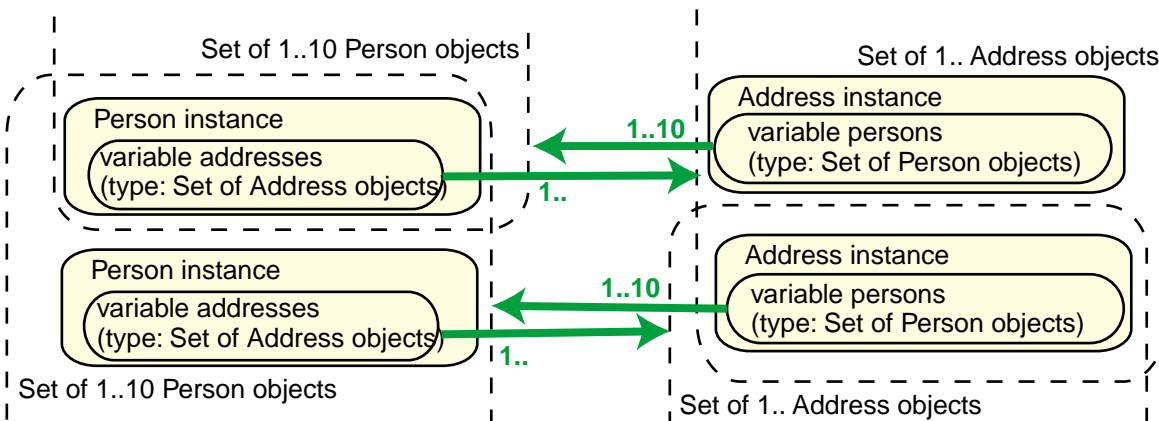


Figure 1.2.12. M<->N relationship example

---

## 1.3. Transactions

This chapter describes the basic concepts of transactions as implemented in OBF.

---

### 1.3.1. What are transactions

A transaction groups together several consecutive actions as though they were a single action. This enables a group of transacted changes which occurred over a given period of time to be either implemented (committed) or aborted at a single instance in time. Thus any inconsistent state that could arise from committing only a portion of a group of commands can be avoided.

#### Bank account example

Assume that DM500 is to be transferred from account A to account B. This single transaction actually consists of several consecutive actions:

- From T1 to T2: Debit DM500 from account A.
- From T2 to T3: Credit DM500 to account B.

This is shown in the following diagram:

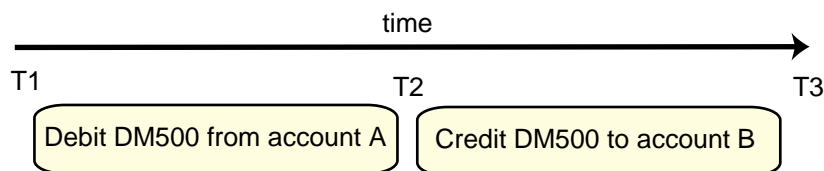


Figure 1.3.1. Debitting account A and crediting account B

If the changes are transacted, then any changes that occurred between T1 and T3 could be aborted in the event that an error occurs (or in the event that it is decided to NOT transfer the money after all).

If no error occurred, then the changes would be committed at time T3.

---

### 1.3.2. Why are transactions not supported in Smalltalk

Transactions are not part of the OOP paradigm of Smalltalk.

---

### 1.3.3. Why are transactions supported by OBF

Transactions are supported by OBF for the following reasons:

- Changes to a variable can be transacted rather than actually being implemented. This can be very convenient, for example, when changes are being made in a dialog. The changes in the dialog could be easily committed / aborted when the "OK" / "CANCEL" button is clicked. And other dialogs displaying some of the same data could display the data with the latest changes or only the changes that have been committed. For this reason, OBF transactions are an important building block for the AFW.
- Concurrent references to a variable can be managed by the OBF **transaction manager**, avoiding concurrency problems with database access. This is implemented by using multiple **transaction contexts**. Only 1 context can be **active** at any time, and only 1 context can have a **lock** on a variable. Thus, a variable can only be modified if the context that has a lock on the variable is the single active context. For this reason, OBF transactions are an important building block for the PFW.

Other uses for the transaction mechanisms are described in later in this chapter.

#### Bank account example

For the above situation, with OBF transaction support, we could have 3 dialogs open that display the account balance for account A.

- Dialog 1 is the dialog where the transfer was initiated.
- Dialog 2 displays the account balance for account A with any pending (uncommitted changes).
- Dialog 3 displays the account balance for account A without pending changes (ie, only committed changes are displayed).

Dialogs 1, 2, and 3 could be assigned to separate processes.

Now if we assumed that the account A balance is **persistent** (ie, stored in a database), then if a 4th process attempted to changed the balance of account A in the databank, a **transaction write conflict** would be generated.

Assume that a 5th process reads the balance of Account A from the database. According to the settings of the context for process 5, the actual value from the databank could be returned, or the current uncommi-

ted (ie, - DM 500) value.

---

## 1.3.4. How transactions are implemented by OBF

### 1.3.4.1. Tools

#### 1.3.4.1.1. Object Net Browser

If changes to a variable need to be transacted, then the variable must be designated as a transacted variable using the OMB (see '3.2. Object Model Browser (OMB)' (page 110) or the ONB (see '3.3. Object Net Browser (NetBrowser)' (page 119)). The OMB/ONB will create the variable accessors for implementing transaction support for the variable.

**Important:** If a variable is not designated as transacted in the OMB/ONB, the variable's accessors will not support transactions.

#### 1.3.4.1.2. Transaction Browser

Any current transactions can be managed using the TB (see '3.6. Transaction Browser' (page 127)).

### 1.3.4.2. Accessors

Transaction of changes to variables is implemented during program execution via the accessor methods generated for the variable.

#### Code example

The following accessors are typical for a variable with basic transactions:

- Getter:

```
PersonName>>firstName
  ^self transactedReadAccessTo: #firstName currentValue: firstName.
```

- Setter:

```
PersonName>>firstName: anObject
  ^self transactedWriteAccessTo: #firstName newValue: anObject currentValue: firstName.
```

### 1.3.4.3. Object versions (committed, uncommitted targets)

A **object version** is created (or updated if it already exists) when a new object is assigned to an object's transacted variable. The object version contains:

- A reference to the **committed target** object referenced by the variable. "committed" refers to the object that the variable will reference if all transacted changes to the variable are aborted.
- A reference to the last **uncommitted target** object assigned to the variable. "uncommitted" refers to the last target object that was assigned to the variable during the current active **transaction level** (described later in this section).

#### Code example

For example, suppose the following code was executed while no transaction context was active:

```
sourceObject var1: 'version1'.
```

Then a transaction context was activated that has no object version for sourceObject>>var1. Then the following code was executed:

```
sourceObject var1: 'version2'.
```

This would result in the following object version:

```
object version
for sourceObject>>var1
committed target = 'version1'
uncommitted target= 'version2'
```

Figure 1.3.2. Newly created object version

Then the following code was executed:

```
sourceObject var1: 'version3'.
```

This would result in the following object version:

```
object version
for sourceObject>>var1
committed target = 'version1'
uncommitted target= 'version3'
```

Figure 1.3.3. Object version with new uncommitted target

Note that the target object 'version2' is no longer referenced (at least by this object version).

**Dialog example**

A object version could be used to record the changes to a field in a dialog. The field could be used to enter a string object that is assigned to the lastName variable of a Person object. Normally the transaction context would be started when the dialog is opened; the transaction context would be committed if the OK button in the dialog is clicked, or aborted if the Cancel button is clicked.

This is demonstrated in the following diagram:

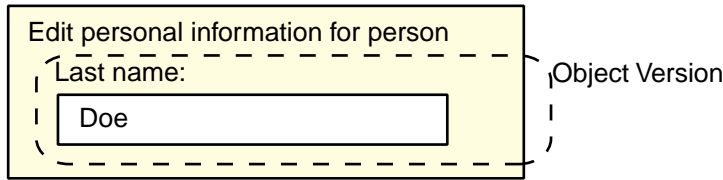


Figure 1.3.4. Object version in a dialog

Entering or deleting a character in the "Last name" field would cause a new uncommitted target (the string in the field) to be assigned to the variable lastName of the Person object.

Note: If the dialog contained a second field (such as for the first name), then a separate object version would be created for the object entered in the second field.

**1.3.4.4. Transaction level 1 (TrLevel1)**

Transaction level 1 is often referred to as "**TrLevel1**" (the name used in the TB).

Note: TrLevel1 is always contained within a transaction **context**. Contexts are examined later in this section. TrLevel1 can also contain nested sublevels; sublevels are also examined later in this section.

**1.3.4.4.1. Object versions in TrLevel1**

TrLevel1 contains 1 object version for each transacted variable (of any source object) that was assigned a new object while the TrLevel1's context was **active**. TrLevel1 can contain any number of object versions.

**TrLevel1 example**

The following example shows a TrLevel1 containing multiple object versions:

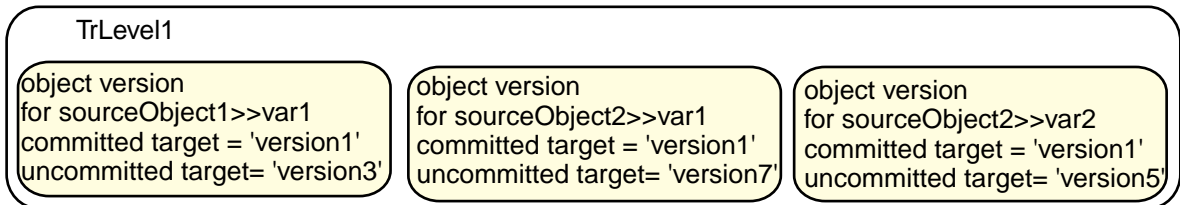


Figure 1.3.5. TrLevel1 with multiple object versions

**Bank account example**

The following diagram shows how the changes to the balance of bank accounts A and B in the bank account example could be transacted in a TrLevel1:

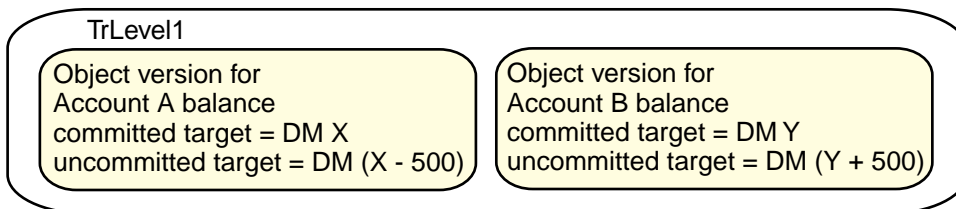


Figure 1.3.6. Object versions in TrLevel1 for balances of accounts A and B

In the above diagram, the **uncommitted target** for the account A balance was created at T1. The **uncommitted target** for the account B balance was created at T2.

**Dialog example**

A single transaction level can represent a dialog. Assigning a new target object to a source object variable

(ie, entering a string in a field, etc.) will generate a new or updated object version for that variable.

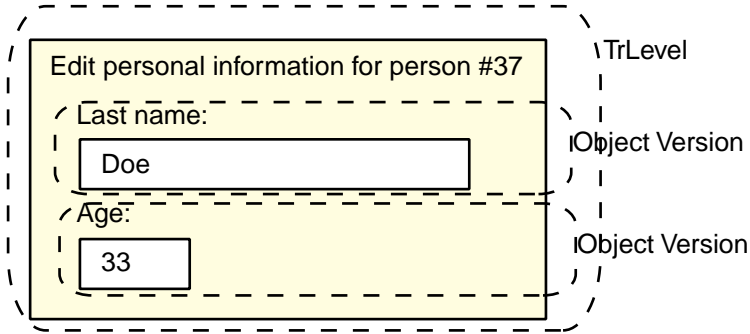


Figure 1.3.7. Transaction level with multiple object versions: Example dialog

Separate object versions will be created for both lastName and age.

### 1.3.4.4.2. Aborting TrLevel1 (aborting context)

The best thing about a TrLevel is that the changes in it can be aborted.

A TrLevel is aborted in order to abort all transacted changes that are recorded in the TrLevel's Object versions.

Aborting TrLevel1 would cause the following:

- The object versions would be dereferenced.
- The context would be aborted (explained later in this section).
- The getters for the variable would return a reference to only the actual target.

This is shown in the following diagram:

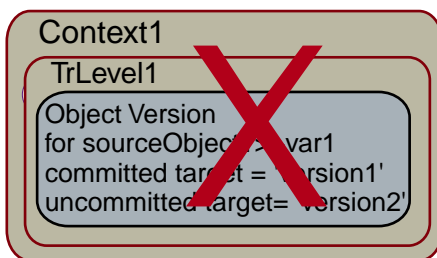


Figure 1.3.8. Aborting TrLevel1

### Bank account example

Assume that in the bank account example someone decided to abort the transfer of money (before T3). TrLevel1 would be aborted, and the object versions would be dereferenced. No transfer of money would occur.

As shown in the following diagram, at time T3 the transaction should be committed or aborted, since all actions involved in the transactions have been finished:

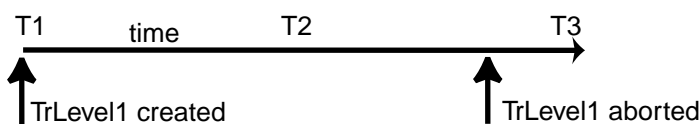


Figure 1.3.9. Bank account example: Aborting changes in TrLevel1 before T3

### Dialog example

A typical scenario where a TrLevel is aborted is clicking the "Cancel" button in a dialog.

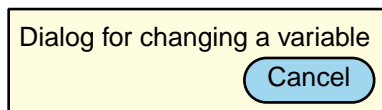


Figure 1.3.10. Aborting a TrLevel: Example dialog

### 1.3.4.4.3. Committing TrLevel1

**Note:** The words "committing" and "committed" are spelled with 1 "t" in the OBF methods ("committed" and "committing").

TrLevel1 is committed in order to commit all transacted changes that are recorded in the TrLevel's Object

versions.

Committing TrLevel1 would cause the following:

- The uncommitted targets in the object versions in the TrLevel1 would become the committed targets.
- The object versions would be dereferenced.
- The context would be aborted (explained later in this section).
- The getters for the variable would return a reference to only the actual object (the newly committed targets).

This is illustrated in the following diagram:

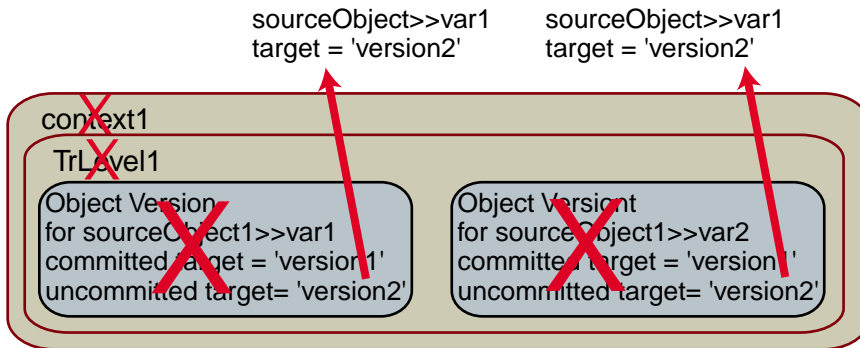


Figure 1.3.11. Committing TrLevel1

### Bank account example

Assume that in the bank account example it was decided that the transfer should go through. The following diagram shows what would happen (assuming that the final button for implementing the transfer was clicked at T3:

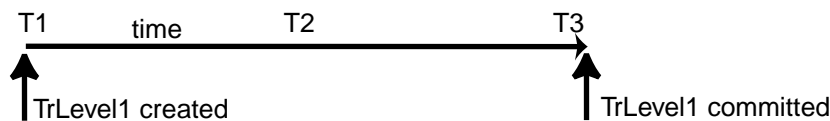


Figure 1.3.12. Bank account example: Committing changes in TrLevel1 at T3

### Dialog example

A typical scenario where a TrLevel is committed is clicking the "OK" button in a dialog.

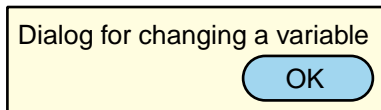


Figure 1.3.13. Committing a TrLevel: Example dialog

### 1.3.4.5. Nested transaction levels

TrLevel1 can contain a nested TrLevel2, which itself can contain a nested TrLevel3, and so on. This is shown in the following diagram:

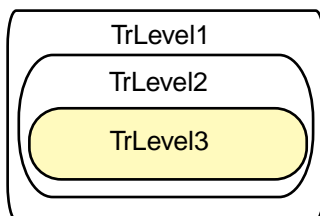


Figure 1.3.14. Nested TrLevels

TrLevel1 is the first TrLevel created.

The highest TrLevel is always the active TrLevel. If a new target object is assigned to a source object transacted variable, then a object version is created (if non-existent) or modified in the highest TrLevel.

### Nested TrLevel Example

If TrLevel2 is the highest TrLevel and a new target object is assigned to a variable, the object version will be updated or created in TrLevel2.

The following diagram shows what would happen if no object version existed in TrLevel1 and a change

was made in TrLevel2:

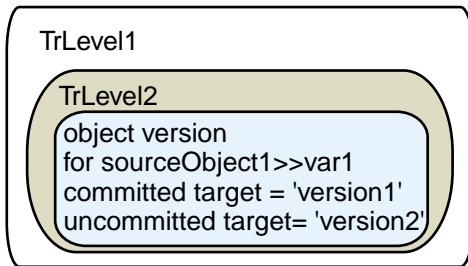


Figure 1.3.15. Object version in TrLevel2

### Dialog example

A typical application of nested TrLevels are subdialogs. Before opening a subdialog, a new TrLevel is created. The object versions for the variables assigned new target objects in the subdialog will belong to the new TrLevel.

This is demonstrated in the following dialog:

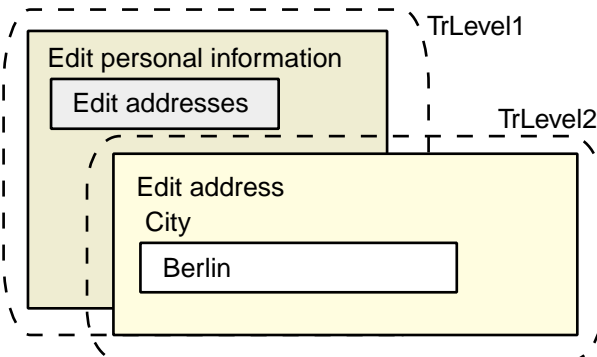


Figure 1.3.16. Nested TrLevels: Example dialogs

The new target object for the city variable will be transacted in TrLevel2.

#### 1.3.4.5.1. Aborting the highest TrLevel (not TrLevel1)

Only the highest TrLevel or ALL TrLevels (in a single context) can be aborted.

Aborting the highest TrLevel (which is not TrLevel1) will cause the object versions in the highest TrLevel and the TrLevel itself to be dereferenced.

### TrLevel example

Assume that you have the following nested TrLevels:

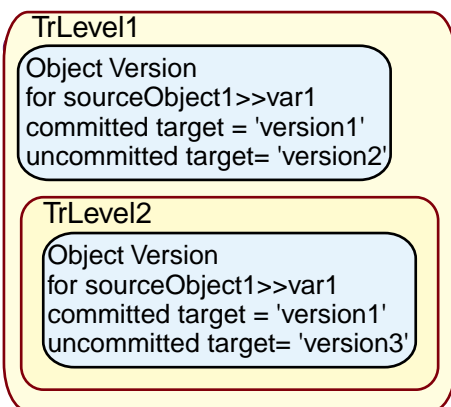


Figure 1.3.17. Object versions in TrLevel1 and TrLevel2



Aborting TrLevel2 would result in the following:

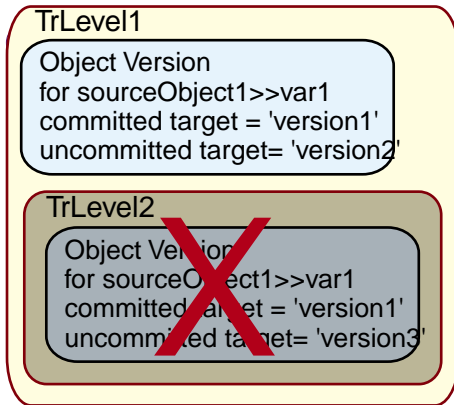


Figure 1.3.18. Aborting TrLevel2

#### Dialog example

A typical situation where the highest TrLevel is aborted is when the "Cancel" button for a subdialog is clicked:

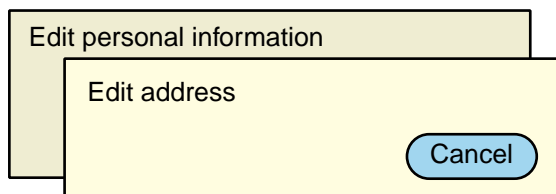


Figure 1.3.19. Aborting highest TrLevel by clicking the Cancel button in the lowest subdialog

#### 1.3.4.5.2. Committing the highest TrLevel (not TrLevel1)

Only the highest TrLevel or ALL TrLevels (in a single context) can be committed.

Committing the highest TrLevel (TrLevelN, which is not TrLevel1) will cause the following:

- The object versions in TrLevelN are carried over into TrLevel N-1.
- If a object version exists for the same variable of the same object in both TrLevels: The object version in TrLevelN-1 is discarded.

#### TrLevel example

The following diagram demonstrates this:

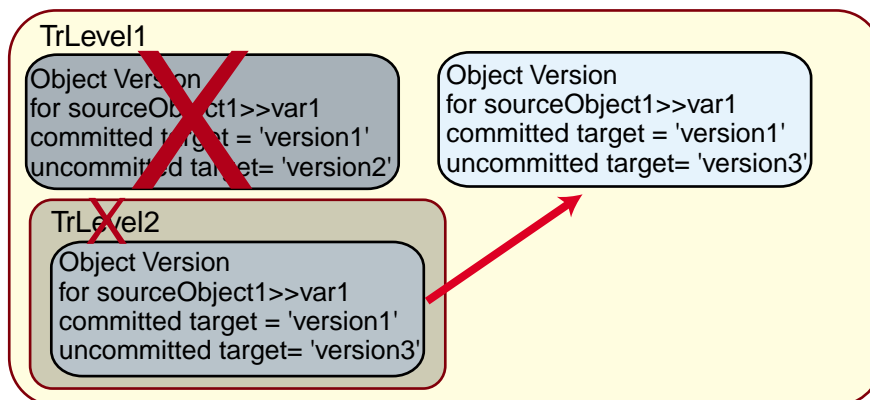


Figure 1.3.20. Committing the highest TrLevel (not TrLevel1)

#### Dialog example

A typical situation where the highest TrLevel is committed is when the "OK" button for a subdialog is clicked:

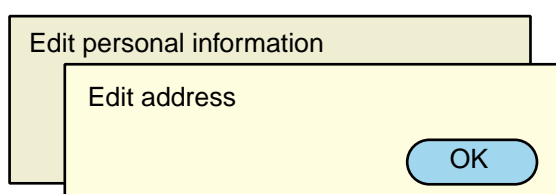


Figure 1.3.21. Committing highest TrLevel by clicking the OK button in the lowest subdialog

### 1.3.4.5.3. Aborting all TrLevels (aborting context)

All TrLevels (in a single context) can be aborted (this is equivalent to aborting the context, which is described later).

Aborting all TrLevels will cause all object versions in all TrLevels to be dereferenced.

#### TrLevel example

Assume that you have the following nested TrLevels:

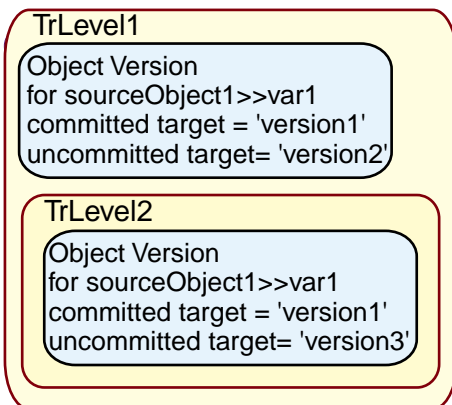


Figure 1.3.22. Object versions in TrLevel1 and TrLevel2

Aborting all TrLevels would result in the following:

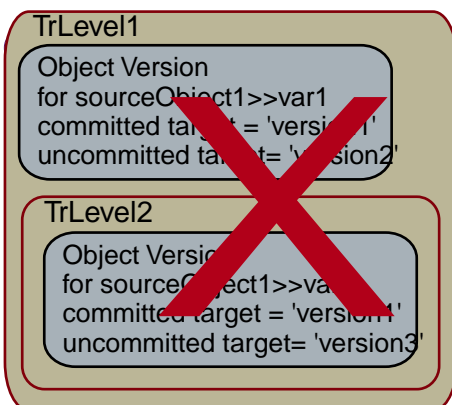


Figure 1.3.23. Aborting all TrLevels (1 and 2)

### 1.3.4.5.4. Committing all TrLevels (committing context)

All TrLevels (in a single context) can be committed (this is equivalent to committing the context, which is described later).

Committing all TrLevels will cause the following:

- The object versions in the highest level are carried over into the second highest level.
- If a object version exists for the same variable of the same object in both of these TrLevels: The object version in the second highest TrLevel is discarded.
- The highest TrLevel is discarded.
- The above steps are repeated until only TrLevel1 remains.
- TrLevel1 is committed (as described earlier).

#### TrLevel example

The following diagram demonstrates this:

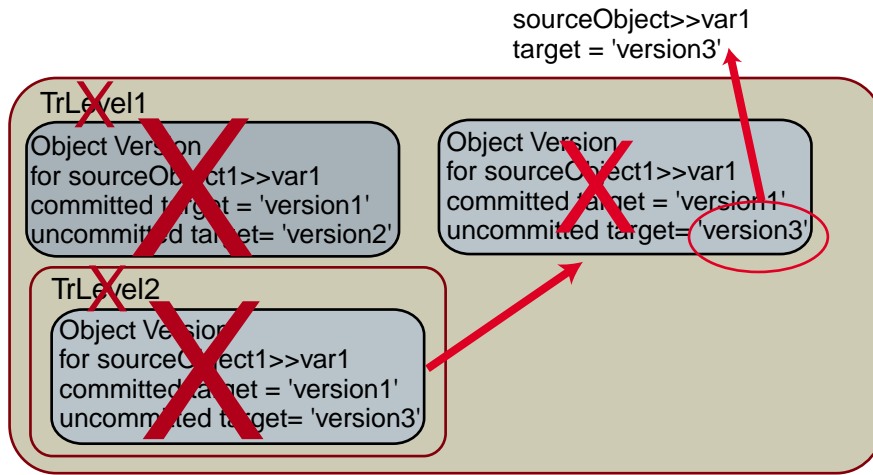


Figure 1.3.24. Committing all TrLevels (1 and 2)

### 1.3.4.6. Single context

A **transaction context** is a logical entity designed to contain a single TrLevel1 and any of its sublevels. TrLevel1 and its sublevels can only exist within a single context. Messages for performing functions with the highest TrLevel (or all TrLevels) are sent to the TrLevel's (TrLevels') context instance.

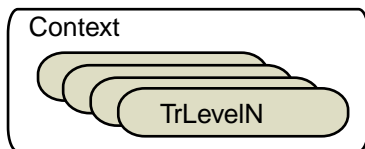


Figure 1.3.25. Context with TrLevels

#### 1.3.4.6.1. Active context

If the context is **active**, then any changes to transacted variables will cause an object version for that variable to be updated (if already exists) or created (if doesn't exist) in the highest TrLevel of the active context.

#### 1.3.4.6.2. Running context

The context is running if it has a TrLevel1.

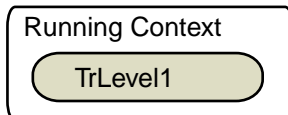


Figure 1.3.26. Running context

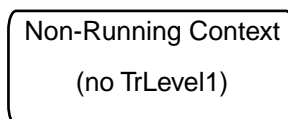


Figure 1.3.27. Non-running context

#### 1.3.4.6.3. Aborting context

When a context is aborted:

- All TrLevels in the context are aborted.
- The context is dereferenced.

#### 1.3.4.6.4. Committing context

When the context is committed:

- All TrLevels in the context are committed.
- The context is dereferenced.

### 1.3.4.7. Multiple contexts

Multiple contexts can exist.

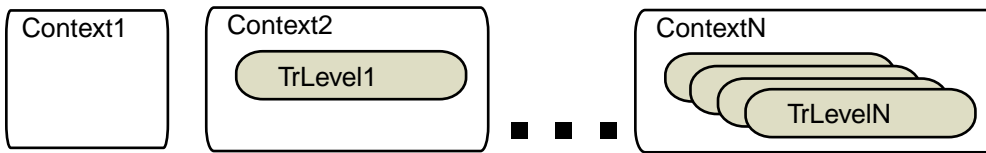


Figure 1.3.28. Multiple contexts

### 1.3.4.7.1. Active context

Only 1 context can be **active** at any time. Changes to a transacted variable are transacted within the active context.

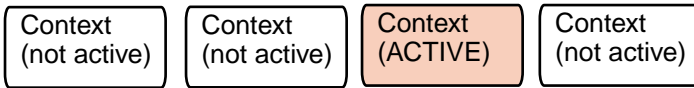


Figure 1.3.29. Single active context in multiple-context environment

Activating a context will deactivate the active context (if an already active context exists).

**Note:** If a context is active and a variable is **locked** by a non-active context: If a new object is assigned to the variable: An error occurs. This is explained in '1.3.4.7.7.1. Variable lock: Definition' (page 37).

### 1.3.4.7.2. Running context

As for a single context environment, in a multiple-context environment a context is **running** if it has a TrLevel1. Multiple contexts can be running / non-running simultaneously.

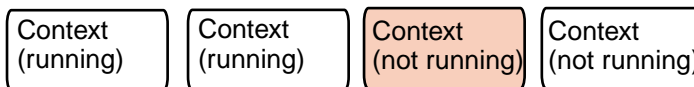


Figure 1.3.30. Running / non-running contexts in multiple-context environment

### 1.3.4.7.3. Hierarchical contexts

A context can contain other contexts. Such contexts are called **hierarchical**.

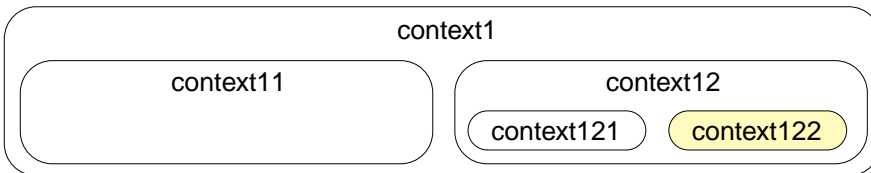


Figure 1.3.31. Hierarchical contexts

Hierarchical contexts can be used for nested dialogs.

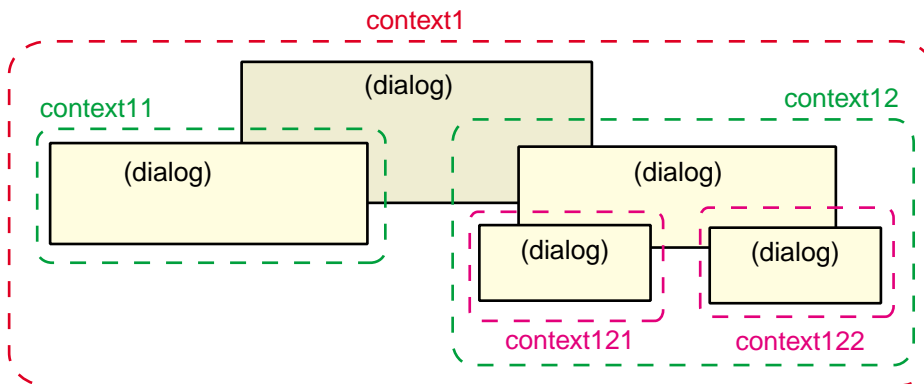


Figure 1.3.32. Hierarchical contexts' dialogs

### Differences between using hierarchical contexts or TrLevels in a single context to transact changes

There are 2 advantages to using hierarchical contexts rather than TrLevels in a single context to transact changes:

- Hierarchical contexts allow not only multiple levels of nested dialogs, but also multiple dialogs on each

level (using TrLevels would allow only 1 dialog on each level).

- Hierarchical contexts allow input in any dialog (using TrLevels would allow input in only the last dialog opened).

#### 1.3.4.7.4. Non-hierarchical contexts

If contexts do not contain other contexts, then the contexts are **non-hierarchical**.

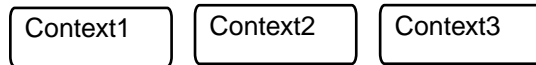


Figure 1.3.33. Non-hierarchical contexts

#### 1.3.4.7.5. Parent-child contexts

2 hierarchical contexts can possibly have a **parent-child** relationship (note: "relationships" is used here as simply a descriptive term and has nothing to do with the technical term "object relationships"). A **child context** is contained within the **parent context**. can be direct or involve several "generations".

For the example hierarchical contexts shown above, the parent child relationships would be the following.

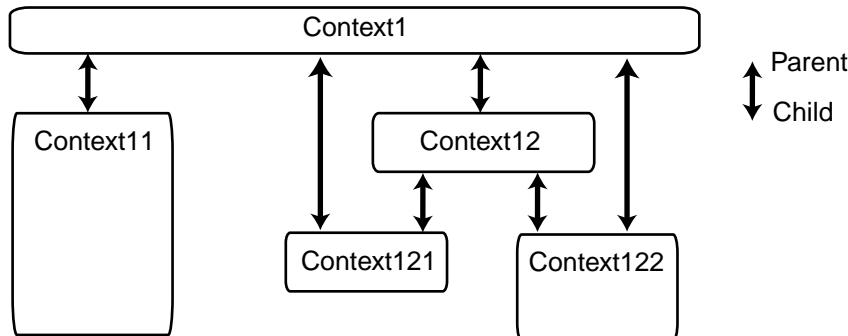


Figure 1.3.34. Parent-child relationships between hierarchical contexts

#### 1.3.4.7.6. Concurrent contexts

2 contexts are said to be **concurrent** if neither context is contained in the other.

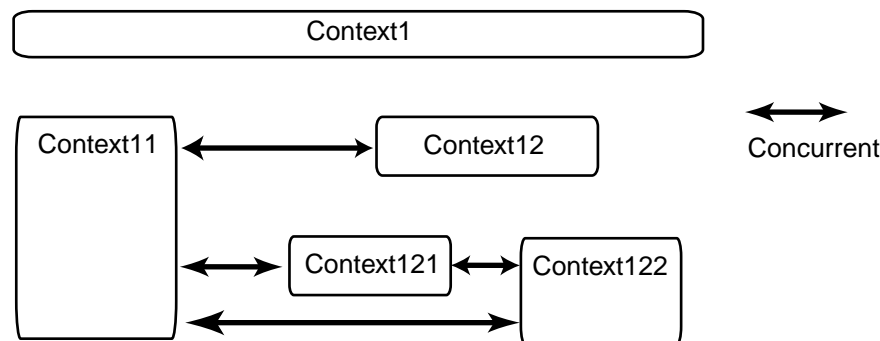


Figure 1.3.35. Concurrent relationships between hierarchical contexts

Note: Non-hierarchical contexts are always concurrent.

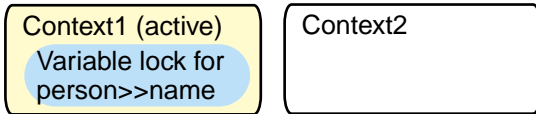
#### 1.3.4.7.7. Setting a transacted variable

Setting a transacted variable (using the variable's setter method to assign a new object to the variable) in a multiple-context environment is the same as in a single-context environment with one very important difference: The concept of a variable lock.

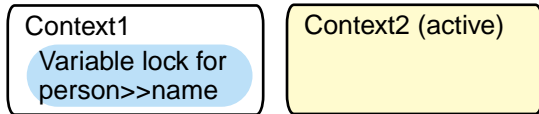
##### 1.3.4.7.7.1. Variable lock: Definition

If a context has a **variable lock** on a variable, then the variable can ONLY be modified in the following 2 situations:

- If the context with the variable lock is active.



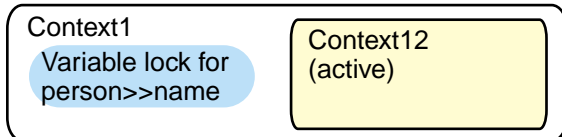
person name: 'new' "no error"



person name: 'new' "error"

Figure 1.3.36. Variable lock: Variable can be changed when context with lock active

- If the context with the variable lock is a non-active parent of the active child.



person name: 'new' "no error"

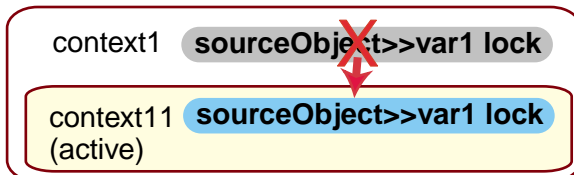
Figure 1.3.37. Variable lock: Variable can be changed when non-active parent has lock

**Note:** If a change is made to a variable that is locked by a parent and a child is active, then the lock is **transferred** to the child (described below).

#### 1.3.4.7.7.2. Getting a variable lock

There are only 3 situations in which a context can obtain a lock on a variable:

- The context is active, the variable is not locked and the variable is changed (An Object Version for the variable in the highest TrLevel of the active context is created).
- A child context is active, the variable is locked by a parent context and the variable is changed. The variable locked is **transferred** to the child. This is shown in the following diagram:



sourceObject var1: 'new' "no error; lock xferred to context11"

Figure 1.3.38. Lock of a variable transferred from parent to child context

**Note:** After the parent has lost the lock to a child, if a change to the variable is attempted while the parent is active: An exception is thrown.

- A direct child context is committed or aborted. The variable locks within the child context are transferred to the parent context, as shown in the following diagram:

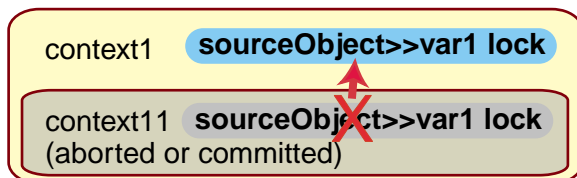


Figure 1.3.39. Transfer of variable lock from child context to parent context when child committed/aborted

#### 1.3.4.7.7.3. Losing a variable lock

There are only 2 situations in which a context with a variable lock that is not committed/aborted can lose a lock on a variable:

- The lowest TrLevel (not TrLevel1) in the context that contains an object version for the variable is aborted.
- The variable lock is transferred to a child context.

#### 1.3.4.7.8. Getting a transacted variable

Getting a transacted variable (using the variable's getter method) in a multiple-context environment involves more factors than in a single-context environment. The object returned by a variable getter (the committed target or the uncommitted target) depends on the following:

- The **context isolation** (described later) of the active context. The context isolation of each context is one of the following:
  - **isolated**
  - **uncommittedRead**
- The relationship between the [active context] / [context with the variable lock]. The relationship can one of the following:
  - **concurrent / concurrent**
  - **parent / child**
  - **child / parent**

This section describes isolated and uncommittedRead contexts and then describes what the variable getter returns in various situations.

Note: Context isolation is only significant when more than 1 context exists.

#### 1.3.4.7.8.1. isolated context

If the active context is **isolated**:

- A getter for a variable locked by a non-active **concurrent** or **child** context returns the **committed target** of the Object Version.
- A getter for a variable locked by a non-active **parent** context returns the **uncommitted target** of the Object Version.

#### 1.3.4.7.8.2. uncommittedRead context

If the active context is **uncommittedRead**:

- A getter for a variable locked by a non-active context returns the **uncommitted target** of the Object Version.

Whether or not a context is uncommitted is not significant when only 1 context exists.

**Caution:** Note the spelling of "uncommittedRead".

#### 1.3.4.7.8.3. Getting: Variable locked by active

If the variable is locked by the active context, then the committed target will be returned.

#### 1.3.4.7.8.4. Getting: Active = uncommittedRead / Variable locked by non-active concurrent

If the active context is uncommittedRead and the variable is locked by a non-active concurrent context, then the variable getter returns the **uncommitted target** from the object version for the variable in [ the highest TrLevel that has an object version for the variable ].

#### Example

context1 has the lock on sourceObject>>var1. context2 is active and uncommittedRead. context1 TrLevel3 is the highest level; however, it contains no object version for the variable. TrLevel2 contains an object version.

The getter for sourceObject>>var1 returns the uncommitted target from the object version in context1 TrLevel2.

This is shown in the following diagram:

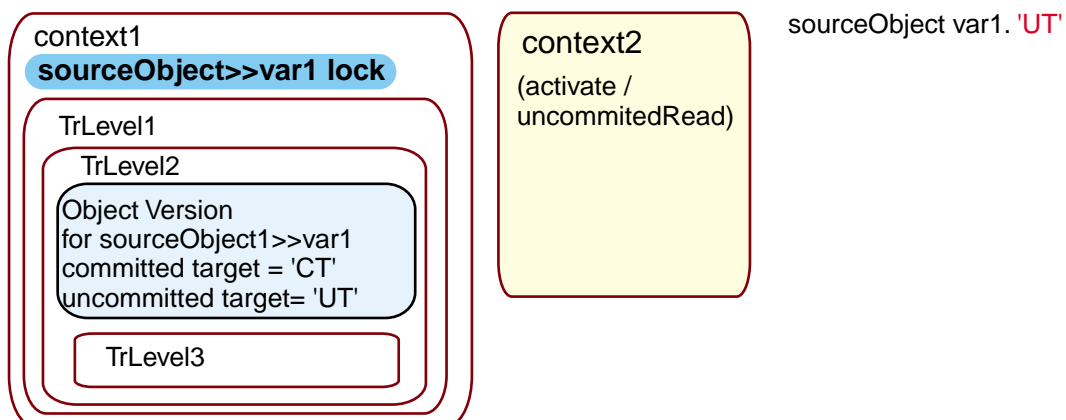


Figure 1.3.40. Uncommitted target returned when active uncommittedRead and variable locked by non-active concurrent

#### 1.3.4.7.8.5. Getting: Active = uncommittedRead / Variable locked by non-active parent

If the variable is locked by a non-active parent context, then the variable getter returns the **uncommitted target** from the object version for the variable in [ the highest TrLevel that has an object version for the variable ] (the Context Isolation of the active child context has no effect).

**Example**

This is shown in the following diagram:

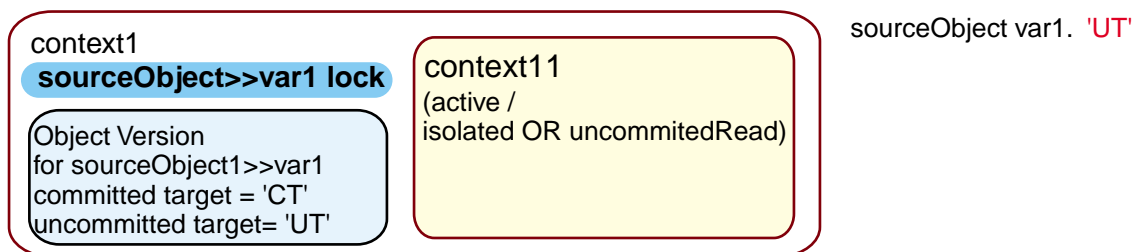


Figure 1.3.41. Uncommitted target returned when variable locked by non-active parent

**1.3.4.7.8.6. Getting: Active = uncommittedRead / Variable locked by non-active child**

If the active context is uncommittedRead and the variable is locked by a non-active child context, then the variable getter returns the **uncommitted target** from the object version for the variable in [ the highest TrLevel that has an object version for the variable ] .

**Example**

This is shown in the following diagram:

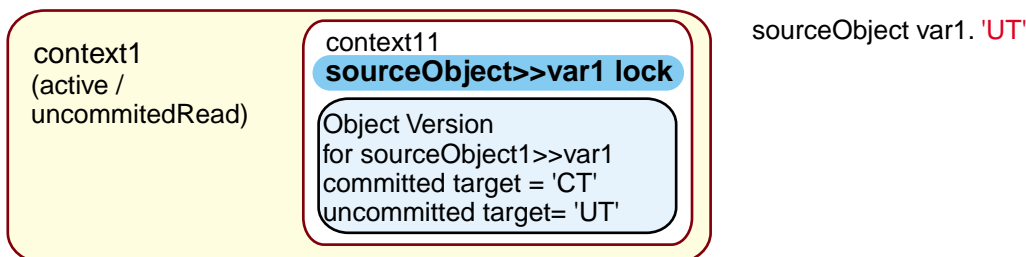


Figure 1.3.42. Uncommitted target returned when active uncommittedRead and variable locked by non-active child

**1.3.4.7.8.7. Getting: Active = Isolated / Variable locked by non-active concurrent**

If the active context is isolated and the variable is locked by a non-active concurrent context, then the variable getter returns the **committed target** from the object version for the variable.

**Example**

context1 has the lock on sourceObject>>var1. context2 is active and isolated. context1 TrLevel3 is the highest level; however, it contains no object version for the variable. TrLevel2 contains an object version. The getter for sourceObject>>var1 returns the **committed target** from the object version in TrLevel2. This is shown in the following diagram:

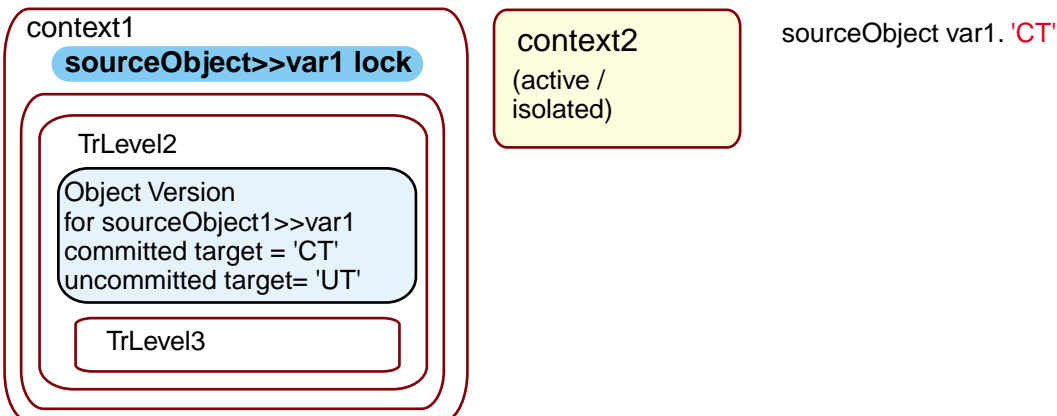


Figure 1.3.43. Committed target returned when active isolated and variable locked by non-active concurrent

**1.3.4.7.8.8. Getting: Active = Isolated / Variable locked by non-active parent**

If the variable is locked by a non-active parent context, then the variable getter returns the **uncommitted target** from the object version for the variable in [ the highest TrLevel that has an object version for the



variable ] (the Context Isolation of the active child context has no effect).  
 This is shown in the following diagram:

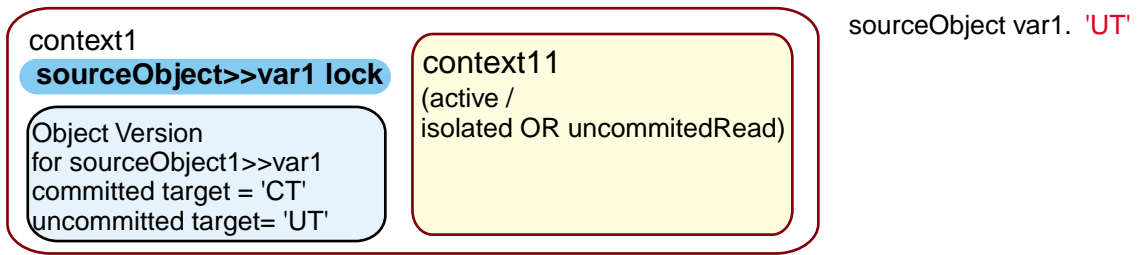


Figure 1.3.44. Uncommitted target returned when variable locked by non-active parent

### 1.3.4.7.8.9. Getting: Active = Isolated / Variable locked by non-active child

If the active context is isolated and the variable is locked by a non-active child context, then the variable getter returns the **committed target** from the object version for the variable in [ the highest TrLevel that has an object version for the variable ].

#### Example

This is shown in the following diagram:

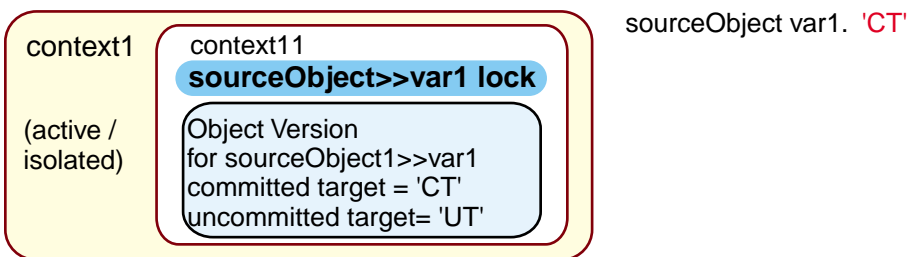


Figure 1.3.45. Committed target returned when active isolated and variable locked by non-active child

### 1.3.4.7.9. Aborting a context

Aborting a context in a multi-context environment involves more factors than in a single-context environment. This is particularly true for parent-child contexts.

#### 1.3.4.7.9.1. Aborting a context with only concurrent relationships to other contexts

Aborting a context that has only concurrent relationships with other contexts has no effect on the other contexts.

#### 1.3.4.7.9.2. Aborting a context with parent and/or child relationships to other contexts

Aborting a context that has child contexts causes all child contexts of that context to be aborted also.

A variable lock in [ the aborted context or a child of the aborted context ] is transferred to nearest parent of the aborted context that has an object version for the variable. If none of the parent contexts of the aborted context has an object version for the variable, then the lock is removed.

#### Example

In the following example, context111 is aborted. obj1 >> var1 lock is transferred to context11, since context1 has an ObjectVersion for that variable (the ObjectVersion for obj1 >> var1 in context1111 is dereferenced). The lock on obj2 >> var3 is released, since none of the remaining (after aborting) parents have an ObjectVersion for the variable..

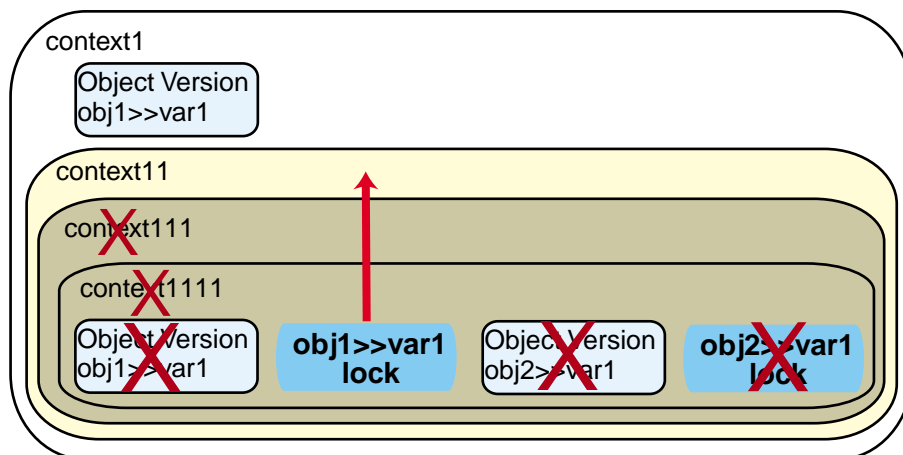


Figure 1.3.46. Aborting context with parent and child relationships to other contexts

### 1.3.4.7.10. Committing a context

#### 1.3.4.7.10.1. Committing a context with only concurrent relationships to other contexts

Committing a context that has only concurrent relationships with other contexts has no effect on the other contexts.

#### 1.3.4.7.10.2. Committing a context with parent and/or child relationships to other contexts

Committing a context that has child contexts causes the following:

- All child contexts of the context are committed.
- The context is committed.

Variable locks and ObjectVersions of the child contexts are transferred to the parent of the committed context.

#### Example

In the following example, context111 is committed. The following occurs:

- context111 is committed.
  - The ObjectVersion for obj1>>var1 is moved to context111. The previous ObjectVersion in context111 is dereferenced.
  - The obj1>>var1 lock is acquired by context111.
  - context1111 is dereferenced.
- context111 is committed.
  - The ObjectVersion for obj1>>var1 is moved to context11.
  - The obj1>>var1 lock is acquired by context11.
  - context111 is dereferenced.

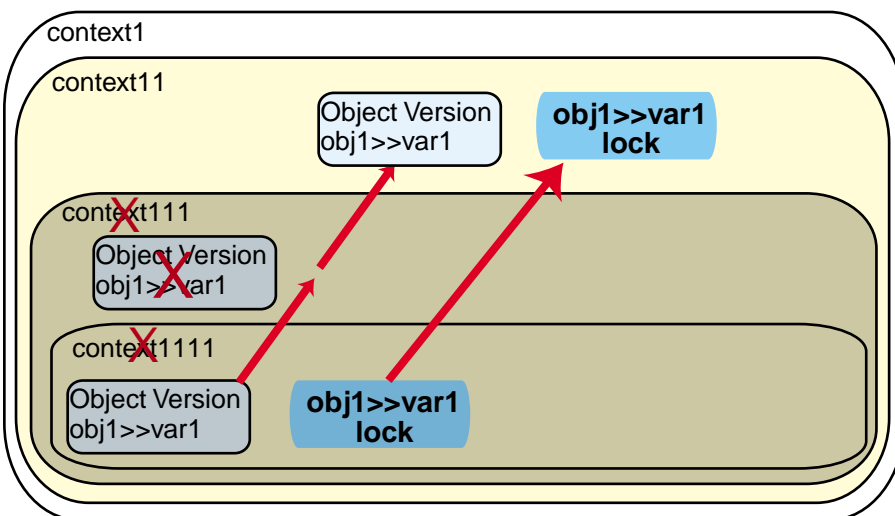


Figure 1.3.47. Committing a context with parent and child relationships to other contexts

### 1.3.4.7.11. Dialog examples

#### 1.3.4.7.11.1. Concurrent contexts

The following diagram shows 2 dialogs that are accessing the same object (Address with id 37). The objects referenced by 2 of the variables (cityName and state) of the Address object are shown. Activating the dialog (by clicking on it) activates the context associated with that dialog.

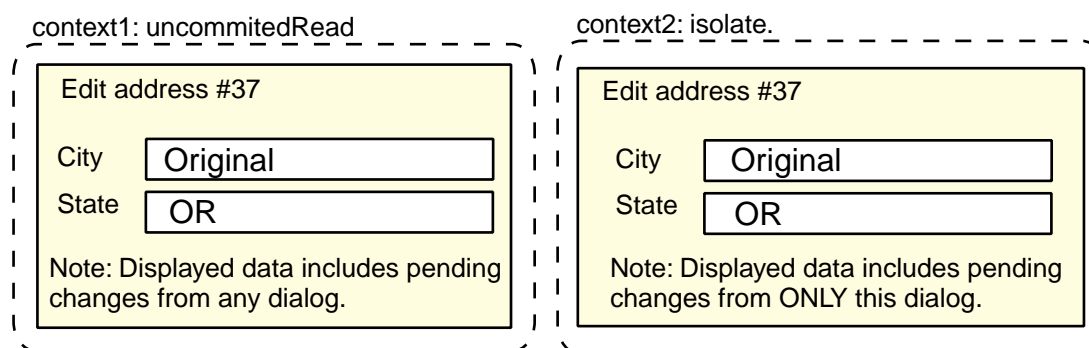


Figure 1.3.48. Example dialogs for uncommittedRead and isolated contexts

In the next diagram, the right window was clicked, activating context2. The city name was changed. This locks the cityName variable for the Address object with id 37 to context2. Note that the change is not yet reflected in the left window, since the changes will only be shown after context1 has been activated.

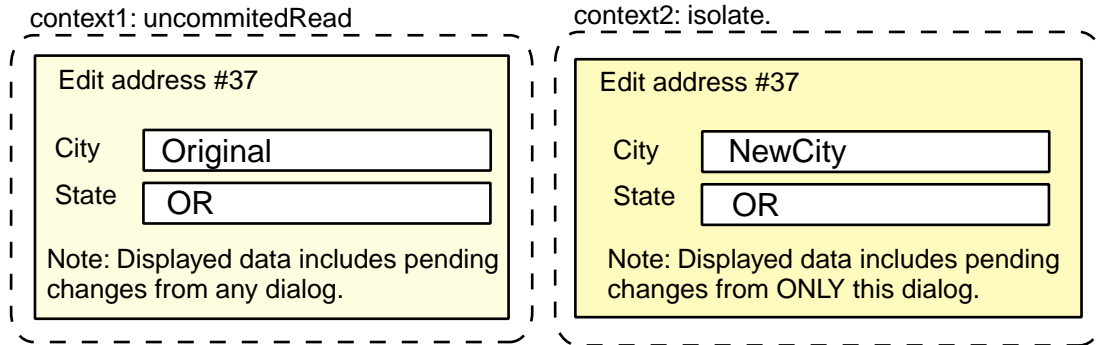


Figure 1.3.49. Example dialogs: Change in isolated context

In the next diagram, the left window has been clicked on. The new city name from context2 is shown in the context1 window.

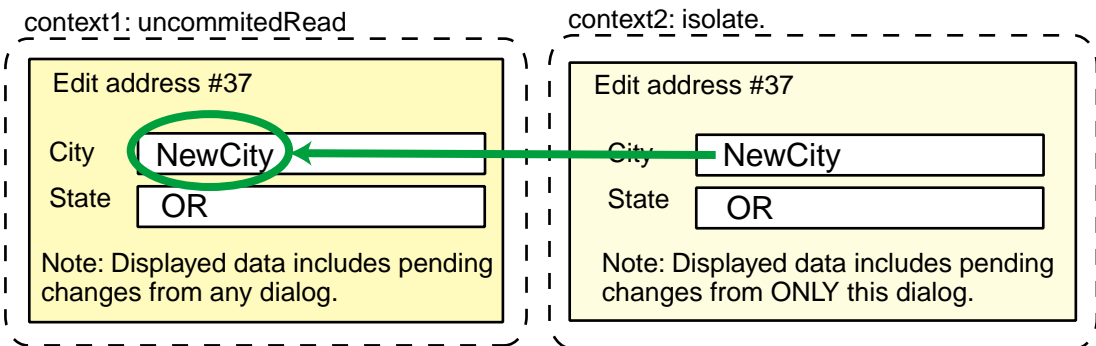


Figure 1.3.50. Example dialogs: Dialog for uncommittedRead context obtains focus

Assigning a different object (such as 'AnotherNewCity') to cityName in a context other than context2 would throw a transaction write exception as soon as another context was chosen. You could enter the new city name in the context1 dialog, but then clicking on the context2 dialog would throw the exception:

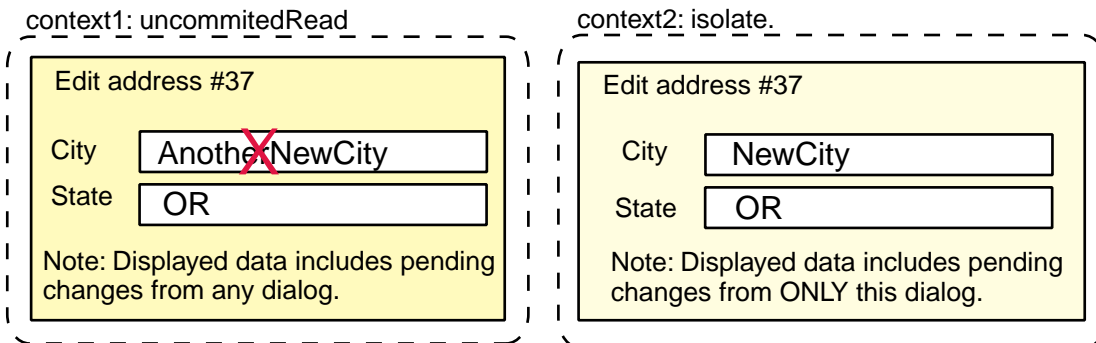


Figure 1.3.51. Example dialogs: Attempted change of a variable locked by other context

In the following diagram, state has been changed in context1 to 'NS'. After assigning the new object and

clicking on the context2 dialog, the change is not shown in the context2 dialog.

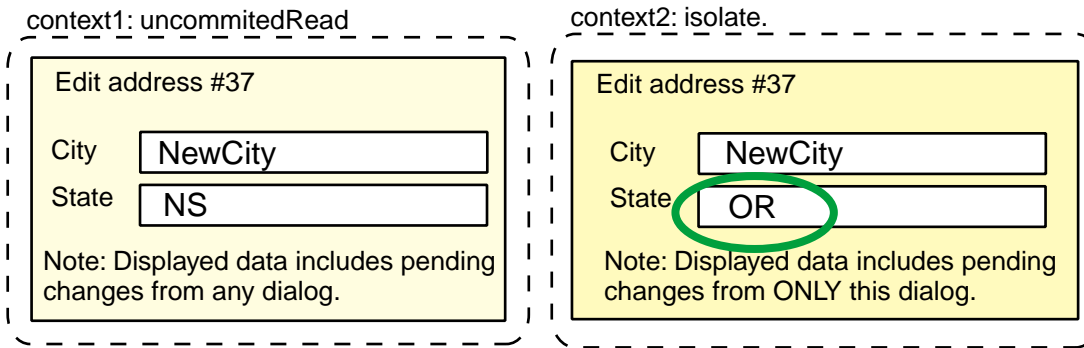


Figure 1.3.52. Example dialogs: Uncommitted change in other dialog is not reflected in the isolated dialog

Finally, the following diagram shows what would happen if context1 was committed and then the context2 dialog was activated: The changes committed in context1 would be shown in the context2 dialog.

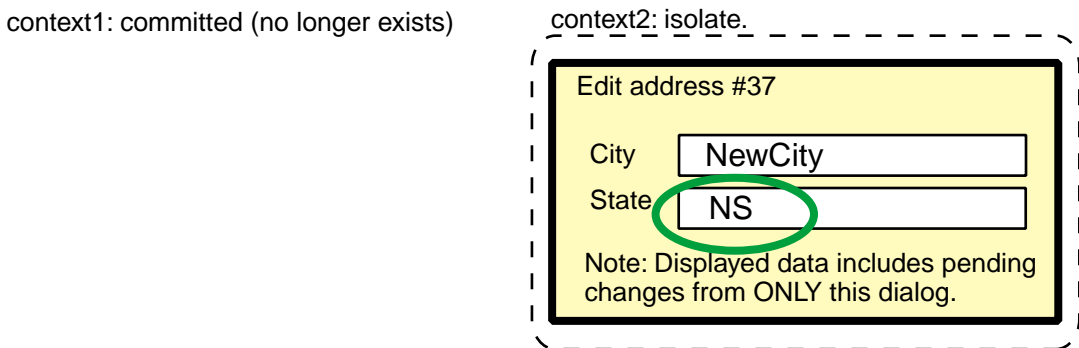


Figure 1.3.53. Committed change is shown in isolated dialog

#### 1.3.4.7.11.2. Parent-child contexts

The following diagram shows 3 dialogs. The middle dialog was the first dialog opened and is assigned context1. The left and right dialogs (context11 and context12) were both opened by clicking the button "Edit address". context11 and context12 are child contexts of context1. All 3 dialogs have 2 fields for entering a cityName object and a stateName object for a Person object with id 61. Note that context1 and context11 are isolated; context12 is uncommittedRead.

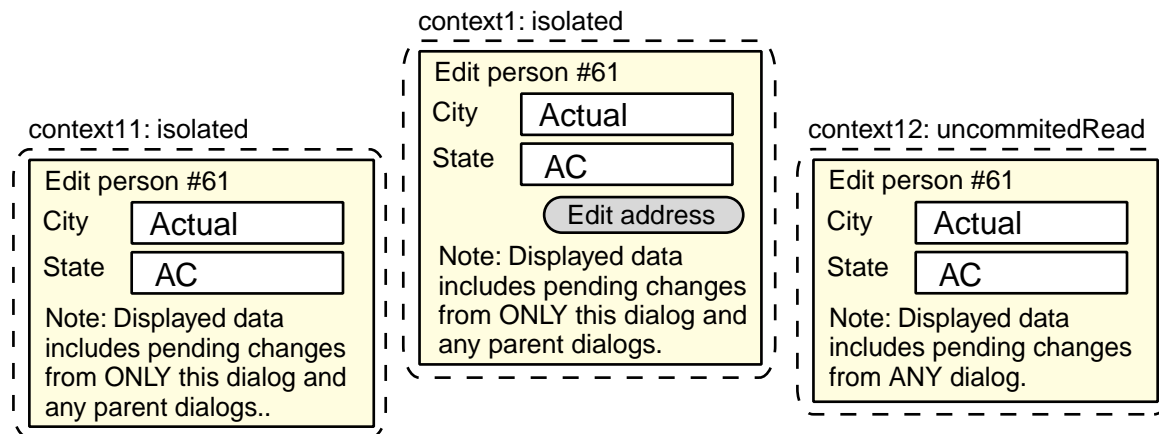


Figure 1.3.54. isolated and committedRead contexts: Example dialogs.

In the following diagram, cityName in context1 is changed. Note that a getter message sent while a child context is active to an object (Person with id 61) that is locked to the parent context will always return the uncommitted target. This is shown in the following diagram (assume that context11 and context12 dialogs

have been activated after the change in context1 dialog):

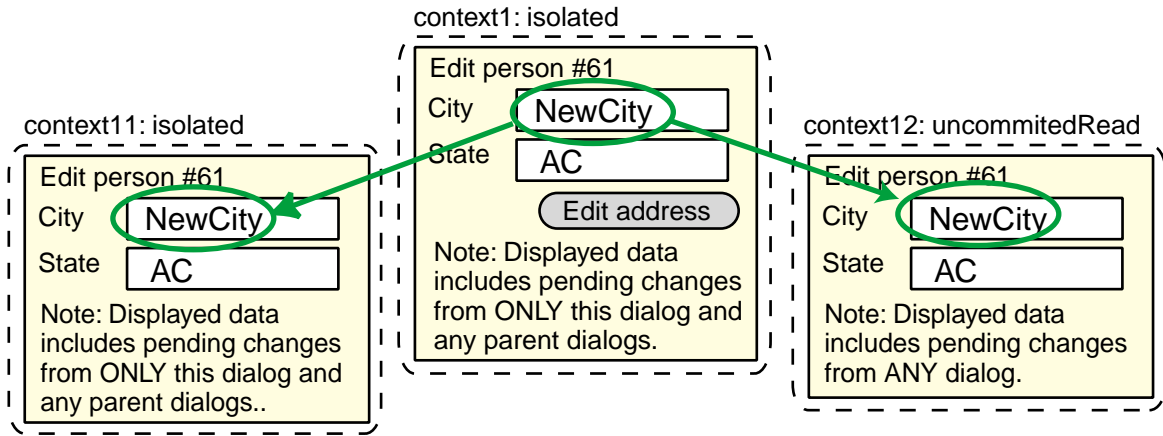


Figure 1.3.55. Change in the parent context dialog is displayed in the child context dialogs.

In the following diagram, cityName has been changed in context12 dialog. This action removes the lock on cityName from context1 and gives it to context12. Assume that context1 and context11 dialogs have been activated once after this. The new cityName object is not shown in either dialog, since the getter message sent when the dialogs were activated was sent within an isolated context.

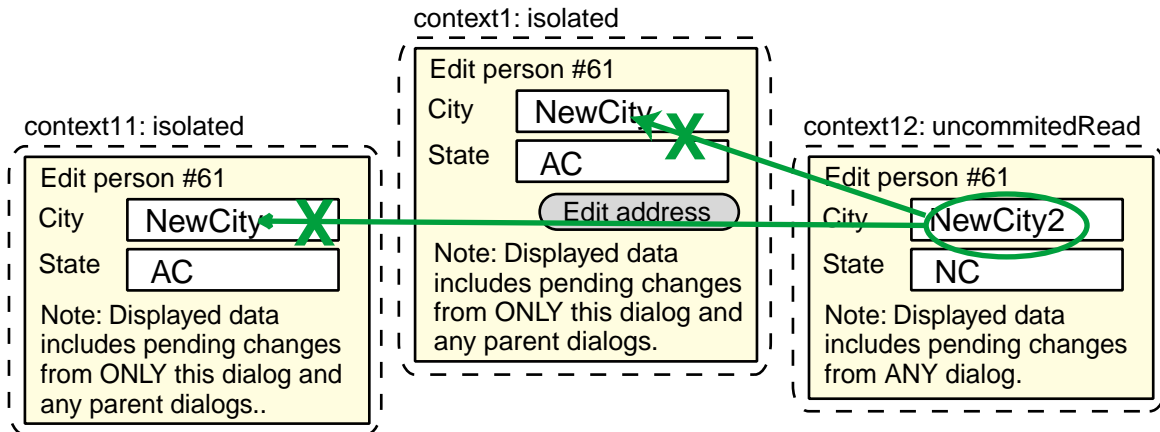


Figure 1.3.56. How changes in an uncommittedRead child dialog are displayed in sibling and parent dialogs

In the following diagram, state is change in the context11 dialog, causing the lock on state to be taken from context1 and given to context11. context1 does not show the change since it is isolated. The change is reflected in context12.

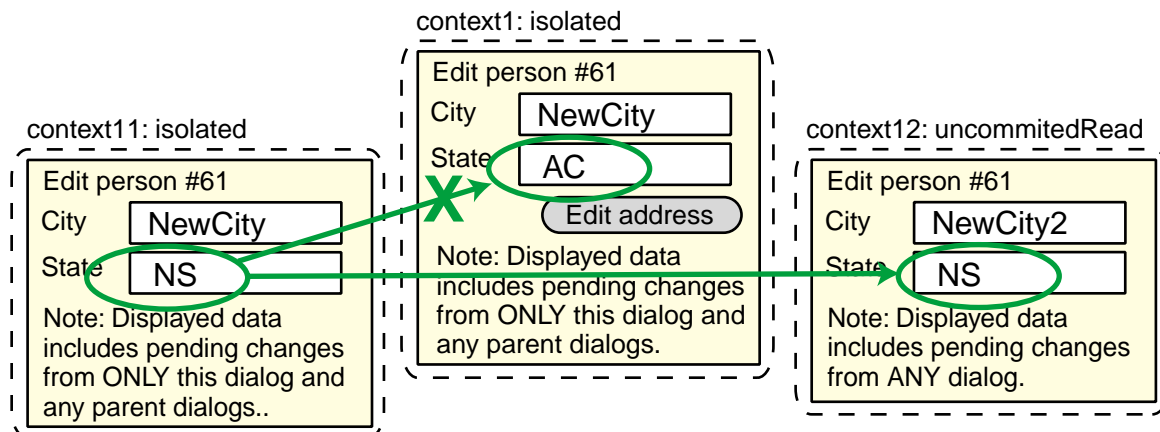


Figure 1.3.57. How changes in an isolated child dialog are displayed in sibling and parent dialogs

The locks on a variable cannot be overtaken by the parent. The only way a parent can get the lock back is when the dialog is closed (and the child context is committed / aborted). Thus, cityName and state cannot

be changed in the context1 dialog. This is shown in the following diagram.

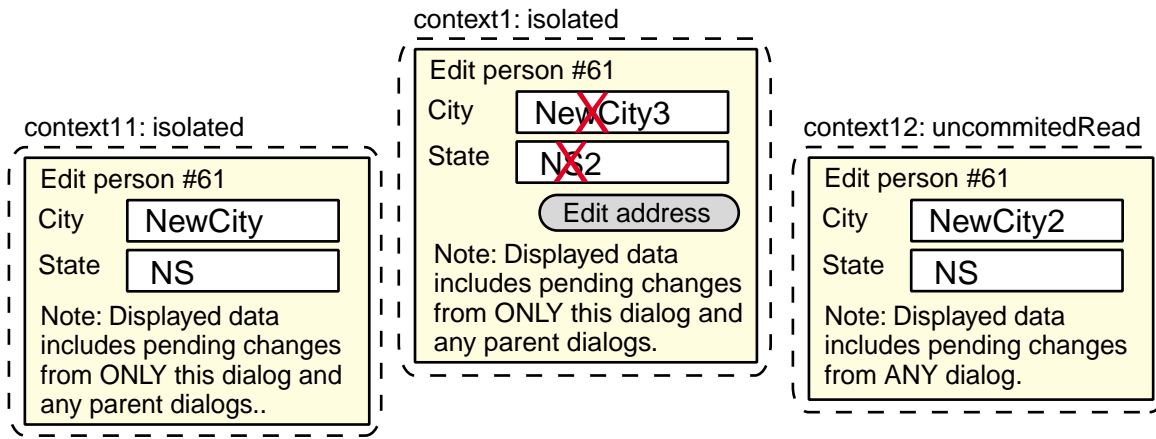


Figure 1.3.58. Variables locked by child contexts cannot be changed in parent context

### 1.3.4.8. Transaction Manager

The transaction manager manages all contexts. The transaction manager is a single instance of the class MicFwTransactionManager.

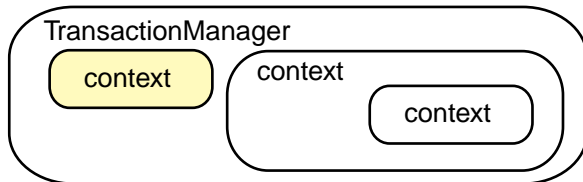


Figure 1.3.59. Transaction manager

# 2

# Tutorial





---

## 2.1. Tutorial overview

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

This section of the User Guide presents a step-by-step tutorial for persons with little or no previous OBF experience (and perhaps limited Smalltalk or object-oriented programming experience). By working through this tutorial, you will quickly be able to exploit OBF functionality and be able to easily grasp more advanced OBF concepts introduced later in this User Guide.

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

### Using the example code

- **'2.2. Using the example code (obf\_ex.txt)' (page 50)**. Explains how to use the example code provided in the accompanying obf\_ex.txt (OBF Examples) file.

### Creating the tutorial application

- **'2.3. Create ZyxTutorial application' (page 51)**. Explains how to create an application with the required prerequisites.

### Models / classes / variables (with OMB)

- **'2.4. Create new model / class (OMB)' (page 52)**. Demonstrates how to create a new model (and create a class in the model).
- **'2.5. Reopen / modify model' (page 54)**. Demonstrates how to reopen a model and how to modify a model.
- **'2.6. Export model' (page 55)**. Demonstrates how a model can be exported to .st and .xml files.
- **'2.7. Save model to VisualAge' (page 56)**. Demonstrates how to "save" the model to VA. Saving involves actually generating the code in VA for the classes/variables in the model.
- **'2.8. Classes' (page 57)**. Demonstrates the various aspects of creating classes with the OMB.
- **'2.9. Variables / Framework accessors' (page 59)**. Demonstrates how to create variables in OMB. The names of the accessors can also be modified in the OMB.

### Variable typing (non-relationships) (with OMB)

- **'2.10. Variable typing (Integer) (with manual type validation, manual/automatic type conversion)' (page 63)**. Demonstrates how to specify variable typing. Also describes how the type of a variable can be manually and automatically checked/converted.
- **'2.11. Variable typing (Date)' (page 67)**. A second example of variable typing.
- **'2.12. Persistent / Key variables' (page 69)**. Explains the persistent and key properties of a variable. Such aspects of variable typing are important for persistence.

### Relationships / Class nets (with OMB)

- **'2.13. ->1 relationship' (page 71)**. Demonstrates how to create a simple ->1 relationship. Analyzes the significance of such relationships for object persistence.
- **'2.14. Adding class nets to the model' (page 75)**. Class nets can be added to a model. What comprises a class net and how to add to the model is explained in this chapter.
- **'2.15. ->N relationship' (page 77)**. Demonstrates how to create a simple ->N relationship. Analyzes the significance of such relationships for object persistence.
- **'2.16. 1<->1 relationship' (page 81)**. Demonstrates how to create a simple 1<->1 relationship. Analyzes the significance of such relationships for object persistence.
- **'2.17. 1<->N relationship' (page 83)**. Demonstrates how to create a simple 1<->N relationship. Analyzes the significance of such relationships for object persistence.
- **'2.18. M<->N relationship' (page 85)**. Demonstrates how to create a simple M<->N relationship. Analyzes the significance of such relationships for object persistence.

### Transactions (with TB)

- **'2.19. 1 context: 1 (non-nested) TrLevel' (page 88)**. The most simple transaction example involves a context with 1 transaction level. Many basic transaction concepts are introduced in this chapter.
- **'2.20. 1 context: Multiple (nested) TrLevels' (page 94)**. A context can have nested transaction levels.
- **'2.21. Multiple contexts: Concurrent' (page 99)**. Multiple context can exist simultaneously.
- **'2.22. Multiple contexts: Parent / Child' (page 102)**. Contexts can be nested. Such contexts are known as hierarchical contexts and are explored in this chapter.

---

## 2.2. Using the example code (obf\_ex.txt)

---

### 2.2.1. Example code in this User Guide

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

```
OBF CH 3.1. Working with transaction contexts."  
[  
  Smalltalk at: #ZyxContext put: MicFwTransactionManager newContext.  
]
```

---

### 2.2.2. Executing example code in obf\_ex.txt

If the code block is marked with the text "**OBF CH (#)**" (Object Behavior Framework User's Guide Chapter #), then the block is included in the file **obf\_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 **obf\_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 classes 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.

## 2.3. Create ZyxTutorial application

In this chapter you will:

- Create application ZyxTutorial. All of the classes created during this tutorial will be assigned to your tutorial application.
- Specify required prerequisites for application ZyxTutorial

### 2.3.1. Create application ZyxTutorial

- 3.1. In **Visual Age Organizer**: Select **Options / Full menus** to display the full menu set in VAST.
- 3.2. Select **Applications / New**. The **New Application** dialog is displayed.
- 3.3. In the **Name** field enter "**ZyxTutorial**".
- 3.4. Make sure that the checkbox **Subapplication of** is not checked.
- 3.5. Click **OK**. The application ZyxTutorial is selected in the list of applications.

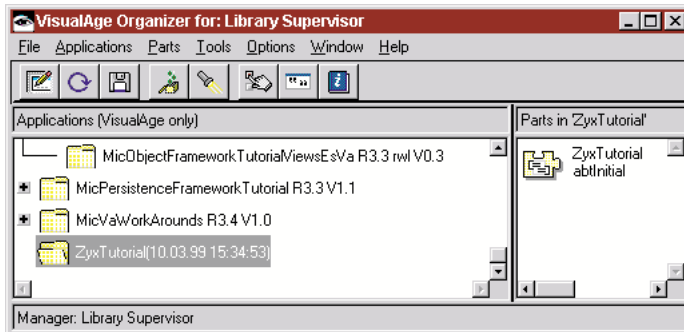


Figure 2.3.1. ZyxTutorial in the application list

### 2.3.2. Specify Prerequisites

- 3.6. From the main menu: Select **Application / Prerequisites**. The **Prerequisites** dialog appears.
- 3.7. Click **Change**. The **Selection required** dialog appears.
- 3.8. Click on **MicFwApplicationModelObjects**.
- 3.9. Click on **>>**. MicFwApplicationModelObjects appears in the list of prerequisite applications.

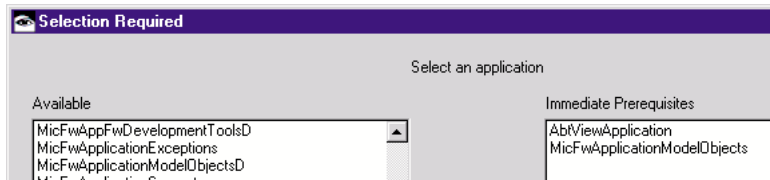


Figure 2.3.2. Prerequisites for ZyxTutorial

- 3.10. Click **OK**. The following dialog appears.

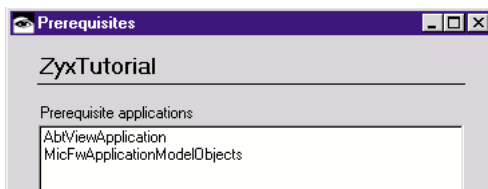


Figure 2.3.3. Prerequisites for ZyxTutorial

- 3.11. Click **OK**.

## 2.4. Create new model / class (OMB)

A model is similar to a sketch. Classes, variables and relationships can be "created" within the model without actually being created in VisualAge. The benefits of this are many, including the fact that incomplete relationships can be created and stored without generating errors in VA.

In this chapter you will:

- Create a new model.
- Create a class in the model (not in VA).
- Save the model (to an .ome file, not to VA).
- Verify that the class does not exist in VA.

### 2.4.1. Create new model

- 4.1. From the **System Transcript**: Select **micFrameworks / Open Object Model Browser....** The **Information required** dialog appears, with the request **Select root class** with the default entry of "\*".
- 4.2. Click **OK**. The **Selection required** dialog appears, with the request **Choose one of the following**. A list of subclasses of MicFwDomainObject is displayed.
- 4.3. Double-click on any class. The **Object Model Browser - Browsing: none** dialog appears with the class net of the selected class:

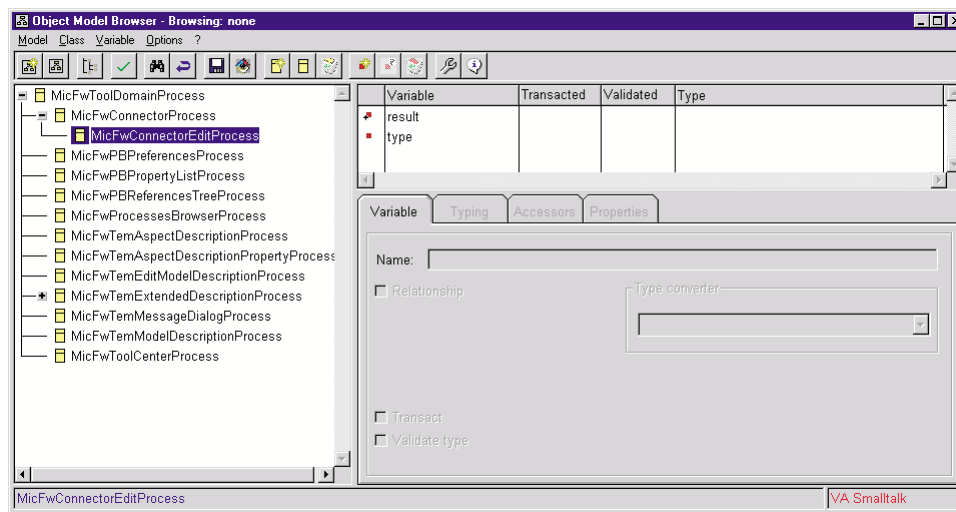


Figure 2.4.1. OMB main dialog (with class net for MicFwConnectorEditProcess)

- 4.4. From the OMB main menu: Select **Model / New model**. A dialog appears with the request **Please enter the name of the new model**.
- 4.5. Enter **ZyxModel1**.
- 4.6. Click **OK**. The OMB dialog is opened for the new (empty) model:

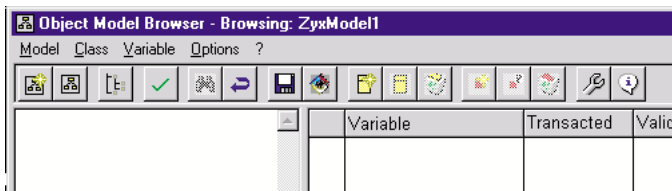


Figure 2.4.2. OMB main dialog (with new model)

---

## 2.4.2. Create a class within the model (not in VA)

4.7. Select **Class / New class...** The **Class specification** dialog tab **Definition** appears.

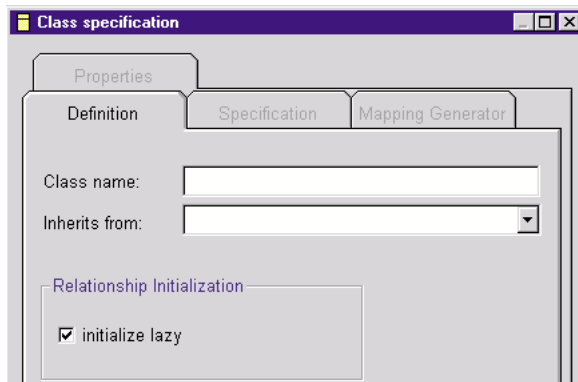


Figure 2.4.3. Dialog "Class specification" tab "Definition"

4.8. In the field **Class name::** Enter **ZyxClass1**.

4.9. From the drop-down list **Inherits from::** Enter **MicFwDomainObject**.

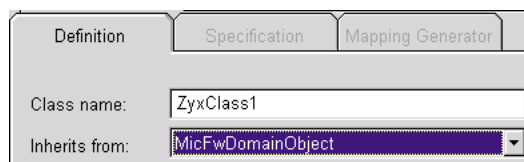


Figure 2.4.4. Class name and parent class specification in dialog "Class specification"

4.10. Select tab **Specification**.

4.11. Click on the **directory button** to the right of the entry field **Application::**. The **Selection Required** dialog appears.

4.12. Double-click on **ZyxTutorial**. ZyxTutorial is now selected as the application for ZyxClass1.

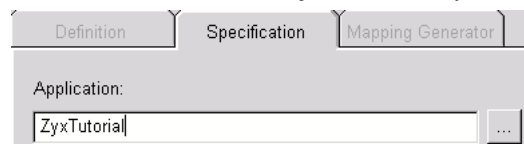


Figure 2.4.5. ZyxTutorial specified as the application for ZyxClass1 in dialog "Class specification" tab "Specification"

4.13. Click **OK**. ZyxClass1 is now displayed in the class net list in the OMB.

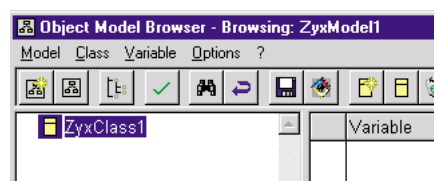


Figure 2.4.6. ZyxClass1 in the OMB class net list

---

## 2.4.3. Save the model

4.14. Select **Model / Save model...** The dialog **Save to...** appears with the default model file name **ZyxModel1.ome** (the default directory is the root directory for VisualAge).

4.15. Click **Open**. The file **ZyxModel1.ome** is created.

---

## 2.4.4. Verify that class ZyxClass1 does not exist in VA

4.16. From the **System Transcript**: Select **Tools / Browse class**. The dialog **Information required** appears.

4.17. Enter **ZyxClass1**.

4.18. Click **OK**. The message '**ZyxClass1**' is not the name of a class. Try again?' appears.

4.19. Click **No**.

---

## 2.5. Reopen / modify model

This section demonstrates the basics of opening and modifying models.

In this chapter you will:

- Reopen the model.
- Modify the model and discard the changes.

---

### 2.5.1. Reopen (overwrite / reopen) the model

A model is closed by:

- Creating a new model (**Model / New model...**)
- Opening a model (**Model / Open model...**)
- Closing the OMB (**Model / Close**)

5.1. Select **Model / Open model....** The message **Save changes?** appears.

5.2. Click **Yes**. The dialog **Save to...** appears with the default model file name **ZyxModel1.ome** (the default directory is the root directory for VisualAge).

5.3. Click **Open**. The message **The file [drive:directory]\ZyxModel1.ome already exists? Overwrite it?**.

5.4. Click **Yes**. The model is saved (overwritten).

---

### 2.5.2. Modify model / Discard all changes

5.5. Select **Class / Class specification....**

5.6. Change the parent class (**Inherits from:**) to **MicFwPersistentObject**.

5.7. Click **OK**.

5.8. Select **Model / Discard all changes**. The message **Discard all changes?** appears.

5.9. Click **Yes**.

5.10. Select **ZyxClass1**.

5.11. Select **Class / Class specification....** Note that the parent class is **MicFwDomainObject**.

5.12. Click **Cancel**.

---

## 2.6. Export model

A model can be exported to:

- Smalltalk .st files (file [classname].st is created for each class in the model)
- .xml file (a single .xml file for alls classes in the model)

In this chapter you will:

- Export the model to .st files (an .st file for each model)
- Export the model to a single .xml file.

---

### 2.6.1. Export to .st files (.st file for each class)

- 6.1. Select **Model / Save to file....** The dialog **Save To...** appears with filename **ZyxClass1.st**. Note: If more than 1 class exists in the model: A .st file is created for each class; however, the dialog **Save To...** will appear only for the first class.
- 6.2. Click **Open**. ZyxClass1.st is created (default directory is the VA image directory).

---

### 2.6.2. Export to an .xml file (single .xml file for all classes)

- 6.3. Select **Model / Tools / Generate XML**. The dialog **Save To...** appears with filename **ZyxClass1.xml**. Note: The name of the .xml file is by default the name of the first class listed in OMB.
- 6.4. Click **Open**. ZyxClass1.xml is created (default directory is the VA image directory).

---

## 2.7. Save model to VisualAge

A model can be saved (ie, the classes/variables specified in the model are actually created in the VisualAge image) to VisualAge. 2 methods for saving to VisualAge exist:

- Saving to VisualAge from within the OMB
- Within VA: Using **File in** to input each .st file into the image

---

### 2.7.1. Saving to VA from within OMB

7.1. Select **Model / Save to VA**. The message **Please wait. Saving source code** appears while the model is being saved.

7.2. In the **VisualAge Organizer**: Select application **ZyxTutorial**. Note that ZyxClass1 is listed as an application class. Note: If ZyxClass1 already exists in the image for a different application: ZyxClass1 will not be saved to VA and no warning message will be issued.

---

### 2.7.2. Using "File in" to save contents of .st file to VA

7.3. Delete **ZyxClass1** from the image.

7.4. From the **System Transcript**: Select **File / Open**. The dialog **Open File** appears.

7.5. Double-click on **ZyxClass1.st**. A **Workspace** is opened with the contents of the file:

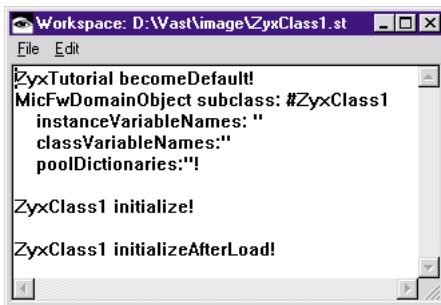


Figure 2.7.1. Contents of ZyxClass1.st in Workspace dialog

7.6. Select all of the workspace contents.

7.7. From the Workspace menu: Select **Edit / File in**.

7.8. Close the Workspace.

7.9. In the **VisualAge Organizer**: Select application **ZyxTutorial**. Note that ZyxClass1 is listed as an application class. Note: If ZyxClass1 already exists in the image for a different application: ZyxClass1 will not be saved to VA and no warning message will be issued.



---

## 2.8. Classes

This chapter describes the basic functions for creating, saving, and deleting classes. In this chapter you will:

- Specify the default application for all classes
- Create subclasses in the model
- Remove classes from the model
- Add classes from VA to the model
- Find a class in the model

---

### 2.8.1. Default application for new classes

- 8.1. In the **OMB**: Select **Options / Preferences....** The dialog **Options** appears.
- 8.2. Select tab **Defaults**.
- 8.3. In the field **Default application::** Select **ZyxTutorial**.
- 8.4. Select radio button **Use default for new class**. When a new class is created, the class will be assigned to the application specified in the **Default application** field.

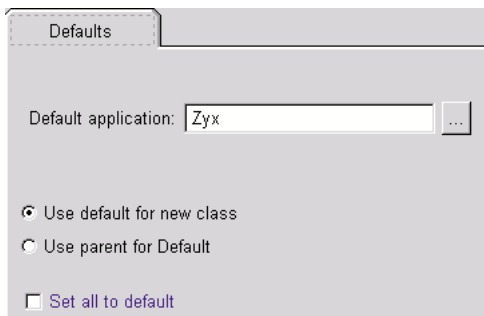


Figure 2.8.1. Specifying default application for new classes

**Note 1:** If **Use parent for Default** is selected: When a new class is created, the class will be assigned to the same application as the parent class.

**Note 2:** If checkbox **Set all to default** is checked: All classes in the OMB are assigned to the specified default application.

**Note 3:** The application for a class can be changed by selecting **Class / Class specification...** + selecting tab **Specification** + selecting the required application.

---

### 2.8.2. Create ZyxClass1 subclass ZyxClass11

By default, a new class is added as a subclass of the class that is currently selected in the OMB.

- 8.5. Click on **ZyxClass1**.
- 8.6. Select **Class / New class....**
- 8.7. For **Class name::** Enter **ZyxClass11** (ZyxClass1 subclass 1). Note that **Inherits from:** is already set to **ZyxClass1**.
- 8.8. Select another tab to change the focus. The **OK** key is enabled.
- 8.9. Click **OK**. ZyxClass11 is added as a subclass of ZyxClass1 (in the model, not in VA).

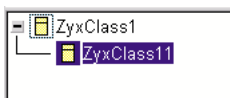


Figure 2.8.2. ZyxClass11 as a subclass of ZyxClass1 in the OMB (not in VA)

Note: A class can be assigned a new parent class by Drag&Dropping the class onto the new parent class.

---

### 2.8.3. Create MicFwDomainObject subclass ZyxClass2

- 8.10. Select **Class / New class....**
- 8.11. For **Class name::** Enter **ZyxClass2**.
- 8.12. For **Inherits from:** Select **MicFwDomainObject**.

8.13. Click **OK**. ZyxClass2 is added as a subclass of MicFwDomainObject (in the model, not in VA).

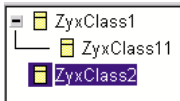


Figure 2.8.3. ZyxClass2 as a subclass of MicFwDomainObject in the OMB (not in VA)

---

#### 2.8.4. Remove classes from model

- 8.14. Select **Model / Save to VA** (the classes will be added from VA later).
- 8.15. Select **ZyxClass1**.
- 8.16. Select **Class / Remove class**. The message **Cannot remove because class "ZyxClass1" has subclass(es)** appears.
- 8.17. Click **OK** to close the message dialog.
- 8.18. Hold down the **Ctrl-Key**.
- 8.19. Select with the mouse **ZyxClass1, ZyxClass11**.
- 8.20. Select **Class / Remove class**. The message **Remove class(es) "ZyxClass11 ZyxClass1" from model?** appears.
- 8.21. Click **Yes**. The classes are removed from the model (but not from VA).

---

#### 2.8.5. Add MicFwDomainObject subclass ZyxClass1 from VA to model

Adding a class to a model does not add the class's subclasses.

- 8.22. Select **Class / Add class(es)....** The message **Select root class** appears. "" (all MicFwDomainObject subclasses) is the default.
- 8.23. Click **OK**. The **Selection required** dialog appears. Note that ZyxClass1/ZyxClass11 are not in the class list.
- 8.24. Select **Model / Save to VA**.
- 8.25. Select **Class / Add class(es)....**
- 8.26. Click **OK**. The **Selection required** dialog appears. Note that ZyxClass1/ZyxClass11 are in the class list.
- 8.27. Double-click on **ZyxClass1**. Note that the subclass ZyxClass11 was not added to the model.

---

#### 2.8.6. Add ZyxClass1 subclass ZyxClass11 from VA to model

Adding a subclass to a model adds class's parent classes (up to and not including MicFwDomainObject).

- 8.28. Select **Model / Discard all changes**. The message **Discard all changes?** appears.
- 8.29. Click **Yes**.
- 8.30. Select **Class / Add class(es)....**
- 8.31. Click **OK**. The **Selection required** dialog appears. Note that ZyxClass1/ZyxClass11 are in the class list.
- 8.32. Double-click on **ZyxClass11**. Note that the subclass ZyxClass11 and the parent class ZyxClass1 were added to the model.

---

#### 2.8.7. Finding a class in the model

- 8.33. Select **Class / Find class....** The message **Enter the name of the class to find?** appears.
- 8.34. Enter **ZyxClass2**.
- 8.35. Click **OK**. ZyxClass2 is highlighted in the OMB.

## 2.9. Variables / Framework accessors

This chapter describes variables and the accessors created for variables. In this chapter you will:

- Add variables to a class
- Analyze the Framework accessors created for the variables
- Modify accessors
- Modify the default specification for accessors for all variables
- Remove and restore a variable
- Make a variable a virtual variable
- Redefine and remove the redefine of a variable

### 2.9.1. Add variable var1, var2 to ZyxClass1

- 9.1. In the **OMB**: Select **ZyxClass1**.
- 9.2. Select **Variable / Add variable...**. The message **Enter the name(s) or the new instance variable(s)** appears.
- 9.3. Enter **var1 var2**.
- 9.4. Click **OK**. The variables appear in the OMB as variables for ZyxClass1

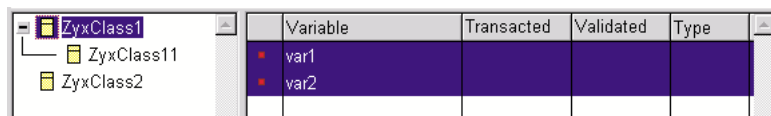


Figure 2.9.1. ZyxClass1>>var1, var2 in the OMB (not in VA)

and as inherited variables for ZyxClass11.

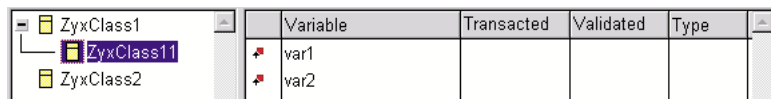


Figure 2.9.2. ZyxClass11 inherited variables var1, var2 in the OMB (not in VA)

- 9.5. Save the model to **ZyxModel1.ome**.

### 2.9.2. Framework accessors generated when model variables saved to VA

- 9.6. Select **Model / Save to VA**.
- 9.7. In the **VisualAge Organizer**: Double click on class **ZyxClass1** (in application **ZyxTutorial**). The Script Editor on ZyxClass1 is opened. Select all categories of **instance methods** for **ZyxClass1**.

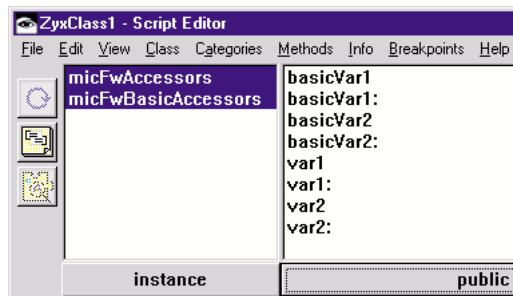


Figure 2.9.3. ZyxClass1 instance methods for var1, var2

The following methods were created for var1 when the OMB contents were saved to VA:

```
basicVar1
  ^var1
basicVar1: anObject
  var1 := anObject
var1
  ^self readAccessTo: #var1 currentValue: var1.
var1: anObject
  self writeAccessTo: #var1 newValue: anObject currentValue: var1.
```

The basic accessors (basicVar1, basicVar1:) provide direct access to the variables and should normally not be used. The framework accessors (var1, var1:) provide access to the variable through framework mechanisms.

### 2.9.3. Modify accessors for ZyxClass1>>var1

- 9.8. Select **ZyxClass1**.
- 9.9. Select **var1**.
- 9.10. Select tab **Accessors**. The default accessor method names for ZyxClass1>>var1 are shown:

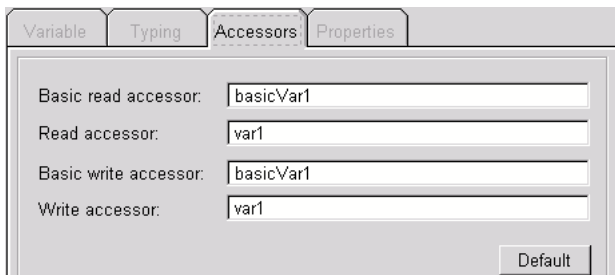


Figure 2.9.4. ZyxClass1>>var1 default accessor methods

- 9.11. Change **Basic read accessor** to **basicVar1X**.
- 9.12. Save to VA. Note that the instance methods now include **basicVar1X**:

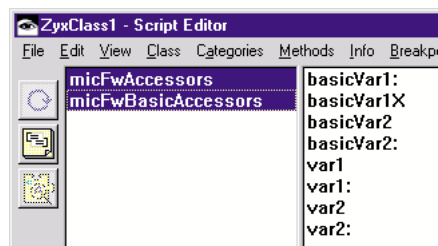


Figure 2.9.5. ZyxClass1>>var1 instance method basicVar1X

### 2.9.4. Modify accessor prefix defaults for all classes

- 9.13. Select **Options / Preferences**.
- 9.14. Select tab **Accessor prefixes**. The default accessor method prefixes for all classes are shown:

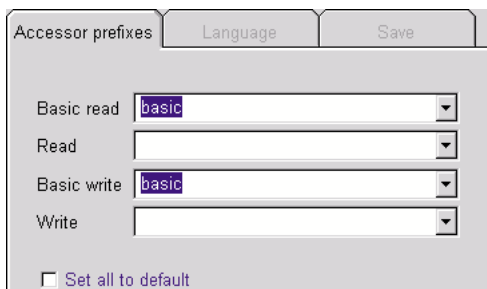


Figure 2.9.6. Default accessor method prefixes for all classes

- 9.15. Change **Basic read** to **basicX**.
- 9.16. Check checkbox **Set all to default**.
- 9.17. Click **OK**. Note that the accessors for all variables have been changed to the defaults specified:

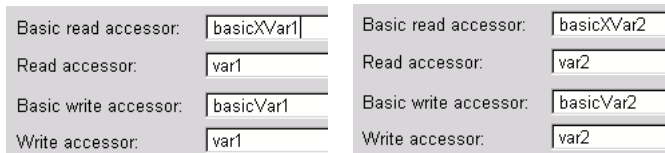


Figure 2.9.7. ZyxClass1>>var1, var2 accessor method names

- 9.18. Change the default for **Basic read** back to **basic**.

---

## 2.9.5. Remove var2

- 9.19. In the **OMB**: Select **ZyxClass11**.
- 9.20. Select **var2**. Note that menu item **Variable / Remove variable** is disabled, since var2 is an inherited variable.
- 9.21. In the **OMB**: Select **ZyxClass1**.
- 9.22. Select **var2**.
- 9.23. Select **Variable / Remove variable**. The message **Do you really want to remove "var2" from "ZyxClass1"?** appears.
- 9.24. Click **Yes**. Note that var2 is removed (in the model).
- 9.25. Select **Model / Save to VA**.
- 9.26. Refresh the Script Editor on **ZyxClass1**. Note that the accessors for var2 no longer exist.

---

## 2.9.6. Restore var2; Make a virtual variable

- 9.27. Select **Model / Discard all changes**.
  - 9.28. Confirm the discarding of all changes. Note that var2 appears again in the model.
  - 9.29. Select **ZyxClass1**.
  - 9.30. Select **var2**.
  - 9.31. Select **Variable / Make virtual variable**. The message **Make "var2" a virtual variable in "ZyxClass1"?** appears.
- Click **Yes**. Note that the icons for var2 in ZyxClass1 and ZyxClass11 are marked with a "V" for virtual.

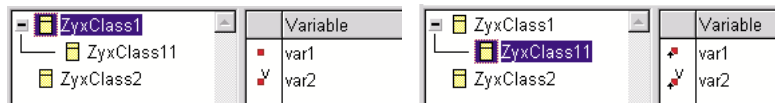


Figure 2.9.8. "V" marking (for virtual) for var2 in ZyxClass1, ZyxClass11 in OMB

- 9.32. Select **Model / Save to VA**.
- 9.33. In the **VisualAge Organizer**: Double click on class **ZyxClass1** (in application **ZyxTutorial**). Note that there are no accessors for var2.

---

## 2.9.7. Make var2 a non-virtual variable; Redefine var2 in subclass ZyxClass11

- 9.34. Select **ZyxClass1**.
- 9.35. Select **var2**.
- 9.36. Select **Variable / Make virtual variable** (this menu item is checked; selecting will cause the item to be unchecked). The message **Do you really want to remove "var2" from "ZyxClass1"?** appears.
- 9.37. Click **Yes**. Note that var2 is no longer marked as virtual.
- 9.38. Select **ZyxClass11**.
- 9.39. Select **var2**.
- 9.40. Select **Variable / Redefine variable**. The message **Redefine "var2" in "ZyxClass11"?** appears.
- 9.41. Click **Yes**. Note that the icon for var2 in ZyxClass11 are marked with a "R" for redefine.



Figure 2.9.9. "R" marking (for redefine) for var2 in ZyxClass11 in OMB

- 9.42. Save the model to VA.
- 9.43. Open the Script Editor on **ZyxClass11**. Note that accessors were created for var2 in ZyxClass11.

---

## 2.9.8. Remove redefine of var2 in subclass ZyxClass11

- 9.44. Select **ZyxClass11**.
- 9.45. Select **var2**.
- 9.46. Select **Variable / Remove redefine variable**. The message **Do you really want to remove "var2" from "ZyxClass11"?** appears.
- 9.47. Click **Yes**. Note that the "R" marking for the icon for var2 in ZyxClass11 is gone.

9.48. Save the model to VA.

9.49. Open the Script Editor on **ZyxClass11**. Note that the accessors for var2 are gone.

## 2.10. Variable typing (Integer) (with manual type validation, manual/automatic type conversion)

This section demonstrates basic variable typing (including manual type validation and manual/automatic type conversion). In this section you will:

- Delete previous classes / create class **ZyxPerson**, variable **ZyxPerson>>id**.
- Type **ZyxPerson>>id** as **Integer**.
- Analyze **accessors** for the typed variable.
- Test with **no validation** (and **no conversion**)
- Test **manual validation** (message **isValid**) using **MicFwTypeConverter**.
- Test **manual validation** (message **isValid**) using **MicFwEmptyTypeConverter**.
- Test **manual conversion** (message **convertValues**) using **MicFwTypeConverter**
- Test **manual conversion** (message **convertValues**) using **MicFwEmptyTypeConverter**.
- Specify **automatic conversion** (checkbox **validateType**)
- Analyze **accessors** for the **automatic conversion**
- Test **automatic conversion** using **MicFwTypeConverter**.
- Test **automatic conversion** using **MicFwEmptyTypeConverter**.

### 2.10.1. Delete classes / Create ZyxPerson, ZyxPerson>>id

- 10.1. Delete all classes (ZyxClass1, ZyxClass11, ZyxClass2) from the model.
- 10.2. Add class **ZyxPerson** with parent class **MicFwDomainObject** and application **ZyxTutorial**.
- 10.3. Add variable **ZyxPerson>>id**.

### 2.10.2. Type ZyxPerson>>id as Integer

- 10.4. Select variable **ZyxPerson>>id**.
- 10.5. Select tab **Typing**.
- 10.6. From the drop-down list **Typing**: Select **Integer**.

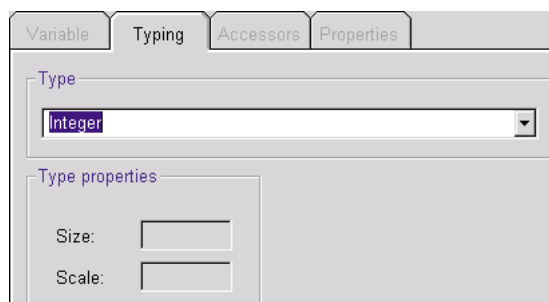


Figure 2.10.1. ZyxPerson>>id typed as Integer

Note: The **Size** and **Scale** fields can be left blank (default values are used).

- 10.7. Save the model.
- 10.8. Save the model to VA. Note: The classes deleted from the model are not deleted from VA.

### 2.10.3. Framework accessors for typed variable

10.9. In the **VisualAge Organizer**: Double click on class **ZyxPerson** (in application **ZyxTutorial**). The Script Editor on ZyxClass1 is opened. Note the following methods:

```
id
^self readAccessTo: #id currentValue: id.
id: anObject
self writeAccessTo: #id newValue: anObject currentValue: id.
```

The accessors are the same as previously.

### 2.10.4. Test with no validation

Assign Integer, String, Array to ZyxPerson>>id (no errors).

- 10.10. In the workspace **Execute** the following code:

```

OBF CH 2.10.#1. Assign Integer, String, Array to ZyxPerson>>id.
No errors."
[
ZyxPerson new
  id: 1234;          "(self/id = 1234)"
  inspect.
ZyxPerson new
  id: 'abcd';       "(self/id = 'abcd')"
  inspect.
ZyxPerson new
  id: #(1 2);       "(self/id = (1 2))"
  inspect
]

```

---

## 2.10.5. Test manual validation (message isTypeValid)

### 2.10.5.1. Assign Integer to ZyxPerson>>id (no error)

10.11. In the workspace **Display** the following code:

```

OBF CH 2.10.#2. Assign Integer to ZyxPerson>>id.
Manual validation / type valid."
[
ZyxPerson new
  id: 1234;
  isTypeValid.      "(true)"
]

```

### 2.10.5.2. Assign String to ZyxPerson>>id (no error)

A **String** can be assigned to ZyxPerson>>id.

10.12. In the workspace **Display** the following code:

```

OBF CH 2.10.#3. Assign String to ZyxPerson>>id.
Manual validation / type valid."
[
ZyxPerson new
  id: 'abcd';
  isTypeValid.      "(true)"
]

```

### 2.10.5.3. Assign Array to ZyxPerson>>id (no error; type invalid)

An **Array** can be assigned to ZyxPerson>>id. However, the type is not valid.

10.13. In the workspace **Display** the following code:

```

OBF CH 2.10.#4. Assign Array to ZyxPerson>>id.
Manual validation / type invalid."
[
ZyxPerson new
  id: #(1 2);
  isTypeValid.      "(false)"
]

```

---

## 2.10.6. Test manual conversion (message convertValues) using MicFwTypeConverter

The message convertValues will cause an object to attempt to convert any objects referenced by its typed variables into the type of objects specified by the variables' types.

10.14. MicFwTypeConverter is the default converter.



Figure 2.10.2. Default converter (MicFwTypeConverter) for ZyxPerson>>id



### 2.10.6.1. String (of digits)

A **String of digits** assigned to `ZyxPerson>>id` can be converted.

10.15. In the workspace **Inspect** the following code:

```
OBF CH 2.10.#5. Assign String (of integers) to ZyxPerson>>id and convertValues.  
No error."  
[  
  ZyxPerson new  
    id: '1234';  
    convertValues;  
    id  
      "(self = 1234)"  
]
```

### 2.10.6.2. String (of letters)

A **String of letters** assigned to `ZyxPerson>>id` can be converted. However, the resultant object is `Integer 0`.

10.16. In the workspace **Inspect** the following code:

```
OBF CH 2.10.#6. Assign String (of letters)to ZyxPerson>>id and convertValues.  
No error."  
[  
  ZyxPerson new  
    id: 'abcd';  
    convertValues;  
    id  
      "(self = 0)"  
]
```

### 2.10.6.3. Array (debugger)

A **Array** can be assigned to `ZyxPerson>>id`. However, the `convertValues` throws an exception.

10.17. In the workspace **Execute** the following code:

```
OBF CH 2.10.#7. Assign Array to ZyxPerson>>id and convertValues (exception).  
Debugger."  
[  
  ZyxPerson new  
    id: #(1 2);  
    convertValues.      "debugger: type error"  
]
```

The Debugger appears with the message `Type error - expected: Integer found: (1 2)`.

---

## 2.10.7. Test manual conversion (message `convertValues`) using `MicFwTypeConverter`

### 2.10.7.1. Change the type converter for `ZyxPerson>>id`

10.18. Select **ZyxPerson**.

10.19. Select **id**.

10.20. Select tab **Variable**.

10.21. From drop-down list **Type converter**: Select **MicFwEmptyTypeConverter**.

10.22. Save the model.

10.23. Save to VA.

### 2.10.7.2. String (of letters)

A **String of letters** assigned to `ZyxPerson>>id` cannot be converted by `MicFwEmptyTypeConverter`.

10.24. In the workspace **Execute** the following code:

```
OBF CH 2.10.#8. Assign String (of letters)to ZyxPerson>>id and convertValues.  
Exception (MicFwEmptyTypeConverter cannot convert)."  
[  
  ZyxPerson new  
    id: 'abcd';  
    convertValues "debugger"  
]
```

The Debugger appears with the message `Type error - expected: Integer found: "abcd"`.

---

## 2.10.8. Specify automatic conversion (checkbox validateType)

- 10.25. Select **ZyxPerson**.
- 10.26. Select **id**.
- 10.27. Select tab **Variable**.
- 10.28. Check checkbox **Validate type**.
- 10.29. Save the model.
- 10.30. Save to VA.

---

## 2.10.9. Framework accessors for automatic conversion

10.31. In the **VisualAge Organizer**: Double click on class **ZyxPerson** (in application **ZyxTutorial**). The Script Editor on **ZyxClass1** is opened. Note the following methods:

```
id
  ^self readAccessTo: #id currentValue: id.
id: anObject
  self validatedWriteAccessTo: #id newValue: anObject currentValue: id.
```

The read accessor is unchanged. The write accessor sends the new message `validatedWriteAccessTo:newValue:currentValue:`.

---

## 2.10.10. Test automatic conversion using MicFwTypeConverter

- 10.32. Change the **Type converter** for **ZyxPerson>>id** to **MicFwTypeConverter**.
- 10.33. Save to VA.

### 2.10.10.1. Assign String to ZyxPerson>>id (no error; auto-conversion to Integer)

10.34. In the workspace **Inspect** the following code:

```
OBF CH 2.10.#9. Assign String to ZyxPerson>>id.
Auto-conversion / no error."
[
ZyxPerson new
  id: 'abcd'          "(self / id = 0)"
]
```

### 2.10.10.2. Assign Array to ZyxPerson>>id (error)

Assigning an **Array** causes an exception.

10.35. In the workspace **Execute** the following code:

```
OBF CH 2.10.#10. Assign Array to ZyxPerson>>id.
Auto-conversion / exception."
[
ZyxPerson new
  id: #(1 2)
]
```

The Debugger appears with the message `Type error - expected: Integer found: (1 2)`.

---

## 2.10.11. Test automatic conversion using MicFwEmptyTypeConverter

- 10.36. Change the **Type converter** for **ZyxPerson>>id** to **MicFwEmptyTypeConverter**.
- 10.37. Save to VA.

### 2.10.11.1. Assign String to ZyxPerson>>id (no error; auto-conversion to Integer)

10.38. In the workspace **Inspect** the following code:

```
OBF CH 2.10.#11. Assign String to ZyxPerson>>id.
Auto-conversion with MicFwEmptyTypeConverter / exception."
[
ZyxPerson new
  id: 'abcd'          "debugger"
]
```

The Debugger appears with the message `Type error - expected: Integer found: "abcd"`.

---

## 2.11. Variable typing (Date)

This section again demonstrates basic variable typing. In this section you will:

- Add variable **ZyxPerson>>dateOfBirth** with type **Date**.
- Test (using **obf\_ex.txt** examples).

---

### 2.11.1. Create ZyxPerson>>dateOfBirth with type Date

11.1. Add variable **ZyxPerson>>dateOfBirth**.

11.2. Set the **Type** to **Date**.

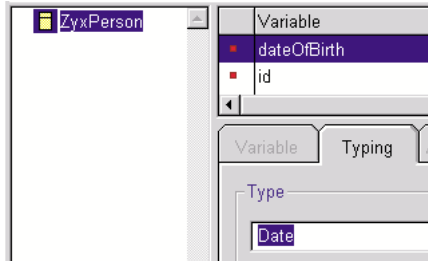


Figure 2.11.1. ZyxPerson>>dateOfBirth with type Date

11.3. Save the model.

11.4. Save to VA.

---

### 2.11.2. Test

#### 2.11.2.1. Assign String to ZyxPerson>>dateOfBirth (no error)

**ZyxPerson>>dateOfBirth** is typed as **Date**. However, **Validate Type** was not checked in the ONB. Therefore, any type of object can be assigned to **ZyxPerson>>dateOfBirth**.

11.5. In the workspace execute the following code:

```
OBF CH 2.11.#1. Assign String (of chars) to Date variable.
No validation.
No conversion / No error / Type is NOT valid."
[
ZyxPerson new
  dateOfBirth: '1 Jan 2000';
  inspect;           "(dateOfBirth = '1 Jan 2000')"
  isTypeValid.      "(false)"
]
```

#### 2.11.2.2. Assign String (of characters representing invalid Date) to ZyxPerson>>id and convertValues

In the following example, a **String** object (containing characters NOT representing a valid **Date**) is assigned to **ZyxPerson>>dateOfBirth** (without validation). Sending the **convertValues** message to the **ZyxPerson** object throws an exception.

11.6. In the workspace execute the following code:

```
OBF CH 2.11.#2. Assign String (containing invalid Date) to Date variable.
No validation.
Attempt to convert with convertValues: Opens debugger."
[
ZyxPerson new
  dateOfBirth: '1234';
  inspect;           "(self/dateOfBirth = '1234')"
  convertValues.    "(debugger: type error)"
]
```

#### 2.11.2.3. Assign String (of characters representing valid Date) to ZyxPerson>>id and convertValues

In the following example, a **String** object (containing characters representing a valid **Date**) is assigned to **ZyxPerson>>dateOfBirth** (without validation). Sending the **convertValues** message to the **ZyxPerson**

object causes a Date object to be created from the String object and assigned to the variable.

11.7. In the workspace execute the following code:

OBF CH 2.11.#3. Assign String (containing valid Date) to Date variable.

No validation.

Convert with convertValues.

Conversion / no error / type valid."

```
[
ZyxPerson new
  dateOfBirth: '1 Jan 2000';
  convertValues;
  inspect;           "(self/dateOfBirth = 01.01.00)"
  isValid.          "(true)"
]
```

#### 2.11.2.4. Assign String (containing valide date) to ZyxPerson>>dateOfBirth with Validation

11.8. Enable **Validate** for **ZyxPerson>>dateOfBirth** (do not forget to save the class).

In the following example, a String object (containing characters representing a valid Date) is assigned to **ZyxPerson>>dateOfBirth** (with validation).

11.9. In the workspace execute the following code:

OBF CH 2.11.#4. Assign String (containg valid Date) to Date variable.

Validation.

Conversion.

No error."

```
[
ZyxPerson new
  dateOfBirth: '1 Jan 2000';
  inspect;           "(dateOfBirth = 01.01.00)"
  isValid.          "(true)"
]
```

---

## 2.12. Persistent / Key variables

In this section you will:

- Specify a variable as being persistent
- Analyze the Framework accessors for the variable
- Specify a variable as being a key variable
- Analyze how persistent and key variables can be used in the Persistence Framework

---

### 2.12.1. Specify ZyxPerson>>id as persistent

- 12.1. Select **ZyxPerson>>id**.
- 12.2. Select tab **Properties**.
- 12.3. Click in row **persistent** column **Value**. Radio buttons appear:



Figure 2.12.1. Radio buttons for Key property for ZyxPerson>>id

- 12.4. Select **True**.
- 12.5. Click to change the focus. Note that Value for key is now "True".
- 12.6. Save the model.
- 12.7. Save to VA.

---

### 2.12.2. Framework accessors for Persistent variable

12.8. In the **VisualAge Organizer**: Double click on class **ZyxPerson** (in application **ZyxTutorial**). The Script Editor on ZyxPerson is opened. Note the following methods:

```
id
  ^self readAccessTo: #id currentValue: id.
id: anObject
  self validatedWriteAccessTo: #id newValue: anObject currentValue: id.
```

The methods are unchanged. However, internally support for persistence has now been added.

---

### 2.12.3. Specify ZyxPerson>>id as a key variable

- 13.1. Select **ZyxPerson>>id**.
- 13.2. Select tab **Properties**.
- 13.3. Click in row **key** column **Value**. Radio buttons appear.
- 13.4. Select **True**.
- 13.5. Click to change the focus. Note that Value for key is now "True".
- 13.6. Save the model.
- 13.7. Save to VA.

---

### 2.12.4. Specify ZyxPerson>>dateOfBirth as a persistent variable

- 14.1. Select **ZyxPerson>>dateOfBirth**.
- 14.2. Select tab **Properties**.
- 14.3. Click in row **persistent** column **Value**. Radio buttons appear.
- 14.4. Select **True**.
- 14.5. Click to change the focus. Note that Value for persistent is now "True".
- 14.6. Save the model.
- 14.7. Save to VA.

---

### 2.12.5. Application: Key variable for persistence object

You now know how to specify a variable as type Integer. One possible application for such functionality is creating a key variable for a persistent object.

Persistent objects are objects that are stored in a database. Database columns require stringent typing. Therefore, since the OBF ensures that `ZyxPerson>>id` always references an Integer object, `ZyxPerson>>id` can easily be made persistent.

By specifying `ZyxPerson>>id` as a Key variable, the Integer object referenced by a `ZyxPerson` instance could be used as a **Primary Key** for identifying the `ZyxPerson` instance in the table (each instance would be stored in a row in the table).

The following diagram shows how a persistent Smalltalk object (a `ZyxPerson` instance) and an entry in a database table would be related.

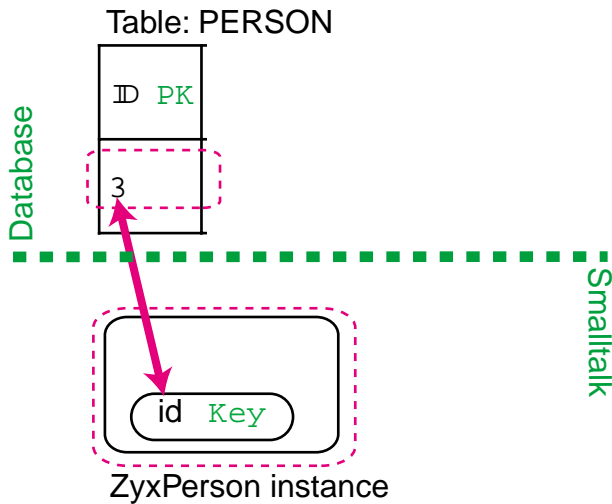


Figure 2.12.2. Persistent variable `ZyxPerson>>id` and entry in database table PERSON

**Note:** Detailed information about key variables and persistent objects are outside the scope of this tutorial. For an introduction, see the *Frameworks Getting Started Manual*. For detailed information, consult the *Persistence Framework User's Guide*.

### 2.12.6. Application: Date variable for persistence object

You now know how to specify a variable as type Date. One possible application for such functionality is for a persistence object.

The following diagram shows how a persistent Smalltalk object (a `ZyxPerson` instance with variables `id` and `dateOfBirth`) and an entry in a database table would be related.

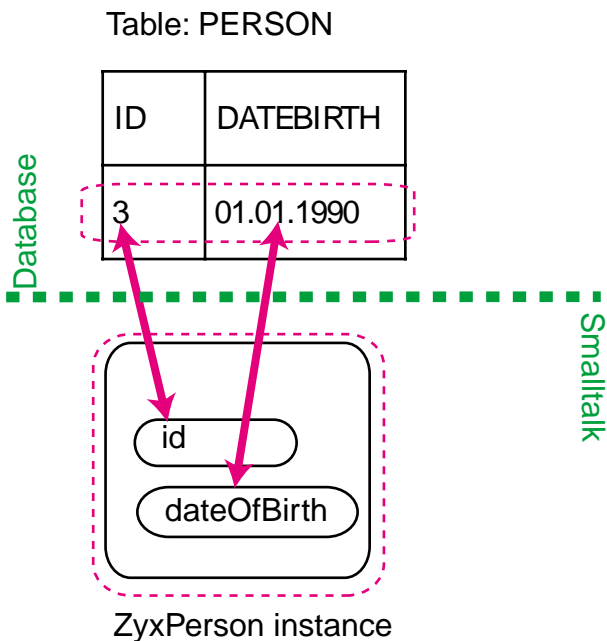


Figure 2.12.3. Persistent variable `ZyxPerson>>dateOfBirth` and entry in database table PERSON column DATEBIRTH

## 2.13. ->1 relationship

This section demonstrates how to create a ->1 relationship between 2 classes.

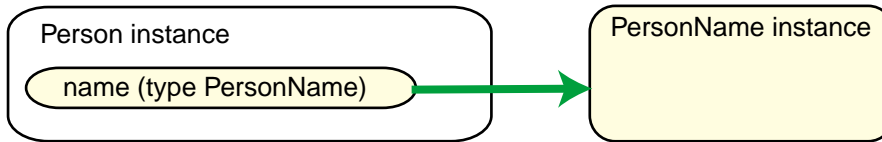


Figure 2.13.1. ->1 relationship

In this section you will:

- Create **MicFwDomainObject** subclass **ZyxName**, **ZyxName>>firstName**, **ZyxName>>lastName**.
- Add variable **ZyxPerson>>name**.
- Establish a ->1 (primitive) relationship from **ZyxPerson>>name** to 1..1 **ZyxName** objects
- Test.
- Analyze a possible application.

### 2.13.1. Create MicFwDomainObject subclass ZyxName with variables firstName, lastName

15.1. In **OMB**: Create **MicFwDomainObject** subclass **ZyxName**.

15.2. Add **ZyxName>>firstName** with type **String**.

15.3. Enter for **Size** value **50**. Note: The focus must be changed before the entry field is available. Click on a different tab to change the focus.

15.4. Add **ZyxName>>lastName** with type **String** (size 50).

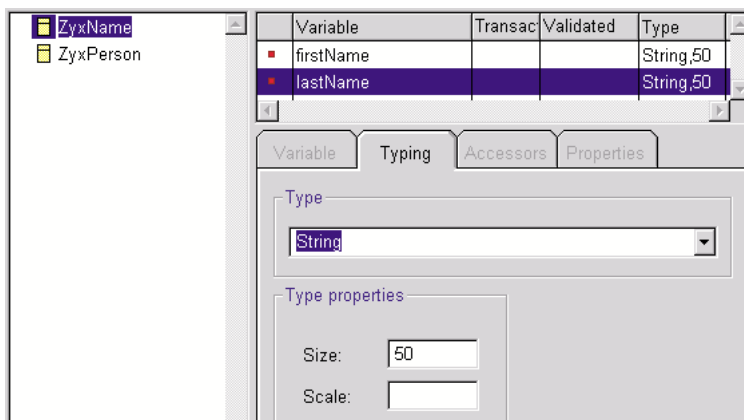


Figure 2.13.2. Defining ZyxName variables in OMB

### 2.13.2. Add variable ZyxPerson>>name

15.5. Add variable **ZyxPerson>>name**.

### 2.13.3. Establish a ->1 (primitive) relationship from ZyxPerson>>name to 1..1 ZyxName objects

The relationship will be specified as ->1. This means that the source object has a variable that references the target object, but the target object has no variable that references the source object. This type of relationship is *primitive*.

#### 2.13.3.1. Specify for source variable: relationship type

15.6. Select **name**.

15.7. Select tab **Variable**.

15.8. Check the checkbox **Relationship**.

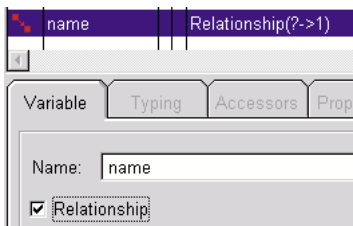


Figure 2.13.3. Checkbox "Relationship" in the OMB

15.9. Select tab **Typing**. Note that the contents of the tab have changed.

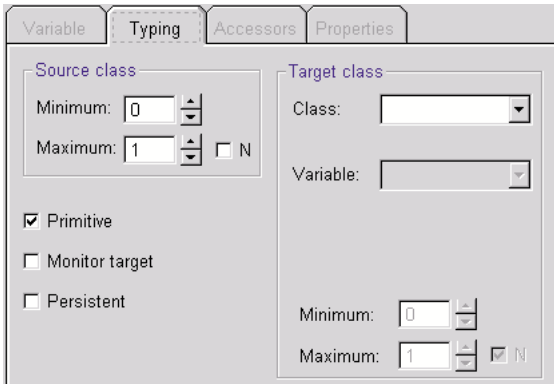


Figure 2.13.4. Tab "Typing" for a relationship

15.10. Check the checkbox **Primitive** (this checkbox is already checked by default). This specifies an **->N** relationship.

### 2.13.3.2. Specify for source variable: target class

15.11. From the **Target class** drop-down list **Class**: Select **ZyxName**.

### 2.13.3.3. Specify for source variable: # of allowed target classes

An **->N** relationship can be further specified as a **->1** relationship by setting the cardinality of the relationship to **1..1**.

15.12. Set the **Source class minimum** (of referenced target classes) to 1. This means that the variable "name" for each instance of **ZyxPerson** should reference minimum 1 instance of **ZyxName**.

15.13. Set the **Source class maximum** (of referenced target classes) to 1. This means that the variable "name" for each instance of **ZyxPerson** should reference maximum 1 instance of **ZyxName**.

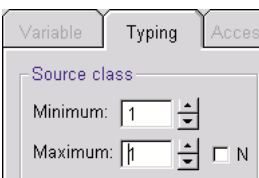


Figure 2.13.5. Source class minimum/maximum (# of allowed target classes)

15.14. Save the model.

15.15. Save to VA.

## 2.13.4. Test

### 2.13.4.1. Assign Integer to **ZyxPerson**>>name (no error)

**Validate Type** was not checked in the ONB for **ZyxPerson**>>name. Therefore, any type of object can be assigned to **ZyxPerson**>>name.

15.16. In the workspace **Display** the following code:

```
OBF CH 2.13.#1. Assign Integer to source variable of ->1 relationship.
```

```
No validation.
```

```
No error."
```

```
[
```

```
ZyxPerson new
```

```
  name: 5678;
```



```

inspect;           "self/name/target = 5678"
isTypeValid.      "(true)"
]

```

### 2.13.4.2. Assign objects to ZyxPerson>>name with Validation enabled

15.17. Check the checkbox **Validate type** for **ZyxPerson>>name**.

15.18. Save the model.

15.19. Save to VA.

#### 2.13.4.2.1. Assign Integer to ZyxPerson>>name with Validation

In the following example, an Integer object is assigned to **ZyxPerson>>name** (with validation). An exception is thrown.

15.20. In the workspace execute the following code:

```

OBF CH 2.13.#2. Assign Integer to source variable of ->1 relationship.
Validation.
Debugger."
[
ZyxPerson new
  name: 5678. "debugger: 'Type error - expected: ZyxName'."
]

```

#### 2.13.4.2.2. Assign ZyxName instance to ZyxPerson>>name with Validation

In the following example, a ZyxName instance is assigned to **ZyxPerson>>name** (with validation).

15.21. In the workspace **Execute** the following code:

```

OBF CH 2.13.#3. Assign ZyxName to source variable of ->1 relationship.
Validation.
No error."
[
ZyxPerson new
  name: ZyxName new;
inspect.           "self id target = a ZyxName"
]

```

## 2.13.5. Application: ->1 relationship

You now know how to specify a variable as a source variable for a ->1 relationship. One possible application for such functionality is for a persistence object.

The following diagram shows how 2 persistent Smalltalk objects (->1 relationship source ZyxPerson instance with variables id and name, and ->1 relationship target ZyxName with variables firstName and lastName) and an entry in a database table would be related.

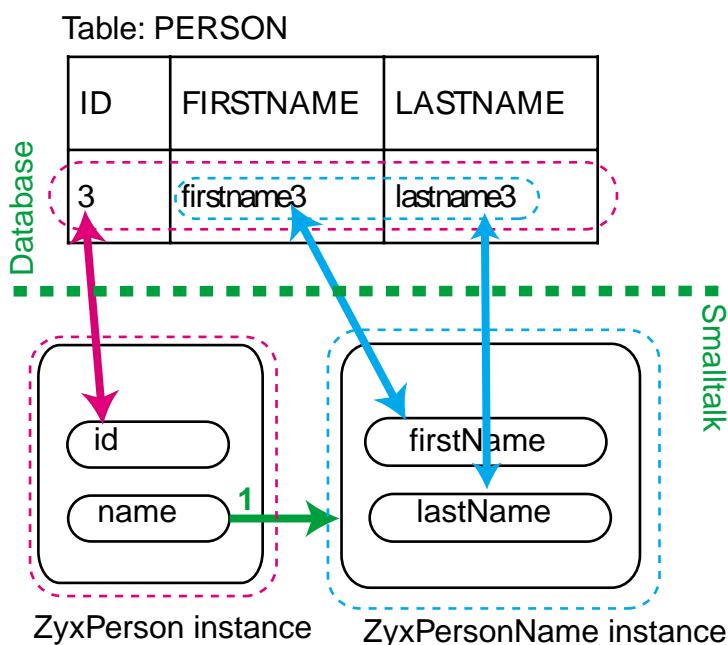


Figure 2.13.6. ->1 relationship source ZyxPerson>>name and target ZyxName in database table PERSON

Note that on the table side that ZyxName does not have its own table. For the table, only the simple variable types (Integer, String, etc) that are recognized by the database have meaning.

Note that the context for both the ZyxPerson and ZyxName objects are loaded from the same table. ->1 relationships are typically used when information should be stored in a single table but represented by multiple objects.

---

## 2.14. Adding class nets to the model

This section demonstrates how to add class nets to the model.

A **class net** is a set of all classes that are related to each other in one of the following ways:

- Parent / child class (parent up to and not including MicFwDomainObject).
- A class references another class via a relationship.

Note: A class net is determined starting with the specified class. If the net includes ClassB that is referenced in a primitive relationship by ClassA (ie, ClassA is not referenced by any variable in ClassB), then ClassA is NOT added to the class net (at least not because of the ClassA variable that references ClassB).

In this section you will:

- Create **ZyxName** subclass **ZyxNameSubclass**.
- Add **ZyxNameSubclass** class net to the model
- Add **ZyxName** class net to the model
- Add **ZyxPerson** class net to the model

---

### 2.14.1. Create ZyxName subclass ZyxNameSubclass; Save to VA

16.1. Create **ZyxName** subclass **ZyxNameSubclass**.

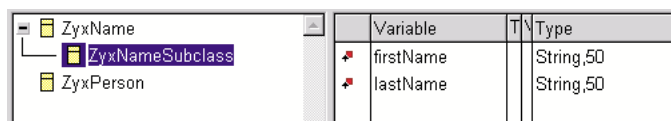


Figure 2.14.1. ZyxNameSubclass in OMB

16.2. Save to VA.

---

### 2.14.2. Add ZyxNameSubclass class net to model

16.3. Delete all classes in the model.

16.4. Save the model.

16.5. Select **Class / Add class net....** The message **Select root class** appears with default "".

16.6. Click **OK**.

16.7. From the list of classes: Double-click on **ZyxNameSubclass**. The following classes are added to the model:

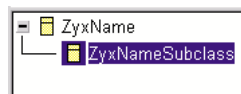


Figure 2.14.2. Object net for ZyxNameSubclass

---

### 2.14.3. Add ZyxName class net to model

16.8. Delete all classes in the model.

16.9. Save the model.

16.10. Select **Class / Add class net....** The message **Select root class** appears with default "".

16.11. Click **OK**.

16.12. From the list of classes: Double-click on **ZyxName**. The following classes are added to the model:

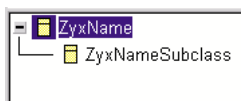


Figure 2.14.3. Object net for ZyxName

---

### 2.14.4. Add ZyxPerson class net to model

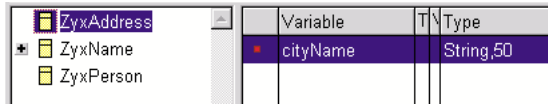
16.13. Delete all classes in the model.

16.14. Save the model.

16.15. Select **Class / Add class net....** The message **Select root class** appears with default "".

16.16. Click **OK**.

16.17. From the list of classes: Double-click on **ZyxPerson**. The following classes are added to the model:



| Variable | T | Type      |
|----------|---|-----------|
| cityName |   | String,50 |

Figure 2.14.4. Object net for ZyxPerson

## 2.15. ->N relationship

This section demonstrates how to create a ->N relationship between 2 classes. The relationship differs from that in the previous section in that the cardinality maximum (max number of Target objects) will be > 1.

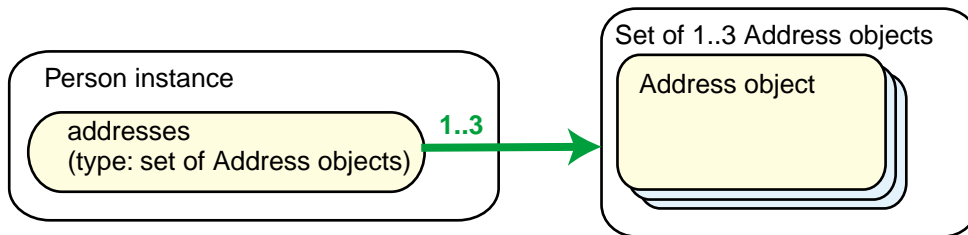


Figure 2.15.1. ->N relationship

In this section you will:

- Create **MicFwDomainObject** subclass **ZyxAddress**, **ZyxAddress>>cityName**.
- Add variable **ZyxPerson>>addresses**.
- Establish ->N (primitive) relationship from **ZyxPerson>>addresses** to **1..3 ZyxAddress** objects
- Test.
- Analyze a possible application.

### 2.15.1. Create MicFwDomainObject subclass ZyxAddress with variable cityName

17.1. In **OMB**: Create **MicFwDomainObject** subclass **ZyxAddress**.

17.2. Add **ZyxAddress>>cityName** with type **String**.

17.3. Enter for **Size** value **50**. Note: The focus must be changed before the entry field is available. Click on a different tab to change the focus.

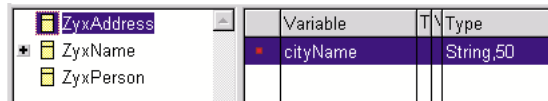


Figure 2.15.2. Defining ZyxAddress variables in OMB

### 2.15.2. Add variable ZyxPerson>>addresses

17.4. Add variable **ZyxPerson>>addresses**.

### 2.15.3. Establish a ->N (primitive) relationship from ZyxPerson>>addresses to 1..3 ZyxAddress objects

The relationship will be specified as ->N. This means that the source object has a variable that references the target objects, but a target object has no variable that references the source object. This type of relationship is **primitive**.

#### 2.15.3.1. Specify for source variable addresses: relationship type

17.5. Select **addresses**.

17.6. Select tab **Variable**.

17.7. Check the checkbox **Relationship**.

17.8. Select tab **Typing**. Note that the contents of the tab have changed.

17.9. Check the checkbox **Primitive** (this checkbox is already checked by default). This specifies an ->N relationship.

#### 2.15.3.2. Specify for source variable: target class

17.10. From the **Target class** drop-down list **Class**: Select **ZyxAddress**.

#### 2.15.3.3. Specify for source variable: # of allowed target classes

17.11. Set the **Source class minimum** (of referenced target classes) to 1. This means that the variable "addresses" for each instance of **ZyxPerson** should reference minimum 1 instance of **ZyxAddress**.

17.12. Set the **Source class maximum** (of referenced target classes) to 3. This means that the variable

"addresses" for each instance of ZyxPerson should reference maximum 3 instances of ZyxAddress.

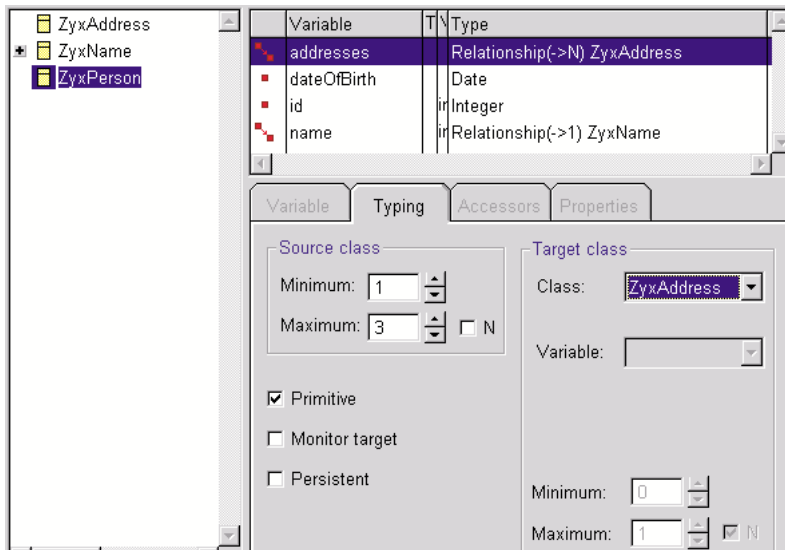


Figure 2.15.3. Source class minimum/maximum (# of allowed target classes)

17.13. Save the model.

17.14. Save to VA.

## 2.15.4. Test

### 2.15.4.1. No validation

17.15. Assign **ZyxPerson>>addresses** an Integer object with **Validate Type** disabled. No error occurs.

OBF CH 2.15.#1. Assign Integer to variable of type Set(1..3) of ZyxAddress.  
No validation.

No error / type valid."

```
[
| aPerson |
aPerson := ZyxPerson new.
aPerson addresses add: 1234.
aPerson inspect.           "self/addresses/target = IdentitySet(1234)"
aPerson isValid.          "(true)"
]
```

17.16. Assign **ZyxPerson>>addresses** 4 objects with **Validate Type** disabled using the **add:** message (the standard Frameworks message for adding a member to a relationship set). Note that with **ValidateType** disabled no exception is thrown, although the relationship for **ZyxPerson>>addresses** specifies a max of 3 objects.

OBF CH 2.15.#2. Assign 4 objects to variable of type Set(1..3) of ZyxAddress.  
No validation.

No error / Type valid."

```
[
| aPerson |
aPerson := ZyxPerson new.
aPerson addresses add: 12.
aPerson addresses add: #($a $b).
aPerson addresses add: 56.
aPerson addresses add: 78.
aPerson inspect.           "self/addresses/target = IdentitySet(12 ($a $b) 56 78)"
aPerson isValid.          "(true)"
]
```

### 2.15.4.2. Validation

17.17. Enable **Validate** for **ZyxPerson>>addresses**.

17.18. Save the model.

17.19. Save to VA.

17.20. Assign **ZyxPerson>>addresses** an Integer object. Exception thrown.

OBF CH 2.15.#3. Assign Integer to variable of type Set(1..3) of ZyxAddress.  
Validation.

Exception."

```
[
| aPerson |
  aPerson := ZyxPerson new.
  aPerson addresses add: 1234.
  aPerson inspect.      "exception: type error"
]
```

#### 17.21. Assign **ZyxPerson>>addresses** 1 **ZyxAddress** object.

OBF CH 2.15.#4. Assign 1 ZyxAddress to variable type Set(1..3) of ZyxAddress.  
Validation.

No error / Type valid."

```
[
| aPerson |
aPerson := ZyxPerson new.
  aPerson addresses add: ZyxAddress new.
  aPerson inspect.      "self/addresses/target = IdentitySet(a ZyxAddress)"
  aPerson isTypeValid. "(true)"
]
```

#### 17.22. Assign **ZyxPerson>>addresses** 3 **ZyxAddress** objects with **Validate Type**.

OBF CH 2.15.#5. Assign 3 ZyxAddress to variable type Set(1..3) of ZyxAddress.  
Validation.

No error / Type valid."

```
[
| aPerson |
aPerson := ZyxPerson new.
  aPerson addresses add: ZyxAddress new.
  aPerson addresses add: ZyxAddress new.
  aPerson addresses add: ZyxAddress new.
  aPerson inspect.      "self/addresses/target =
                        IdentitySet(a ZyxAddress a ZyxAddress a ZyxAddress)"
  aPerson isTypeValid. "(true)"
]
```

#### 17.23. Assign **ZyxPerson>>addresses** 4 **ZyxAddress** objects. A cardinality exception is thrown.

OBF CH 2.15.#6. Assign 4 ZyxAddress to variable type Set(1..3) of ZyxAddress.  
Validation.

Cardinality exception.

```
[
| aPerson |
aPerson := ZyxPerson new.
  aPerson addresses add: ZyxAddress new.
  aPerson addresses add: ZyxAddress new.
  aPerson addresses add: ZyxAddress new.
  aPerson addresses add: ZyxAddress new. "exception: cardinality violation"
]
```

---

### 2.15.5. Review how the OBF classes and methods you created can be used by PFW.

->N relationships are required by PFW for mapping objects to tables.

The accessors created for ZyxPerson>>address specify:

- The type for the variable is ZyxAddress.
- A ->(1..3) relationship specifies that ZyxPerson>>address references 1..3 ZyxAdderss objects.

The PFW can now use these classes to make the object persistent. The following diagram shows how the

Smalltalk objects and the database tables would be related.

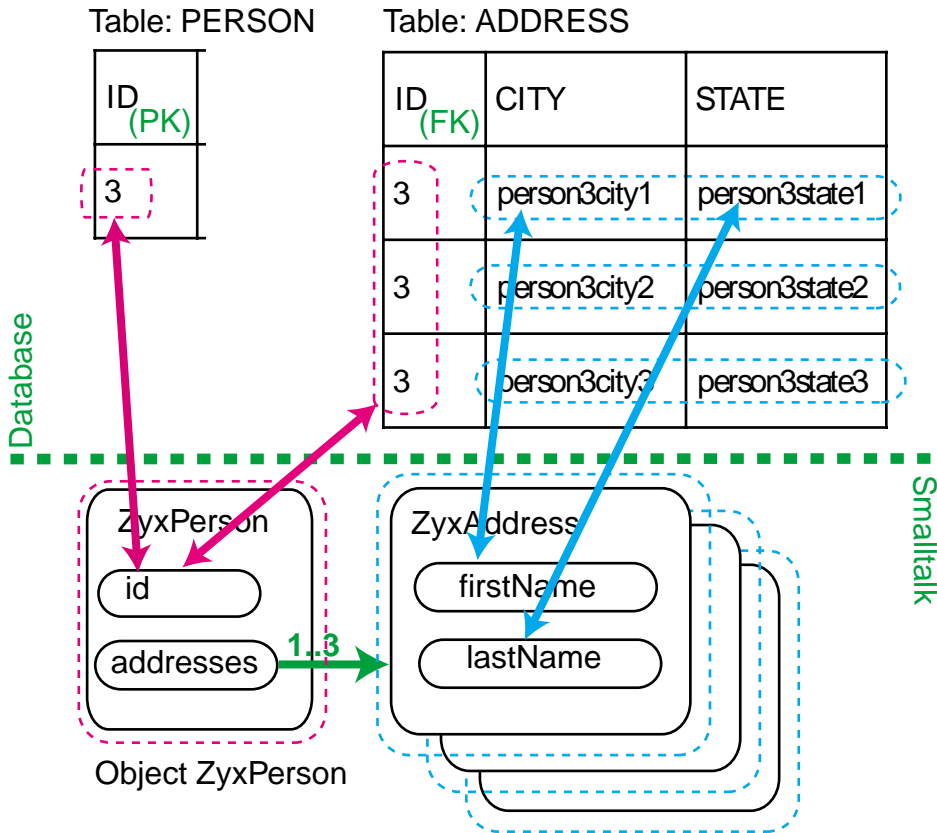


Figure 2.15.4. The ->N relationship between ZyxPerson and ZyxAddress in table ADDRESS

Note that on the table side the type ZyxAddress is stored as a separate object. The primary key of the PERSON table is the Foreign Key of the ADDRESS table.



## 2.16. 1<->1 relationship

This section demonstrates how to implement a 1<->1 relationship between 2 classes.

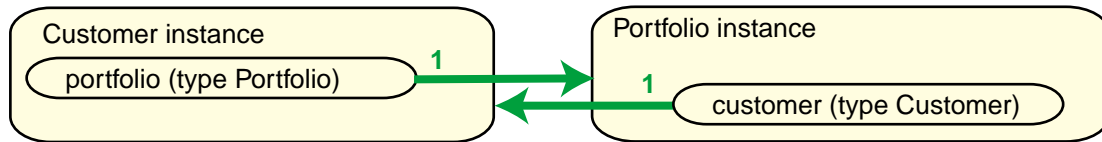


Figure 2.16.1. 1<->1 relationship

In this section you will:

- Create classes and variables (create **ZyxCustomer**, **ZyxCustomer>>portfolio**, **ZyxPortfolio**, **ZyxPortfolio>>customer**).
- Set **ZyxCustomer>>portfolio** type to **ZyxPortfolio** and **ZyxPortfolio>>customer** type to **ZyxCustomer**.
- Set cardinality of **ZyxCustomer>>portfolio** to 1..1 and set cardinality of **ZyxPortfolio>>customer** to 1..1.
- Review how the OBF classes and methods you created might be used by PFW.

### 2.16.1. Create classes and variables.

- 18.1. Create **ZyxPerson** subclass **ZyxCustomer**.
- 18.2. Create **ZyxCustomer>>portfolio**.
- 18.3. Create **MicFwDomainObject** subclass **ZyxPortfolio**.
- 18.4. Create **ZyxPortfolio>>customer**.

### 2.16.2. Set **ZyxCustomer>>portfolio** type as 1<->1 relationship

- 18.5. Specify **ZyxCustomer>>portfolio** as **Relationship** (checkbox **Relationship**).
- 18.6. Specify **Source class minimum...maximum** as 1..1.
- 18.7. Specify **Target class** as **ZyxPortfolio**.
- 18.8. Specify **variable** as **Customer**.
- 18.9. Specify **Target class minimum...maximum** as 1..1.

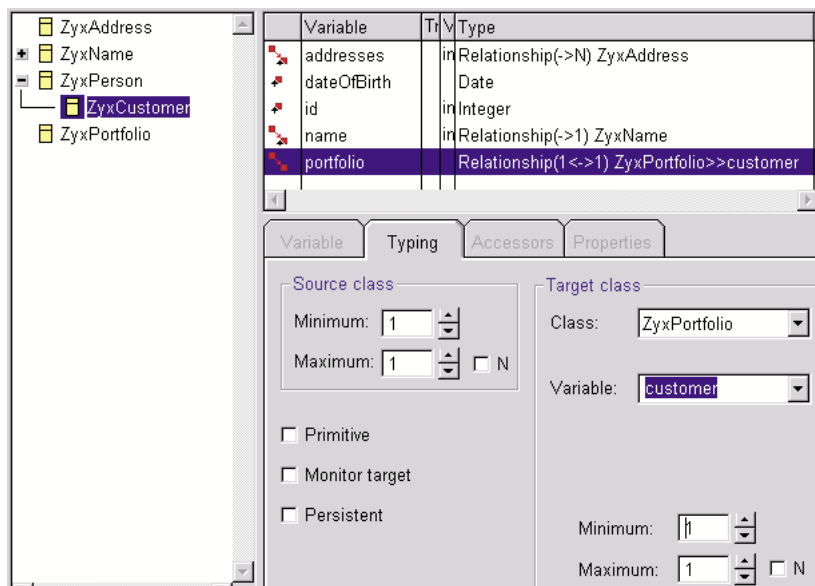


Figure 2.16.2. **ZyxCustomer>>portfolio** definition

18.10. Display tab **Typing** for **ZyxPortfolio>>customer**.

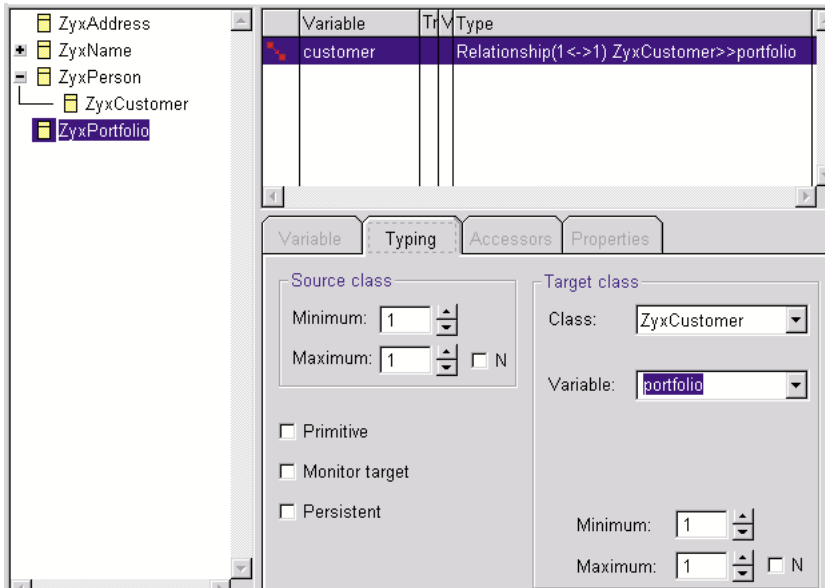


Figure 2.16.3. *ZyxCustomer>>portfolio* definition

- 18.11. Save model.
- 18.12. Save to VA.

**2.16.3. Review how the OBF classes and methods you created can be used by PFW.**

1<->1 relationships are required by PFW for mapping objects to tables.

The following diagram shows how the Smalltalk objects instantiated from the classes you just created and the database tables would be related.

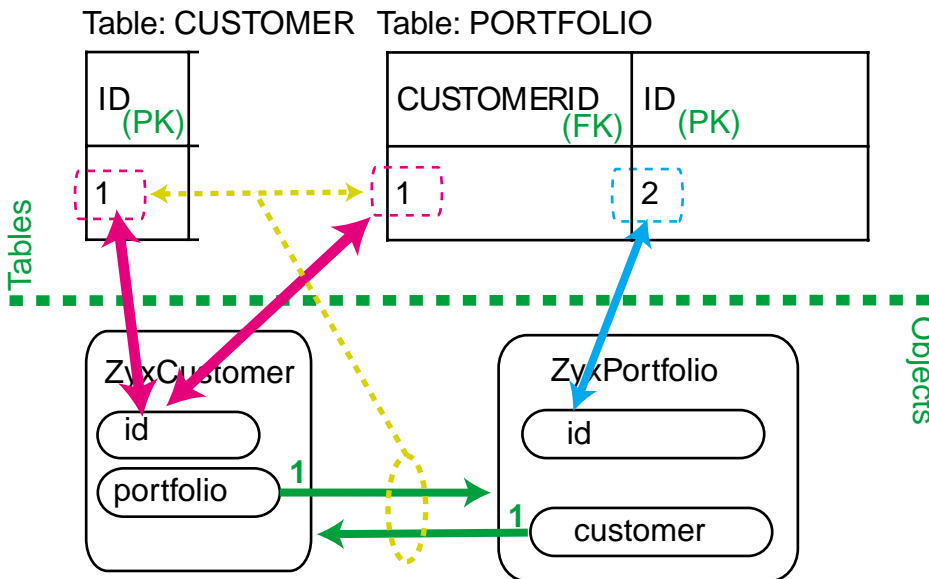


Figure 2.16.4. 1<->1 relationship between *ZyxCustomer* and *ZyxPortfolio* in tables *CUSTOMER* and *PORTFOLIO*

Note that table *PORTFOLIO* has the the FK since *ZyxPortfolio* was specified as the target object.  
 Note that the connection between objects is represented in the database by the *CUSTOMER* PK and the *PORTFOLIO* FK match.

## 2.17. 1<->N relationship

This chapter demonstrates how to implement a 1<->N relationship between 2 classes.

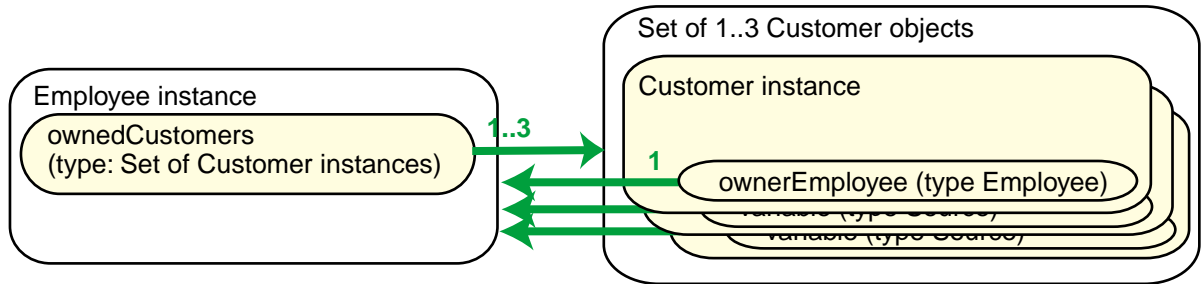


Figure 2.17.1. 1<->N relationship

In this section you will:

- Create classes and variables (create **ZyxEmployee**, **ZyxEmployee>>ownedCustomers**, **ZyxCustomer>>ownerEmployee**).
- Set **ZyxEmployee>>ownedCustomers** type to **ZyxCustomer** and **ZyxCustomer>>ownerEmployee** type to **ZyxEmployee**.
- Set cardinality of **ZyxEmployee>>ownedCustomers** to 1..3 and set cardinality of **ZyxCustomer>>ownerEmployee** to 1..1.
- Review how the OBF classes and methods you created might be used by PFW.

### 2.17.1. Create classes and variables.

- 19.1. Create **ZyxPerson** subclass **ZyxEmployee**.
- 19.2. Create **ZyxEmployee>>ownedCustomers**.
- 19.3. Create **ZyxCustomer>>ownerEmployee**.

### 2.17.2. Set **ZyxEmployee>>ownedCustomers** type as 1<->N relationship

- 19.4. Specify **ZyxEmployee>>ownedCustomers** as **Relationship** (checkbox **Relationship**).
- 19.5. Specify **Source class minimum...maximum** as 1..3.
- 19.6. Specify **Target class** as **ZyxCustomer**.
- 19.7. Specify **variable** as **ownerEmployee**.
- 19.8. Specify **Target class minimum...maximum** as 1..1.

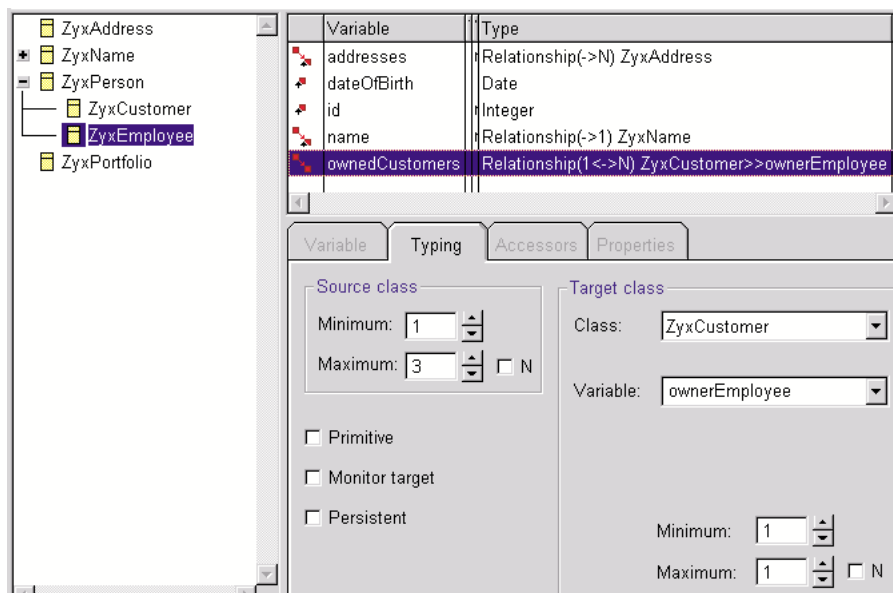


Figure 2.17.2. **ZyxEmployee>>ownedCustomers** definition

19.9. Display tab **Typing** for **ZyxCustomer>>ownerEmployee**.

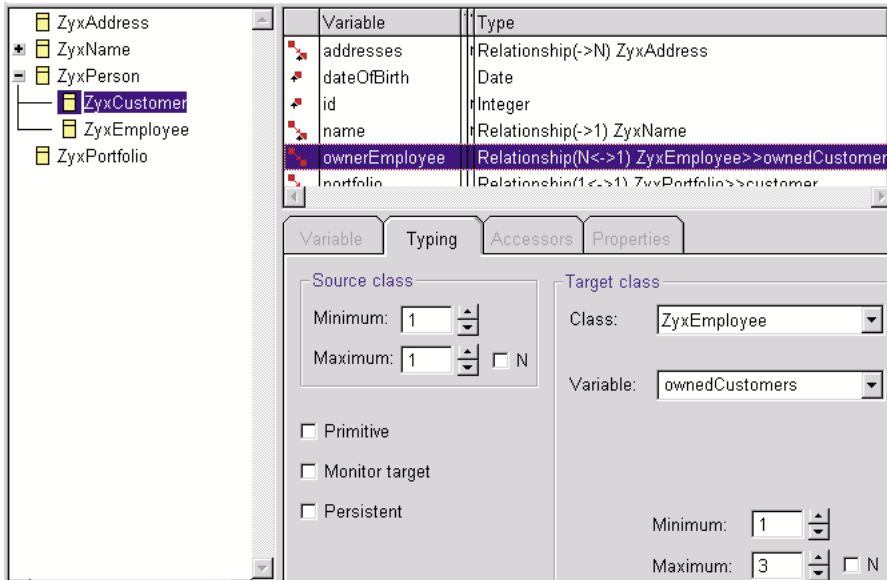


Figure 2.17.3. *ZyxCustomer>>ownderEmployee* definition

- 19.10. Save model.
- 19.11. Save to VA.

**2.17.3. Review how the OBF classes and methods you created can be used by PFW.**

The following diagram shows how the Smalltalk objects instantiated from the classes you just created and the database tables would be related.

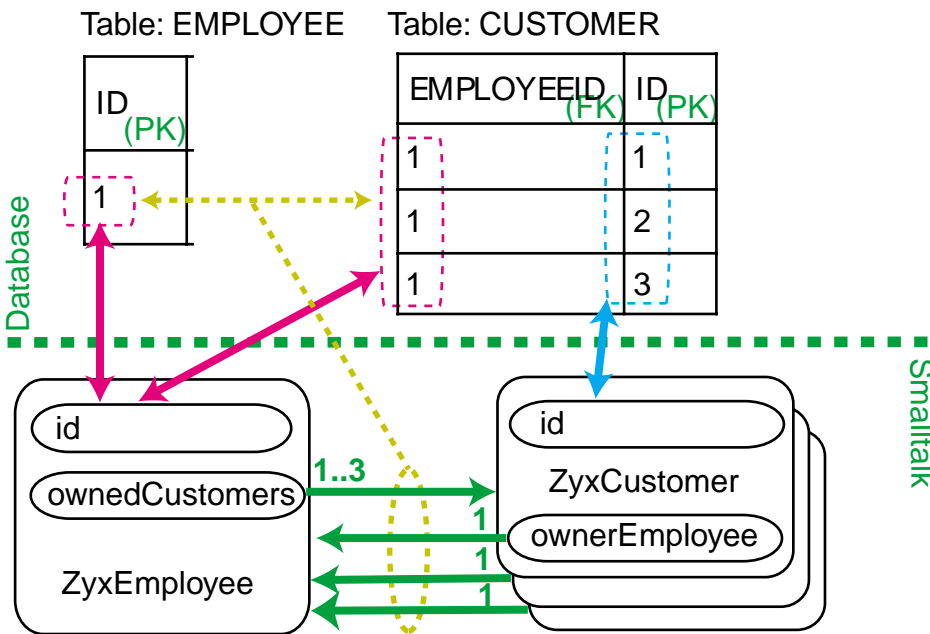


Figure 2.17.4. 1<->N relationship between *ZyxCustomer* *ZyxEmployee* in tables *EMPLOYEE* and *CUSTOMER*

Note that table *CUSTOMER* has the the FK since *ZyxCustomer* was specified as the target object.  
 Note that the connection between objects is represented in the database by the *EMPLOYEE* PK and the *CUSTOMER* FK match.

## 2.18. M<->N relationship

In this chapter you will create the following M<->N relationship.

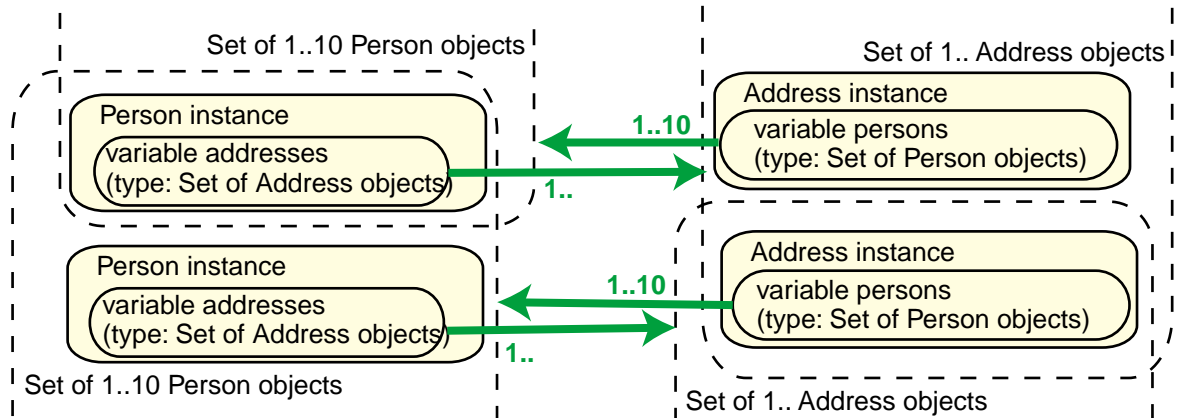


Figure 2.18.1. M<->N relationship

In this chapter you will:

- Create classes and variables (create **ZyxAddress>>persons**).
- Set **ZyxPerson>>addresses** type to **ZyxAddress** and **ZyxAddress>>persons** type to **ZyxPerson**.
- Set cardinality of **ZyxPerson>>addresses** to 1..N and set cardinality of **ZyxAddress>>persons** to 1..10.
- Review how the OBF classes and methods you created would be used by PFW.

### 2.18.1. Create classes and variables.

19.12. Create **ZyxAddress>>persons**.

### 2.18.2. Set **ZyxPerson>>addresses** type as M<->N relationship

19.13. For **ZyxPerson>>addresses**: Uncheck **Primitive**.

19.14. Specify **variable** as **persons**.

19.15. Specify **Source class minimum...maximum** as 1..N.

19.16. Specify **Target class minimum...maximum** as 1..10.

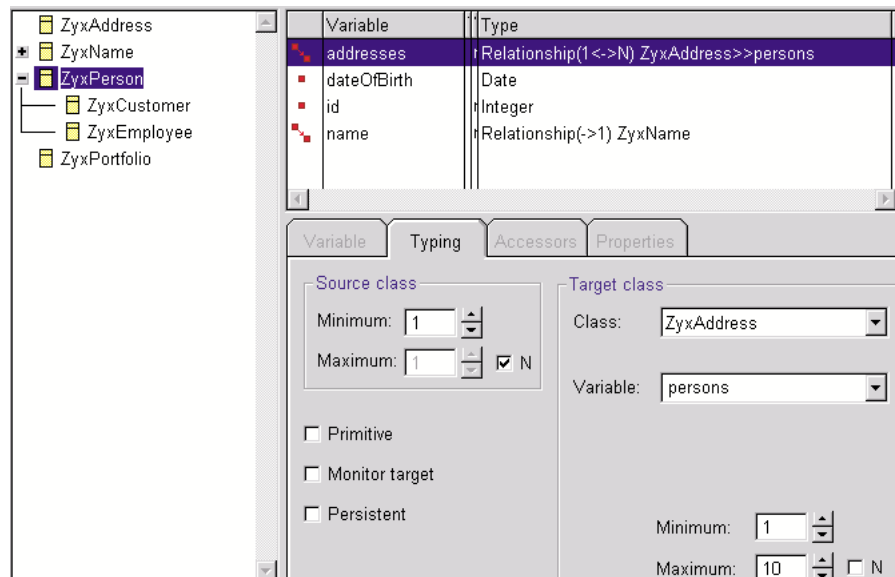


Figure 2.18.2. **ZyxPerson>>addresses** definition

19.17. Display tab **Typing** for **ZyxAddress>>persons**.

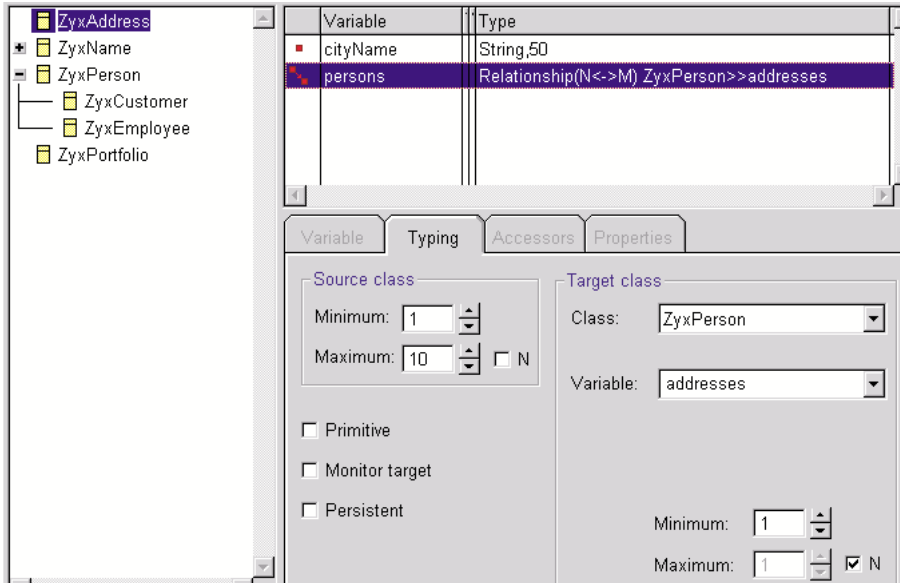


Figure 2.18.3. *ZyxAddress>>persons* definition

- 19.18. Save model.
- 19.19. Save to VA.

**2.18.3. Review how the OBF classes and methods you created can be used by PFW.**

The following diagram shows how the Smalltalk objects instantiated from the classes you just created and the database tables would be related.

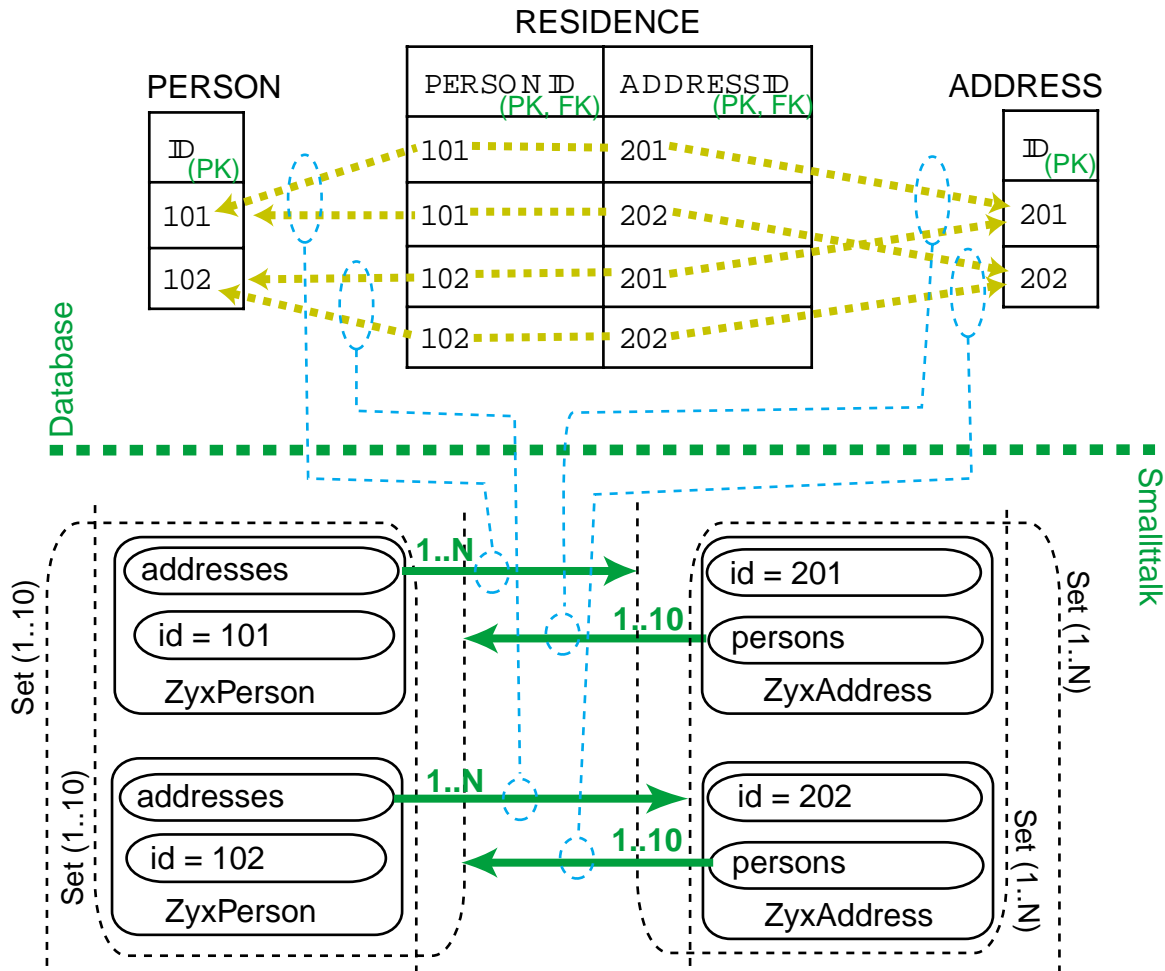


Figure 2.18.4. *M<->N* relationship between *ZyxPerson* and *ZyxAddress* in tables *PERSON*, *RESIDENCE*, and *ADDRESS*

For M<->N relationships, a 3rd table is required. In this case, the table is RESIDENCE. Each row in residence represent a unique pair of source and target objects. Both keys in RESIDENCE are FK and PK. Note that in table RESIDENT column PERSON ID a number can appear in 1..N rows (a person can have any number of addresses). However, in column ADDRESSID a number can appear in 1..10 rows (an address can only have from 1..10 persons).

## 2.19. 1 context: 1 (non-nested) TrLevel

In this section you will:

- Create and view (in the Transaction Browser) a non-running, non-active context
- Specify a variable as being transacted (creates the required Framework accessors)
- Assign an object to a variable (with no active context)
- Assign an object to a variable (with an active context)
- Commit TrLevel1
- Abort TrLevel1
- Commit the context
- Abort the context

### 2.19.1. Cleaning up

During the examples, if at any point you wish to start from scratch, execute the following lines to abort all transactions in all contexts and remove keys from the system dictionary:

```
OBF CH 2.19.#1. Commands for cleaning up transactions."  
[  
  MicFwTransactionManager abortAll.  
  Smalltalk removeKey: #con1 ifAbsent: [].  
  Smalltalk removeKey: #cus1 ifAbsent: [].  
  Smalltalk removeKey: #cus2 ifAbsent: [].  
]
```

### 2.19.2. Create non-running, non-active context

20.1. Create a context and store in the system dictionary:

```
OBF CH 2.19.#2. Create transaction context"  
[  
  Smalltalk at: #con1 put: MicFwTransactionManager newTransactionContext.  
]
```

The context, having been just created, is:

- **Not running.** TrLevel1 does not exist (message beginTransactionContext has not been sent to con1 yet).
- **Not active.** The current context of the transaction manager is not con1 (message activate has not been sent to con1 yet).

#### 2.19.2.1. Open Transaction Browser

20.2. Open the Transaction Browser:

```
[  
  MicFwTransactionsBrowser openBrowser.  
]
```

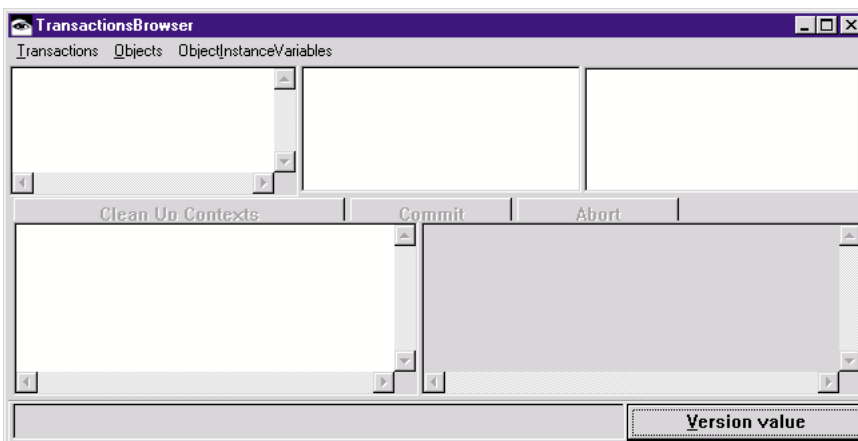


Figure 2.19.1. Transactions Browser

Note that no transactions are displayed since no transactions are running (ie, with TrLevel1 or higher).



Note: The transaction browser can also be opened from the System Transcript menu by selecting **mic-Frameworks / Browse Transactions**.

### 2.19.3. Specify ZyxName>>firstName as transacted

A variable must be specified as transacted (using the OMB) for transacted changes for that variable. Specifying the variable as transacted creates the required accessors with transaction support for the variable.

20.3. In the OMB: For **ZyxName>>firstName**: Check the checkbox **Transacted**.

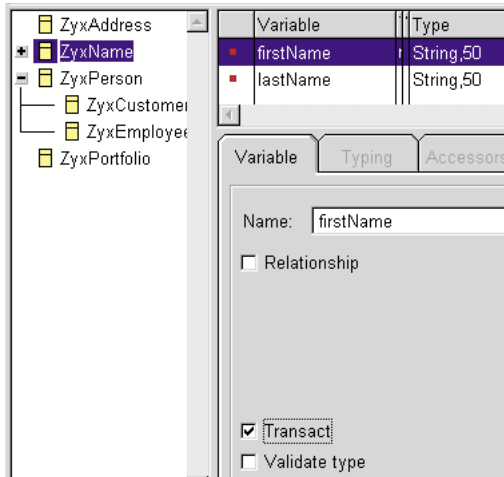


Figure 2.19.2. Specifying ZyxName>>firstName as transacted

20.4. Save the model.

20.5. Save to VA.

### 2.19.4. Assign an object to a variable (with no active context)

Any changes to a transacted variable will not be transacted if no context is running or active.

20.6. Create a new Customer object and assign a name object to the transacted variable "name":

OFB CH 2.19.#3. Non-running, non-active transaction context"

```
[
  Smalltalk at: #cus1 put: ZyxCustomer new.
]
[
  cus1 name: ZyxName new.
  cus1 name firstName: 'clFirstNameV1'.
]
```

20.7. Verify that the context is not running or active:

```
[
  con1 isRunning.          "false"
]
[
  con1 isActive.          "false"
]
```

20.8. Refresh the Transaction Browser dialog (from the main menu select **Transactions / Update**). Note that there are no running or active contexts.

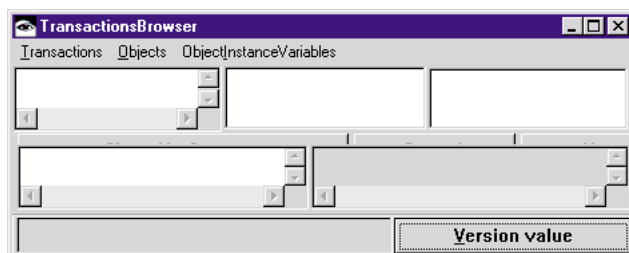


Figure 2.19.3. No running/active contexts in TB

---

## 2.19.5. Assign an object to a variable (with active context)

### 2.19.5.1. beginTransaction

20.9. Run and activate context by sending message beginTransaction to context:

```
OBF CH 2.19.#4. Run / activate context (beginTransaction)"
[
  con1 beginTransaction.
  con1 isRunning.          "true"
]
[
  con1 isActive.          "true"
]
```

### 2.19.5.2. Inspect changes to context, manager

20.10. Refresh the Transaction Browser. Note that 1 transaction context and 1 level are shown. However, no object version or object aspects are listed.

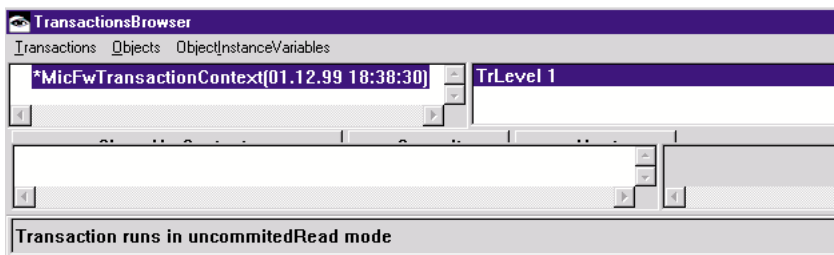


Figure 2.19.4. Running (TrLevel1) / active ("\*") context in TB

### 2.19.5.3. Example: Running / Active contexts and applications

Typically a context would be created and the beginTransaction message sent to the context when a user dialog for data entry is being opened.

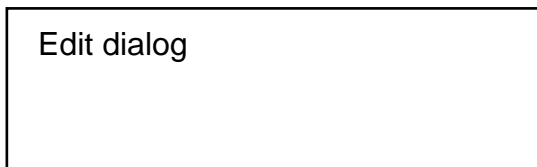


Figure 2.19.5. Typical dialog for data entry

The context would be:

- Running (the dialog is open).
- Active (the dialog is the active dialog).

### 2.19.5.4. Assign object to variable

20.11. Assign object to firstName.

```
OBF CH 2.19.#5. Active context: Assign object to transacted variable."
[
  cus1 name firstName: 'cus1FirstNameV2'.
]
```

### 2.19.5.5. Object version uncommitted/committed targets in Transaction Browser

20.12. Refresh the Transaction Browser display. An object version for "a ZyxName" is now shown. The **uncommitted target (version value)** is shown ('cus1NameFirstV2').

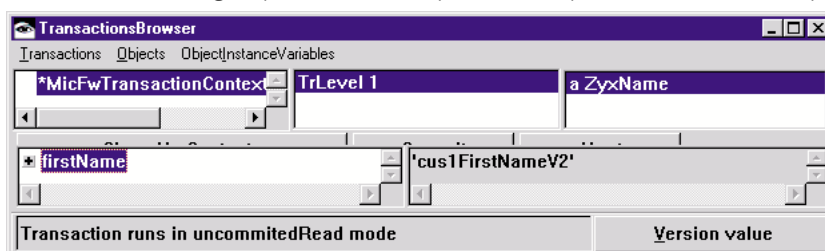


Figure 2.19.6. Uncommitted target (version value) for ZyxName in TB

20.13. Click on **Version value**. The button becomes **Variable value**. The *committed target (variable value)* is shown ('cus1NameFirstV1').

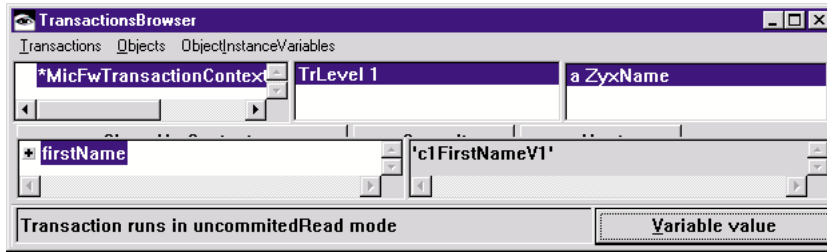


Figure 2.19.7. Committed target (variable value) for ZyxName in TB

The object version object returns 1 of 2 versions of a target object depending on the context isolation of the active context in which the getter message was sent. The following diagram illustrates the structure of an object version:

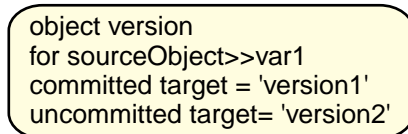


Figure 2.19.8. Object version

### 2.19.5.6. Example: Assigning object to transacted variable in active context with TrLevel1

Typically a context would be created and the beginTransaction message sent to the context when a user dialog for data entry is being opened.

A typical example of a context with TrLevel1 is the dialog in the following diagram. The characters entered in the text field become the new string object assigned as the uncommitted target. However, this uncommitted target ('cus1NameFirstV2') does not become the committed target ('cus1NameFirstV1') while the context is not committed.

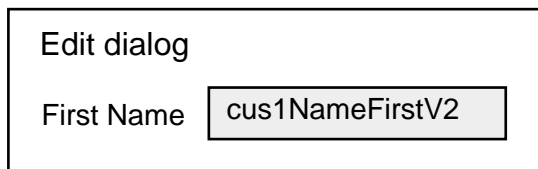


Figure 2.19.9. Assigning object to transacted variable in active context with TrLevel1: Example dialog

Note: Actually a new uncommitted target would be created each time a character is entered in the above field.

### 2.19.6. TrLevel1: Committing

Committing TrLevel1 causes the following:

- The transacted objects become the actual objects assigned to the variables.
- TrLevel1 ceases to exist.
- The context is closed (ceases to exist).

20.14. Commit TrLevel1.

```

OBF CH 2.19.#6. Committing TrLevel1."
[
  con1 commitTransaction.
]
[
  cus1 name firstName.      "'cus1FirstNameV2'"
]

```

### 2.19.6.1. Inspect changes to transaction objects

20.15. Refresh the view in Transaction Browser. Note that no transaction variables exist.

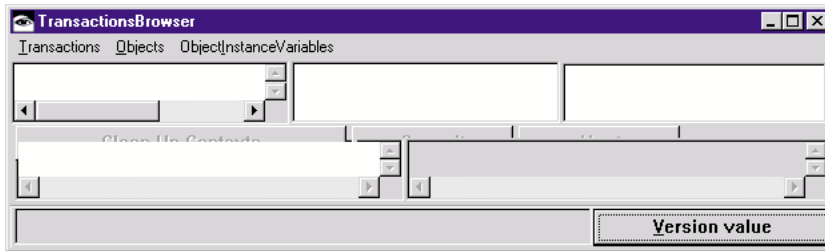


Figure 2.19.10. TrLevel1 committed: TB empty

### 2.19.6.2. Example: Committing TrLevel1

Pressing the OK button in the dialog show below would send the commitTransaction message to the context for the dialog.

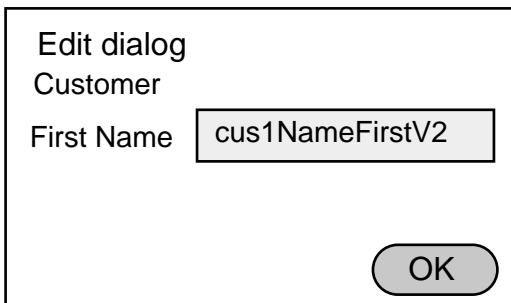


Figure 2.19.11. Committing TrLevel1: Example dialog

---

## 2.19.7. TrLevel1: Aborting

Aborting TrLevel1 causes the following:

- Any uncommitted targets is object versions assigned to the TrLevel1 are dereferenced.
- TrLevel1 ceases to exist.
- The context is closed (ceases to exist).

### 2.19.7.1. Run / activate context (beginTransaction)

20.16. Run and activate context by sending message beginTransaction to context:

```
OBF CH 2.19.#7. Aborting TrLevel1
[
  con1 beginTransaction.
```

### 2.19.7.2. Assign objects to variables

20.17. Create a second Customer instance and assign new objects to both Customer instances.

```
cus1 name firstName: 'cus1FirstNameV3'.
]
```

### 2.19.7.3. Abort the transaction

20.18. Abort the transacted changes:

```
[
  con1 abortTransaction.
]
[
  cus1 name firstName.      "'cus1FirstNameV2'"
]
```

### 2.19.7.4. Example: Aborting TrLevel1

Pressing the CANCEL button in the dialog show below would send the abortTransaction message to the

context for the dialog.

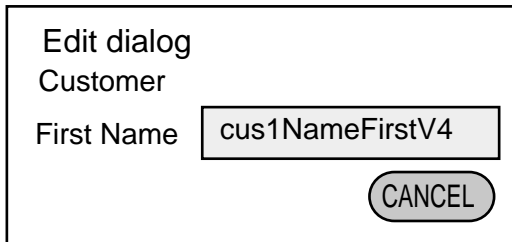


Figure 2.19.12. Aborting TrLevel1: Example dialog

---

## 2.19.8. Committing a single-level context

Committing a context has exactly the same effect as committing TrLevel1.

20.19. Run and activate context by sending message beginTransaction to context:

```
OBF CH 2.19.#8. Committing a context
```

```
[  
  con1 beginTransaction.
```

20.20. Assign objects to variables.

```
  cus1 name firstName: 'cus1FirstNameV4'.
```

20.21. A context is committed with the commitTransaction message.

```
  con1 commitTransaction.
```

```
]
```

---

## 2.19.9. Aborting a single-level context

Aborting a context has exactly the same effect as aborting TrLevel1.

20.22. Begin transaction and assign objects:

```
OBF CH 2.19.#9. Aborting a context
```

```
[  
  con1 beginTransaction.
```

20.23. Create a second Customer instance and assign new objects to both Customer instances.

```
  cus1 name firstName: 'cus1FirstNameV5'.
```

20.24. A context is aborted with the abortTransaction message.

```
  con1 abortTransaction.
```

```
]
```

## 2.20. 1 context: Multiple (nested) TrLevels

In this section you will:

- Create a context with 2 transaction levels
- Abort TrLevel2
- Abort all TrLevels (same as aborting context)
- Commit TrLevel2
- Commit all TrLevels (same as committing context)
- Abort context
- Commit context

### 2.20.1. Create context, TrLevel1, TrLevel2

21.1. Specify **ZyxName>>lastName** as **Transacted**.

21.2. Save the model.

21.3. Save to VA.

21.4. Create a context with 2 TrLevels. TrLevel1 has 1 object version, TrLevel2 has 2 object versions:

OBF CH 2.20.#1. Create context with 2-levels."

```
[
  cus1 name: ZyxName new.
  cus1 name firstName: 'c1FirstNameTL0'.
  cus1 name lastName: 'c1LastNameTL0'.
  con1 beginTransaction.
  cus1 name firstName: 'c1FirstNameTL1'.
  con1 beginTransaction.
  cus1 name firstName: 'c1FirstNameTL2'.
  cus1 name lastName: 'c1LastNameTL2'.
]
```

#### 2.20.1.1. Inspect changes to transaction objects

21.5. Refresh the view in Transaction Browser.

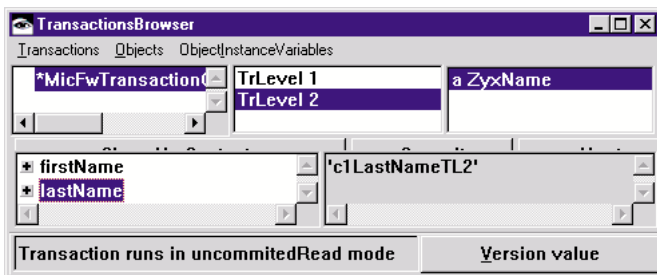


Figure 2.20.1. 2 TrLevels in TB

#### 2.20.1.2. Example: 2 TrLevels

The code above could represent the situation show in the following diagram:

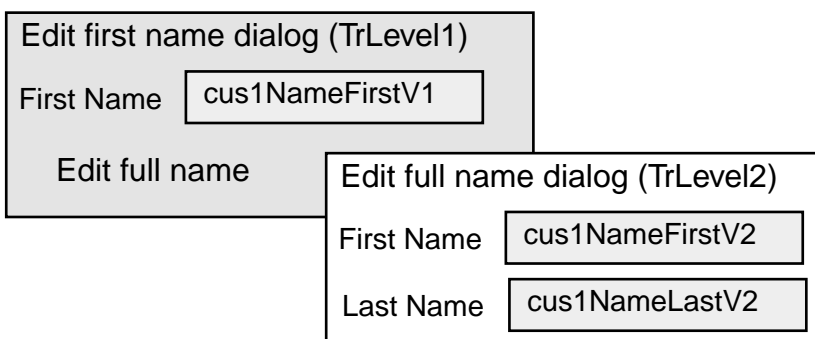


Figure 2.20.2. 2 TrLevels: Example dialogs.

The context would be created and the first beginTransaction message would be sent when the first dialog is opened. The second beginTransaction message would be sent when the second dialog is opened. The

firstName: and lastName: messages would be sent when text is entered in the text fields.

Note: The assignment of an object can only be recorded in the object version in the highest TrLevel. This is equivalent to saying that data can be entered only in the the last sub-dialog opened.

---

## 2.20.2. Abort highest level (TrLevel2)

21.6. Abort TrLevel2 by sending the abortTransaction message to the context:

```
OBF CH 2.20.#2. Abort TrLevel2 of 2-level context."  
[  
  con1 abortTransaction.  
]
```

21.7. Check the objects referenced by variables:

```
[  
  cus1 name firstName      "c1NameFirstTL1"  
]  
[  
  cus1 name lastName      "c1NameLastTL0"  
]
```

### 2.20.2.1. Inspect changes to transaction objects

21.8. Refresh the view in Transaction Browser. Note that only TrLevel1 exists.

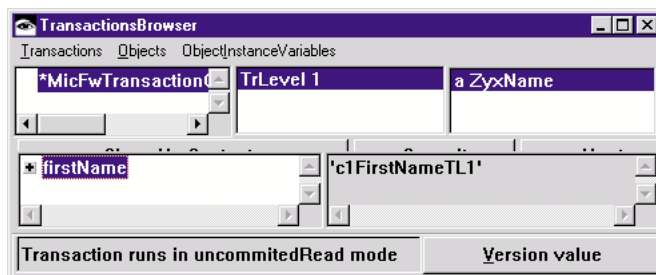


Figure 2.20.3. 1 TrLevel in TB

### 2.20.2.2. Example: Abort highest level (TrLevel2)

The code above could represent pressing the CANCEL button:

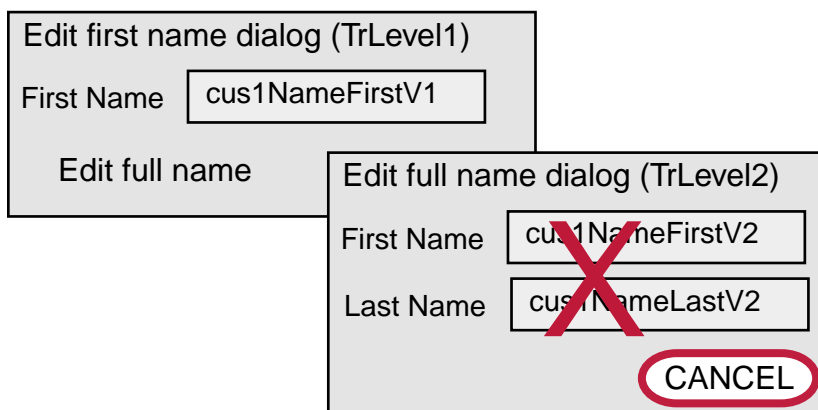


Figure 2.20.4. Aborting highest TrLevel: Example dialogs.

The changes that were entered in the TrLevel2 dialog would be lost.

---

## 2.20.3. 2-level transaction: Abort all TrLevels

21.9. Create TrLevel2 and abort both levels:

```
OBF CH 2.20.#3. Abort all levels of 2-level context."  
[  
  con1 beginTransaction.  
  cus1 name firstName: 'c1FirstNameTL3'.  
  cus1 name lastName: 'c1LastNameTL3'.  
  con1 abortToTop.  
]
```

### 2.20.3.1. Inspect changes to transaction objects

21.10. Refresh the view in Transaction Browser. Note that no TrLevels exist.

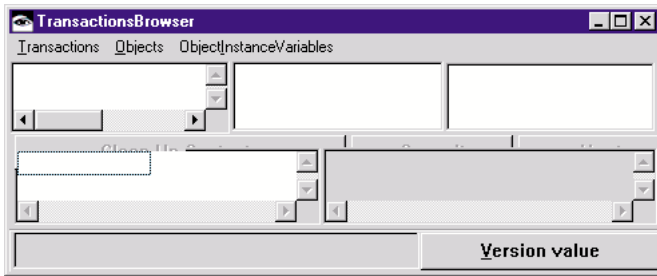


Figure 2.20.5. No TrLevels in TB

### 2.20.3.2. Example: Abort all levels

The code above could represent pressing the CANCEL ALL button as shown in the following diagram:

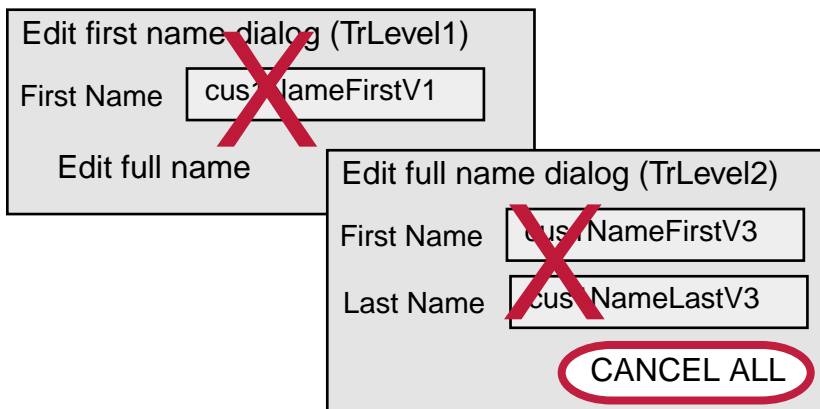


Figure 2.20.6. Aborting all TrLevels: Example dialogs.

The changes that were entered in the dialogs would be lost.

---

## 2.20.4. Commit highest level (TrLevel2)

21.11. Create object version and 2 TrLevels:

```
OBF CH 2.20.#4. Create context with 2-levels."
```

```
[
  cus1 name: ZyxName new.
  cus1 name firstName: 'c1FirstNameTL0'.
  cus1 name lastName: 'c1LastNameTL0'.
  con1 beginTransaction.
  cus1 name firstName: 'c1FirstNameTL1'.
  con1 beginTransaction.
  cus1 name firstName: 'c1FirstNameTL2'.
  cus1 name lastName: 'c1LastNameTL2'.
]
```

21.12. Commit TrLevel2 by sending the commitTransaction message to the context:

```
OBF CH 2.20.#5. Commit TrLevel2 of 2-level context."
```

```
[
  con1 commitTransaction.
]
```

### 2.20.4.1. Inspect changes to transaction objects

21.13. Check the objects referenced by variables:

```
[
  cus1 name firstName      "c1NameFirstTL2"
]
[
  cus1 name lastName       "c1NameLastTL2"
]
```



21.14. Refresh the view in Transaction Browser. Note that only TrLevel1 exists. The version value for first-Name is 'c1FirstNameTL2'. The variable value, however, is still 'c1FirstNameTL0'.

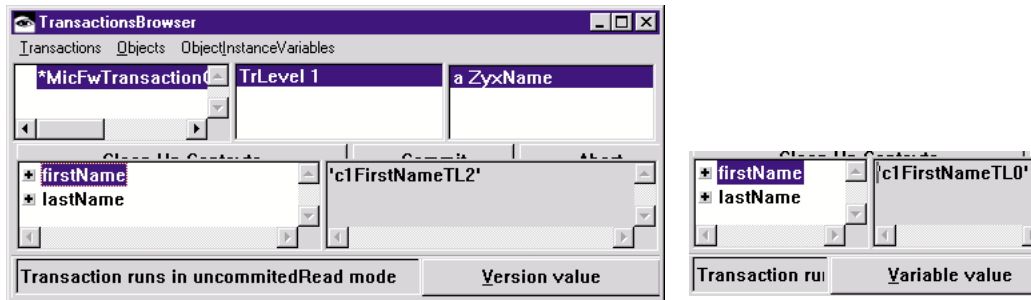


Figure 2.20.7. 1 TrLevel in TB

### 2.20.4.2. Example: Commit highest level (TrLevel2)

The code above could represent pressing the OK button:

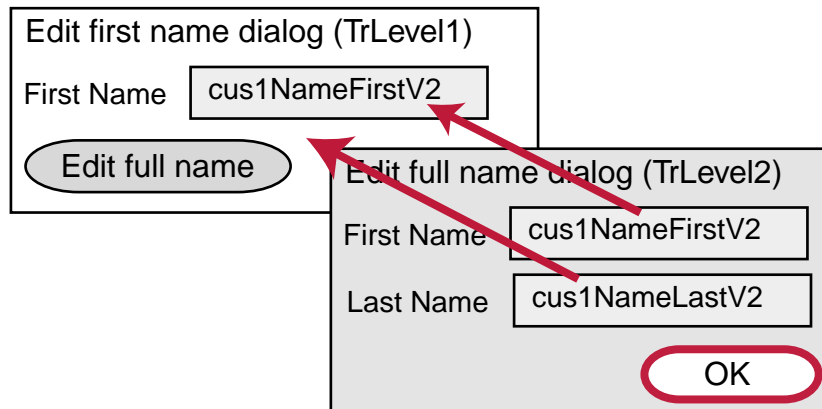


Figure 2.20.8. Committing highest TrLevel: Example dialogs.

The objects that were assigned to variables in TrLevel1 would be lost if objects were written to the same variables in TrLevel2.

## 2.20.5. 2-level transaction: Commit all TrLevels

21.15. Create TrLevel2 and commit both levels:

```

OBF CH 2.20.#6. Commit all levels of 2-level context."
[
    con1 beginTransaction.
    cus1 name firstName: 'c1FirstNameTL3'.
    cus1 name lastName: 'c1LastNameTL3'.
    con1 commitToTop.
]

```

### 2.20.5.1. Inspect changes

21.16. Check the objects referenced by variables:

```

[
    cus1 name firstName      "c1NameFirstTL3"
]
[
    cus1 name lastName      "c1NameLastTL3"
]

```

21.17. Refresh the view in Transaction Browser. Note that no transaction contexts exist.

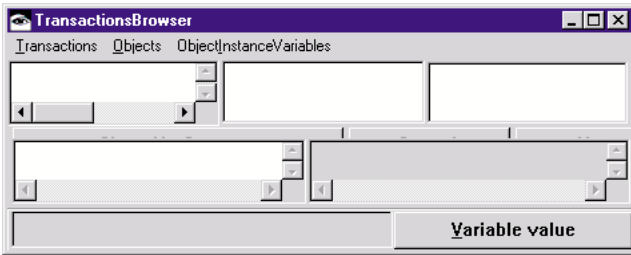


Figure 2.20.9. No transaction contexts in TB

### 2.20.5.2. Example: Commit all levels

The code above could represent pressing the ACCEPT ALL button as shown in the following diagram:

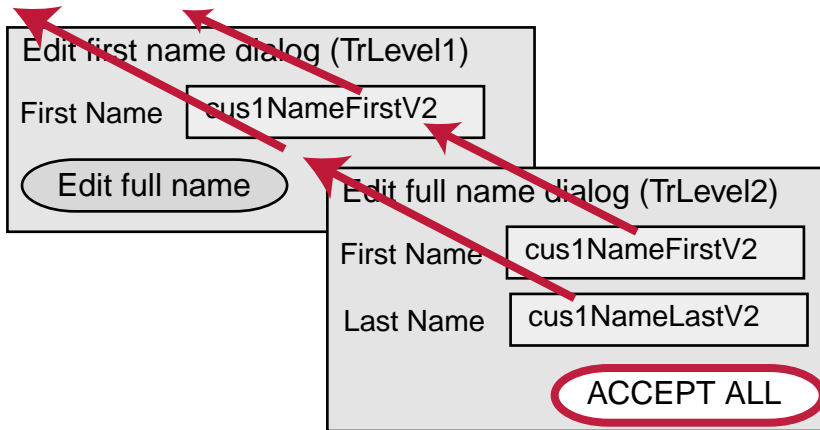


Figure 2.20.10. Committing all TrLevels: Example dialogs

The uncommitted objects assigned to the variables would become the actual (committed) objects. Thus, the latest changes entered in the dialogs would be confirmed.

---

### 2.20.6. Aborting a multi-level context

Aborting a multi-level context has exactly the same effect as aborting all TrLevels.

---

### 2.20.7. Committing a multi-level context

Committing a multi-level context has exactly the same effect as committing all TrLevels.

## 2.21. Multiple contexts: Concurrent

In this section you will:

- Create 2 concurrent contexts in model uncommittedRead
- Check the objects returned by variable accessors
- Create 2 concurrent contexts in model isolate
- Check the objects returned by variable accessors
- Attempt to change a variable locked by another context

### 2.21.1. Create 2 (concurrent) contexts; mode uncommittedRead (default)

#### 2.21.1.1. Create contexts

22.1. Create 2 contexts:

```
OBF CH 2.21.#1. Create 2 contexts."  
[  
  Smalltalk at: #con1 put: MicFwTransactionManager newTransactionContext.  
  Smalltalk at: #con2 put: MicFwTransactionManager newTransactionContext.  
]  
[  
  cus1 name: ZyxName new.  
  cus1 name firstName: 'c1FirstNameV0'.  
  cus1 name lastName: 'c1LastNameV0'.  
  con1 beginTransaction.  
  cus1 name firstName: 'c1FirstNameCon1V1'.  
  con2 beginTransaction.  
  cus1 name lastName: 'c1LastNameCon2V1'.  
]  
]
```

#### 2.21.1.2. Inspect changes

22.2. Check the objects referenced by variables:

```
[  
  cus1 name firstName      "c1FirstNameCon1V1"  
]  
[  
  cus1 name lastName      "c1LastNameCon2V1"  
]  
]
```

22.3. Refresh the view in Transaction Browser. Note that 2 contexts exist.

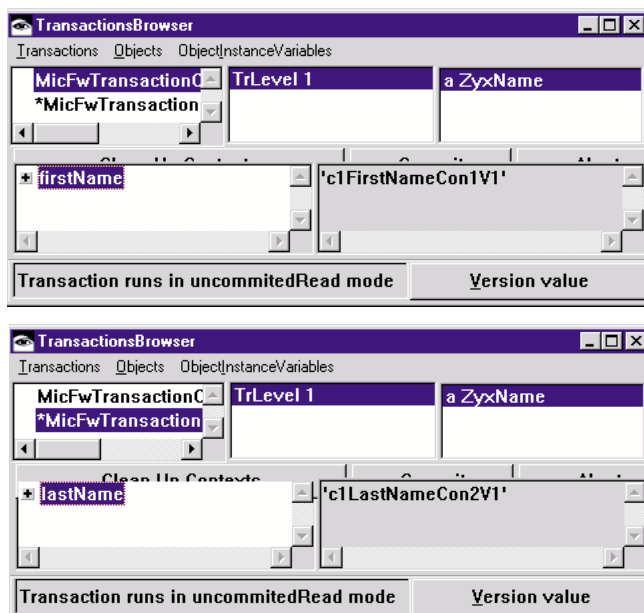


Figure 2.21.1. 2 contexts in TB

Note the following:

- Both contexts are running in uncommittedRead mode (described later).
- con2 is active (con2 began after con1; only 1 context can be active)

### 2.21.1.3. Example: 2 concurrent contexts

The code above could represent the situation show in the following diagram:

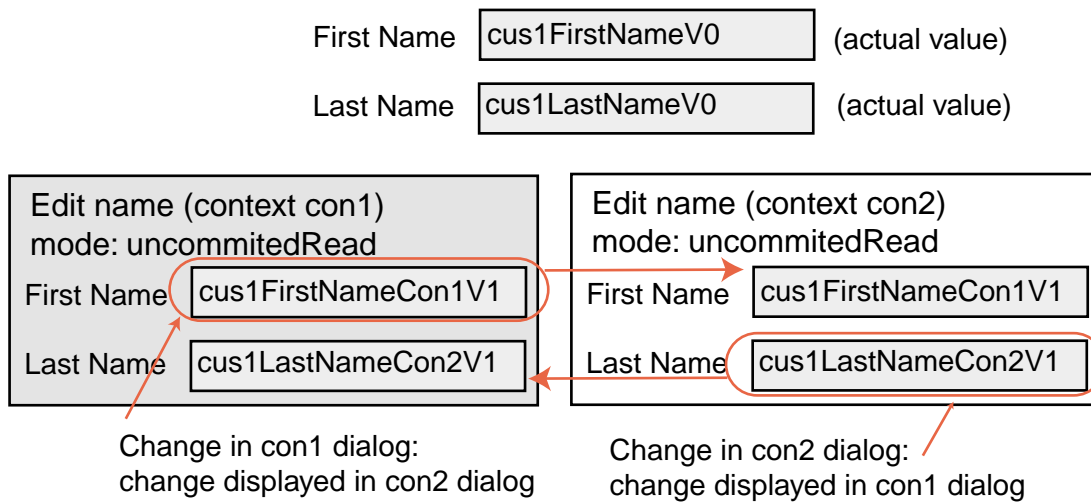


Figure 2.21.2. 2 concurrent contexts in uncommittedRead mode: Example dialogs

Note that both contexts show the transacted object referenced by the variables.

## 2.21.2. 2 concurrent contexts; mode isolate

### 2.21.2.1. Change the mode of the contexts

22.4. Change the mode of the contexts:

```
OBF CH 2.21.#2. Change mode of contexts to isolated."
[
  con1 isolate.
  con2 isolate.
]
```

### 2.21.2.2. Inspect changes

22.5. Check the objects referenced by variables:

```
[
  con1 activate.
  cus1 name firstName      "c1FirstNameCon1V1"
]
[
  cus1 name lastName      "c1LastNameV0"
]
[
  con2 activate.
  cus1 name firstName      "c1FirstNameV0"
]
[
  cus1 name lastName      "c1LastNameCon2V1"
]
```

22.6. Refresh the view in Transaction Browser. Note that the only change in the browser is the designation of the contexts as isolated.

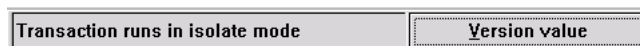


Figure 2.21.3. Designation of isolated mode in the TB status bar for a context

Note the following:

- In isolated mode: A context only sees changes that were made while the context was active.

### 2.21.2.3. Example: Isolated mode

The code above could represent the situation show in the following diagram:

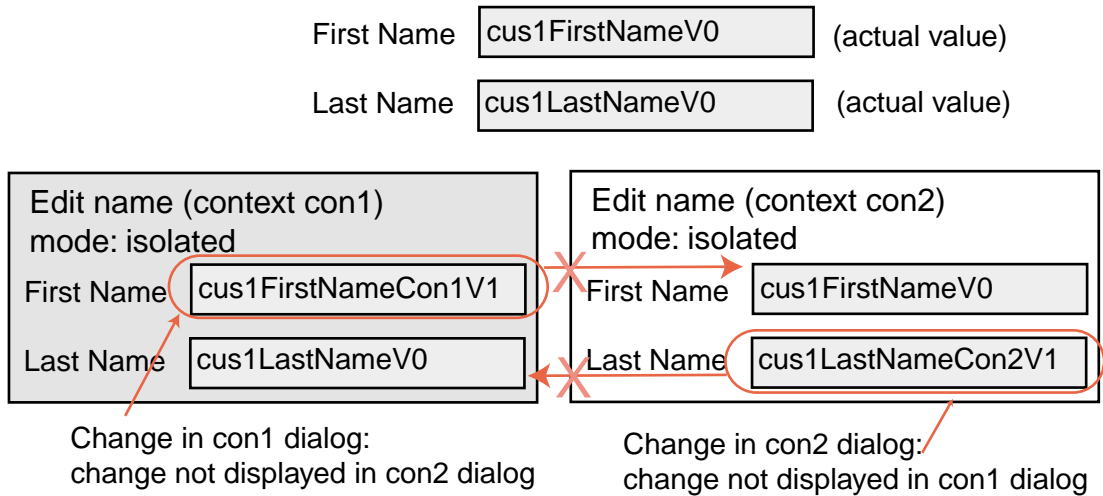


Figure 2.21.4. 2 concurrent contexts in isolated mode: Example dialogs

### 2.21.3. Attempt to change variable in context that is locked by other context

A context cannot change a variable that is locked by a concurrent context.

22.7. Attempt to change a variable locked by another context.

```
[
  con2 activate.
  cus1 name firstName: 'xxx'. "exception: Transaction write conflict."
]
[
  con1 activate.
  cus1 name lastName: 'xxx'. "exception: Transaction write conflict."
]
```

#### 2.21.3.1. Example: Attempt to change variable in parent or sibling that is locked by other context

The code above could represent the situation show in the following diagram:

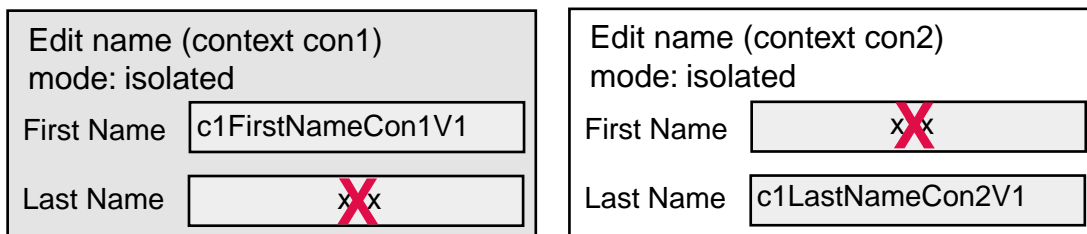


Figure 2.21.5. Unallowed changes to locked variables in concurrent contexts: Example dialogs

---

## 2.22. Multiple contexts: Parent / Child

In this chapter you will:

- Create a context with a child and a grandchild contexts
- Test in uncommittedRead and isolate modes for a parent / child
- Analyze how a variable lock can be transferred

---

### 2.22.1. Context and 2 child contexts (child and grandchild)

#### 2.22.1.1. Create contexts

23.1. Create context and 2 child contexts:

```
OBF CH 2.22.#1. Create child and grandchild context for con1."
[
  con1 abort.
  con2 abort.
]
[
  Smalltalk at: #con11 put: con1 newTransactionContext.
]
[
  Smalltalk at: #con111 put: con11 newTransactionContext.
]
[
  cus1 name: ZyxCName new.
  cus1 name firstName: 'c1FirstNameV0'.
  cus1 name lastName: 'c1LastNameV0'.
  con1 beginTransaction.
  cus1 name firstName: 'c1FirstNameCon1V1'.
  cus1 name lastName: 'c1LastNameCon1V1'.
  con11 beginTransaction.
  con111 beginTransaction.
  cus1 name lastName: 'c1LastNameCon111V2'.
]
```

## 2.22.1.2. Inspect changes

23.2. Refresh the view in Transaction Browser. Note that 3 contexts exist.

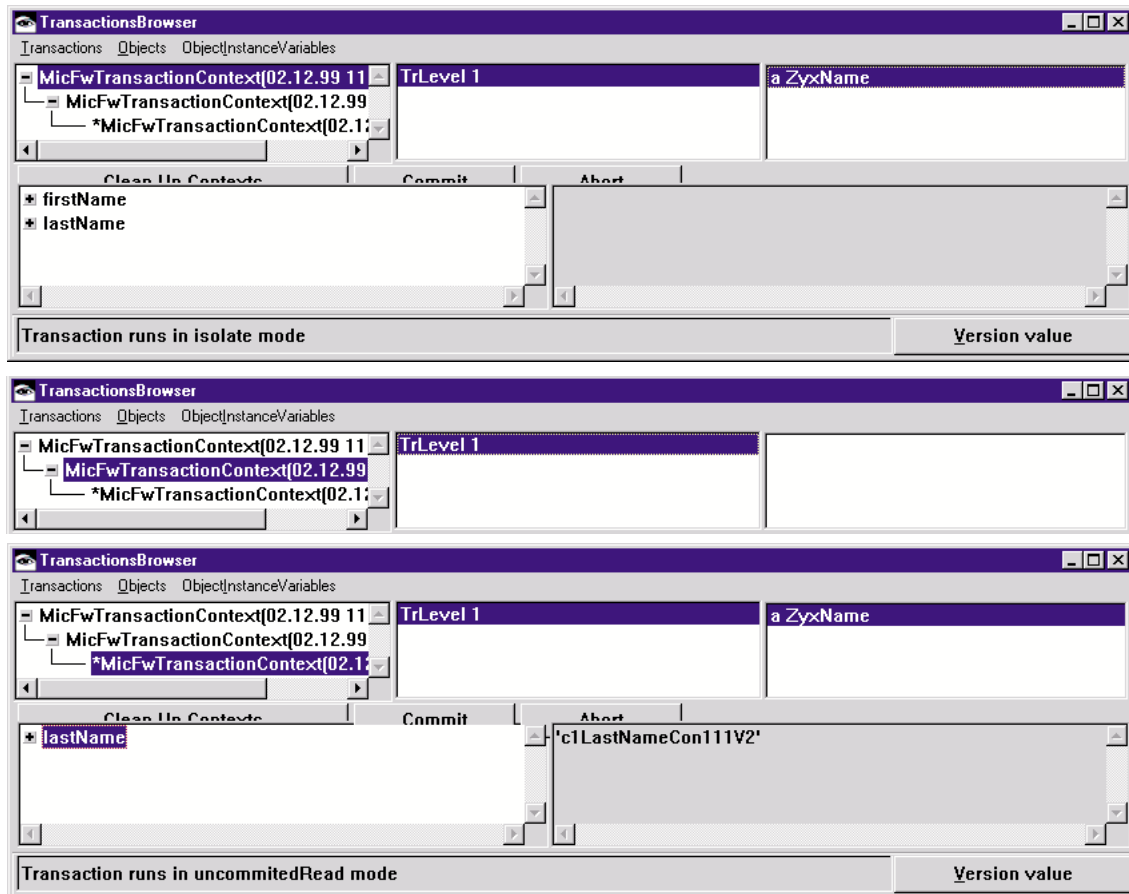


Figure 2.22.1. Parent, child, grandchild contexts in TB

## 2.22.1.3. Example: Parent, child, grandchild context

The code above could represent the situation show in the following diagram:

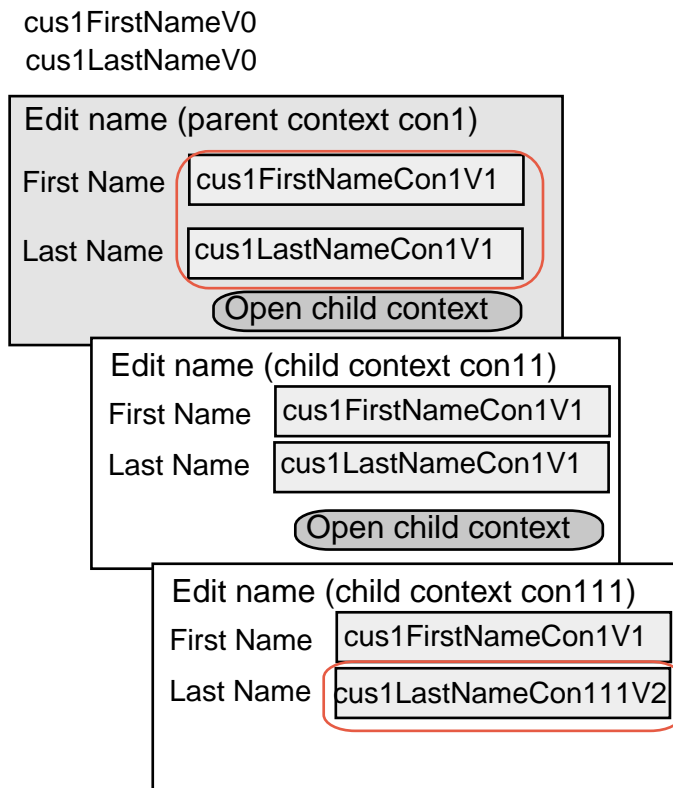


Figure 2.22.2. Parent context with child, grandchild contexts: Example dialogs

## 2.22.2. Test in uncommittedRead and isolate modes for a parent / child

### 2.22.2.1. uncommittedRead when variable locked by non-active parent

23.3. Set con11 to uncommittedRead and then test the value returned for firstName (locked by con1):

```
OBF CH 2.22.#2. uncommittedRead when var locked by non-active parent."  
[  
  con11 uncommittedRead.  
  con11 activate.  
  cus1 name firstName "c1FirstNameCon1V1".  
]
```

### 2.22.2.2. uncommittedRead when variable locked by non-active child

23.4. Test the value returned for lastName (locked by con111):

```
OBF CH 2.22.#3. uncommittedRead when var locked by non-active child."  
[  
  cus1 name lastName "c1LastNameCon111V2".  
]
```

### 2.22.2.3. isolated read variable locked by non-active parent

23.5. Set con11 to isolated and then test the value returned for firstName (locked by con1):

```
OBF CH 2.22.#4. isolated read when var locked by non-active parent."  
[  
  con11 isolate.  
  cus1 name firstName "c1FirstNameCon1V1".  
]
```

### 2.22.2.4. isolated read when variable locked by non-active child

23.6. Test the value returned for lastName (locked by con111):

```
OBF CH 2.22.#5. isolated read when var locked by non-active child."  
[  
  cus1 name lastName " 'c1LastNameCon1V1' ".  
]
```

### 2.22.2.5. Example: Parent, child, grandchild context with uncommitted, isolated read

The code above could represent the situation show in the following diagram:

```
cus1FirstNameV0  
cus1LastNameV0
```

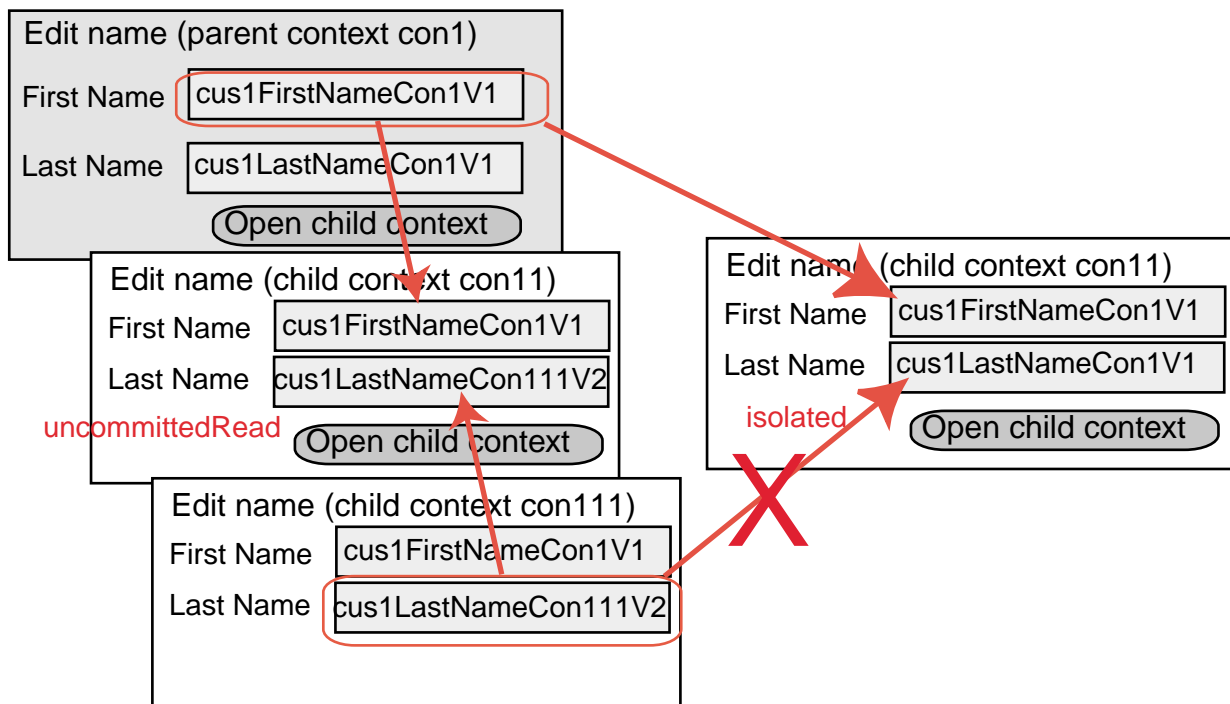


Figure 2.22.3. Parent context with child, grandchild contexts: uncommittedRead, isolated example dialogs



## 2.22.3. Analyze how a variable lock can be transferred

### 2.22.3.1. Transfer of variable lock to child

23.7. Modify firstName in con11:

```
OBF CH 2.22.#6. Transfer variable lock from con1 to con11."  
[  
  con11 activate.  
  cus1 name firstName: 'clFirstNameCon11V2'.  
  con1 activate.  
  cus1 name firstName: 'xxx' "transaction write conflict exception".  
]
```

### 2.22.3.2. No transfer of variable lock to parent

23.8. Attempt to modify lastName in con11:

```
OBF CH 2.22.#7. No transfer variable lock from con111 to con11."  
[  
  con11 activate.  
  cus1 name lastName: 'xxx' "transaction write conflict exception".  
]
```

### 2.22.3.3. Examples: Transfer of variable lock

The code above could represent the situation show in the following diagram:

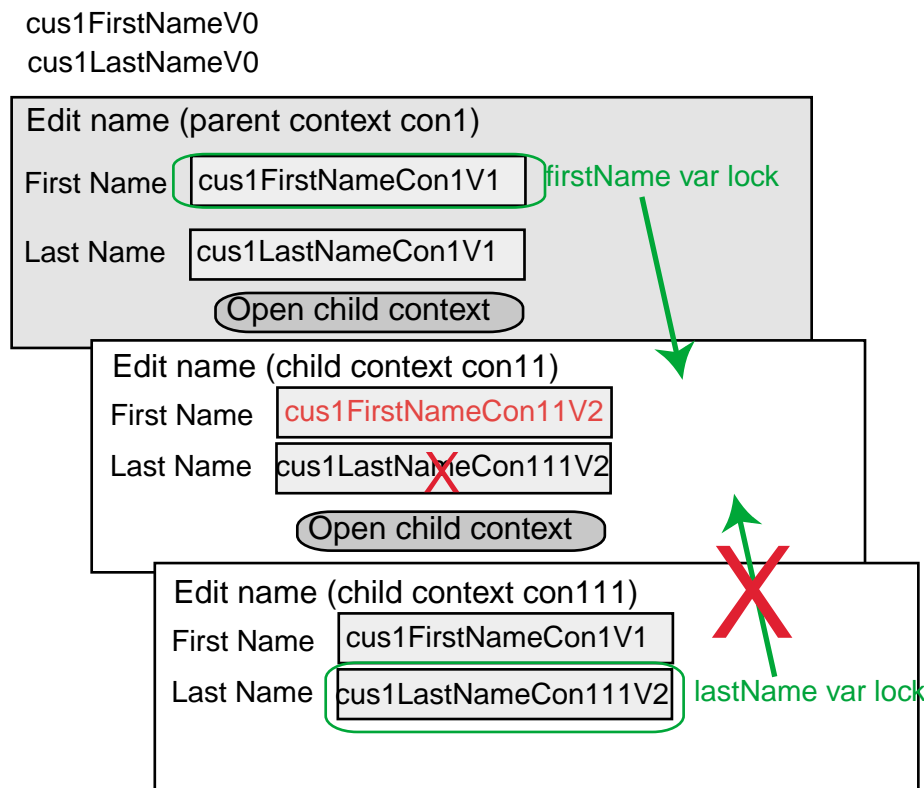


Figure 2.22.4. Transfer of variable locks between parent and child contexts



# 3

# Tools



---

## 3.1. Introduction

This part of the User's Guide describes the visual tools of OBF. It includes overviews and short examples that demonstrate the functionality of each element in each tool.

The described tools include:

- '3.2. Object Model Browser (OMB)' (page 110).
- '3.3. Object Net Browser (NetBrowser)' (page 119)
- '3.4. Type Editor' (page 124)
- '3.5. Relationship Editor' (page 125)
- '3.6. Transaction Browser' (page 127)

## 3.2. Object Model Browser (OMB)

### 3.2.1. Primary functions

The primary functions of OMB include:

- Creating an Object Model
- Adding/deleting classes/variables to the model
- Specifying default attributes for classes/variables
- Exporting a model to other formats (.st, .xml)
- Importing/saving a model to VA
- Specifying variable typing/relationships

### 3.2.2. Opening OMB

#### 3.2.2.1. From Transcript menu

Note: When opening from the System Transcript menu, the OMB must be opened on a class (OMB cannot be simply opened without specifying a class). To open a model file (.ome) in the OMB, simply open the OMB on any class and then open the model file from within OMB.

2.1. In the System Transcript: Select **micFrameworks / Open Object Model Browser...**

2.2. In response to the message "Select root class":

2.2.1 Enter the name of a class that is in the Object Net.

2.2.2 Click **OK**.

**OR**

2.2.3 Click **OK**. A list of available classes is displayed (the displayed classes are all subclasses of MicFwDomainObject).

2.2.4 Double-click on a class in the list.

The OMB dialog appears:

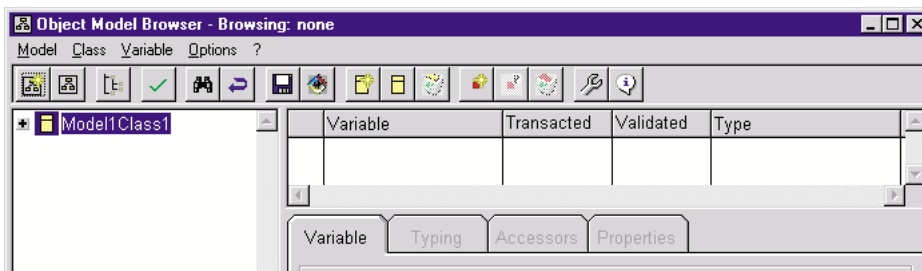


Figure 3.2.1. OMB dialog (opened on class Model1Class1 from the Transcript menu)

#### 3.2.2.2. From transcript (Smalltalk)

2.3. Execute the following line in a transcript window:

```
MicFwTemModelDescriptionProcess open
```

The OMB dialog appears (without an object net):

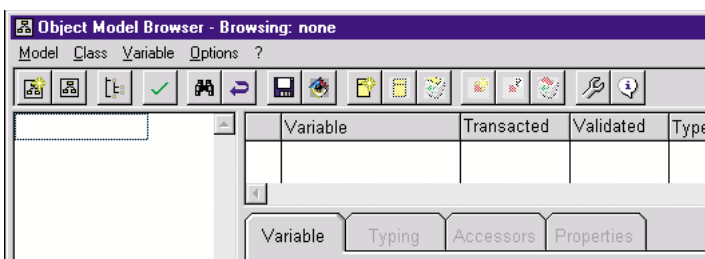


Figure 3.2.2. OMB dialog (opened with MicFwTemModelDescriptionProcess open)

### 3.2.3. OMB dialog

The following shows the major parts of the OMB dialog.

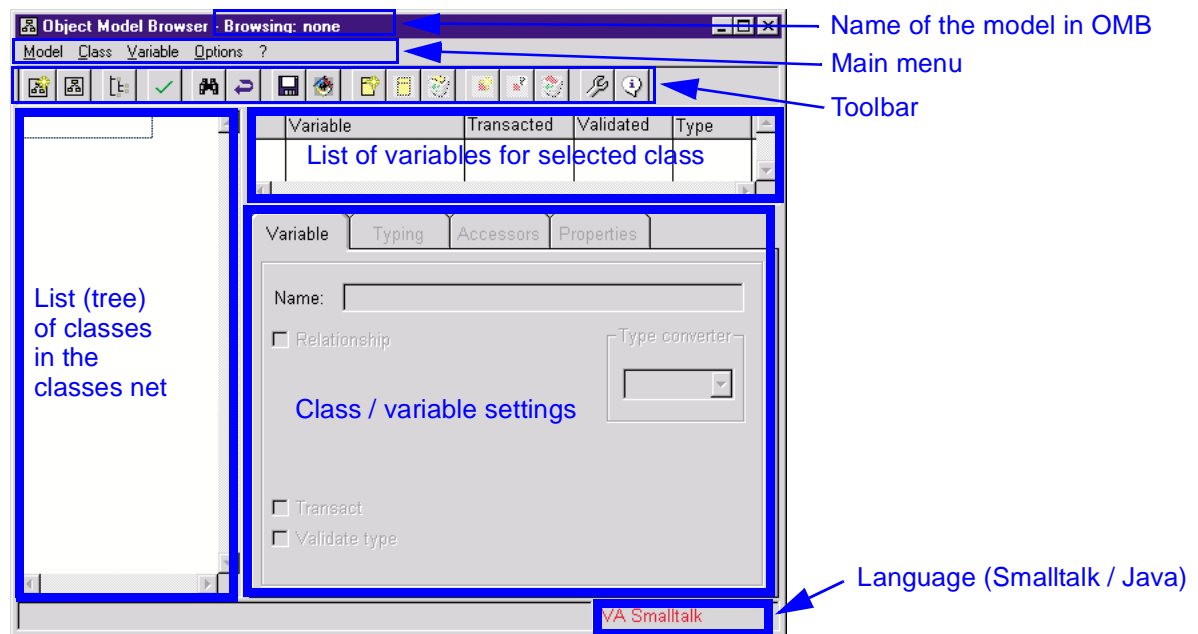


Figure 3.2.3. Major parts of the OMB dialog

#### 3.2.3.1. Main menu submenus

##### 3.2.3.1.1. Submenu Model

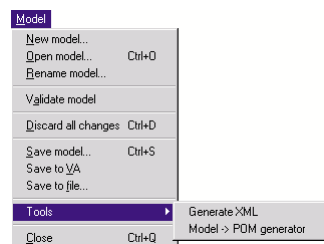


Figure 3.2.4. Submenu Model

###### 3.2.3.1.1.1. Model / New model...

Closes the open model (prompts to save if not saved).  
Prompts for the name of the new model (the model will be saved as name.ome).  
Opens the new (empty) model in OMB.

###### 3.2.3.1.1.2. Model / Open model... Ctrl+O

Closes the open model (prompts to save if not saved).  
Opens the **Open file...** dialog for selecting an .ome file.

###### 3.2.3.1.1.3. Model / Rename model...

Prompts for the new name of the model open in the OMB.

###### 3.2.3.1.1.4. Model / Validate model...

Validates (checks for errors) the ExtendedDescriptions for the classes in the model.

###### 3.2.3.1.1.5. Model / Discard all changes Ctrl+D

Validates (checks for errors) the ExtendedDescriptions for the classes in the model.

###### 3.2.3.1.1.6. Model / Save model... Ctrl+S

Saves the model to an .ome file. Note: If the .ome file already exists, a prompt appears asking whether or not to overwrite the file.

###### 3.2.3.1.1.7. Model / Save to VA

Saves the the Smalltalk content of the model (classes, variables, relationships, variable typing, etc.) to the Smalltalk image.

Note: The following can be saved:

- The entire contents of the model.
- Only the changes since the model was last saved to VA.

This is set by selecting Options / Preferences and then tab Save.

#### 3.2.3.1.1.8. Model / Save to file...

Opens a **Save to...** dialog. Clicking **OK** will then save the specification for each class to a .st file (with the same name as the name of the class).

#### 3.2.3.1.1.9. Model / Tools / Generate XML

Opens a **Save to...** dialog. Clicking **OK** will then save the specification for all classes to a single .xml file.

#### 3.2.3.1.1.10. Model / Tools / Model -> POM generator

This menu item is only applicable when using Java.

#### 3.2.3.1.1.11. Model / Close Ctrl-Q

Closes the OMB.

Note: If there are unsaved changes in the model: The **Save to...** dialog appears for saving the model.

### 3.2.3.1.2. Submenu Class

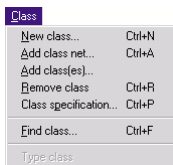


Figure 3.2.5. Submenu Class

The Class menu can also be opened as a context menu by right-clicking in the class list subdialog.

#### 3.2.3.1.2.1. Class / New class... Ctrl+N

Creates a subclass for the class currently selected. Opens the **Class specification** dialog for entering required and optional information about the new class (class name, etc.)

#### 3.2.3.1.2.2. Class / Add class net... Ctrl+A

Opens the **Information required** dialog. In response:

- Enter the name of a class that is in the Object Net.
- Click **OK**.
- **OR**
- Click **OK**. A list of available classes is displayed (the displayed classes are all subclasses of MicFwDomainObject).
- Double-click on a class in the list.

#### 3.2.3.1.2.3. Class / Add class(es)

Opens the **Information required** dialog. In response:

- Enter the name of a class that should be added to the model.
- Click **OK**.
- **OR**
- Click **OK**. A list of available classes is displayed (the displayed classes are all subclasses of MicFwDomainObject which are NOT already in the model).
- Double-click on a class in the list.

#### 3.2.3.1.2.4. Class / Remove class Ctrl+R

Removes the currently selected class from the model.

Note: If the selected class has subclasses in the model: The subclasses must be deleted before the class can be deleted.

#### 3.2.3.1.2.5. Class / Class specification... Ctrl+P



Opens the **Class specification** dialog for the selected class:

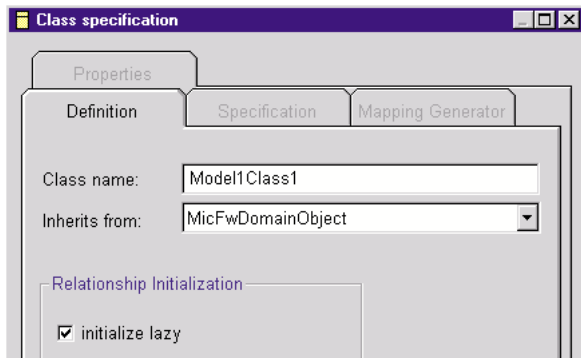


Figure 3.2.6. Class specification dialog

#### 3.2.3.1.2.6. Class / Find class... Ctrl+F

Opens the **Information required** dialog. In response:

- Enter the name of a class that should be found in the model.
- Click **OK**.

**OR**

- Click **OK**. A list of all classes in the model is displayed.
- Double-click on a class in the list.

#### 3.2.3.1.2.7. Class / Type class

Used for specifying the parent class of a class that has been imported from Java.

#### 3.2.3.1.3. Submenu Variable

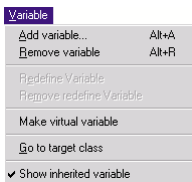


Figure 3.2.7. Submenu Variable

The Variable menu can also be opened as a context menu by right-clicking in the variable list subdialog.

##### 3.2.3.1.3.1. Variable / Add variable... Alt+A

Opens the **Information required** dialog. In response

- Enter the name of a class that is in the Object Net.
- Click **OK**.

A variable with the name is created for the class.

##### 3.2.3.1.3.2. Variable / Remove variable Alt+R

Deletes the selected variable from the class.

Note: If this option is greyed out: The selected variable is an inherited variable.

##### 3.2.3.1.3.3. Variable / Redefine Variable

The selected inherited variable is redefined (the variable is defined for the class (no longer inherited)).

##### 3.2.3.1.3.4. Variable / Remove redefine Variable

The definition in the selected class for the selected variable is removed. The variable is now defined by the definition in the parent class.

##### 3.2.3.1.3.5. Variable / Make virtual variable

The selected variable is created for the class without creating the Frameworks accessor methods.

Note: A variable can be made virtual only if the variable is:

- Not redefined in any subclasses.
- Not defined in any parent class.

##### 3.2.3.1.3.6. Variable / Go to target class

The target class for the variable is selected.

##### 3.2.3.1.3.7. Variable / Show inherited variable

If not checked: Inherited variables are not displayed.

Note: Redefined variables are displayed.

### 3.2.3.1.4. Submenu Options

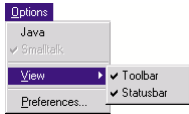


Figure 3.2.8. Submenu Options

#### 3.2.3.1.4.1. Options / Java

If checked: The OMB options are set for Java.

#### 3.2.3.1.4.2. Options / Smalltalk

If checked: The OMB options are set for Smalltalk.

#### 3.2.3.1.4.3. Options / View / Toolbar

If checked: The toolbar is displayed.

#### 3.2.3.1.4.4. Options / View / Statusbar

If checked: The status bar is displayed.

#### 3.2.3.1.4.5. Options / Preferences...

Opens the **Options** dialog.

### 3.2.3.1.5. Submenu ?



Figure 3.2.9. Submenu ?

#### 3.2.3.1.5.1. ? / About...

Displays information about PMS Micado.

## 3.2.3.2. Toolbar



Figure 3.2.10. OMB Toolbar

#### 3.2.3.2.1. New Model

Same as menu selection **Model / New model...**

#### 3.2.3.2.2. Open Model

Same as menu selection **Model / Open model...**

#### 3.2.3.2.3. Add class net

Same as menu selection **Class / Add class net....**

#### 3.2.3.2.4. Validate Model

Same as menu selection **Model / Validate model.**

#### 3.2.3.2.5. Find class

Same as menu selection **Class / Find class....**

#### 3.2.3.2.6. Discard all changes

Same as menu selection **Model / Discard all changes.**

#### 3.2.3.2.7. Save to file

Same as menu selection **Model / Save to file....**

### 3.2.3.2.8. Save to VA

Same as menu selection **Model / Save to VA**.

### 3.2.3.2.9. New class

Same as menu selection **Class / New class...**

### 3.2.3.2.10. Open class specification

Same as menu selection **Class / Class specification...**

### 3.2.3.2.11. Remove class

Same as menu selection **Class / Remove class**.

### 3.2.3.2.12. Add new variable

Same as menu selection **Variable / Add variable...**

### 3.2.3.2.13. Redefine variable

Same as menu selection **Variable / Redefine variable**.

### 3.2.3.2.14. Remove variable

Same as menu selection **Variable / Remove variable**.

### 3.2.3.2.15. Options

Same as menu selection **Options / Preferences...**

### 3.2.3.2.16. Info

Same as menu selection **? / About**.

## 3.2.3.3. Class tree

The class tree displays all objects in the object net in the model.

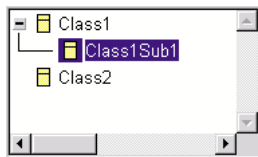


Figure 3.2.11. OMB classes tree

## 3.2.3.4. Variable list

The variable list display all variables for the selected class.





|   | Variable | Transacted | Validated | Type |
|---|----------|------------|-----------|------|
|  | var1     |            |           |      |
|  | var2     |            |           |      |
|  | var3     |            |           |      |
|  | var4     |            |           |      |

Figure 3.2.12. OMB variable list

## 3.2.3.5. Tab Variable

Tab **Variable** contains the following:

- Entry field for the **name** of the variable
- Checkbox for specifying if the variable is a simple type or **relationship**
- Drop-down list for selecting the **TypeConverter**.
- Checkbox for specifying if the variable is **transacted**.

- Checkbox for specifying if the variable type should be automatically converted and validated.

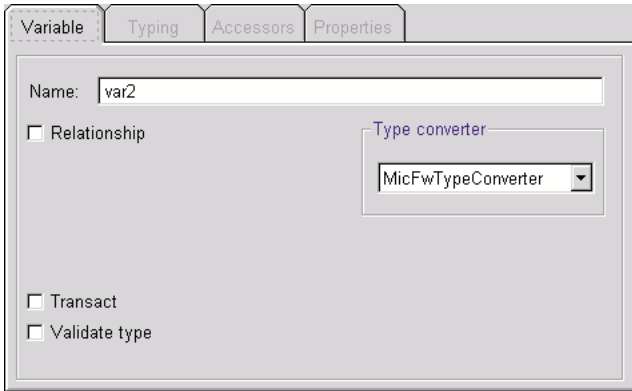


Figure 3.2.13. OMB tab Variable

### 3.2.3.6. Tab Typing (not relationship)

Tab **Typing** (not for relationship) contains the following:

- Drop-down list for selecting the **type** of the variable
- Other fields for specifying **available properties** of the selected type

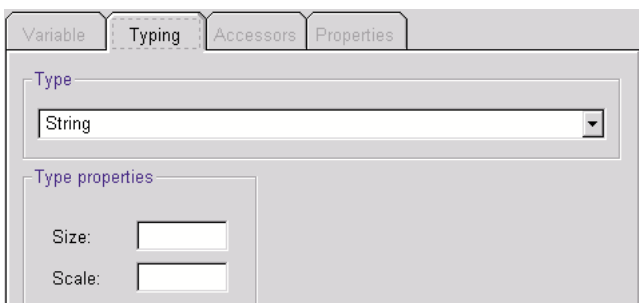


Figure 3.2.14. OMB tab typing (not relationship)

### 3.2.3.7. Tab Typing (relationship)

Tab **Typing** (for **relationship**) contains the following:

- Source class **minimum** and **maximum** selection fields. This specifies the minimum and maximum number of target class objects that can be referenced by the Source class variable.
- Checkbox **Primitive** for specifying a primitive relationship.
- Checkbox **Monitor target** for specifying if the target should be monitored.
- Checkbox **Persistent** for specifying if the source variable is persistent.
- Drop-down list for selecting the **Target class**.
- If the relationship is not primitive: Drop-down list for selecting the **target class variable**.
- Target class **minimum** and **maximum** selection fields. This specifies the minimum and maximum number of source class objects that can be referenced by the Target class variable.

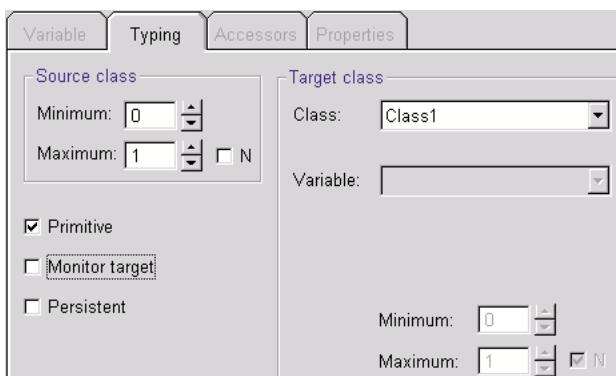


Figure 3.2.15. OMB tab typing (relationship)

### 3.2.3.8. Tab Accessors

Tab **Accessors** contains the following:

- Comboboxes for specifying the **names of the framework accessors** for a variable

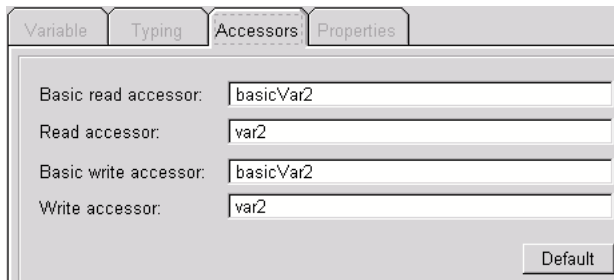


Figure 3.2.16. OMB tab Accessors

### 3.2.3.9. Tab Properties

Tab **Properties** contains the following:

- Radiobutton groups for specifying the values of variable properties



Figure 3.2.17. OMB tab Properties

### 3.2.3.10. Status bar

The **Status bar** contains the following:

- The class name.
- Smalltalk or Java (development environment)



Figure 3.2.18. OMB Status bar

### 3.2.3.11. Options dialog

#### 3.2.3.12. Tab Accessor prefixes

The **Options Tab Accessor prefixes** contains the following:

- Comboboxes for specifying the accessor default prefixes
- A checkbox for specifying that the names of all accessors should be changed to match the prefixes

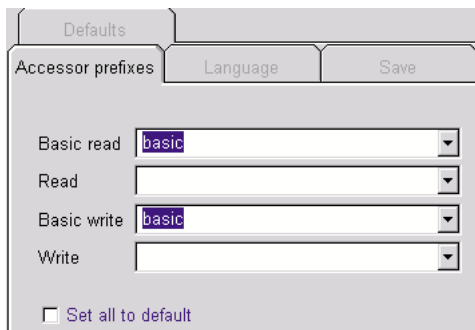


Figure 3.2.19. OMB dialog Options tab Accessor prefixes

#### 3.2.3.13. Tab Language

The **Options Tab Language** contains the following:

- A radiogroup for selecting the **language** of the environment

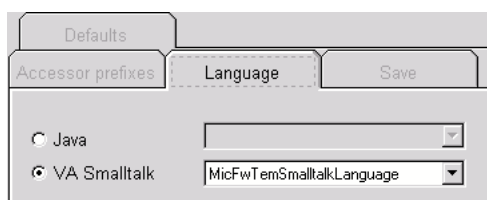


Figure 3.2.20. OMB dialog Options tab Language

### 3.2.3.14. Tab Save

The **Options Tab Save** contains the following:

- A radio group for specifying what is saved when saving to VA (**Save changes** will cause only those changes to be saved to VA that are new since the model was last saved to VA).

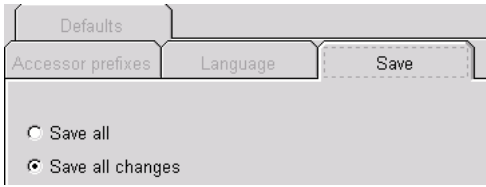


Figure 3.2.21. OMB dialog Options tab Save

### 3.2.3.15. Tab Defaults

The **Options Tab Defaults** contains the following:

- An entry field for specifying the **Default application**
- A radio group for specifying if a new class will be by default **subclass**ed from MicFwDomainObject or from the class currently selected in the OMB
- Checkbox for specifying that all classes in the model be **set to the default**

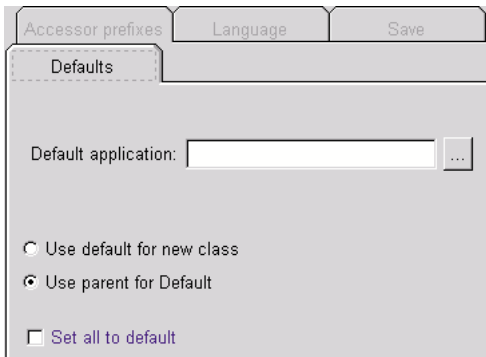


Figure 3.2.22. OMB dialog Options tab Defaults

---

## 3.3. Object Net Browser (NetBrowser)

---

### 3.3.1. Primary functions

The primary functions of the Net Browser are:

- Display a class's object net.
- Add/delete variables in a class.
- Specify a type or relationship for a class variable.
- Verify the validity of a variable's ExtendedDescription.
- Specify if a variable is a key.
- Specify if the variable type is automatically validated.
- Specify if the variable supports transaction/persistence.

Note: The Net Browser can only be opened on subclasses of MicFwApplicationObject.

---

### 3.3.2. Opening the Net Browser:

#### 3.3.2.1. From the System Transcript menu

- 2.1. In the System Transcript: Select **micFrameworks / Browse Object Net...**
- 2.2. In response to the message "Select root class": Enter **MicFwTutorialIdentifiedObject**.
- 2.3. Click **OK**.

The NetBrowser window is opened:

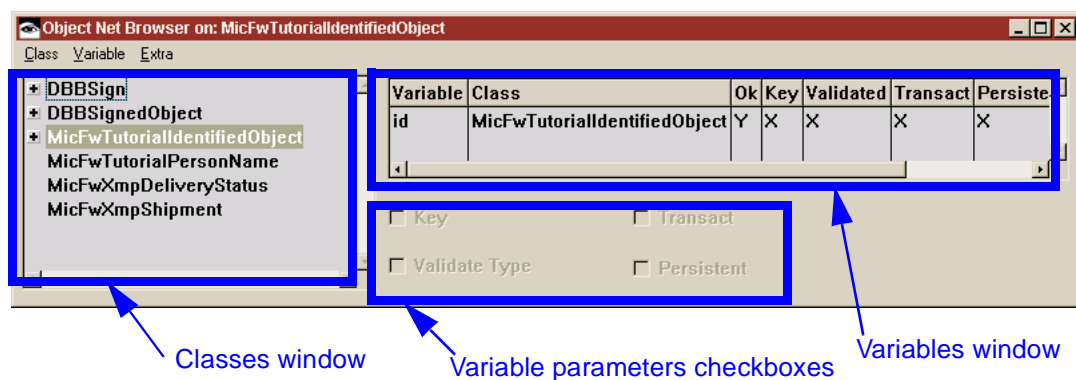


Figure 3.3.1. Object Net Browser dialog

#### 3.3.2.2. From a Transcript window

- 2.4. Execute the following line in a transcript window:

```
OBFC CH Open Net Browser.  
MicFwNetBrowser openBrowser
```

---

### 3.3.3. Classes window

The classes window list all classes and subclasses in the object net.

#### 3.3.3.1. Add MicFwTutorialIdentifiedObject subclasses

- 2.5. Add MicFwTutorialIdentifiedObject subclasses **AaaClass1** and **AaaClass2** (as shown earlier in this manual).
- 2.6. Refresh the classes window display by clicking on the plus sign to the left of "MicFwTutorialIdentifiedObject".

Note that the 2 new classes are displayed in the Net Browser, since they were subclassed from MicFwTutorialIdentifiedObject.

---

### 3.3.4. Net Browser submenu Class

- 2.7. Display the Class submenu by clicking on Class in the menu OR by right-clicking the mouse in the

classes window



Figure 3.3.2. ONB: Submenu Class

### 3.3.4.1. Find Alt+F

Find a class listed in the classes window.

- 2.8. Right click in the classes window.
- 2.9. Select **Find**.

### 3.3.4.2. Browse...

Browse a class listed in the classes window

- 2.10. Select a class in the classes window.
- 2.11. Right click in the classes window.
- 2.12. Select **Browse**.

### 3.3.4.3. Browse another net Ctrl+B

Browse a different object net

- 2.13. Right click in the classes window.
- 2.14. Select **Browse another net**.
- 2.15. In response to "Select root class": Enter the **root class** of the net you want to view.

### 3.3.4.4. Validate Descriptions

Validate ExtendedDescription for variables of all classes in the displayed net. This is not necessary if all changes to instance variable definitions were carried out within the the framework (Net Browser). However, if manual (ie, without using the Net Browser) changes are made, mistakes can occur. Therefore, if there is any doubt about the validity of the variable descriptions, the descriptions should be verified. To do this:

- 2.16. Right click in the classes window.
- 2.17. Select **Validate descriptions**.

### 3.3.4.5. Reset classes...

This option remove any unsaved changes for the selected class(es). **Important:** If a variable is deleted and the change was not saved, resetting the classes will not restore the variable.

- 2.18. In the classes window: Select **AaaClass1**.
- 2.19. Click on **id**.
- 2.20. Uncheck the checkbox **Key**.
- 2.21. In response to the question: "Do you want to redefine the description...": Click **Yes**. Note that **id** is now displaed with a black font, since it is no longer inherited from a superclass.
- 2.22. Right click in the classes window.
- 2.23. Select **Reset Classes**.
- 2.24. A list of all classes which have unsaved changes is displayed. Double-click on **AaaClass1**.  
Note that the changes have been removed.

### 3.3.4.6. Reset all classes... Alt+C

Similar to **Reset classes...**, except that all unsaved changes in all classes are removed without prompting for the class names.

### 3.3.4.7. Save Ctrl+S

Saves all changes that have been made since the last change.

**Important:** The display in Net Browser does not reflect the actual status of the objects. Although changes are displayed in the Net Browser without saving, no changes are implemented until saved.



### 3.3.5. Net Browser submenu Variable

2.25. Display the Variable submenu by clicking on Variable in the menu OR by right-clicking the mouse in the variables window

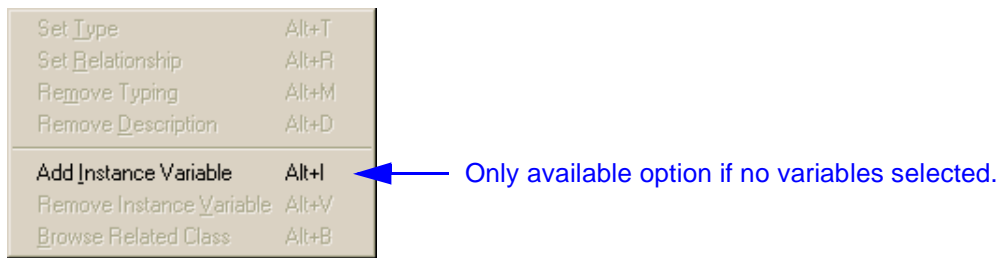


Figure 3.3.3. ONB: Submenu Variable

#### 3.3.5.1. Add Instance Variable Alt+I

Add an instance variable within the framework for the class selected in the classes window.

- 2.26. In the classes window: Select **AaaClass1**.
- 2.27. Right-click in the variables window.
- 2.28. Select **Add Instance Variable**.
- 2.29. Enter **class1var1** as the variable name.
- 2.30. Click **OK**.

Note that the new variable is displayed in the variables window.

#### 3.3.5.2. Set type Alt+T

Specify instance variable type as simple type.

- 2.31. Select **class1var1**.
- 2.32. Right click in the variables window. Note the enabled options in the menu:

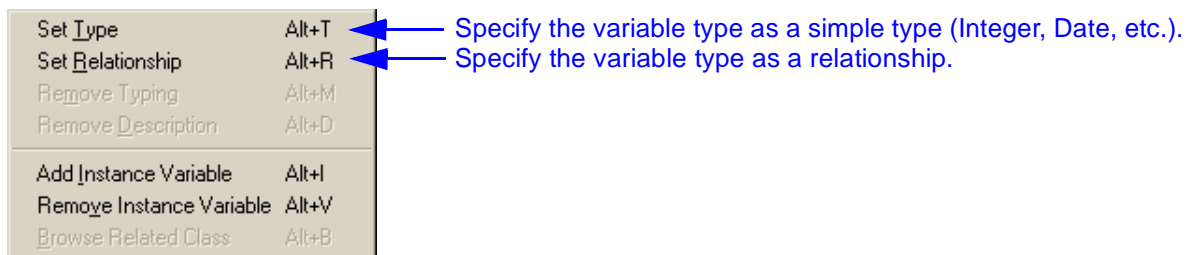


Figure 3.3.4. ONB pop-up dialog: Setting variable type

- 2.33. Select **Set Type**. The **Type Editor on <class1var1>** is opened (the Type Editor will be explained in detail later in this part).
- 2.34. Double-click on **Integer**.

Note in the variables window that the type is now "Integer".

#### 3.3.5.3. Remove Typing Alt+M

- 2.35. Select **class1var1**.
- 2.36. Right click in the variables window. Note the enabled options in the menu:

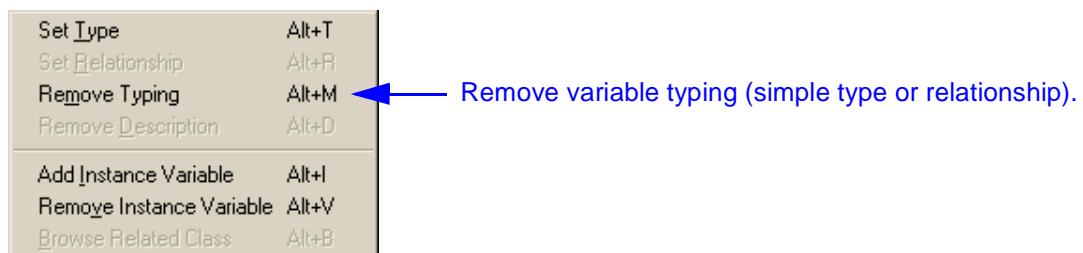


Figure 3.3.5. ONB pop-up dialog: Removing variable type

2.37. Select **Remove Typing**. Note in the variables window that there is now no type.

### 3.3.5.4. Set Relationships Alt+R

2.38. Select **class1var1**.

2.39. Right click in the variables window.

2.40. Select **SetRelationship**.

2.41. The **Relationship Editor on variable <class1var1>** appears. The Relationship Editor will be described later in this part.

2.42. In the **Target class** drop-list box: Select **AaaClass2**.

2.43. Check the checkbox **Primitive**.

2.44. Click **OK**.

Note in the variables window that the type is now "Relationship ->1" and the target is AaaClass2.

### 3.3.5.5. Browse Related Class Alt+B

When a variable is typed as a relationship, the class referenced by that variable is referred to as the "related class".

2.45. Select **class1var1**.

2.46. Right click in the variables window. Note the enabled options in the menu:

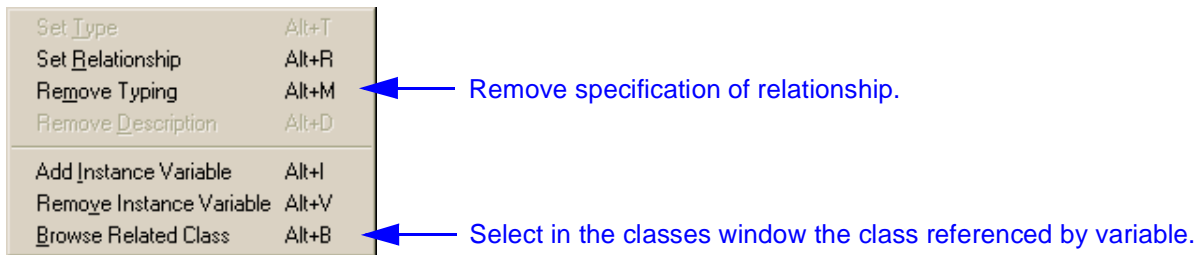


Figure 3.3.6. ONB pop-up dialog: Browse related class

2.47. Select **Browse Related Class**.

AaaClass2 is now selected in the classes window. Note that AaaClass2 is the target of a primitive relationship. Therefore, you cannot browse to AaaClass1 from AaaClass2.

### 3.3.5.6. Remove Description Alt+D

If an inherited variable is redefined in a subclass, a MicFwVariableDescription object is added to the ExtendedDescription for the subclass. This variable description can be removed and the inherited definition of the variable restored by selecting **Remove Description**.

2.48. Select **AaaClass1**.

2.49. Select **id**.

2.50. Uncheck checkbox **Key**.

2.51. In response to "Do you want to redefine the description of...": Click **Yes**.

2.52. Right click in the variables window. Note the enabled options in the menu:

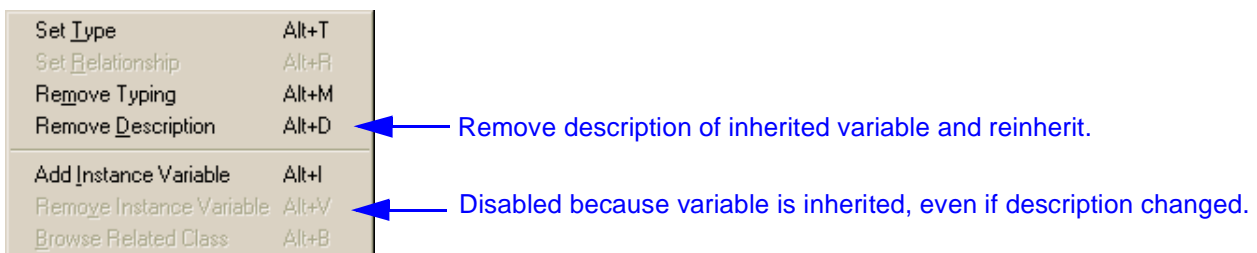


Figure 3.3.7. ONB pop-up dialog: Remove description

2.53. Select **Remove Description**.

2.54. In response to "Remove the redefinition of id and inherit the definition of the superclass?": Click **Yes**.

Note that the inherited definition has been restored.

### 3.3.5.7. Remove Instance Variable Alt+V

Only non-inherited instance variables can be deleted.

2.55. Select **AaaClass1**.

2.56. Select **class1var1**.

2.57. Right click.

2.58. Select **Remove Instance Variable**.

2.59. In response to "Do you really want to remove (class1var1) from AaaClass1?": Click **Yes**.

Note that the variable has been deleted from the variables window.

---

## 3.3.6. Variable parameters

### 3.3.6.1. Key

If the **Key** checkbox is checked, the variable is a key for the class. All key variables for an instance of the class must uniquely identify the instance (ie, no 2 instances can have the same values for all key values).

#### During runtime

The key parameter has no effect during runtime. It is used only by the Persistence Framework (e.g., by the POM-Generator).

### 3.3.6.2. Validate Type

If checked, **Validate Type** will cause automatic validation of the type of an object assigned to the variable. If the object is not of the required type and cannot be converted to the required type, an exception is generated.

#### For relationships

The type of assigned target object and the cardinality will be checked immediately.

### 3.3.6.3. Transact

If checked, **Transact** allows the variable to be transacted.

**Important:** Changes to a variable will not be transacted if the variable is not marked as transacted.

### 3.3.6.4. Persistent

If checked, **Persistent** provides the variable with capability for persistence.

Note: If **Persistent** is checked, **Transact** is also checked automatically.

---

## 3.4. Type Editor

---

### 3.4.1. Primary functions

The primary functions of the Type Editor are to select:

- A simple type for the variable from a list of available simple types.
- The type converter for the variable.
- If applicable to the selected variable type:
  - Size (for String, etc.)
  - Scale (precision for Float, etc.)

---

### 3.4.2. Opening the Type Editor

There are 2 ways to open the Type Editor from the Net Browser:

- Double-clicking on a variable with a simple type.
- Selecting **Set Type Alt+T** from the variable window pop-up menu.

The Type Editor window is opened:

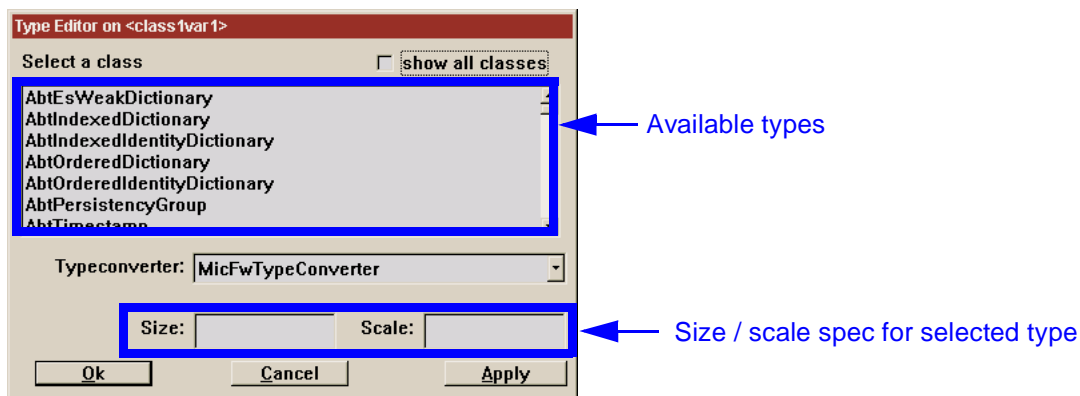


Figure 3.4.1. Type Editor

---

### 3.4.3. Fields / checkboxes in the type editor

#### 3.4.3.1. show all classes

Check this checkbox to list all classes as available types.

Uncheck this checkbox to list only those classes that are normally selected as simple types.

#### 3.4.3.2. Available types

The **Select a class** drop-list box displays the list of classes from which a variable type can be selected. There are 2 types of lists:

- With **show all classes** NOT checked: The classes compose a set of classes that are considered as normal types (Integer, Date, etc.).
- With **show all classes** checked: All subclasses of MicFwApplicationObject are displayed. This checkbox is not normally checked in order to shorten the list of types.

#### 3.4.3.3. Type converter

The type converter for this variable. The type converter converts any object assigned to the variable that is not of the type specified required by the variable. If the type converter cannot convert an assigned object, an exception is generated.

#### 3.4.3.4. Type parameters

The type parameters are required for certain types. For example:

- Size could be specified for a String object as 20 (characters).
- Scale could be specified for a Float object as 5 (5 decimal points accuracy).

## 3.5. Relationship Editor

### 3.5.1. Primary functions

The primary functions of the Relationship Editor are:

- Select the type of relationship (primitive or bidirectional).
- Select the target class.
- If bidirectional relationship: Select target class variable that will reference source class.
- Specify cardinality of target class.
- If bidirectional relationship: Specify cardinality of source class.
- Specify type converter.
- Specify if target should be monitored.

### 3.5.2. Opening the Relationship Editor

The Relationship Editor can be opened from the NetBrowser in 1 of 2 ways:

- Select a variable. Right click. Select **Set Relationship** from the pop-up window.
- Double-click on a variable that already has a relationship as its type.

The class of the variable that the Relationship Editor was opened on is always the **source** class.

#### 3.5.2.1. Open the relationship editor:

- 3.1. Open the Net Browser.
- 3.2. Create variables `AaaClass1>>class1var1` and `AaaClass2>>class2var1`.
- 3.3. Select `AaaClass1>>class1var1`.
- 3.4. Right-click.
- 3.5. Select **Set Relationship**. The Relationship Editor is opened with `AaaClass1` as the Source class and `class1var1` as the source relationship variable.

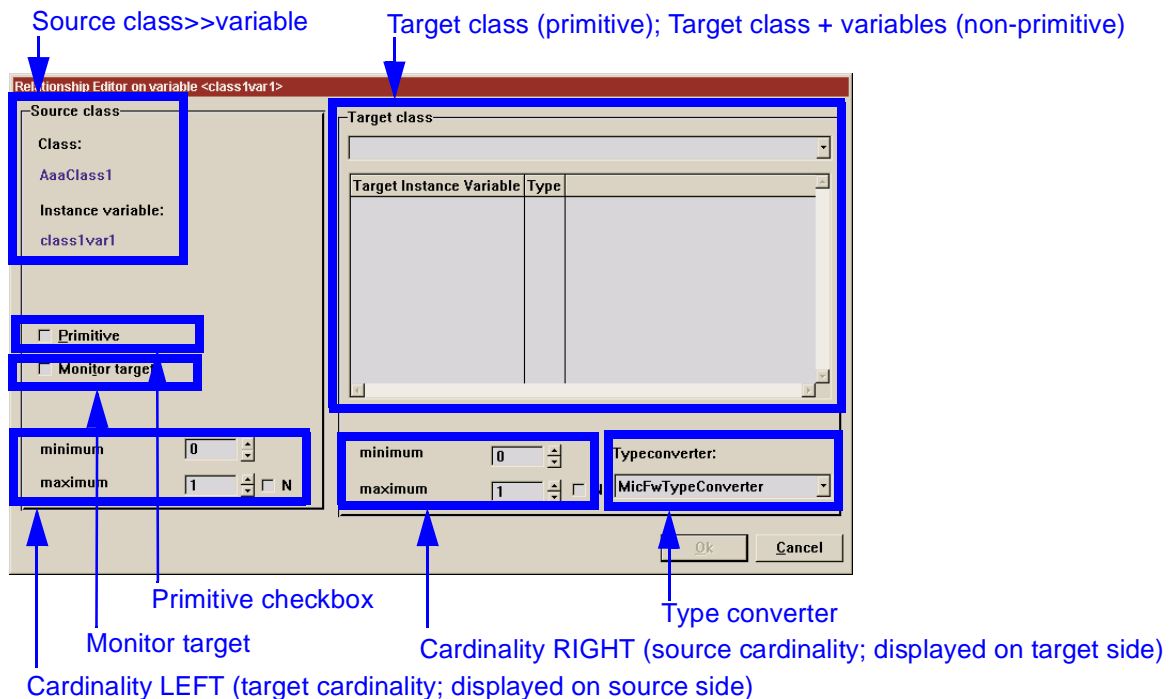


Figure 3.5.1. Relationship editor

### 3.5.3. Fields in the Relationship Editor

#### 3.5.3.1. Source class>>variable

The source class and instance variable is always the class>>variable that the Relationship Editor was opened on. If you open a second Relationship Editor window on the class>>variable that is displayed in one Relationship Editor window as the target, then the source and target class>>variables will be reversed.

### 3.5.3.2. Target class + variables window

The target class and all target class variables are displayed in this window. If a target class>>variable is selected, then it has been already defined as the target variable in the relationship with the source class>>variable.

#### 3.5.3.2.1. Select target and target>>variable

3.6. From the "Target class" drop-list box: Select **AaaClass2**. Note that in the "Target Instance Variable" list that variables `id` and `class2var1` as displayed.

### 3.5.3.3. Primitive checkbox

This checkbox will determine if the relationship is primitive or non-primitive (bi-directional).

#### 3.5.3.3.1. Select primitive

3.7. Check the **Primitive** checkbox. Note that the target variables are no longer displayed and that source cardinality (cardinality RIGHT) selectors are disabled.

#### 3.5.3.3.2. Deselect primitive

3.8. Uncheck the **Primitive** checkbox. Note that the target variables are redisplayed and that source cardinality (cardinality RIGHT) selectors are enabled.

### 3.5.3.4. Monitor Target

Monitor target is only supported for to-1 relationships.

This flag is necessary for the aggregated 1-to-1 relationship, where it will be set automatically when mapping an aggregated relationship. For the other to-1-relationships it is optional.

The target object creates a change-event connection to the relationship, so that the other side of the relationship knows, when the target object changed its value. Normally, if a (unidirectional) primitive relationship is mapped, no event connection is created between the relationship proxy and the target object so that changes to the target object can not be detected and propagated to the source object if necessary.

The use of the **#monitor target** flag creates the event connection. For a aggregated 1-to-1 relationship, you must set the `#monitor target` flag, so that changes to the source object can be detected by the FW.

### 3.5.3.5. Target cardinality (cardinality LEFT)

The target cardinality determines the min and max number of target objects the source>>variable should reference. If N is checked, then the max number of target objects is unlimited.

### 3.5.3.6. Source cardinality (cardinality RIGHT)

The source cardinality determines the min and max number of source objects the target>>variable should reference. If N is checked, then the max number of source objects is unlimited.

This option is only available if checkbox **Primitive** is not checked.

### 3.5.3.7. Typeconverter

The type converter for the object referenced by the source>>variable and (non-primitive only) the target>>variable.

---

## 3.6. Transaction Browser

---

### 3.6.1. Primary functions

The primary functions of the Transaction Browser are:

- Display:
  - All contexts
  - Transaction levels (within the selected context)
  - Versioned objects (within the selected transaction level)
  - Transacted variables (within the selected versioned object)
  - Actual or transacted (actual + transacted changes) object (referenced by the selected transacted variable)
- Commit or abort entire contexts or certain transaction levels within a context.

---

### 3.6.2. Opening the Transaction Browser

The Transaction Browser can be opened 2 ways:

- In the System Transcript: Select **micFrameworks / Browse Transactions** from the pop-up window.
- With the following message statement:

```
MicFwTransactionsBrowser openBrowser.
```

#### 3.6.2.1. Open the Transaction Browser:

- 4.1. Execute the following code segment to generate example contexts and transaction levels and open the transaction browser:

```
OBF CH Create transactions, open browser."  
[  
  MicFwTransactionManager abortAll.  
  Smalltalk removeKey: #CONTEXT1 ifAbsent: [].  
  Smalltalk removeKey: #CONTEXT2 ifAbsent: [].  
  Smalltalk removeKey: #Context1Instance ifAbsent: [].  
  Smalltalk removeKey: #Context2Instance ifAbsent: [].  
  
  Smalltalk at: #CONTEXT1 put: MicFwTransactionManager newTransactionContext.  
  Smalltalk at: #CONTEXT2 put: MicFwTransactionManager newTransactionContext.  
  Smalltalk at: #Context1Instance put: SimpleClass new.  
  Smalltalk at: #Context2Instance put: SimpleClass new.  
]  
[  
  Context1Instance simpleString: 'Context1Level0'.  
  Context2Instance simpleString: 'Context2Level0'.  
  CONTEXT1 beginTransaction.  
  Context1Instance simpleString: 'Context1Level1'.  
  CONTEXT2 beginTransaction.  
  Context2Instance simpleString: 'Context2Level1'.  
  CONTEXT2 beginTransaction.  
  Context2Instance simpleString: 'Context2Level2'.  
  MicFwTransactionsBrowser openBrowser.  
]
```

The transaction browser is opened. The following diagram explains the fields in the browser:

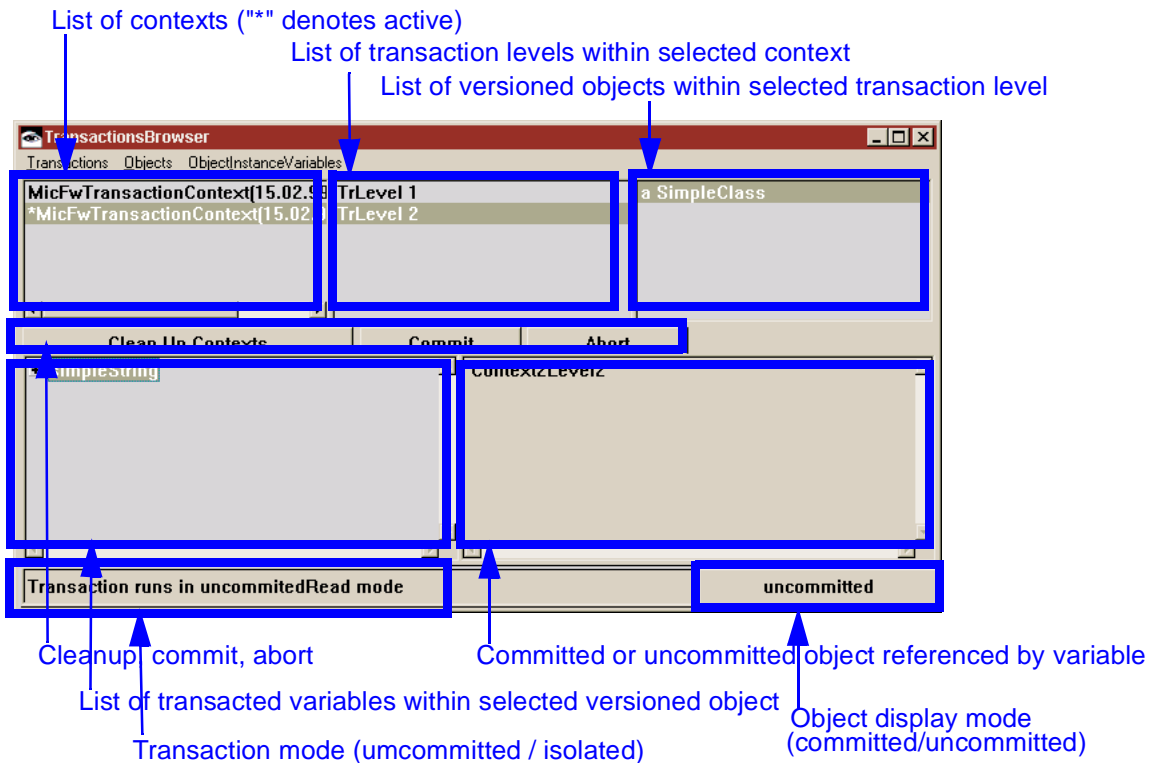


Figure 3.6.1. Transaction Browser

### 3.6.3. Submenu Transactions

4.2. Display the submenu Transactions by clicking in the menu OR by right-clicking the mouse in the List of contexts.

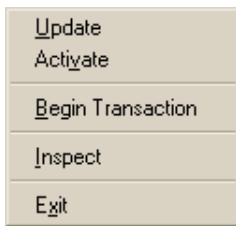


Figure 3.6.2. Transaction Browser: Submenu Transactions

#### 3.6.3.1. Update

The Transaction Browser display is not automatically updated. You can select this menu item to update the display.

Note: The display can also be updated by selecting a different window and reselecting the transaction browser window.

#### 3.6.3.2. Activate/Deactivate

Select this menu item to toggle the active status of the selected context. Remember these points when using this menu selection:

- Selecting when the active context is selected will deactivate the context and not activate any other context (ie, no context is active).
- Selecting when an inactive context is selected will cause the active context to be deactivated and the selected context to be activated.

#### 3.6.3.3. Begin Transaction

Select this menu item to create a new transaction level in the **selected** (not active) context. Not only will a new transaction level be created, but the selected context will become the active context.

#### 3.6.3.4. Inspect

Select this menu item to open an inspector on the selected context (a MicFwTransactionContext object).



### 3.6.3.5. Exit

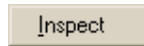
Select to exit the transaction browser. This has no effect on the contexts.

---

## 3.6.4. Submenu Objects

4.3. To display the submenu Objects:

- 4.3.1 Select a context.
- 4.3.2 Select a transaction level.
- 4.3.3 Select a versioned object.
- 4.3.4 Select **Objects** OR right-click the mouse in list of versioned objects.



### 3.6.4.1. Inspect

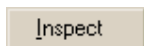
Select to open an inspector on the selected object.

---

## 3.6.5. Submenu ObjectInstanceVariables

4.4. To display the submenu ObjectInstanceVariables:

- 4.4.1 Select a context.
- 4.4.2 Select a transaction level.
- 4.4.3 Select a versioned object.
- 4.4.4 Select a transacted instance variable.
- 4.4.5 Select **ObjectInstanceVariables** OR right-click the mouse in list of instance variables.



### 3.6.5.1. Inspect

Select to open an inspector on the selected instance variable.

---

## 3.6.6. Context / TransactionLevel fields / buttons

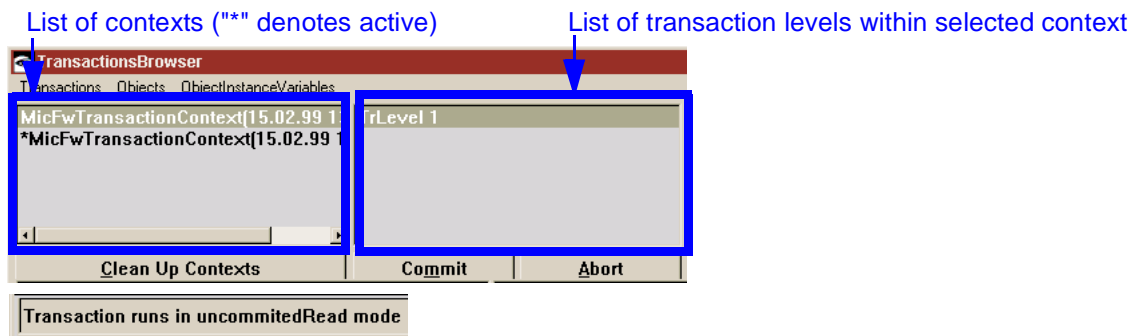


Figure 3.6.3. Transaction Browser: Context / TransactionLevel fields / buttons

### 3.6.6.1. List of contexts

Displays a list of all running contexts (the single active context is marked with "\*" (asterisk)).

### 3.6.6.2. List of transaction levels

Displays a list of transaction levels in the selected context.

### 3.6.6.3. Clean Up Contexts

Click this button to abort all displayed contexts.

### 3.6.6.4. Commit

Click this button to commit the highest transaction level in the selected context (the level itself does not have to be selected). If the selected context has only transaction level 1, then the context will be committed.

### 3.6.6.5. Abort

Click this button to abort the highest transaction level in the selected context (the level itself does not have

to be selected). If the selected context has only transaction level 1, then the context will be aborted.

### 3.6.6.6. Transaction mode

This field displays the transaction mode of the selected context (uncommitted or isolate).

---

### 3.6.7. Versioned objects fields / buttons

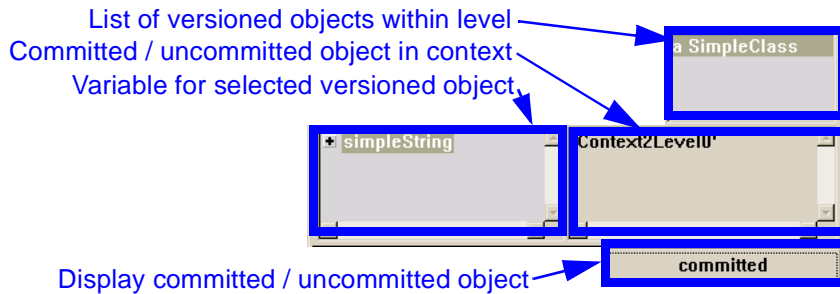


Figure 3.6.4. Transaction Browser: Versioned objects fields / buttons

#### 3.6.7.1. List of versioned objects within level

Displays a list of all objects that have been modified within the selected context and transaction level.

#### 3.6.7.2. Variable for selected versioned object

Displays the (transacted) variable that references the the versioned object.

#### 3.6.7.3. Committed / uncommitted object in context

Displays the actual object (if committed selected) or the transacted version of the object (if uncommitted selected).

#### 3.6.7.4. Display committed / uncommitted object

Toggle this button to display the actual object (committed) or the transacted version of the object (uncommitted). For example, clicking on the **committed** button in the above diagram would display the transacted object:



Figure 3.6.5. Transaction Browser: Display committed / uncommitted object button

# Appendix A

## API



The following is a list of API methods for the Object Behavior Framework.

*MicFwTransactionContext*>>

### **abortToTop**

Abort all child contexts of receiver and then abort all transaction levels in receiver.

*MicFwTransactionContext*>>

### **abortTransaction**

Abort highest transaction level in receiver. If TrLevel1 aborted, then abort all child contexts and receiver.

*MicFwTransactionContext*>>

### **activate**

Receiver becomes the single active context.

*MicFwTransactionContext*>>

### **allCurrentChanges**

Answer a Dictionary with all changes on the current level in the receiver. The changed objects appear as keys, their version containers as values

*MicFwApplicationObject*>>

### **allRelationships**

Answer a collection with the relationship objects for all relationships in the receiver.

*MicOvmObjectVersion*>>

### **allVersionedAspects**

Answer a collection with all aspect identifiers in the receiver.

*MicFwNRelationship*>>

### **any**

Answer a target object or <nil>. If there are more than one target objects in the collection, it is not ensured, that multiple calls of this method would return the same target object.

*MicFwApplicationObject*>>

### **assertTypeValid**

Check whether the receiver conforms to the type declarations that were made on the class level. If any check fails, signal a MicFwTypingError.

*MicFwApplicationObject*>>

### **attributeNamed: instVarSymbol**

Answer the value of attribute @instVarSymbol by performing the basic read accessor.

*MicFwApplicationObject*>>

### **attributeNamed: instVarSymbol put: newValue**

Write access to instance variable @instVarSymbol with the new value @newValue.

*MicFwTransactionContext*>>

### **beginTransaction**

Begin a new transaction level.

*MicFwTransactionContext*>>

### **committedRead**

Set the mode of the receiver to #isolate. No uncommitted changes from other transaction contexts will be visible through this context.

*MicFwTransactionContext*>>

### **commitToTop**

Commit all transaction levels of receiver and then commit all child contexts of receiver.

*MicFwTransactionContext*>>

### **commitTransaction**

Attempt to commit the current transaction level. Let all interested parties prepare for the commit if necessary (#aboutToCommit). If the method answers true, proceed. Go to the next lower transaction level.

*MicFwTransactedObject*>>

### **commitVersion: aMicFwObjectVersion inContext: aTransactionContext**

Notify the receiver about a commit of @aMicFwObjectVersion in context @aTransactionContext. Trigger the @commit event on a final commit. The commit has already been completed and @aTransactionContext's transactionLevel is already decremented.

*MicFwTransactedObject*>>

## **commitVersion: aMicFwObjectVersion inContext: aTransactionContext inRelationship: aMicFwRelationship**

Commit the receiver's relationship @aMicFwRelationship to the version contained in aMicFwObjectVersion.

*MicFwApplicationObject*>>

### **convertValues**

Make the receiver conform to the type declarations that were made on the class level by sending #as<type> to all declared instance variables. If any conversion fails (i.e. message not understood), the normal Error exception will result. If any conversion results in a value of wrong size, signal a MicFwSizeError.

*MicFwTransactionContext*>>

### **couldLock: anAspect in: anObject**

Answer true if a lock could be obtained for @anAspect in @anObject (i.e. no other context holds a version), false otherwise.

*MicFwTransactionContext class*>>

### **current**

Answer the current transaction context or nil if none active. This is an alternative to MicFwTransactionManager global currentContext.

*MicFwTransactionManager*>>

### **currentContext**

Return the active context.

*MicFwProcessOrientedTransactionManager*>>

### **currentContext**

Answer the current context of the current smalltalk process.

*MicFwTransactionManager*>>

### **currentContext**

Answer the current context.

*MicFwTransactionManager class*>>

### **currentContext**

Answer the contexts that is currently active.

*MicFwTransactionManager class*>>

### **currentContext**

Return active context (or nil).

*MicFwTransactionContext*>>

### **deactivate**

If the receiver is the single active context: Deactivate the receiver (ie, no context is active).

*MicFwApplicationObject class*>>

### **defaultTypeConverter**

Answer the default type converter for the receiver. Can be overriding by subclasses.

*MicFw1Relationship*>>

### **delete**

Delete the relationship maintained by the receiver. Also delete the inverse references.

*MicFwNRelationship*>>

### **delete**

Delete the relationship maintained by the receiver. Also delete the inverse references.

*MicFwRelationship*>>

### **delete**

Delete the relationship maintained by the receiver, i.e. remove all references to objects. Also delete the inverse references

*MicFwTrNRelationship*>>

### **delete**

Delete the relationship maintained by the receiver. Also delete the inverse references

*MicFw1Relationship*>>

### **getTarget**

Returns the target object of the ->1 relationship.

*MicFwNRelationship>>*

**getTarget**

Returns a Collection of target objects referenced by the ->N relationship.

*MicFwRelationship>>*

**getTarget**

Returns the target object (the actual object or a collection, depending on the type of relationship) of the relationship.

*MicFwTr1Relationship>>*

**getTarget**

Returns the transacted target object of the ->1 relationship.

*MicFw1Relationship>>*

**getTargetNoTr**

For an non-transacted relationship: Returns the object referenced by the ->1 relationship. For a transacted relationship: Returns the variable value (not the transacted object).

*MicFwNRelationship>>*

**getTargetNoTr**

For an non-transacted relationship: Returns the object version of the Collection of objects referenced by the ->N relationship. For a transacted relationship: Returns the variable value of the Collection (not the transacted object).

*MicFwRelationship>>*

**getTargetNoTr**

For an non-transacted relationship: Returns the object referenced by the relationship. For a transacted relationship: Returns the variable value (not the transacted object).

*MicFwTransactionManager class>>*

**global**

Return single instance of receiver.

*MicFwTransactionManager>>*

**hasCurrentContext**

Return True if there is a active context.

*MicFwQProjectionResult>>*

**hasOrder**

Answer true if the receiver as an order definition.

*MicFwApplicationObject class>>*

**hasRedefinitionFor: thisInstVar**

Answer true if the receiver redefined the typing/description for instance variable @thisInstVar, else answer false (= no description found/description is inherited/own variable).

*MicFwTransactionManager>>*

**hasRunningTransaction**

Return True if the active context has a TrLevel1.

*MicOvmObjectVersion>>*

**hasVersionFor: anAspect**

Answer true if the receiver contains a versioned value for <anAspect>, false otherwise.

*MicFwDomainObject class>>*

**initializeValidation**

Class method that sends all validateWrite:using: messages for all instance variables of the class that should be validated.

*MicFwRelationship>>*

**is1ToN**

Answer whether the receiver describes a 1-to-N relationship.

*MicFwTransactionContext>>*

**isActive**

Answers True if receiver is the single active context.

*MicFwApplicationObject>>*

**isAnyModified**

Check if the current transaction context (if any) holds a version for the receiver or for an relationship that originates from the receiver.

*MicFwTransactionContext>>*

### **isIsolated**

Answers True if receiver mode is isolated (not uncommittedRead).

*MicFwRelationship>>*

### **isMicFwRelationship**

Answer whether the receiver is a relationship object.

*MicFwTransactedObject>>*

### **isModified**

Check if the current transaction context (if any) holds a version for the receiver.

*MicFwRelationship>>*

### **isNToM**

Answer whether the receiver describes a N-to-M relationship.

*MicFwTransactionContext>>*

### **isolate**

Set the mode of the receiver to #isolate. No uncommitted changes from other transaction contexts will be visible through this context.

*MicFwTransactionContext>>*

### **isolateFor: aBlock**

temporarily isolate the receiver context while executing <aBlock>. Reset to the previously isolation afterwards. Answer the result of <aBlock>.

*MicFwRelationship>>*

### **isPrimitive**

Answer whether the receiver is a primitive (one-sided) relationship object.

*MicFwApplicationObject>>*

### **isReIValid**

Check whether the receiver conforms to the relationship cardinality declarations that were made on the class level. If any check fails, answer false. Otherwise, answer true.

*MicFwTransactionContext>>*

### **isRunning**

Answer whether the receiver is in an uncommitted state.

*MicFw1Relationship>>*

### **isTo1**

Answer whether the receiver describes a to-1 relationship.

*MicFwRelationship>>*

### **isTo1**

Answer whether the receiver describes a to-1 relationship.

*MicFwNRelationship>>*

### **isToN**

Answer whether the receiver describes a to-N relationship.

*MicFwRelationship>>*

### **isToN**

Answer whether the receiver describes a to-N relationship.

*MicFwApplicationObject class>>*

### **isTypable**

Answer whether the receiver contain type declaration.

*MicFwApplicationObject>>*

### **isTypeValid**

Check whether the receiver conforms to the type declarations that were made on the class level. If any check fails, answer false. Otherwise, answer true.

*MicFw1Relationship>>*

### **isValid**

Answer true if the receiver conforms to the declared cardinality constraints, false otherwise.



*MicFwNRelationship*>>

**isValid**

Answer true if the receiver conforms to the declared cardinality constraints, false otherwise.

*MicFwRelationship*>>

**isValid**

Answer true if the receiver conforms to the declared cardinality constraints, false otherwise.

*MicFwApplicationObject class*>>

**newNoTransact**

Create an instance of the receiver that is not sensitive to transaction handling.

*MicFwDomainObject*>>

**newPersistent**

Creates a persistent instance of the DO.

*MicFwTransactionContext*>>

**newTransactionContext**

Create new context as child of receiver.

*MicFwTransactionManager*>>

**newTransactionContext**

Create a new `TransactionContext` with this manager as its owner and manager. This context isn't activated automatically but can be activated by sending `#beginTransaction` to the context.

*MicFwTransactionManager*>>

**newTransactionContext**

Create a new `TransactionContext` with this manager as its owner and manager. This context isn't activated automatically but can be activated by sending `#beginTransaction` to the context.

*MicOvmObjectVersion*>>

**newValueFor: anAspect**

Answer the new value stored for `<anAspect>`.

*MicFwQArrayParser*>>

**nextPutMessage: aSymbol**

*MicFwNRelationship*>>

**orderAscending**

Define the receiver to be sorted on the attributes named in `@arrayOfSelectors` in ascending order. The attributes are assumed to be base attributes (not relationships).

*MicFwNRelationship*>>

**orderBy: selectorsOrClause**

Define the sort order for the receiver. `@selectorsOrClause` can be either an array of selectors, optionally postfixed with `#asc` or `#desc` designators, or a previously created `MicFwOrderClause`. Answer the order-  
Clause.

*MicFwNRelationship*>>

**orderDescending: arrayOfSelectors**

Define the receiver to be sorted on the attributes named in `@arrayOfSelectors` in descending order. The attributes are assumed to be base attributes (not relationships).

*MicOvmObjectVersion*>>

**recordNewVersion: aValue for: anAspect**

Set `@aValue` as the new value for attribute `@anAspect`.

*MicFwApplicationObject class*>>

**relationshipFor: aSymbol**

Answer a relationship for attribute `<aSymbol>`, or `<nil>` if none.

*MicFw1Relationship*>>

**releaseTarget**

Remove the committed target from the receiver without any transacted changes.

*MicFwNRelationship*>>

**releaseTarget**

Remove the committed target from the receiver without any transacted changes.

*MicFwRelationship*>>

**releaseTarget**

Remove the committed target from the receiver without any transacted changes.

*MicFwNRelationship*>>

**removeAll**

Obsolete. Remove all objects from the relationship maintained by the receiver. Also delete the inverse references.

*MicFwNRelationship*>>

**removeOrder**

The order clause for the handled object was removed. Remove the sorted order from the target collection.

*MicFwTransactionContext*>>

**removeVersionsFor: anAspect in: anObject**

Remove all version information for `anAspect` in `anObject` in all transaction levels. Information will also be removed in child contexts. @param anObject the object containing the desired aspect @param anAspect the aspect for which to remove version information.

*MicFwTransactionContext*>>

**removeVersionsFor: anObject**

Remove all version information accumulated for `anObject`. Information will also be removed in child contexts. @param anObject the object for which to remove version information @return true when a version has been removed in one of the contexts.

*MicFwTransactionContext class*>>

**running**

Answer the currently running transaction context or nil if none. This is an alternative to MicFwTransactionManager global runningContext.

*MicFwTransactionManager*>>

**runningContext**

Answer the current context if it has a transaction running, or nil otherwise.

*MicFwTransactionManager class*>>

**runningContexts**

Return the running context (or nil).

*MicFw1Relationship*>>

**setTo: anObject**

Create the relationship connection between @anObject and #thisObject. Validate the type of @anObject and the connection, if the receiver's #validateType is <true>. Delegate to connection handler.

*MicFwApplicationObject class*>>

**setTypeConverter: aTypeConverterClass for: aSelector**

Set the converter class for the instance variable named @aSelector.

*MicFwTransactionManager class*>>

**terminate**

Abort and destroy the default instance of the receiver.

*MicFwNRelationship*>>

**transact**

Answer true if the relationship access handler handles transacted accesses, false otherwise.

*MicFwTr1Relationship*>>

**transact**

Answer true if the relationship access handler handles transacted accesses, false otherwise.

*MicFwNRelationship*>>

**transact: aBoolean**

Set the transaction property for the receiver. If @aBoolean = true -> handle accesses transacted.

*MicFwTr1Relationship*>>

**transact: aBoolean**

Set the transaction property for the receiver. If @aBoolean = true -> handle accesses transacted.

*MicFwTransactedObject*>>

**transactionContext**

Answer the current TransactionContext for the receiver. This is the default implementation, that answers the currentContext from the receiver's transactionManager You may overwrite this method to return a specific one.

*MicFwTransactionContext>>*

**transactionLevel**

Return the active (highest) transaction level of receiver (as Integer).

*MicFwTransactedObject>>*

**transactionManager**

Answer the TransactionManager for the receiver. This is the default implementation, that answers the global TransactionManager. You may overwrite this method to return a specific one.

*MicFwTransactedObject>>*

**triggerUncommittedChange: instVar value: aValue**

trigger the uncommitted change event for an instance variable.

*MicFwApplicationObject class>>*

**typeDescriptionClass**

Answer the class used to hold the type information of an instance variable of the receiver.

*MicFwApplicationObject class>>*

**typingDeclarationFor: variableSelector**

Answer any kind of type declaration (i.e. relationship or base type) for <variableSelector>, or <nil> if none.

*MicFwTransactionContext>>*

**uncommittedRead**

Set the mode of the receiver to #uncommittedRead. Uncommitted changes from other transaction contexts will be visible.

*MicFwApplicationObject>>*

**unTransact**

Delete all transact flags in the receiver.

*MicFwDomainObject>>*

**validateWrite: #var using: aBlock**

Causes any attempted assignment of an object to the variable var to be allowed only if the object to be assigned meets the criteria specified in the block aBlock.

*MicFwTransactedObject>>*

**writeVersion: aMicFwObjectVersion**

Commit the receiver to the version contained in aMicFwObjectVersion. Write the changes to the attributes. The context calls this method on a level-1-commit (final commit).



# Appendix B

## 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 trans-

acted 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 required 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: uncommittedread 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 Model:** A model of object that can be created in the OMB and then saved to VA.

**Object Model Browser:** A tool for creating and managing object models.

**Object Net Browser:** A visual OBF tool that displays 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.

**OMB:** Object Model Browser.

**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 PMS MICADO 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 PMS MICADO 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).

**Save to VA:** Implementing the contents of an object model in VA code.

**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 PMS MICADO 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.

**uncommittedRead context:** If an `ObjectVersion` exists for `aSourceObject>>aSOVariable` (ie, `aSOVariable` is transacted and an `UncommittedTargetObject` was assigned to `aSOVariable` while `aContext1` was active): If

aContext2 is an active uncommittedRead context and getter message sOVariable is sent to aSourceObject, the object returned will be anUncommittedTargetObject. See: Isolated context.

Note: The message is spelled "uncommittedRead".

**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 uncommittedReadisolate 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 Tables

|  |     |
|--|-----|
| Table 1: DemoStartView part type labels and part names .....   | 74  |
| Table 2: DemoView part type labels and part names .....        | 74  |
| Table 3: Example table for API information .....               | 90  |
| Table 4: Domain Object Base API instance methods .....         | 91  |
| Table 5: Domain Process Base API class methods .....           | 92  |
| Table 6: Domain Process Base API instance methods .....        | 92  |
| Table 7: ViewPort Base API methods .....                       | 95  |
| Table 8: Example table of API information .....                | 142 |
| Table 9: Domain Object Advanced API class methods .....        | 143 |
| Table 10: Domain Process Advanced API class methods .....      | 145 |
| Table 11: Domain Process Advanced API instance methods .....   | 151 |
| Table 12: Viewport Advanced API class methods .....            | 165 |
| Table 13: Viewport Advanced API instance methods .....         | 169 |
| Table 14: Authorization methods .....                          | 177 |
| Table 15: Group controls Advanced API (instance) methods ..... | 182 |
| Table 16: MicFwBrokerMapper mappings .....                     | 225 |
| Table 17: MicFwBrowserMapper mappings .....                    | 225 |
| Table 18: MicFwGenusMapper mappings .....                      | 226 |
| Table 19: MicFwInterfaceMapper mappings .....                  | 226 |
| Table 20: Table of Locale Categories .....                     | 231 |
| Table 21: Configuration Maps .....                             | 256 |
| Table 22: Prerequisite applications .....                      | 256 |



# List Of Figures

|  |    |
|--|----|
| Figure 1: Initialize relationships lazy checkbox .....   | 13 |
| Figure 1.2.1. Standard type .....  | 23 |
| Figure 1.2.2. Standard type example .....  | 23 |
| Figure 1.2.3. ->1 (primitive) relationship .....   | 24 |
| Figure 1.2.4. ->1 (primitive) relationship example .....   | 24 |
| Figure 1.2.5. ->N (primitive) relationship .....   | 24 |
| Figure 1.2.6. ->N (primitive) relationship example .....   | 24 |
| Figure 1.2.7. 1<->1 relationship .....   | 25 |
| Figure 1.2.8. 1<->1 relationship example .....   | 25 |
| Figure 1.2.9. 1<->N relationship .....   | 25 |
| Figure 1.2.10. 1<->N relationship example .....  | 25 |
| Figure 1.2.11. M<->N relationship .....  | 26 |
| Figure 1.2.12. M<->N relationship example .....  | 26 |
| Figure 1.3.1. Debitting account A and crediting account B .....  | 27 |
| Figure 1.3.2. Newly created object version .....   | 28 |
| Figure 1.3.3. Object version with new uncommitted target .....   | 29 |
| Figure 1.3.4. Object version in a dialog .....   | 29 |
| Figure 1.3.5. TrLevel1 with multiple object versions .....   | 29 |
| Figure 1.3.6. Object versions in TrLevel1 for balances of accounts A and B .....                                     | 29 |
| Figure 1.3.7. Transaction level with multiple object versions: Example dialog .....                                  | 30 |
| Figure 1.3.8. Aborting TrLevel1 .....  | 30 |
| Figure 1.3.9. Bank account example: Aborting changes in TrLevel1 before T3 .....                                     | 30 |
| Figure 1.3.10. Aborting a TrLevel: Example dialog .....  | 30 |
| Figure 1.3.11. Committing TrLevel1 .....   | 31 |
| Figure 1.3.12. Bank account example: Committing changes in TrLevel1 at T3 .....                                      | 31 |
| Figure 1.3.13. Committing a TrLevel: Example dialog .....  | 31 |
| Figure 1.3.14. Nested TrLevels .....   | 31 |
| Figure 1.3.15. Object version in TrLevel2 .....  | 32 |
| Figure 1.3.16. Nested TrLevels: Example dialogs .....  | 32 |
| Figure 1.3.17. Object versions in TrLevel1 and TrLevel2 .....  | 32 |
| Figure 1.3.18. Aborting TrLevel2 .....   | 33 |
| Figure 1.3.19. Aborting highest TrLevel by clicking the Cancel button in the lowest subdialog .....                  | 33 |
| Figure 1.3.20. Committing the highest TrLevel (not TrLevel1) .....   | 33 |
| Figure 1.3.21. Committing highest TrLevel by clicking the OK button in the lowest subdialog .....                    | 34 |
| Figure 1.3.22. Object versions in TrLevel1 and TrLevel2 .....  | 34 |
| Figure 1.3.23. Aborting all TrLevels (1 and 2) .....   | 34 |
| Figure 1.3.24. Committing all TrLevels (1 and 2) .....   | 35 |
| Figure 1.3.25. Context with TrLevels .....   | 35 |
| Figure 1.3.26. Running context .....   | 35 |
| Figure 1.3.27. Non-running context .....   | 35 |
| Figure 1.3.28. Multiple contexts .....   | 36 |
| Figure 1.3.29. Single active context in multiple-context environment .....   | 36 |
| Figure 1.3.30. Running / non-running contexts in multiple-context environment .....                                  | 36 |
| Figure 1.3.31. Hierarchical contexts .....   | 36 |
| Figure 1.3.32. Hierarchical contexts' dialogs .....  | 36 |
| Figure 1.3.33. Non-hierarchical contexts .....   | 37 |
| Figure 1.3.34. Parent-child relationships between hierarchical contexts .....  | 37 |
| Figure 1.3.35. Concurrent relationships between hierarchical contexts .....  | 37 |
| Figure 1.3.36. Variable lock: Variable can be changed when context with lock active .....                            | 38 |
| Figure 1.3.37. Variable lock: Variable can be changed when non-active parent has lock .....                          | 38 |
| Figure 1.3.38. Lock of a variable transferred from parent to child context .....                                     | 38 |
| Figure 1.3.39. Transfer of variable lock from child context to parent context when child committed/aborted ....      | 38 |
| Figure 1.3.40. Uncommitted target returned when active uncommitted Read and variable locked by non-active concurrent | 39 |
| Figure 1.3.41. Uncommitted target returned when variable locked by non-active parent .....                           | 40 |
| Figure 1.3.42. Uncommitted target returned when active uncommitted Read and variable locked by non-active child      | 40 |

|  |    |
|--|----|
| Figure 1.3.43. Committed target returned when active isolated and variable locked by non-active concurrent               | 40 |
| Figure 1.3.44. Uncommitted target returned when variable locked by non-active parent                                     | 41 |
| Figure 1.3.45. Committed target returned when active isolated and variable locked by non-active child                    | 41 |
| Figure 1.3.46. Aborting context with parent and child relationships to other contexts                                    | 42 |
| Figure 1.3.47. Committing a context with parent and child relationships to other contexts                                | 42 |
| Figure 1.3.48. Example dialogs for uncommittedRead and isolated contexts   | 42 |
| Figure 1.3.49. Example dialogs: Change in isolated context   | 43 |
| Figure 1.3.50. Example dialogs: Dialog for uncommittedRead context obtains focus   | 43 |
| Figure 1.3.51. Example dialogs: Attempted change of a variable locked by other context                                   | 43 |
| Figure 1.3.52. Example dialogs: Uncommitted change in other dialog is not reflected in the isolated dialog               | 44 |
| Figure 1.3.53. Committed change is shown in isolated dialog  | 44 |
| Figure 1.3.54. isolated and committedRead contexts: Example dialogs.   | 44 |
| Figure 1.3.55. Change in the parent context dialog is displayed in the child context dialogs.                            | 45 |
| Figure 1.3.56. How changes in an uncommittedRead child dialog are displayed in sibling and parent dialogs                | 45 |
| Figure 1.3.57. How changes in an isolated child dialog are displayed in sibling and parent dialogs                       | 45 |
| Figure 1.3.58. Variables locked by child contexts cannot be changed in parent context                                    | 46 |
| Figure 1.3.59. Transaction manager   | 46 |
| Figure 2.3.1. ZyxTutorial in the application list  | 51 |
| Figure 2.3.2. Prerequisites for ZyxTutorial  | 51 |
| Figure 2.3.3. Prerequisites for ZyxTutorial  | 51 |
| Figure 2.4.1. OMB main dialog (with class net for MicFwConnectorEditProcess)   | 52 |
| Figure 2.4.2. OMB main dialog (with new model)   | 52 |
| Figure 2.4.3. Dialog "Class specification" tab "Definition"  | 53 |
| Figure 2.4.4. Class name and parent class specification in dialog "Class specification"                                  | 53 |
| Figure 2.4.5. ZyxTutorial specified as the application for ZyxClass1 in dialog "Class specification" tab "Specification" | 53 |
| Figure 2.4.6. ZyxClass1 in the OMB class net list  | 53 |
| Figure 2.7.1. Contents of ZyxClass1.st in Workspace dialog   | 56 |
| Figure 2.8.1. Specifying default application for new classes   | 57 |
| Figure 2.8.2. ZyxClass11 as a subclass of ZyxClass1 in the OMB (not in VA)   | 57 |
| Figure 2.8.3. ZyxClass2 as a subclass of MicFwDomainObject in the OMB (not in VA)  | 58 |
| Figure 2.9.1. ZyxClass1>>var1, var2 in the OMB (not in VA)   | 59 |
| Figure 2.9.2. ZyxClass11 inherited variables var1, var2 in the OMB (not in VA)   | 59 |
| Figure 2.9.3. ZyxClass1 instance methods for var1, var2  | 59 |
| Figure 2.9.4. ZyxClass1>>var1 default accessor methods   | 60 |
| Figure 2.9.5. ZyxClass1>>var1 instance method basicVar1X   | 60 |
| Figure 2.9.6. Default accessor method prefixes for all classes   | 60 |
| Figure 2.9.7. ZyxClass1>>var1, var2 accessor method names  | 60 |
| Figure 2.9.8. "V" marking (for virtual) for var2 in ZyxClass1, ZyxClass11 in OMB   | 61 |
| Figure 2.9.9. "R" marking (for redefine) for var2 in ZyxClass11 in OMB   | 61 |
| Figure 2.10.1. ZyxPerson>>id typed as Integer  | 63 |
| Figure 2.10.2. Default converter (MicFwTypeConverter) for ZyxPerson>>id  | 64 |
| Figure 2.11.1. ZyxPerson>>dateOfBirth with type Date   | 67 |
| Figure 2.12.1. Radio buttons for Key property for ZyxPerson>>id  | 69 |
| Figure 2.12.2. Persistent variable ZyxPerson>>id and entry in database table PERSON                                      | 70 |
| Figure 2.12.3. Persistent variable ZyxPerson>>dateOfBirth and entry in database table PERSON column DATEBIRTH            | 70 |
| Figure 2.13.1. ->1 relationship  | 71 |
| Figure 2.13.2. Defining ZyxName variables in OMB   | 71 |
| Figure 2.13.3. Checkbox "Relationship" in the OMB  | 72 |
| Figure 2.13.4. Tab "Typing" for a relationship   | 72 |
| Figure 2.13.5. Source class minimum/maximum (# of allowed target classes)  | 72 |
| Figure 2.13.6. ->1 relationship source ZyxPerson>>name and target ZyxName in database table PERSON                       | 73 |
| Figure 2.14.1. ZyxNameSubclass in OMB  | 75 |
| Figure 2.14.2. Object net for ZyxNameSubclass  | 75 |
| Figure 2.14.3. Object net for ZyxName  | 75 |
| Figure 2.14.4. Object net for ZyxPerson  | 76 |
| Figure 2.15.1. ->N relationship  | 77 |
| Figure 2.15.2. Defining ZyxAddress variables in OMB  | 77 |
| Figure 2.15.3. Source class minimum/maximum (# of allowed target classes)  | 78 |
| Figure 2.15.4. The ->N relationship between ZyxPerson and ZyxAddress in table ADDRESS                                    | 80 |
| Figure 2.16.1. 1<->1 relationship  | 81 |



|  |     |
|--|-----|
| Figure 2.16.2. ZyxCustomer>>portfolio definition .....   | 81  |
| Figure 2.16.3. ZyxCustomer>>portfolio definition .....   | 82  |
| Figure 2.16.4. 1<->1 relationship between ZyxCustomer and ZyxPortfolio in tables CUSTOMER and PORTFOLIO      | 82  |
| Figure 2.17.1. 1<->N relationship .....  | 83  |
| Figure 2.17.2. ZyxEmployee>>ownedCustomers definition .....  | 83  |
| Figure 2.17.3. ZyxCustomer>>ownderEmployee definition .....  | 84  |
| Figure 2.17.4. 1<->N relationship between ZyxCustomer ZyxEmployee in tables EMPLOYEE and CUSTOMER            | 84  |
| Figure 2.18.1. M<->N relationship .....  | 85  |
| Figure 2.18.2. ZyxPerson>>addresses definition .....   | 85  |
| Figure 2.18.3. ZyxAddress>>persons definition .....  | 86  |
| Figure2.18.4.M<->NrelationshipbetweenZyxPersonandZyxAddressintablesPERSON,RESIDENCE,andADDRESS               | 86  |
| Figure 2.19.1. Transactions Browser .....  | 88  |
| Figure 2.19.2. Specifying ZyxName>>firstName as transacted .....   | 89  |
| Figure 2.19.3. No running/active contexts in TB .....  | 89  |
| Figure 2.19.4. Running (TrLevel1) / active ("*") context in TB .....   | 90  |
| Figure 2.19.5. Typical dialog for data entry .....   | 90  |
| Figure 2.19.6. Uncommitted target (version value) for ZyxName in TB .....                                    | 90  |
| Figure 2.19.7. Committed target (variable value) for ZyxName in TB .....                                     | 91  |
| Figure 2.19.8. Object version .....  | 91  |
| Figure 2.19.9. Assigning object to transacted variable in active context with TrLevel1: Example dialog ..... | 91  |
| Figure 2.19.10. TrLevel1 committed: TB empty .....   | 92  |
| Figure 2.19.11. Committing TrLevel1: Example dialog .....  | 92  |
| Figure 2.19.12. Aborting TrLevel1: Example dialog .....  | 93  |
| Figure 2.20.1. 2 TrLevels in TB .....  | 94  |
| Figure 2.20.2. 2 TrLevels: Example dialogs. ....   | 94  |
| Figure 2.20.3. 1 TrLevel in TB .....   | 95  |
| Figure 2.20.4. Aborting highest TrLevel: Example dialogs. ....   | 95  |
| Figure 2.20.5. No TrLevels in TB .....   | 96  |
| Figure 2.20.6. Aborting all TrLevels: Example dialogs. ....  | 96  |
| Figure 2.20.7. 1 TrLevel in TB .....   | 97  |
| Figure 2.20.8. Committing highest TrLevel: Example dialogs. ....   | 97  |
| Figure 2.20.9. No transaction contexts in TB .....   | 98  |
| Figure 2.20.10. Committing all TrLevels: Example dialogs .....   | 98  |
| Figure 2.21.1. 2 contexts in TB .....  | 99  |
| Figure 2.21.2. 2 concurrent contexts in uncommittedRead mode: Example dialogs .....                          | 100 |
| Figure 2.21.3. Designation of isolated mode in the TB status bar for a context .....                         | 100 |
| Figure 2.21.4. 2 concurrent contexts in isolated mode: Example dialogs .....                                 | 101 |
| Figure 2.21.5. Unallowed changes to locked variables in concurrent contexts: Example dialogs .....           | 101 |
| Figure 2.22.1. Parent, child, grandchild contexts in TB .....  | 103 |
| Figure 2.22.2. Parent context with child, grandchild contexts: Example dialogs .....                         | 103 |
| Figure 2.22.3. Parent context with child, grandchild contexts: uncommittedRead, isolated example dialogs .   | 104 |
| Figure 2.22.4. Transfer of variable locks between parent and child contexts .....                            | 105 |
| Figure 3.2.1. OMB dialog (opened on class Model1Class1 from the Transcript menu) .....                       | 110 |
| Figure 3.2.2. OMB dialog (opened with MicFwTemModelDescriptionProcess open) .....                            | 110 |
| Figure 3.2.3. Major parts of the OMB dialog .....  | 111 |
| Figure 3.2.4. Submenu Model .....  | 111 |
| Figure 3.2.5. Submenu Class .....  | 112 |
| Figure 3.2.6. Class specification dialog .....   | 113 |
| Figure 3.2.7. Submenu Variable .....   | 113 |
| Figure 3.2.8. Submenu Options .....  | 114 |
| Figure 3.2.9. Submenu ? .....  | 114 |
| Figure 3.2.10. OMB Toolbar .....   | 114 |
| Figure 3.2.11. OMB classes tree .....  | 115 |
| Figure 3.2.12. OMB variable list .....   | 115 |
| Figure 3.2.13. OMB tab Variable .....  | 116 |
| Figure 3.2.14. OMB tab typing (not relationship) .....   | 116 |
| Figure 3.2.15. OMB tab typing (relationship) .....   | 116 |
| Figure 3.2.16. OMB tab Accessors .....   | 117 |
| Figure 3.2.17. OMB tab Properties .....  | 117 |

|   |     |
|---|-----|
| Figure 3.2.18. OMB Status bar .....   | 117 |
| Figure 3.2.19. OMB dialog Options tab Accessor prefixes .....                           | 117 |
| Figure 3.2.20. OMB dialog Options tab Language .....                                    | 118 |
| Figure 3.2.21. OMB dialog Options tab Save .....  | 118 |
| Figure 3.2.22. OMB dialog Options tab Defaults .....                                    | 118 |
| Figure 3.3.1. Object Net Browser dialog .....   | 119 |
| Figure 3.3.2. ONB: Submenu Class .....  | 120 |
| Figure 3.3.3. ONB: Submenu Variable .....   | 121 |
| Figure 3.3.4. ONB pop-up dialog: Setting variable type .....                            | 121 |
| Figure 3.3.5. ONB pop-up dialog: Removing variable type .....                           | 122 |
| Figure 3.3.6. ONB pop-up dialog: Browse related class .....                             | 122 |
| Figure 3.3.7. ONB pop-up dialog: Remove description .....                               | 122 |
| Figure 3.4.1. Type Editor .....   | 124 |
| Figure 3.5.1. Relationship editor .....   | 125 |
| Figure 3.6.1. Transaction Browser .....   | 128 |
| Figure 3.6.2. Transaction Browser: Submenu Transactions .....                           | 128 |
| Figure 3.6.3. Transaction Browser: Context / TransactionLevel fields / buttons .....    | 129 |
| Figure 3.6.4. Transaction Browser: Versioned objects fields / buttons .....             | 130 |
| Figure 3.6.5. Transaction Browser: Displaye committed / uncommitted object button ..... | 130 |

# Index

## Symbols

|                        |     |
|------------------------|-----|
| .st .....              | 55  |
| .xml .....             | 55  |
| ->1 relationship ..... | 71  |
| ->N relationship ..... | 77  |
| ? (menu) .....         | 114 |

## Numerics

|                       |    |
|-----------------------|----|
| 0..1 .....            | 24 |
| 11 relationship ..... | 81 |
| 1N relationship ..... | 83 |
| 1-way relationships   |    |
| Defined .....         | 21 |
| 2-way relationships   |    |
| Defined .....         | 21 |

## A

|                                       |          |
|---------------------------------------|----------|
| Abort (transacted changes) .....      | 20       |
| abortToTop .....                      | 95       |
| abortTransaction .....                | 92       |
| About... (menu item) .....            | 114      |
| Abstract Control .....                | 143      |
| Abstract Event .....                  | 143      |
| Abstract Value .....                  | 143      |
| Abstract View .....                   | 143      |
| Accessor prefix defaults .....        | 60       |
| Accessor prefixes (tab) .....         | 60, 117  |
| Accessors .....                       | 23       |
| Accessors (tab) .....                 | 60, 116  |
| active .....                          | 90       |
| Active context .....                  | 27       |
| active context .....                  | 90       |
| Adapter .....                         | 143, 144 |
| Add class net (toolbar icon) .....    | 114      |
| Add class net... (menu item) .....    | 75, 112  |
| Add class(es) (menu item) .....       | 112      |
| Add class(es)... (menu item) .....    | 58       |
| Add new variable (toolbar icon) ..... | 115      |
| Add variable... (menu item) .....     | 59, 113  |
| AFW .....                             | 27       |
| Alt+A .....                           | 113      |
| Alt+R .....                           | 113      |
| Application                           |          |
| (field) .....                         | 53       |
| Application Framework .....           | 19       |
| Applications / New .....              | 51       |
| architecture .....                    | 145      |
| authorization .....                   | 144, 147 |
| Automatic conversion .....            | 66       |

## B

|   |     |
|---|-----|
| Base Connection .....                   | 144 |
| Basic read accessor (entry field) ..... | 60  |
| beginTransaction .....                  | 90  |
| Broker .....                            | 144 |
| Browse Transactions .....               | 89  |

## C

|  |            |
|--|------------|
| Cardinality                              |            |
| Defined .....                            | 24         |
| cityName .....                           | 77         |
| Class (menu) .....                       | 112        |
| Class name                               |            |
| (field) .....                            | 53         |
| class net .....                          | 75         |
| Class specification (dialog) .....       | 113        |
| Class specification... (menu item) ..... | 54, 112    |
| Class tree .....                         | 115        |
| Close (menu item) .....                  | 54, 112    |
| combinable .....                         | 144        |
| Commit (transacted changes) .....        | 20         |
| committed .....                          | 90         |
| Committed target                         |            |
| Defined .....                            | 19, 28     |
| Committing transacted changes .....      | 30         |
| commitToTop .....                        | 97         |
| commitTransaction .....                  | 91         |
| Concurrent .....                         | 37         |
| Concurrent access violation .....        | 22         |
| Concurrent contexts .....                | 99         |
| Configuration maps .....                 | 15         |
| Configuration Maps Browser .....         | 15         |
| Connector .....                          | 144        |
| Context                                  |            |
| Active .....                             | 19, 35, 36 |
| Defined .....                            | 35         |
| Running .....                            | 35, 36     |
| control flow .....                       | 144        |
| Create class (in model) .....            | 53         |
| Create model .....                       | 52         |
| createdExtendedDescription .....         | 23         |
| Ctrl+A .....                             | 112        |
| Ctrl+D .....                             | 111        |
| Ctrl+F .....                             | 113        |
| Ctrl+N .....                             | 112        |
| Ctrl+O .....                             | 111        |
| Ctrl+P .....                             | 112        |
| Ctrl+R .....                             | 112        |
| Ctrl+S .....                             | 111        |
| Ctrl-Q .....                             | 112        |
| customer .....                           | 81         |

---

## D

|  |     |
|--|-----|
| Database interface .....                 | 22  |
| Date (variable type) .....               | 67  |
| dateOfBirth.....                         | 67  |
| Default application .....                | 118 |
| Default application for new classes..... | 57  |
| Default Base Connection.....             | 144 |
| Defaults (tab) .....                     | 118 |
| Delegation Model.....                    | 144 |
| Description object                       |     |
| Structure.....                           | 23  |
| Design-time framework .....              | 22  |
| Discard all changes (in model).....      | 54  |
| Discard all changes (menu item).....     | 111 |
| Discard all changes (toolbar icon).....  | 114 |
| Domain Model .....                       | 144 |
| Domain Object.....                       | 144 |
| Domain Process .....                     | 144 |
| Domain Processes Browser .....           | 145 |

---

## E

|  |    |
|--|----|
| Export to .st files .....                    | 55 |
| Export to an .xml file .....                 | 55 |
| External software component integration .... | 22 |

---

## F

|                                 |     |
|---------------------------------|-----|
| File in .....                   | 56  |
| File in (menu item).....        | 56  |
| Find class (toolbar icon) ..... | 114 |
| Find class... (menu item) ..... | 113 |
| firstName .....                 | 71  |
| Framework.....                  | 145 |
| Framework accessors .....       | 59  |
| Framework Logger .....          | 145 |
| Full menus .....                | 51  |

---

## G

|                                     |     |
|-------------------------------------|-----|
| Generate XML (menu item).....       | 112 |
| Go to target class (menu item)..... | 113 |

---

## H

|                      |    |
|----------------------|----|
| Hierarchical context |    |
| Defined.....         | 36 |

---

## I

|                               |     |
|-------------------------------|-----|
| Info (toolbar icon).....      | 115 |
| Inherits from                 |     |
| (drop-down list) .....        | 53  |
| Instance variable typing      |     |
| Defined.....                  | 19  |
| Integer (variable type) ..... | 63  |
| interaction .....             | 144 |
| Invalid column types .....    | 22  |

|                          |     |
|--------------------------|-----|
| isActive .....           | 90  |
| isolate .....            | 100 |
| Isolated (context) ..... | 39  |
| isRunning .....          | 90  |

---

## J

|                        |     |
|------------------------|-----|
| Java (menu item) ..... | 114 |
|------------------------|-----|

---

## K

|                      |    |
|----------------------|----|
| Key (variable) ..... | 69 |
|----------------------|----|

---

## L

|                                  |     |
|----------------------------------|-----|
| Language (tab).....              | 117 |
| lastName .....                   | 71  |
| Library .....                    | 15  |
| Linked tables .....              | 22  |
| Lock (transacted variable) ..... | 27  |
| Lock (variable).....             | 37  |
| Logger .....                     | 145 |

---

## M

|   |          |
|---|----------|
| Make virtual variable (menu item) ..... | 61, 113  |
| Manual conversion (convertValues) ..... | 64       |
| Mapper .....                            | 145      |
| maximum.....                            | 116      |
| maximum (source class) .....            | 72       |
| mgr45.dat .....                         | 50       |
| MicFwApplicationModelObjects.....       | 51       |
| MicFwDomainObject.....                  | 53       |
| MicFwExtendedDescription .....          | 23       |
| MicFwInstanceVariableDescription .....  | 23       |
| MicFwRelationshipDescription.....       | 23       |
| MicFwTemModelDesriptionProcess .....    | 110      |
| MicFwTypeConverter.....                 | 64       |
| MicFwTypeDescription.....               | 23       |
| minimum.....                            | 116      |
| minimum (source class) .....            | 72       |
| MN relationship .....                   | 85       |
| model.....                              | 144      |
| Model (menu) .....                      | 111      |
| Model (object) .....                    | 20       |
| Model -> POM generator (menu item)..... | 112      |
| Model View Connector.....               | 145      |
| Monitor target (checkbox) .....         | 116      |
| Multiple TrLevels .....                 | 94       |
| MVC .....                               | 144, 145 |

---

## N

|                                |         |
|--------------------------------|---------|
| Nested TrLevels.....           | 94      |
| New Application dialog.....    | 51      |
| New class (toolbar icon).....  | 115     |
| New class... (menu item) ..... | 53, 112 |
| New model (menu item).....     | 52      |
| New Model (toolbar icon).....  | 114     |

|                               |         |
|-------------------------------|---------|
| New model... (menu item)..... | 54, 111 |
| Non-hierarchical context      |         |
| Defined.....                  | 37      |
| Non-virtual variable .....    | 61      |

## O

|   |          |
|---|----------|
| OBF   |          |
| Defined.....                                  | 19       |
| OBF CH (#).....                               | 50       |
| obf_ex.txt.....                               | 50       |
| Object Behavior Framework .....               | 144, 146 |
| Object model .....                            | 20       |
| Object Model Browser .....                    | 20       |
| Object net .....                              | 19       |
| object net.....                               | 145      |
| Object Net Browser .....                      | 20, 144  |
| Object relationships                          |          |
| Defined.....                                  | 19       |
| Object version .....                          | 90       |
| Defined.....                                  | 28       |
| ONB.....                                      | 28       |
| Open class specification (toolbar icon) ..... | 115      |
| Open Model (toolbar icon).....                | 114      |
| Open model... (menu item) .....               | 54, 111  |
| Open Object Model Browser.....                | 52       |
| Open Object Model Browser... (menu item)..... | 110      |
| openBrowser .....                             | 88       |
| Options (dialog).....                         | 117      |
| Options (menu).....                           | 114      |
| Options (toolbar icon).....                   | 115      |
| Overwrite model .....                         | 54       |
| ownedCustomers .....                          | 83       |
| ownerEmployee.....                            | 83       |

## P

|                                       |              |
|---------------------------------------|--------------|
| packaging .....                       | 146          |
| Parent / Child contexts.....          | 102          |
| Persistence                           |              |
| Defined.....                          | 22           |
| Persistence Framework .....           | 19, 144, 146 |
| Persistent (checkbox).....            | 116          |
| Persistent (variable) .....           | 69           |
| Persistent object                     |              |
| Defined.....                          | 19           |
| persons.....                          | 85           |
| PFW.....                              | 22, 27       |
| Platform Adapter .....                | 143          |
| Polymorphism.....                     | 21           |
| portable.....                         | 144          |
| portfolio.....                        | 81           |
| Preferences (menu item).....          | 60           |
| Preferences... (menu item) .....      | 57, 114      |
| Prerequisites (for application) ..... | 51           |
| Primary Key .....                     | 70           |
| Primitive (checkbox).....             | 116          |
| primitive (relationship).....         | 77           |

|                        |     |
|------------------------|-----|
| processing.....        | 144 |
| Properties (tab) ..... | 117 |

## R

|  |                    |
|--|--------------------|
| read trace .....                           | 147                |
| readAccessTo                               |                    |
| .....                                      | 23, 59, 63, 66, 69 |
| real Control .....                         | 143                |
| real View .....                            | 143                |
| real world concepts.....                   | 144                |
| Redefine Variable (menu item).....         | 113                |
| Redefine variable (menu item) .....        | 61                 |
| Redefine variable (toolbar icon).....      | 115                |
| referential integrity .....                | 146                |
| relational databases.....                  | 146                |
| relationship.....                          | 145, 146           |
| Relationship (checkbox) .....              | 72                 |
| Relationship Editor.....                   | 20, 23             |
| Relationships                              |                    |
| ->1 (primitive) .....                      | 23                 |
| ->N (primitive).....                       | 24                 |
| 1 to 1 .....                               | 24                 |
| 1 to N.....                                | 25                 |
| Defined .....                              | 19, 21             |
| M to N.....                                | 25                 |
| Remove class (menu item).....              | 112                |
| Remove class (toolbar icon) .....          | 115                |
| Remove classes from model .....            | 58                 |
| Remove redefine.....                       | 61                 |
| Remove redefine Variable (menu item) ....  | 113                |
| Remove redefine variable (menu item) ..... | 61                 |
| Remove variable (menu item) .....          | 61, 113            |
| Remove variable (toolbar icon).....        | 115                |
| Rename model... (menu item).....           | 111                |
| Reopen model.....                          | 54                 |
| Run-time binding of messages .....         | 21                 |
| runtime executable .....                   | 146                |
| Run-time type verification .....           | 22                 |

## S

|  |        |
|--|--------|
| Save (tab).....                          | 118    |
| Save model (menu item).....              | 111    |
| Save model... (menu item) .....          | 53     |
| Save to file (toolbar icon).....         | 114    |
| Save to file... (menu item).....         | 112    |
| Save to VA (menu item).....              | 111    |
| Save to VA (toolbar icon).....           | 115    |
| Saving (model) to VA.....                | 56     |
| Scale (field) .....                      | 63     |
| Select root class.....                   | 52, 75 |
| Set all to default (checkbox) .....      | 57, 60 |
| set to the default .....                 | 118    |
| Show inherited variable (menu item)..... | 113    |
| Size (field) .....                       | 63     |
| Smalltalk (menu item) .....              | 114    |
| Source object                            |        |
| Defined .....                            | 19     |

|                                    |     |
|------------------------------------|-----|
| Specification (tab).....           | 53  |
| Static binding of messages .....   | 21  |
| Status bar .....                   | 117 |
| Statusbar (menu item).....         | 114 |
| STOPF.....                         | 144 |
| Subapplication of (checkbox) ..... | 51  |

## T

|  |         |
|--|---------|
| target class .....                           | 72      |
| Target class (drop-down list).....           | 116     |
| target class variable (drop-down list) ..... | 116     |
| Target object                                |         |
| Defined.....                                 | 19      |
| TB .....                                     | 28      |
| Toolbar.....                                 | 114     |
| Toolbar (menu item) .....                    | 114     |
| Tools (menu) .....                           | 112     |
| transactedReadAccessTo .....                 | 28      |
| transactedWriteAccessTo .....                | 28      |
| Transaction                                  |         |
| Defined.....                                 | 27      |
| transaction .....                            | 146     |
| Transaction Browser .....                    | 20, 88  |
| Transaction Context .....                    | 144     |
| Defined.....                                 | 35      |
| Transaction context.....                     | 27      |
| Transaction level                            |         |
| Defined.....                                 | 29      |
| Transaction level nested                     |         |
| Defined.....                                 | 31      |
| Transaction Manager                          |         |
| Defined.....                                 | 46      |
| Transaction manager .....                    | 27      |
| Transaction write conflict.....              | 27      |
| Transactions                                 |         |
| Defined.....                                 | 19      |
| TrLevel.....                                 | 88      |
| TrLevel1 .....                               | 29      |
| Type   |         |
| Standard.....                                | 23      |
| Type (instance variable)                     |         |
| Defined.....                                 | 19      |
| Type class (menu item) .....                 | 113     |
| Type Editor .....                            | 20      |
| TypeConverter.....                           | 115     |
| Typing                                       |         |
| Why OBF provides .....                       | 22      |
| typing .....                                 | 146     |
| Typing (tab) .....                           | 63, 116 |

## U

|                                     |        |
|-------------------------------------|--------|
| uncommittedRead .....               | 39, 99 |
| uncommitted .....                   | 90     |
| Uncommitted target .....            | 29     |
| Defined.....                        | 19, 28 |
| Unrecognized message avoidance..... | 22     |
| update .....                        | 147    |

|   |    |
|---|----|
| Use default for new class (radio button)..... | 57 |
|---|----|

## V

|                                    |               |
|------------------------------------|---------------|
| Validate Model (toolbar icon)..... | 114           |
| Validate model... (menu item)..... | 111           |
| validatedWriteAccessTo             |               |
| .....                              | 66, 69        |
| validateType (checkbox).....       | 66            |
| validation .....                   | 144, 147      |
| Variable (menu).....               | 113           |
| Variable (tab).....                | 115           |
| Variable list.....                 | 115           |
| variable lock transfer .....       | 105           |
| Variable types                     |               |
| Defined .....                      | 21            |
| Variable typing .....              | 63            |
| Variable value .....               | 91            |
| version value .....                | 90            |
| View (menu) .....                  | 114           |
| Viewport .....                     | 143, 144, 147 |
| Virtual variable .....             | 61            |

## W

|                |            |
|----------------|------------|
| workflow ..... | 144        |
| writeAccessTo  |            |
| .....          | 23, 59, 63 |

## Z

|                              |    |
|------------------------------|----|
| ZyxAddress .....             | 77 |
| ZyxClass1.st.....            | 56 |
| ZyxClass11 .....             | 57 |
| ZyxClass2 .....              | 57 |
| ZyxCustomer.....             | 81 |
| ZyxEmployee .....            | 83 |
| ZyxModel1.ome.....           | 53 |
| ZyxName .....                | 71 |
| ZyxNameSubclass .....        | 75 |
| ZyxPortfolio .....           | 81 |
| ZyxTutorial application..... | 51 |