



VisualAge Pacbase 2.5

Pacbench C/S \*  
User's Guide – Volume III  
**Graphic Clients**

DDOVA000255A

**Note**

Before using this document, read the general information under "Notices" on the next page.

According to your license agreement, you may consult or download the complete up-to-date collection of the VisualAge Pacbase documentation from the VisualAge Pacbase Support Center at:

<http://www.ibm.com/software/ad/vapacbase/support.htm>

Consult the Catalog section in the Documentation home page to make sure you have the most recent edition of this document.

**Fifth Edition (December 1999)**

This edition applies to the following licensed program:

- VisualAge Pacbase Version 2.5

Comments on publications (including document reference number) should be sent electronically through the Support Center Web site at:

<http://www.ibm.com/software/ad/vapacbase/support.htm>

or to the following postal address:

IBM Paris Laboratory  
VisualAge Pacbase Support  
30, rue du Château des Rentiers  
75640 PARIS Cedex 13  
FRANCE

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 1983, 1999. All rights reserved.

Note to U.S. Government Users – Documentation related to restricted rights – Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

## NOTICES

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

Intellectual Property and Licensing  
International Business Machines Corporation  
North Castle Drive, Armonk, New-York 10504-1785  
USA

Licenses of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of information which has been exchanged, should contact:

IBM Paris Laboratory  
SMC Department  
30, rue du Château des Rentiers  
75640 PARIS Cedex 13  
FRANCE

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

IBM may change this publication, the product described herein, or both.

## TRADEMARKS

IBM is a trademark of International Business Machines Corporation, Inc. AIX, AS/400, CICS, CICS/MVS, CICS/VSE, COBOL/2, DB2, IMS, MQSeries, OS/2, PACBASE, RACF, RS/6000, SQL/DS, TeamConnection, and VisualAge are trademarks of International Business Machines Corporation, Inc. in the United States and/or other countries.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and/or other countries licensed exclusively through X/Open Company Limited.

All other company, product, and service names may be trademarks of their respective owners.



## Contents

*A detailed Table of Contents is given after these contents.*

|   |            |
|---|------------|
| <b>1 Introduction.....</b>  | <b>17</b>  |
| 1.1 Steps of Development.....   | 17         |
| 1.2 Compatibility of Business Components / Proxy Objects.....         | 18         |
| <b>2 VisualAge for Java Generation .....</b>                          | <b>19</b>  |
| 2.1 Delivered Generic classes.....                                    | 19         |
| 2.2 Starting the Generator.....                                       | 21         |
| 2.3 Operating Mode.....   | 22         |
| 2.4 Importing the Generated Classes in VisualAge.....                 | 26         |
| 2.5 Generation Results.....   | 26         |
| <b>3 VisualAge for Smalltalk Generation .....</b>                     | <b>33</b>  |
| 3.1 Delivered Generic Classes.....                                    | 33         |
| 3.2 Generation User Interface .....                                   | 35         |
| 3.3 Generation Results.....   | 41         |
| <b>4 Generation for COM Targets .....</b>                             | <b>45</b>  |
| 4.1 Starting the Generator in Graphical Mode.....                     | 45         |
| 4.2 Starting the Generator in Batch Mode.....                         | 47         |
| 4.3 Generation Results.....   | 48         |
| 4.4 Compilation Results .....   | 49         |
| 4.5 Compiling with Visual C++ Version 5.0 and 6.0 .....               | 49         |
| <b>5 Developing a Client with VisualAge for Java.....</b>             | <b>57</b>  |
| 5.1 General Principles .....  | 57         |
| 5.2 Example of an Applet .....  | 78         |
| 5.3 Specificities of the Development of a Standalone Application..... | 97         |
| 5.4 Error Management .....  | 99         |
| 5.5 Communication Management .....                                    | 110        |
| 5.6 Testing the Application Generated– Packaging.....                 | 118        |
| 5.7 Application Deployment.....                                       | 122        |
| <b>6 Developing a Client with VisualAge for Smalltalk.....</b>        | <b>123</b> |
| 6.1 General Principles .....  | 124        |

|           |  |            |
|-----------|--|------------|
| 6.2       | Example of the Development of a Standard GUI Application .....   | 149        |
| 6.3       | Example of a Web Application .....                               | 166        |
| 6.4       | Error Management .....   | 175        |
| 6.5       | Communication Management .....                                   | 185        |
| 6.6       | Test of the Generated Application– Packaging .....               | 192        |
| 6.7       | Application Deployment.....                                      | 194        |
| <b>7</b>  | <b>Developing a COM Standard Client .....</b>                    | <b>195</b> |
| 7.1       | General Principles .....   | 195        |
| 7.2       | Error Management .....   | 211        |
| 7.3       | Communication Management .....                                   | 216        |
| 7.4       | Application Deployment.....                                      | 221        |
| <b>8</b>  | <b>Pacbench Client/Server Bridge.....</b>                        | <b>223</b> |
| 8.1       | Introduction .....   | 223        |
| 8.2       | Pacbench Client/Server Information Model (Graphic Client) .....  | 224        |
| 8.3       | Operating Principles .....                                       | 233        |
| 8.4       | Backup: Upload to the Repository .....                           | 233        |
| 8.5       | Restoration of Data to the VisualAge Smalltalk Workstation ..... | 244        |
| 8.6       | Purging VisualAge Entities in the Repository .....               | 247        |
| <b>9</b>  | <b>Appendix 1: Parameterizing Prerequisite Software .....</b>    | <b>249</b> |
| 9.1       | IMS CPI-C .....  | 249        |
| 9.2       | CICS CPI-C.....  | 267        |
| 9.3       | CICS ECI.....  | 298        |
| 9.4       | TCP-IP Socket via UNIX .....                                     | 306        |
| 9.5       | Windows/NT TCP-IP Socket .....                                   | 307        |
| 9.6       | CICS TCP/IP Sockets Interface.....                               | 308        |
| 9.7       | Local Middleware .....   | 324        |
| 9.8       | TUXEDO .....   | 325        |
| 9.9       | MQSERIES .....   | 325        |
| 9.10      | CPIC-C/XCP2.....   | 328        |
| <b>10</b> | <b>Appendix 2: Use of a Customized Middleware.....</b>           | <b>343</b> |
| 10.1      | VisualAge for Java .....   | 343        |
| 10.2      | VisualAge for Smalltalk .....                                    | 343        |
| <b>11</b> | <b>Index .....</b>   | <b>347</b> |

## **Table of Contents**

|   |           |
|---|-----------|
| <b>1 Introduction.....</b>                                    | <b>17</b> |
| 1.1 Steps of Development.....                                 | 17        |
| 1.2 Compatibility of Business Components / Proxy Objects..... | 18        |
| <b>2 VisualAge for Java Generation .....</b>                  | <b>19</b> |
| 2.1 Delivered Generic classes.....                            | 19        |
| 2.1.1 On-line Documentation of Generic Classes.....           | 20        |
| 2.2 Starting the Generator.....                               | 21        |
| 2.3 Operating Mode.....                                       | 22        |
| 2.4 Importing the Generated Classes in VisualAge.....         | 26        |
| 2.5 Generation Results.....                                   | 26        |
| 2.5.1 Introduction .....                                      | 26        |
| 2.5.2 Generated Classes .....                                 | 27        |
| 2.5.2.1 On-line Documentation of the Generated Classes        | 29        |
| 2.5.2.2 Customizing Classes                                   | 30        |
| <b>3 VisualAge for Smalltalk Generation .....</b>             | <b>33</b> |
| 3.1 Delivered Generic Classes.....                            | 33        |
| 3.1.1 Communication .....                                     | 33        |
| 3.1.1.1 Middleware Interface                                  | 33        |
| 3.1.1.2 Locations Manager                                     | 33        |
| 3.1.2 Runtime.....  | 33        |
| 3.1.2.1 Local Cache   | 33        |
| 3.1.2.2 Exchange Manager                                      | 34        |
| 3.1.2.3 Communication Manager                                 | 34        |
| 3.1.2.4 Folder View Model                                     | 34        |
| 3.1.3 Edition .....   | 34        |
| 3.2 Generation User Interface .....                           | 35        |
| 3.2.1 Starting the Generator .....                            | 35        |
| 3.2.2 Selecting Files.....                                    | 35        |
| 3.2.3 Description of Menus .....                              | 35        |
| 3.2.3.1 Directories Menu                                      | 36        |
| 3.2.3.2 Proxy files Menu                                      | 36        |
| 3.2.3.3 Options Menu  | 36        |
| 3.2.4 Generation Execution .....                              | 37        |
| 3.2.5 Applications Mapping.....                               | 38        |
| 3.2.6 Generation Report .....                                 | 39        |
| 3.3 Generation Results.....                                   | 41        |
| 3.3.1 Introduction .....                                      | 41        |
| 3.3.2 Generated Classes .....                                 | 41        |
| 3.3.2.1 Generated Classes Codification                        | 41        |
| 3.3.2.2 Contents of Generated Classes                         | 42        |
| 3.3.2.3 Customizing Classes                                   | 43        |
| <b>4 Generation for COM Targets .....</b>                     | <b>45</b> |
| 4.1 Starting the Generator in Graphical Mode.....             | 45        |
| 4.2 Starting the Generator in Batch Mode.....                 | 47        |

|            |   |           |
|------------|---|-----------|
| <b>4.3</b> | <b>Generation Results.....</b>  | <b>48</b> |
| 4.3.1      | Introduction .....  | 48        |
| 4.3.2      | Generated Classes .....   | 48        |
| <b>4.4</b> | <b>Compilation Results .....</b>  | <b>49</b> |
| <b>4.5</b> | <b>Compiling with Visual C++ Version 5.0 and 6.0 .....</b>                | <b>49</b> |
| 4.5.1      | Creating a New Workspace .....  | 49        |
| 4.5.2      | Creating the Static Library Project .....                                 | 50        |
| 4.5.3      | Creating COM Proxy Project .....  | 51        |
| 4.5.4      | Transferring files .....  | 52        |
| 4.5.5      | Adding the Files to the Projects .....                                    | 52        |
| 4.5.6      | Specifying Paths to Compile the Proxy.....                                | 54        |
| 4.5.7      | Compiling the Static Library .....  | 55        |
| 4.5.8      | Setting Parameters for Link-Edit.....                                     | 55        |
| 4.5.9      | Compilation de la Proxy .....   | 56        |
| <b>5</b>   | <b>Developing a Client with VisualAge for Java.....</b>                   | <b>57</b> |
| <b>5.1</b> | <b>General Principles .....</b>   | <b>57</b> |
| 5.1.1      | Visual Representation of Proxy Objects in the Composition Editor.....     | 57        |
| 5.1.2      | Use of Properties .....   | 58        |
| 5.1.2.1    | Local Checks .....  | 58        |
| 5.1.2.2    | Check of the Length of the <code>detail</code> Property Fields .....      | 58        |
| 5.1.2.3    | Selecting the Local or Server Sort Criterion on a List of Instances ..... | 59        |
| 5.1.2.4    | Specification of the Local Sort Criterion .....                           | 59        |
| 5.1.2.5    | Table Model .....   | 60        |
| 5.1.2.6    | Sub-schema Management .....   | 60        |
| 5.1.3      | Use of Methods.....   | 61        |
| 5.1.3.1    | Implementation .....  | 61        |
| 5.1.3.2    | The Different Types of Server Methods .....                               | 61        |
| 5.1.3.3    | Managing Folder Reading .....   | 62        |
| 5.1.3.4    | Folder Update Management .....  | 69        |
| 5.1.3.5    | Asynchronous Methods .....  | 70        |
| 5.1.3.6    | User Service .....  | 73        |
| 5.1.3.7    | Database Logical Lock .....   | 74        |
| 5.1.3.8    | Customization of the Columns of a JTable .....                            | 75        |
| 5.1.3.9    | Management of Data Element Presence .....                                 | 76        |
| 5.1.3.10   | Management of Data Element Check .....                                    | 76        |
| 5.1.3.11   | Sub-Schema Management .....   | 76        |
| 5.1.4      | Use of Events.....  | 77        |
| 5.1.4.1    | Event-driven Management of Large Reading .....                            | 77        |
| 5.1.4.2    | Event-driven Management of Instance Reading .....                         | 78        |
| <b>5.2</b> | <b>Example of an Applet .....</b>   | <b>78</b> |
| 5.2.1      | Introduction .....  | 78        |
| 5.2.2      | Presentation of the End User Interface.....                               | 79        |
| 5.2.3      | Developing the End User Interface with VisualAge Java V1 .....            | 80        |
| 5.2.3.1    | Implementing the Example and Creating the Applet .....                    | 80        |
| 5.2.3.2    | Developing the Customers Window .....                                     | 83        |
| 5.2.3.3    | Developing the Orders Window .....  | 88        |
| 5.2.4      | Developing the End User Interface with VisualAge Java V2 .....            | 89        |
| 5.2.4.1    | Implementing the Example and Creating the Applet .....                    | 89        |
| 5.2.4.2    | Developing the Customers Window .....                                     | 92        |
| 5.2.4.3    | Developing the Orders Window .....  | 96        |
| <b>5.3</b> | <b>Specificities of the Development of a Standalone Application.....</b>  | <b>97</b> |
| 5.3.1      | Introduction .....  | 97        |
| 5.3.2      | Example .....   | 98        |
| <b>5.4</b> | <b>Error Management .....</b>   | <b>99</b> |
| 5.4.1      | Principles.....   | 99        |
| 5.4.1.1    | Introduction .....  | 99        |
| 5.4.1.2    | Programming .....   | 99        |



|            |  |            |
|------------|--|------------|
| 5.4.2      | Local Errors.....  | 100        |
| 5.4.2.1    | List of Errors   | 100        |
| 5.4.3      | Server Errors.....   | 102        |
| 5.4.3.1    | Structure of the Error Key   | 102        |
| 5.4.4      | System Errors .....  | 102        |
| 5.4.4.1    | Structure of the Error Key   | 102        |
| 5.4.5      | Communication Errors .....   | 105        |
| 5.4.5.1    | List of Errors   | 105        |
| 5.4.6      | Example of Error Management.....   | 106        |
| 5.4.6.1    | Introduction   | 106        |
| 5.4.6.2    | Presentation of the non-visual classes in use                                  | 106        |
| 5.4.6.3    | Presentation of the ErrorManagerExample visual class                           | 107        |
| 5.4.6.4    | Code for displaying the error window   | 110        |
| <b>5.5</b> | <b>Communication Management .....</b>  | <b>110</b> |
| 5.5.1      | Middleware Components .....  | 110        |
| 5.5.2      | Processing a Request.....  | 111        |
| 5.5.2.1    | Direct Access to the Middleware or Local Mode                                  | 112        |
| 5.5.2.2    | Access via a Gateway   | 113        |
| 5.5.2.3    | Dynamic Change of the Middleware Access Parameters                             | 114        |
| 5.5.3      | Definition of the Use Context .....  | 115        |
| 5.5.3.1    | Structure of the VAPLOCAT.INI File   | 115        |
| 5.5.3.2    | Implementation   | 117        |
| <b>5.6</b> | <b>Testing the Application Generated– Packaging.....</b>                       | <b>118</b> |
| 5.6.1      | Testing the Generated Application.....   | 118        |
| 5.6.1.1    | Version Compatibility Check  | 118        |
| 5.6.1.2    | Optimized Middleware or Trace Mode   | 118        |
| 5.6.2      | Packaging .....  | 119        |
| 5.6.2.1    | Reminder: Prerequisites  | 119        |
| 5.6.2.2    | Export   | 119        |
| <b>5.7</b> | <b>Application Deployment.....</b>   | <b>122</b> |
| <b>6</b>   | <b>Developing a Client with VisualAge for Smalltalk.....</b>                   | <b>123</b> |
| <b>6.1</b> | <b>General Principles .....</b>  | <b>124</b> |
| 6.1.1      | Using the Proxy Objects in the Composition Editor.....                         | 124        |
| 6.1.1.1    | Unfold Option in the Elementary Proxy Pop-up Menu                              | 124        |
| 6.1.1.2    | Generation Context   | 127        |
| 6.1.1.3    | Displaying Child Elementary Proxy  | 128        |
| 6.1.2      | Use of Attributes .....  | 128        |
| 6.1.2.1    | Parameter Definition of Attributes: Settings Window                            | 128        |
| 6.1.2.2    | Defining a Graphic Representation of the Data Elements in the Detail Attribute | 130        |
| 6.1.2.3    | Check of the Length of the <code>detail</code> Attribute Fields                | 131        |
| 6.1.2.4    | Selecting the Local or Server Sort Criterion on a List of Instances            | 132        |
| 6.1.2.5    | Specific Use of the Quick Form   | 132        |
| 6.1.2.6    | Local Checks   | 133        |
| 6.1.2.7    | Sub-schema Management  | 133        |
| 6.1.3      | Use of Actions.....  | 133        |
| 6.1.3.1    | Implementation   | 133        |
| 6.1.3.2    | The Different Types of Server Actions  | 134        |
| 6.1.3.3    | Managing Folder Reading  | 134        |
| 6.1.3.4    | Folder Update Management   | 142        |
| 6.1.3.5    | Asynchronous Actions   | 143        |
| 6.1.3.6    | User Services  | 145        |
| 6.1.3.7    | Database Logical Lock  | 146        |
| 6.1.3.8    | Management of Data Element Presence  | 147        |
| 6.1.3.9    | Management of Data Element Check   | 147        |
| 6.1.3.10   | Sub-schema Management  | 147        |
| 6.1.4      | Use of Events.....   | 148        |
| 6.1.4.1    | Event-driven Management of Large Reading                                       | 148        |
| 6.1.4.2    | Event-driven Management of Instance Reading                                    | 149        |

|          |  |            |
|----------|--|------------|
| 6.2      | Example of the Development of a Standard GUI Application .....                     | 149        |
| 6.2.1    | Introduction .....   | 149        |
| 6.2.2    | Presentation of the Application - Example.....                                     | 151        |
| 6.2.3    | Development of the Application- Example.....                                       | 155        |
| 6.2.3.1  | Developing the Customers Window  | 155        |
| 6.2.3.2  | Development of the Orders Window   | 161        |
| 6.2.3.3  | Development of the Product References Window                                       | 164        |
| 6.3      | Example of a Web Application .....   | 166        |
| 6.3.1    | Introduction .....   | 166        |
| 6.3.1.1  | Automatic Conversion of HtmlFormData   | 166        |
| 6.3.1.2  | HTML Quick-Form Extension  | 166        |
| 6.3.2    | Managing an Application Context .....  | 167        |
| 6.3.3    | Example: Presentation of the User Interface .....                                  | 167        |
| 6.3.4    | Example: Developing the End User Interface.....                                    | 169        |
| 6.3.4.1  | Developing the Customers Page  | 170        |
| 6.3.4.2  | Programming the CustomersDetail Page   | 173        |
| 6.4      | Error Management .....   | 175        |
| 6.4.1    | Principles.....  | 175        |
| 6.4.1.1  | Introduction   | 175        |
| 6.4.1.2  | Event-driven Error Handling  | 175        |
| 6.4.1.3  | Visual Programming   | 176        |
| 6.4.1.4  | Customization of the Standard Labels   | 176        |
| 6.4.1.5  | Error Key Structure  | 177        |
| 6.4.2    | Errors list.....   | 179        |
| 6.4.2.1  | Local Errors   | 179        |
| 6.4.2.2  | Server errors  | 181        |
| 6.4.2.3  | System Errors  | 181        |
| 6.4.2.4  | Communication Errors   | 182        |
| 6.4.3    | System Errors Generated by the Communication Monitor and the Services Manager..... | 182        |
| 6.4.3.1  | Errors Generated by the Communication Monitor                                      | 182        |
| 6.4.3.2  | Errors Generated by the Services Manager   | 182        |
| 6.4.4    | Example .....  | 184        |
| 6.5      | Communication Management .....   | 185        |
| 6.5.1    | Middleware Components .....  | 185        |
| 6.5.2    | Diagram of the Processing of a Request .....                                       | 187        |
| 6.5.3    | Definition of Use Context .....  | 187        |
| 6.5.3.1  | Structure of the VAPLOCAT.INI File   | 188        |
| 6.5.3.2  | Smalltalk Versions from 2.5 V08 onwards  | 190        |
| 6.5.3.3  | Implementation   | 192        |
| 6.6      | Test of the Generated Application– Packaging .....                                 | 192        |
| 6.6.1    | Test of the Generated Application .....  | 192        |
| 6.6.1.1  | Starting the Application   | 192        |
| 6.6.1.2  | Version Control  | 192        |
| 6.6.1.3  | Debugging Aid or Trace Mode  | 193        |
| 6.6.2    | Packaging .....  | 193        |
| 6.6.2.1  | Packaging an IC  | 193        |
| 6.7      | Application Deployment.....  | 194        |
| <b>7</b> | <b>Developing a COM Standard Client.....</b>                                       | <b>195</b> |
| 7.1      | General Principles .....   | 195        |
| 7.1.1    | Use of Attributes .....  | 195        |
| 7.1.1.1  | Local Checks   | 195        |
| 7.1.1.2  | Sub-schema Management  | 195        |
| 7.1.1.3  | Check of the Length of the <code>detail</code> Attribute Fields                    | 196        |
| 7.1.1.4  | Selecting the Local or Server Sort Criterion on a List of Instances                | 196        |
| 7.1.2    | Use of Actions.....  | 197        |
| 7.1.2.1  | Implementation   | 197        |
| 7.1.2.2  | The Different Types of Server Actions  | 197        |

|            |   |            |
|------------|---|------------|
| 7.1.2.3    | Managing Folder Reading   | 198        |
| 7.1.2.4    | Folder Update Management  | 206        |
| 7.1.2.5    | User Services   | 207        |
| 7.1.2.6    | Database Logical Lock   | 208        |
| 7.1.2.7    | Management of Data Element Presence   | 209        |
| 7.1.2.8    | Management of Data Element Check  | 209        |
| 7.1.2.9    | Sub-schema Management   | 209        |
| 7.1.3      | Use of Events   | 210        |
| 7.1.3.1    | Event-driven Management of Large Reading                                      | 210        |
| 7.1.3.2    | Event-driven Management of Instance Reading                                   | 211        |
| <b>7.2</b> | <b>Error Management</b>   | <b>211</b> |
| 7.2.1      | Local Errors  | 212        |
| 7.2.2      | Server Errors   | 213        |
| 7.2.3      | System Errors   | 213        |
| 7.2.4      | Communication Error Messages  | 214        |
| 7.2.5      | System Errors Generated by the Communication Monitor and the Services Manager | 214        |
| 7.2.5.1    | Errors Generated by the Communication Monitor                                 | 214        |
| 7.2.5.2    | Errors Generated by the Services Manager                                      | 214        |
| <b>7.3</b> | <b>Communication Management</b>   | <b>216</b> |
| 7.3.1      | Middleware Components   | 216        |
| 7.3.2      | Diagram of the Processing of a Request  | 217        |
| 7.3.3      | Definition of the Use Context   | 218        |
| 7.3.3.1    | Structure of the VAPLOCAT.INI File  | 218        |
| 7.3.3.2    | Implementation  | 221        |
| <b>7.4</b> | <b>Application Deployment</b>   | <b>221</b> |
| <b>8</b>   | <b>Pacbench Client/Server Bridge</b>  | <b>223</b> |
| 8.1        | Introduction  | 223        |
| 8.2        | Pacbench Client/Server Information Model (Graphic Client)                     | 224        |
| 8.2.1      | Business Logic Sub-schema   | 225        |
| 8.2.2      | Graphical User Interface Sub-schema   | 226        |
| 8.2.3      | VisualAge Entities  | 227        |
| 8.2.3.1    | Application Entity  | 228        |
| 8.2.3.2    | Part Entity   | 229        |
| 8.2.3.3    | Folder View Proxy Entity  | 230        |
| 8.2.3.4    | Elementary Proxy Entity   | 231        |
| 8.2.3.5    | Logical View Proxy Entity   | 232        |
| 8.3        | Operating Principles  | 233        |
| 8.4        | Backup: Upload to the Repository  | 233        |
| 8.4.1      | Java Local Backup   | 233        |
| 8.4.1.1    | Functions   | 233        |
| 8.4.1.2    | CLASSPATH Principle   | 233        |
| 8.4.1.3    | Starting the Program  | 234        |
| 8.4.1.4    | User Identification   | 235        |
| 8.4.1.5    | Search Options  | 235        |
| 8.4.1.6    | Modifying the Class Path  | 237        |
| 8.4.1.7    | Modifying the List of Packages  | 238        |
| 8.4.1.8    | Selecting Folder View Proxy   | 238        |
| 8.4.1.9    | Selecting and Generating the Output File                                      | 239        |
| 8.4.2      | Smalltalk Local Backup  | 239        |
| 8.4.2.1    | Functions   | 239        |
| 8.4.2.2    | Local Backup Main Window  | 240        |
| 8.4.2.3    | Window Listing Proxy Objects  | 240        |
| 8.4.2.4    | User Context  | 241        |
| 8.4.3      | Host Backup   | 242        |
| 8.4.3.1    | VUP1 Procedure: Computing Entity Codes  | 242        |
| 8.4.3.2    | VUP2 Procedure: Generation of the Update                                      | 243        |

|            |  |            |
|------------|--|------------|
| <b>8.5</b> | <b>Restoration of Data to the VisualAge Smalltalk Workstation .....</b>            | <b>244</b> |
| 8.5.1      | Host Restoration .....   | 244        |
| 8.5.1.1    | VDWN Procedure .....   | 244        |
| 8.5.2      | Local Restoration into VisualAge Pacbase for Smalltalk .....                       | 246        |
| 8.5.2.1    | Functionalities .....  | 246        |
| 8.5.2.2    | Access to the Local Restoration Window .....                                       | 246        |
| 8.5.2.3    | Pacbench C/S Restore window .....  | 246        |
| <b>8.6</b> | <b>Purging VisualAge Entities in the Repository .....</b>                          | <b>247</b> |
| 8.6.1      | Functions.....   | 247        |
| 8.6.2      | User Input.....  | 247        |
| 8.6.3      | Results .....  | 247        |
| <b>9</b>   | <b>Appendix 1: Parameterizing Prerequisite Software .....</b>                      | <b>249</b> |
| <b>9.1</b> | <b>IMS CPI-C .....</b>   | <b>249</b> |
| 9.1.1      | MVS and IMS Configuration .....  | 249        |
| 9.1.1.1    | VTAM Definitions .....   | 249        |
| 9.1.1.2    | APPC/MVS Definitions .....   | 251        |
| 9.1.1.3    | IMS Definitions .....  | 253        |
| 9.1.2      | OS/2 Configuration .....   | 254        |
| 9.1.2.1    | Communication Manager/2: APPC Configuration .....                                  | 254        |
| 9.1.3      | WINDOWS 95/NT Configuration.....   | 259        |
| 9.1.3.1    | Personal Communications: APPC Configuration .....                                  | 259        |
| <b>9.2</b> | <b>CICS CPI-C .....</b>  | <b>267</b> |
| 9.2.1      | MVS and CICS Configuration .....   | 267        |
| 9.2.1.1    | VTAM Definitions .....   | 267        |
| 9.2.1.2    | APPC/MVS Definitions .....   | 269        |
| 9.2.1.3    | CICS Definitions .....   | 269        |
| 9.2.2      | OS/2 Configuration .....   | 272        |
| 9.2.2.1    | Communication Manager/2: APPC configuration .....                                  | 272        |
| 9.2.3      | WINDOWS 95/NT Configuration.....   | 274        |
| 9.2.3.1    | Personal Communications : APPC configuration .....                                 | 274        |
| 9.2.4      | Windows NT Configuration .....   | 283        |
| 9.2.4.1    | SNA Server 3.0A_Configuration .....  | 283        |
| <b>9.3</b> | <b>CICS ECI.....</b>   | <b>298</b> |
| 9.3.1      | MVS and CICS Configuration .....   | 298        |
| 9.3.1.1    | VTAM Definitions .....   | 298        |
| 9.3.1.2    | APPC/MVS Definitions .....   | 300        |
| 9.3.1.3    | CICS Definitions .....   | 300        |
| 9.3.2      | OS/2 Configuration .....   | 303        |
| 9.3.2.1    | Communication Manager/2: APPC configuration .....                                  | 303        |
| 9.3.2.2    | CICS OS/2 .....  | 305        |
| <b>9.4</b> | <b>TCP-IP Socket via UNIX .....</b>  | <b>306</b> |
| 9.4.1      | Configuration.....   | 306        |
| 9.4.1.1    | Makefile Example .....   | 306        |
| 9.4.1.2    | Execution of the Listener on UNIX .....  | 306        |
| <b>9.5</b> | <b>Windows/NT TCP-IP Socket .....</b>  | <b>307</b> |
| 9.5.1      | Configuration.....   | 307        |
| 9.5.1.1    | Microfocus Compiling Parameterization on Windows/NT .....                          | 307        |
| 9.5.1.2    | Execution of the Listener on Windows/NT .....                                      | 307        |
| <b>9.6</b> | <b>CICS TCP/IP Sockets Interface.....</b>  | <b>308</b> |
| 9.6.1      | Configuration CICS TCP/IP .....  | 308        |
| 9.6.1.1    | Prerequisites .....  | 308        |
| 9.6.1.2    | CICS Startup .....   | 308        |
| 9.6.1.3    | Definition of CICS TCP/IP transactions .....                                       | 308        |
| 9.6.1.4    | Definition of CICS TCP/IP programs .....   | 310        |
| 9.6.1.5    | Definition of the DCT Table .....  | 314        |
| 9.6.1.6    | Definition and initialization of Configuration files (*TCP/IP Version 3.2.0) ..... | 314        |
| 9.6.1.7    | Definition of the PLT table (*TCP/IP V320) .....                                   | 319        |

|           |  |            |
|-----------|--|------------|
| 9.6.2     | Configuration TCP/IP MVS/ESA .....   | 319        |
| 9.6.2.1   | Modification of the TCP/IP Configuration .....                                     | 319        |
| 9.6.2.2   | TCPJOBNAME Parameter in the <i>hlq</i> .TCPIP.DATA File .....                      | 320        |
| 9.6.3     | Manual Start and Stop of CICS TCP/IP .....   | 320        |
| 9.6.3.1   | Start of CICS TCP/IP .....   | 320        |
| 9.6.3.2   | Stop of CICS TCP/IP .....  | 321        |
| 9.6.4     | Cobol Compilation.....   | 322        |
| 9.6.4.1   | Cobol Compilation of the VisualAge Pacbase Communications Monitor Program .....    | 322        |
| 9.6.5     | CICS definitions for the VisualAge Pacbase application .....                       | 322        |
| 9.6.5.1   | Definition of Transaction Code .....   | 322        |
| 9.6.5.2   | Definition of the Communications Monitor Program .....                             | 322        |
| 9.6.5.3   | Definition of the VSAM Workfile .....  | 323        |
| 9.6.6     | Client Configuration .....   | 323        |
| 9.6.6.1   | VAPLOCAT.INI File Parameters .....   | 323        |
| 9.6.6.2   | VP Middleware .....  | 324        |
| 9.6.6.3   | Traces .....   | 324        |
| 9.7       | Local Middleware .....   | 324        |
| 9.7.1     | Configuration.....   | 324        |
| 9.7.1.1   | Microfocus OS/2 Compiling Parameterization .....                                   | 324        |
| 9.8       | TUXEDO .....   | 325        |
| 9.8.1     | Client.....  | 325        |
| 9.8.2     | Server.....  | 325        |
| 9.9       | MQSERIES .....   | 325        |
| 9.9.1     | CICS Adapter .....   | 325        |
| 9.10      | CPIC-C/XCP2.....   | 328        |
| 9.10.1    | CPI-C/XCP2 – TDS Configuration .....   | 328        |
| 9.10.1.1  | Prerequisites .....  | 328        |
| 9.10.1.2  | Frontal CNP7 Configuration .....   | 328        |
| 9.10.1.3  | GCOS7 Network Configuration (NETGEN command) .....                                 | 328        |
| 9.10.1.4  | GCOS7 Configuration .....  | 329        |
| 9.10.1.5  | TDS Configuration .....  | 330        |
| 9.10.2    | CPI-C/OSI Configuration on AIX Server .....  | 332        |
| 9.10.2.1  | Definition of the XCP2 user profile on AIX (required configuring CPI-C/XCP2) ..... | 332        |
| 9.10.2.2  | CPI-C/OSI Configuration .....  | 333        |
| <b>10</b> | <b>Appendix 2: Use of a Customized Middleware.....</b>                             | <b>343</b> |
| 10.1      | VisualAge for Java .....   | 343        |
| 10.2      | VisualAge for Smalltalk .....  | 343        |
| 10.2.1    | Schema of Inheritance of Server Call Classes .....                                 | 343        |
| 10.2.2    | Instantiating Objects from the Pacbench Client/Server middleware.....              | 344        |
| 10.2.3    | Actions to Load again for Integrating Specific Middleware.....                     | 344        |
| 10.2.3.1  | Action callServer: with: with: with .....  | 344        |
| 10.2.3.2  | Creation of a Local User Buffer .....  | 345        |
| <b>11</b> | <b>Index .....</b>   | <b>347</b> |



## ***Foreword***

### **Contents of the Volume**

The object of this manual is to guide the developer in creating client GUI applications using Pacbench Client/Server Functionalities. The VisualAge Pacbase Business Logic generates the server part of these applications. The VisualAge Workstation is used to develop Java or Smalltalk clients that can be executed on a PC or accessed via the Web. The client applications can also be developed with any other development tool using COM technology, and for a Java Client, with any Java development tool using JDK version 1.1.

- The introduction gives you a brief description of the different steps of a graphic client development, from the generation of components extracted from the Repository to the graphic construction of applications.
- You will find then in a specific chapter, a detailed description of the generation of Proxies for each client (Java, Smalltalk and COM).
- The following chapter deals with the using principles of elements that make up the public interface for generated components and for each type of development environment. It also gives a presentation of error and communication management. Chapters dedicated to Java and Smalltalk clients provide examples with detailed comments on the development of standard GUI and Web applications. They also provide information on to test and package an application.
- You will then find a chapter on the Pacbench Client/Server Bridge. With VisualAge Pacbase for Java Bridge, the cross-references of the graphic components are saved in VisualAge Pacbase Repository. With VisualAge Pacbase for Smalltalk Bridge, the source and cross-references of the graphic components are saved in the Repository and restored into the VisualAge workstation.
- Appendix 1 gives details on how to parameterize external software and Appendix 2 gives advice on how to customize the middleware.

### **Prerequisites**

In this Volume, we assume that the reader is familiar with the main principles used in the development of Client/Server applications with VisualAge Pacbase. If not, see the *Pacbench C/S User's Guide, Vol. I: Concepts - Architectures - Environments*.

The *Pacbench C/S User's Guide, Vol. II – Business Logic* covers the development of server components that you may integrate to the client application as Proxy objects. A thorough reading of this Volume is not necessary to develop graphic clients, but it would help you to have a better understanding of Proxy objects.

Since this manual cannot include all the information relating to the development of Client/Server applications with VisualAge Pacbase, you will need to refer to the following manuals:

- *Pacbench Client/Server Reference Manual – Public interface of Generated Components,*
- *VisualAge Pacbase Workstation Reference Manual,*
- *Pacbench Client/Server Bridge Operations Manual* specific to your platform,
- Documentation associated with VisualAge for Smalltalk Workstation, including:
  - ♦ *VisualAge for Smalltalk - Getting Started,*
  - ♦ *VisualAge for Smalltalk - User's Guide,*
  - ♦ *VisualAge for Smalltalk - User's Reference,*
  - ♦ *VisualAge Web Connection - User's Guide,*
  - ♦ *IBM Smalltalk: Introduction to Object-Oriented Programming with IBM Smalltalk,*
  - ♦ *IBM Smalltalk: Programmer's Reference Manual,*
  - ♦ *IBM Smalltalk User's Guide.*
- Documentation associated with VisualAge for Java workstation, including HTML on-line documentation or the documentation associated with any other Java tool used.
- Documentation associated with your COM development environment.

### **Typographical conventions in use**

The Courier New font is used for any character strings you can enter, displayed by the product or corresponding to generated codes.

Italics is used for titles of publications or Chapter in cross-references.

The following symbols are used to point out:



a note, a remark.



cross-reference to another location in the documentation.



a helpful hint or tip, a useful piece of information.



an action to be carried out in a Tool or an Editor.



that you must proceed with caution (risky or irreversible action, etc.).

### **Terminology standards**

- FVP refers to a Folder View Proxy.
- LVP refers to a Logical View Proxy.
- Folder View Proxy, Elementary Proxy and Logical View Proxy are usually referred to as Proxy objects and Proxy components.



# 1 Introduction

Pacbench C/S enables you to develop GUI applications in VisualAge for Java, VisualAge for Smalltalk and also in any other development environment, which uses Microsoft COM specifications.

With the VisualAge workstation, besides standard GUI applications executable on a PC, you can create applications accessible from a Web browser (Intranet or Internet).

Whatever development environment you are working in, the principle is based on the use of a generated Proxy component which, once imported, is used to remotely command services on the corresponding Business Component(s).

You can use Pacbench C/S to harmoniously integrate servers developed using the Business Logic functionality with clients developed in a VisualAge or a COM standard environment. Pacbench C/S provides all the elements necessary to achieve a smooth interfacing between servers and clients.

☞ For further information on the Business Component, refer to the *Pacbench C/S User's Guide, Vol. II: Business Logic*.

It is also important to provide a unique data storage space to take advantage of the integration capabilities, ensuring global mechanisms, in particular reusability and cross references. Storing data in the VisualAge Pacbase Repository also benefits from the management of versions and security procedures (archiving, backup, and reorganization).

The repository is therefore the reference storage space for graphic clients developed in VisualAge.

All objects handled in VisualAge can be stored in the Repository and subsequently restored in the VisualAge workstation (for Smalltalk only). This is the underlying principle of the Pacbench Client/Server Bridge.

## 1.1 Steps of Development

Developing a graphic application in Pacbench C/S consists in developing Server Components first, and Client Components then.

☞ The development of servers – including the development of Business Components, and possibly the Folder and Folder View development in the VisualAge Pacbase workstation, the extraction of Business Components (in the case of a single-view development) or a Folder View – is documented in the *Pacbench C/S User's Guide, Vol. II: Business Logic*.

The development of Client consists in:

- generating/Importing Proxies

The generation/importation phase produces classes whose objects can be re-used by the client applications to be developed in the Client workstation. In VisualAge it is carried out by a generator controlled by a GUI interface.

This generator takes the input extraction result initially transferred on to the client development workstation. The elements, which make up the Folder View or the Business Component(s), are then grouped together in the Proxy object. All the elements required to call associated Business Components will be present and initialized in the client application.

For more details on the generation features, refer to Chapter 2 for a Java Client, Chapter 3 for a Smalltalk Client or Chapter 4 for a COM standard Client

- Developing a client application

Developing a client application implies integrating Proxies and calling their services.

You will find all the information on the actions corresponding to these services in section 5.1.2.5 for a Java Client, 6.1.2.7 for a Smalltalk Client or 7.1.1.2 for a COM standard Client.

You will also find an example with comments on the development of a standard and web application in Subchapters 5.2 and 5.3 for a Java Client, 6.2 and 6.3 for a Smalltalk Client.

Developing a the client application also implies creating a graphic application based on the standard tools of your development workstation and possibly writing codes. These subjects are not dealt with in this manual. Refer to the VisualAge documentation or the documentation associated with any other Client development tool.

## 1.2 Compatibility of Business Components / Proxy Objects

Business Components and Proxy objects must be generated with the same version number, for a difference of version can produce discrepancies showing inconsistency in the client application.

The version number is different in the following cases:

- If you have generated a new version of the Business Component without re-generating the associated Proxy object,
- Or if the generated graphic application, including the new Proxy object, has not been implemented in VisualAge,
- Or if the generated Business Component has not been implemented.

To avoid possible discrepancies, you must implement version control by setting an option in the Business Component. This option sends an error message if a discrepancy between the version numbers is detected when the VisualAge Client calls the Business Component.

The option handling the version numbers is the **NUVERS** option. For more details, refer to the *Pacbench C/S User's Guide, Vol. II: Business Logic*.

## 2 VisualAge for Java Generation

To develop a Pacbench C/S Client in VisualAge, you will use generated classes and a great deal of generic classes, which are delivered with the product to avoid the multiplication of elements at each new generation. Contrary to the generated classes documented further, generic classes do not depend on the characteristics of the processed Logical View.

### 2.1 Delivered Generic classes



You must make sure that generic classes are installed on your workstation before starting developing your application.

The Pacbench C/S generic classes are delivered in the 3 following packages:

- `com.ibm.vap.generic`

This package contains:

- The Parent classes of the `ProxyLv` generated classes, that is to say, the following classes:
  - ♦ `ProxyLv`
  - ♦ `HierarchicalProxyLv`
  - ♦ `DependentProxyLv`
  - ♦ `Folder`
  - ♦ `ReferenceProxyLv`
- The `Data` generic classes (Parent of `selectionCriteria` and `DataDescription` generated classes)
- The classes of the local cache, that is to say, the following classes:
  - ♦ `Node`
  - ♦ `HierarchicalNode`
  - ♦ `DependentNode`
  - ♦ `RootNode`
  - ♦ `ReferenceNode`
- Pacbench C/S exceptions and errors, that is to say, the following classes:
  - ♦ `VapException`
  - ♦ `LocalException`
  - ♦ `ServerException`
  - ♦ `CommunicationError`
  - ♦ `SystemError`
- The layout classes in the form of a list of `DataDescription` generated classes (handled by the `rows` property of Proxy objects).
- The classes describing the properties of the Proxy objects as defined in VisualAge Pacbase:
  - ♦ `VapProxyProperties`
  - ♦ `VapHierarchicalProxyProperties`
  - ♦ `VapDependentProxyProperties`
  - ♦ `VapFolderProperties`
  - ♦ `VapReferenceProxyProperties`

- `com.ibm.vap.exchange`

This package contains the classes handling the Exchange Manager.

### 2.1.1 On-line Documentation of Generic Classes

An HTML-formatted documentation is delivered with the Pacbench C/S runtime. This documentation deals with:

- Generic classes, parent of the `ProxyLv` and `data` generated classes used on execution,
- Errors and exceptions raised by these execution classes,
- The beans associated with VA Pac Data Element-type properties, used in the Composition Editor for a quick mapping of the data.

In the `awt` palette, these beans are:

- ♦ `Pacbase Text Field`
- ♦ `Pacbase Integer Field`
- ♦ `Pacbase Decimal Field`
- ♦ `Pacbase Date Field`
- ♦ `Pacbase Time Field`
- ♦ `Pacbase Long Field`
- ♦ `Pacbase Text Choice`
- ♦ `Pacbase Integer Choice`
- ♦ `Pacbase Decimal Choice`
- ♦ `Pacbase Date Choice`
- ♦ `Pacbase Time Choice`
- ♦ `Pacbase Long Choice`

In the `swing` palette, these beans are:

- ♦ `Pacbase Swing Text Field`
- ♦ `Pacbase Swing Integer Field`
- ♦ `Pacbase Swing Decimal Field`
- ♦ `Pacbase Swing Date Field`
- ♦ `Pacbase Swing Time Field`
- ♦ `Pacbase Swing Long Field`
- ♦ `Pacbase Swing Text ComboBox`
- ♦ `Pacbase Swing Integer ComboBox`
- ♦ `Pacbase Swing Decimal ComboBox`
- ♦ `Pacbase Swing Date ComboBox`
- ♦ `Pacbase Swing Time ComboBox`
- ♦ `Pacbase Swing Long ComboBox`
- ♦ `Pacbase Swing Date RadioButtonGroup`
- ♦ `Pacbase Swing Decimal RadioButtonGroup`
- ♦ `Pacbase Swing Integer RadioButtonGroup`

## 2.2 Starting the Generator

To start the generator, follow these steps:

- In a DOS or OS/2 window, execute the following command  
`java com.ibm.vap.generator.view.Main [-lang [fr] [en]]`

☞ To be able to use this command, the JDK (Java Developer ToolKit) must be installed on your workstation and the `java` command must be declared in the `PATH`.

The `lang` option enables you to select the language of the generator's graphic interface: `fr` for French, `en` for English.

The language of the system is used by default.

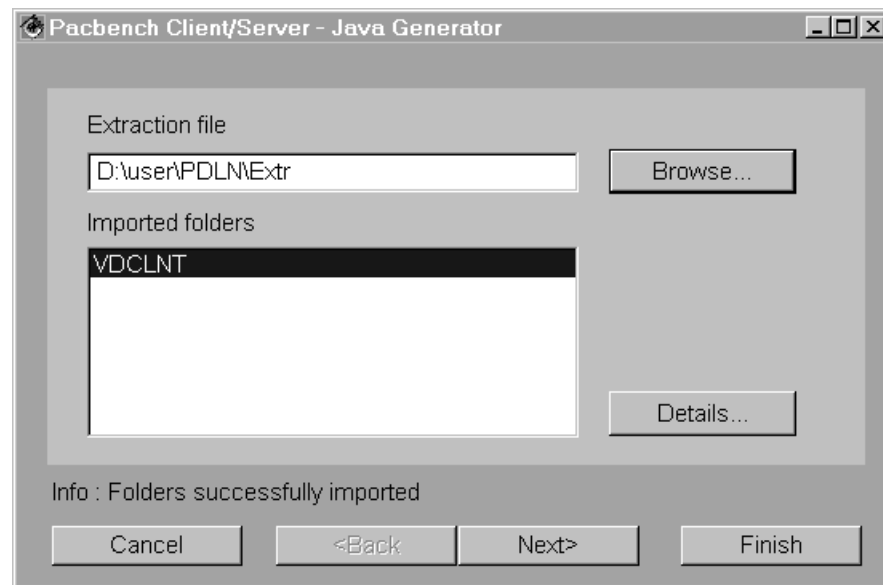
The generator is also provided with the following generation options:

- ♦ `-input INPUT_FILE`: this option allows you to specify the extraction file.
- ♦ `-output OUTPUT_DIR`: this option allows you to specify the directory in which the generated classes will be stored.
- ♦ `-proxy PROXY_PACKAGE`: this option allows you to specify the Java package in which the classes corresponding to the Proxy components will be generated.
- ♦ `-data DATA_PACKAGE`: this option allows you to specify the Java package in which the data classes will be generated.
- ♦ `-beans`: this option is used to generate classes whose objects are in compliance with the specifications of JavaBean components.
- ♦ `-swing`: this option is used to generate all the data required for the operations of Swing graphic classes.
- ♦ `-eab`: this option is used to generate classes able to interface with IBM EAB classes.

☞ All these options correspond to generation parameters in the graphic interface of the generator; for more details, refer to section *Operating Mode*.

- Or, start the `vapgen.exe` program from Windows File Manager or from Windows NT Explorer program.
- If the generator has been imported previously in the VisualAge workstation:
  - ♦ In the Workbench, select the Project in which the generator has been imported.
  - ♦ To choose the language of the generator, select the package `com.ibm.vap.generator.view` and then the `Main` class. Display the properties of this class. In the window which opens then, in the `Command Line Argument` field, enter the language option as follows: `-lang en` or `-lang fr`. If this field is left blank, the generator's interface will be in English.
  - ♦ To start the generator, display the pop-up menu of the project. Select the `Run` sub-menu, and choose `Run Main...` and then `Main`, or click on the `Run` icon in the tool bar and then select `Main`.

The generator's main detail is displayed:



## 2.3 Operating Mode

- **Selecting the Folder Views to be generated**

You need to enter the access path to the extraction file in the **Extraction file** field. To do this, you can:

- Enter the access path in the edit box, or
- Click on the **browse...** button. A window is displayed in which you can choose the complete file path by using the different list boxes.

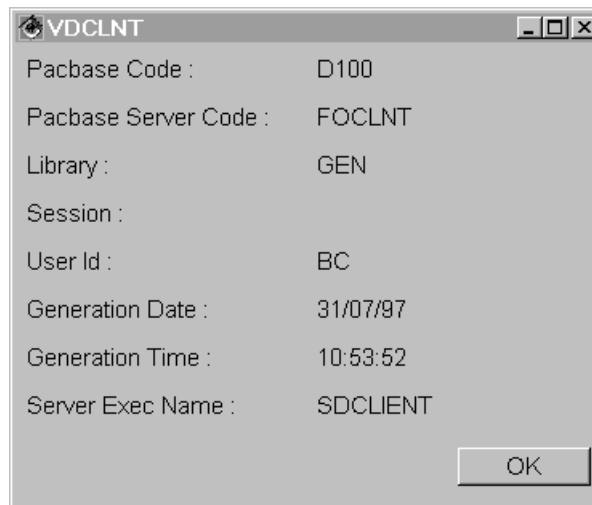
Once you have entered the file name, the list of Folder View(s) contained in the extraction file is displayed in the **Imported Folders** field.

If there are several Folder Views in the list, they are all automatically selected. In this case, you can deselect those that you don't want to generate.

- ☞ The **Folders successfully imported** message means that the extraction file is correct and that the Folder Views it contains have been identified and are ready for generation.

- **Other functionalities available in the detail**

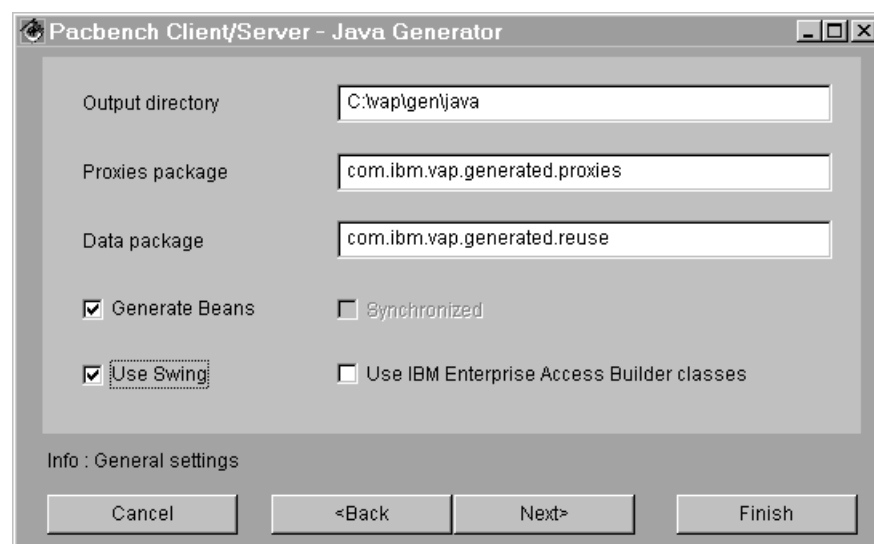
- The **Details...** button is used to display the **GVC** generation context of the selected Folder View.




- Clicking the **Next** button displays a detail in which you will specify generation parameters. For more details on the default parameters, refer to the following paragraph.
- If you want to generate immediately all selected the Folder Views, by using the default parameters, click the **Finish** button. A message box is displayed, you are asked to confirm the creation of an output file if it does not exist yet. Click OK. Generation begins.

- **Generation options**

If you have clicked on **Next** in the main detail, the following detail is displayed:



This detail is used to enter the generation parameters. It contains the default parameters.

- The **Output directory** field contains the access path to the directory in which the classes are generated.
- The **Proxy objects Package** field contains the name of the Java package in which the classes corresponding to the Proxy components are generated.
- The **Data Package** field contains the name of the Java package in which the data classes are generated.
- If the **Generate beans** option is activated, the generator creates beans or Java standard classes.
  - ☞ This option is used to generate classes whose objects are comply with the specifications of JavaBeans, used in visual programming. Refer to Subchapter 2.5, *Generation Results*.
- If the **synchronized** option is activated, the generator creates classes supporting multithreading.
  - ☞ For example, it will be possible for Proxy components to be called simultaneously by several programs or *threads*.
- If the **Use Swing** option is activated, the generator creates all the data required for the operations of the Swing graphic classes integrated into VisualAge Java from version 2.0 onwards.
  - ☞ It will be possible to connect the **rows** property of the Proxy components to the data of a JTable (Swing component with rows and columns).
- If the **Use IBM Enterprise Access Builder classes** option is activated, the generator creates classes able to interface with IBM EAB classes.
  - ☞ It will be possible to connect the **rows** property of the Proxy components to an EAB container data in Visual Composition Editor.
    -  This option generates errors if it is used with VisualAge Java from version 2.0 onwards.

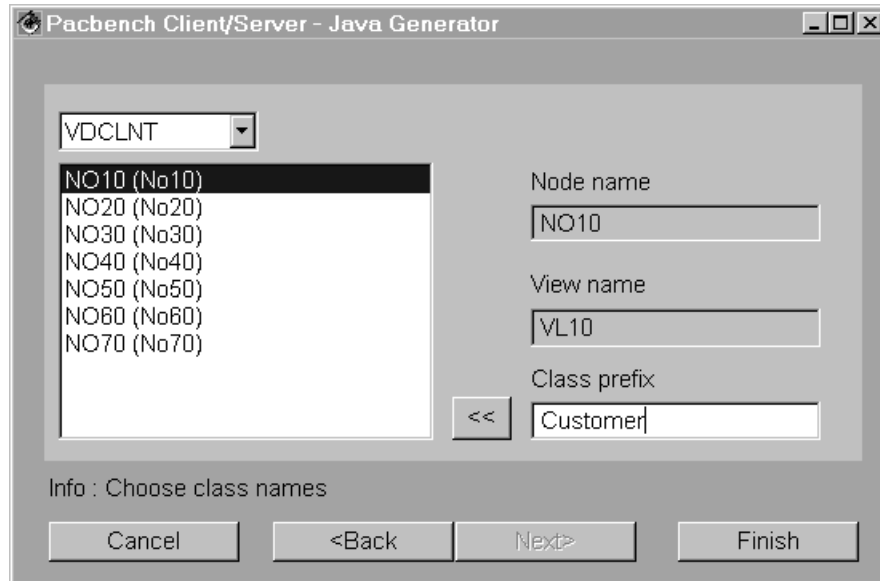
This detail also contains the following push buttons:

- The **Back** button displays the main detail.
- The **Next** button displays the detail in which the generated classes prefixes can be modified.
- If you want to generate immediatly all the selected Folder Views, by using the default classes name, click the **Finish** button. A message box is displayed, you are asked to confirm the creation of the output directory if it has not been created yet. Click **OK**; generation begins.




- **Modifying the prefixes of the generated classes**

If you clicked on **Next** in the generation parameters detail, the following detail is displayed:



This detail is used to modify the prefixes of the generated classes when the prefixes assigned by the server do not suit you (for example: because they are not explicit enough). By default, the prefix is the name of the Logical View associated with the node.

The drop-down list contains the list of all the Folder Views selected for generation in the main detail.

- The list contains all the nodes that make up the Folder View selected in the drop-down list. Each node can be identified by its code and by the Logical View's name (prefix).
- For each node selected in the list, the right part of the detail gives the name of the node and that of the associated Logical View.
- To modify a node's prefix, follow these steps:
  - ♦ Select the node in the list,
  - ♦ Enter a new prefix in the **Class prefix** field,
  - ♦ Click on  to assign the prefix to the node, or press the **Enter** key to assign the prefix to the node and select automatically the next node in the list.

**EXAMPLE:**

If a node's prefix is **Customer**, the classes generated for this node are coded as follows:

- ♦ **CustomerProxyLv**
- ♦ **CustomerDataDescription**
- ♦ **CustomerSelectionCriteria**
- ♦ etc.



You must make sure of the uniqueness and validity of the class names.

This detail also contains the following buttons:

- The **Back** button displays the previous detail.
- The **Finish** button starts generation. A message box is displayed, you are asked to confirm the creation of the output directory if it has not been created yet. Click OK; generation begins.

## 2.4 Importing the Generated Classes in VisualAge

To import the generated classes in VisualAge, proceed as follows:

- From VisualAge Workbench, select the **Import...** choice in the **File** menu.
- In the **SmartGuide - Type of import** window, enter the name of the projet in which classes are imported or select, via the **Browse** button, a project which has already been created.
  - ☞ We advise you not to import generated classes in a VisualAge Pacbase or Java standard project.
- As for the import type, select the **Entire Directory (Including resources)** option.
  - ☞ The **Java files** option can also be selected. In this case, the icons specific to Pacbench Client/Server will not be available.
- Click **Next** to get to the following detail. Select, via the **Browse** button, the directory in which classes have been generated.
  - ☞ Use the directory which was selected for generation in the **Output directoy** field, in the generation option detail. In our example, it is **C:\vap\gen\java**.

## 2.5 Generation Results

The generated elements always depend on the type of service carried out by the Business Component. They will not be the same depending on whether the Business Component updates or just reads.

The generated file only contains the classes which depend on the characteristics of the processed Logical View.

### 2.5.1 Introduction

Once generation is over, the following files are created:

- The **VAPLOCAT.INI** location file is created in the generation output directory.



This file must be completed manually. For more information on how to proceed, refer to section **5.5.3**.

This file must be entered at the gateway start.

- The source files of the generated classes and resources required for editing them.
  - ☞ If you have checked the **Generate beans** option, a **BeanInfo** class is generated for each execution class (**ProxyLv** and **data**). The **BeanInfo** class associated with a class (bean) bears the name of the this class, with a **BeanInfo** suffix at the end.

For example, when a `CustomerProxyLv` class (Root Proxy in our example) is generated, a `CustomerProxyLvBeanInfo`, is systematically generated.

A `BeanInfo` class contains editing information (label, icon, etc.) for the associated execution class. It contains public methods that give information on the associated bean class, such as the class name, properties, methods and events available on the bean.

## 2.5.2 Generated Classes

In VisualAge, the elements generated by Pacbench C/S correspond to classes whose coding consists of a prefix and a hard-coded part assigned by the generator. The prefix corresponds to the Logical View's name, whose maximum length is 36 characters.



If you are not satisfied with the server's prefix, you can change it during the generation. For more details, refer to the point *Modifying the Prefixes of the Generated Classes*, in subchapter 2.3.

- `[Prefix]Data` Class

This class represents the description of a Logical View instance. It contains a set of properties corresponding to the Logical View Data Elements.

- `[Prefix]SelectionCriteria` class

This class represents the description of the selection criteria. It contains a set of properties corresponding to identifier data elements and extraction parameters associated with the Logical View.

- `[Prefix]Buffer` Class

This class represents the description of the contextual information. It contains a set of properties corresponding to the user buffer Data Elements.

- `[Prefix]DataUpdate` class

This class inherits from the `[Prefix]Data` class. Compared to its parent class, it holds 2 additional properties whose values vary according to the modifications in progress. These two properties are the following ones:

- ♦ `action` : update action which can be `Read`, `Modified`, `Created` or `Deleted`. This property is only visible in the list of the `UpdatedFolders` property.
- ♦ `updatedInstancesCount` : number of useful server updates associated with the Folder instance concerned and whose value can be 1 to n.

- `[Prefix]UserData` class

This class inherits from the `[Prefix]Data` class. It contains an additional property that corresponds to the key Data Element of the parent instance.

- `[Prefix]TableModel` class

This class is generated only if you checked the `Use Swing` option at generation time. It inherits from the `Pacbase TableModel` class and implements the `TableModel` swing component, used to fill in a swing table with data. This generated class is used to display the list of instances of the `[Prefix]Data` class generated in the swing table.

- **[Prefix]UpdateTableModel** class

This class is generated only if you checked the **Use Swing** option at generation time. It inherits from the **Pacbase UpdateTableModel** class and implements the **TableModel** swing component, used to fill in a swing table with data. This generated class is used to display the list of instances of the **[Prefix]DataUpdate** class generated in the swing table.

### 2.5.2.1 On-line Documentation of the Generated Classes

The documentation on the generated classes' public interface is directly integrated into the code in the form of comments.

You can therefore consult this documentation from the source of the desired element in the Workbench or a VisualAge browser.

You can also generate this documentation by using the Javadoc Facility delivered with JDK.

#### 2.5.2.1.1 Generating Documentation

You can generate documentation associated with a project, a package, a class or even a method.

To start Javadoc, you have two possibilities:

- From the Workbench or a browser in VisualAge:
  - ♦ select the desired project, package, class or method.
  - ♦ open the associated pop-up menu and select the **Generate javadoc** choice. The documentation is generated by default in the VisualAge directory, ...\**Ide**\javadoc Subdirectory.
- From a DOS or OS/2 window, by using the default parameters of the Proxy components generator, execute the following command:

```
javadoc -classpath c:\vap\generated\java -d c:\doc -public  
com.ibm.vap.generated.data com.ibm.vap.generated.proxies
```

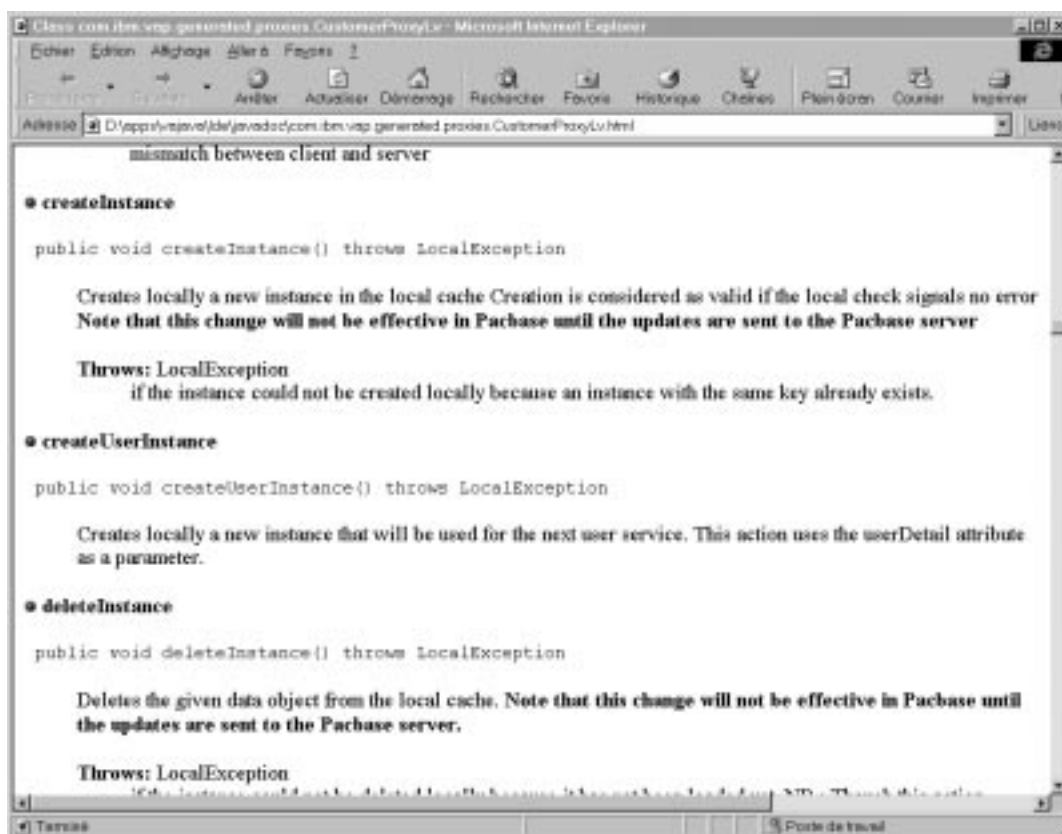
#### 2.5.2.1.2 Results

The generated documentation only contains information useful to the developer, for it corresponds exactly to each generated class.

Only the documentation of the public interface elements - properties or methods –**actually generated**– is extracted. It is an HTML-formatted documentation including hypertext links.

For each method, besides the comments, its signature – parameter(s), return code and exceptions – is extracted as well.

For information only, an example is presented:



☞ You can refer to the *Graphic Clients Reference Manual: Public Interface of Generated Components* where properties, methods and events are documented thematically.

### 2.5.2.2 Customizing Classes

All generated classes inherit from generic classes.

Generic classes are loaded upon the installation of the product.

If you want to implement new functions in the generated Proxy Components, you must follow this inheritance system to be able to create new classes allocated to these functions. Do not modify parent and generated classes.

#### 2.5.2.2.1 Customizing ProxyLv Classes

You can add a behavior common to all Root, Reference or Depending Proxy objects.

You just have to create a class inheriting from **DependentProxyLv**, **FolderProxyLv** or **ReferenceProxyLv**, then implement a new functionality and modify the parent class of the generated Proxy objects.

☞ You need to repeat this modification at each generation.

### ***2.5.2.2.2 Customization of Data Classes***

You can change the **Data** class used by a given Proxy.

To do so, you have to define a class inheriting from the **DataDescription** class, then perform the required implementation, and modify the **newData()** and **newData(String[] values)** methods of the Proxy.





## 3 VisualAge for Smalltalk Generation

To develop a Pacbench C/S Client in VisualAge, you will use generated classes and a great deal of generic classes, which are delivered with the product to avoid the multiplication of elements at each new generation. Contrary to the generated classes documented further, generic classes do not depend on the characteristics of the processed Logical View.



On installation, all these classes are not displayed in the VisualAge Organizer. To reach your insertion point, you must first display the classes. To do this, from the **Applications** menu, select **View** then **Show All Applications**.

### 3.1 Delivered Generic Classes



You must make sure that generic classes are installed on your workstation before starting developing your application.



The generic classes from the 2.5 V08 version onwards (**VpCS** type) cannot be used with classes generated with a lower version. If you want to use these new generic classes, you must regenerate all the classes of your application to be able to execute this application.

#### 3.1.1 Communication

The Client communication relies on on a Middleware interface of Pacbench C/S. Additionally, it is possible to specify different locations for the various targets.

These classes are required for using the product, whether at execution, edition or generation.

##### 3.1.1.1 Middleware Interface

The middleware interface enables to communicate with the DLLs provided with the product to send or receive messages to/from the server.

The classes that implement this interface are stored in the **VapCommApi** application.

##### 3.1.1.2 Locations Manager

The locations manager is responsible for managing the various server configurations that are available.

The classes that implement the locations manager are stored in the **VpCSLocationsApp** application.

#### 3.1.2 Runtime

The runtime generic classes are required for the execution of any application using a Pacbench C/S Client. These classes are stored in several applications or sub-applications according to the implemented functionality.

##### 3.1.2.1 Local Cache

The local cache is used to store and to manage the Client instances that are either retrieved from or to be sent to the server.

The classes that implement the local cache are stored in the **VapCacheManagerSubApp** sub-application in the **VapRunGenericPartsApp** application.

### 3.1.2.2 Exchange Manager

The exchange manager is responsible for handling the requests between the Client and the middleware.

The classes that implement the exchange manager are stored in the **VapExchangeManagerSubApp** sub-application in the **VapRunGenericPartsApp** application.

### 3.1.2.3 Communication Manager

The communication manager makes the interface between the exchange manager and the middleware easier.

The classes that implement the communication manager are stored in the **VapCommunication** sub-application in the **VapRunGenericPartsApp** application.

☞ The class **VapServerManagement** is the entry point for the communication processing. This class holds a child class **VapLUWServerManagement** which is responsible for handling the communication.

This class is specialized by the **VapLUWServerUserManagement** which enables users to implement their own communication protocol.

☞ Pour des informations sur la personnalisation du middleware, reportez-vous au chapitre 10, sous-chapitre *VisualAge for Smalltalk*.

### 3.1.2.4 Folder View Model

The Folder View model defines a set of objects allowing to retrieve and to handle data from the server.

The classes that implement this model are stored in the **VpcsCommonRuntimeApp** and **VpcsRuntimeApp** applications and the depending sub-applications.

☞ To ensure the compatibility with versions lower than 2.5 V08, the former classes of the model are kept in the **VapGenericProxysSubApp** sub-application in the **VapRunGenericPartsApp** application.

Among these classes, the **VapUserServiceError** class, related to error handling, can be customized.

## 3.1.3 Edition

The edition classes allow you to handle and manipulate the GUI interface. These classes are only used to edit the objects in the Composition Editor.

In this edition step, it is supposed that the generation phase has been executed, but mais it is not necessary to have loaded the generator to use these classes.

The edition classes are stored in the **VpcsCommonEditionApp** and **VpcsEditionApp** applications and the depending sub-applications.

☞ To ensure the compatibility with versions lower than 2.5 V08, the former edition classes are kept in the **VapEditGenericPartsApp** application and the depending sub-applications.

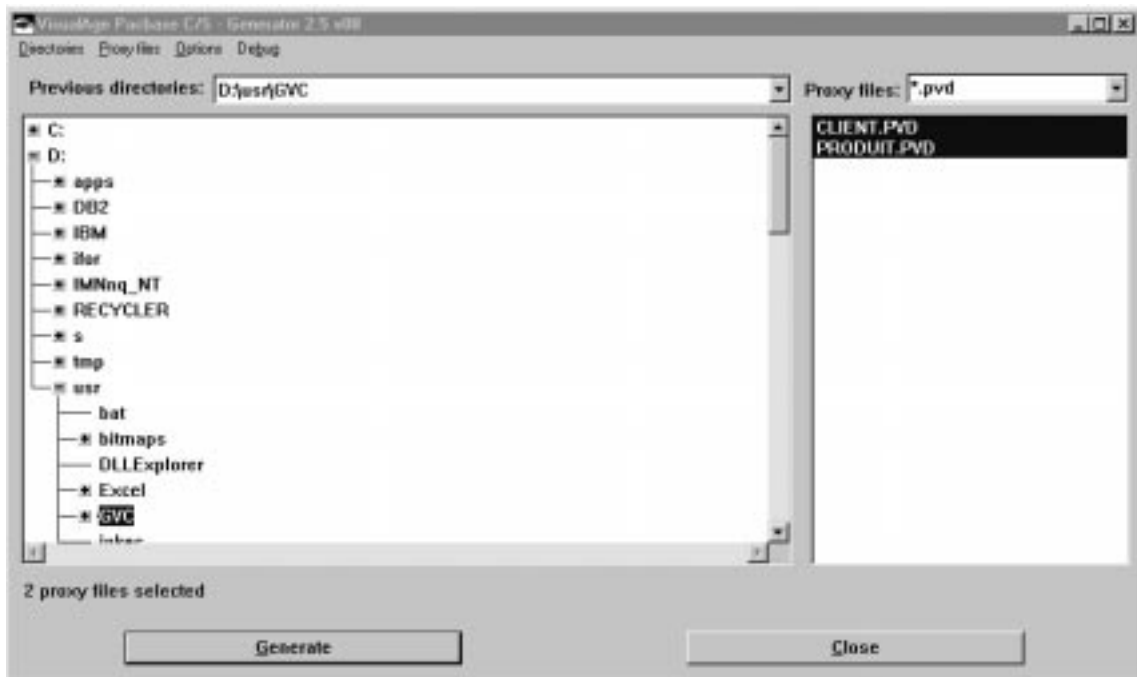
## 3.2 Generation User Interface

### 3.2.1 Starting the Generator

The Pacbench C/S generator generates Folder View Proxy objects.

You start the generation from the Pacbench C/S **Generator** window which can be accessed via the **Pacbase C/S Generator** option of the **Tools** menu in the VisualAge Organizer.

The main window opens:



### 3.2.2 Selecting Files

- You first need to enter the access path to the file(s) to be generated:
  - select them in the **Previous directories** drop-down list displaying the memorized directories.
 

This drop-down list is implicitly updated when you clicked on the **Generate** button. It can also be explicitly updated using **Directories** menu or the popup menu.
  - or select them using the directory tree.
- Then, you need to select the file(s) to be generated in the list. The number of selected valid files are displayed on the information line at the bottom of the generation window.



You can filter the files to be displayed in the list by selecting a generic file name in the **Proxy files** combo-box. The user can add customized filters which are memorized in the list when starting the generation. The user can also delete filters from the list using the **Proxy files** menu or the popup menu.

### 3.2.3 Description of Menus

The generation user interface contains three main menus.

### 3.2.3.1 Directories Menu

This menu contains the following options:

- **Add Selected Directory To Recall List:** adds the selected directory to the drop-down list.
- **Remove Directory From Recall List:** deletes the displayed directory from the drop-down list.

### 3.2.3.2 Proxy files Menu

- **Generate:** generates the selected file(s). This option is enabled if at least one of the selected files is valid.
- **View Log:** displays the last generation report of the selected file(s). This option is enabled if there is a generation report for at least one of the selected files. The generation reports are stored in the same directory as the files to be generated, has the same name as the corresponding file to be generated but their extension is **.log**.
- **Remove Proxy File Wildcard:** allows you to delete a generic file name from the **Proxy Files** combo-box.



The options from this menu can also be accessed using the popup menu of the list of files to be generated.

### 3.2.3.3 Options Menu

This menu comprises the following general options:

- **View Log After Generation:** displays or not the generation report just after the generation ends (**default: enabled**)
- **Force Library Update:** allows you to force the update of properties in the Smalltalk library even when no delta is stated (**default: disabled**)
- **Generate UserContext Classes:** generates the **UserContext** classes (**default: enabled**).
- **Optimize Generated Code:** optimizes the generation of properties for Proxy-type classes (**default: disabled**). When this option is used, only the minimum functional properties are generated.

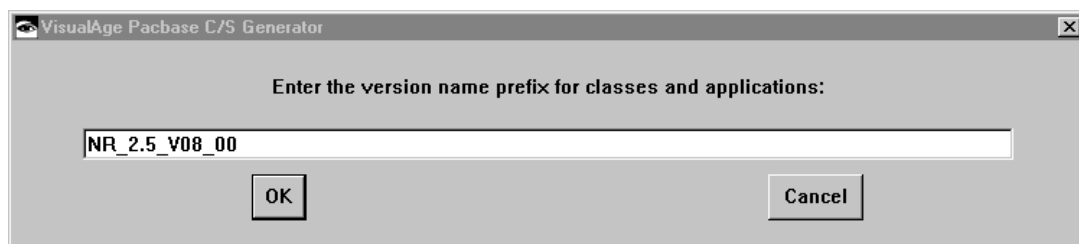


When enabled, some of the optional properties are not generated.

- **Bypass Repository Verification:** allows to de-activate the check on the origin of the file to be (VisualAge Pacbase database and library) (**default: disabled**). If this option is set and the check fails, the generation is not stopped; but a warning message will be displayed in the generation report.

The **Options** menu also comprises options for handling the applications during generation.

- **Set Version Name:** opens a dialog box allowing you to specify the version code that will be used for versioning the classes and applications:



The following options allow you to refine the generator behavior in relation to the applications and sub-applications involved in the generation.

- **Application Edition Creation Mode**

This option comprises a sub-menu with the following choices:

- ♦ **Always Create New Edition:** when activated, an edition will be automatically created for the versioned applications.
- ♦ **Confirm Before Creating Edition:** you are asked to confirm the creation of an edition for the versioned applications (**default mode**).
- ♦ **Confirm Before Creating Scratch Edition:** you are asked to confirm the creation of a scratch edition for the versioned applications.
- ♦ **Stop Generation If Not an Edition:** the generation immediately stops as soon as an application is not in edition mode.

- **Application Edition Versioning Mode**

This option comprises a sub-menu with the following choices:

- ♦ **Always Version:** the applications in edition will be automatically versioned.
- ♦ **Confirm Before Versioning:** you are asked to confirm the versioning of the applications in edition.
- ♦ **Never Version:** the applications will not be versioned (**default mode**).

☞ These choices are not available when the user has selected the scratch edition mode for the applications.

- **Application Version Release Mode**

This option comprises a sub-menu with the following choices:

- ♦ **Always Release:** the versioned applications will be automatically released.
- ♦ **Confirm Before Releasing:** the versioned applications will be automatically released after confirmation by the user.
- ♦ **Never Version:** the applications will not be released release (**default mode**).

☞ These choices are not available if the user has selected the scratch edition mode for the applications.

The values of all these options which are automatically saved will be the current values the next time you start the generator. The default values indicated are initial values.

### 3.2.4 Generation Execution

The message bar below the two buttons displays information about how the generation progresses and possibly, error messages.

Once the different parameters and options are specified in the main window, start the generation clicking on the **Generate** button.



The **Generate** button is disabled if none of the selected files is valid.

If at least one of the classes to be generated is not in the image, a dialog box opens up to allow you to map the new classes to an application.

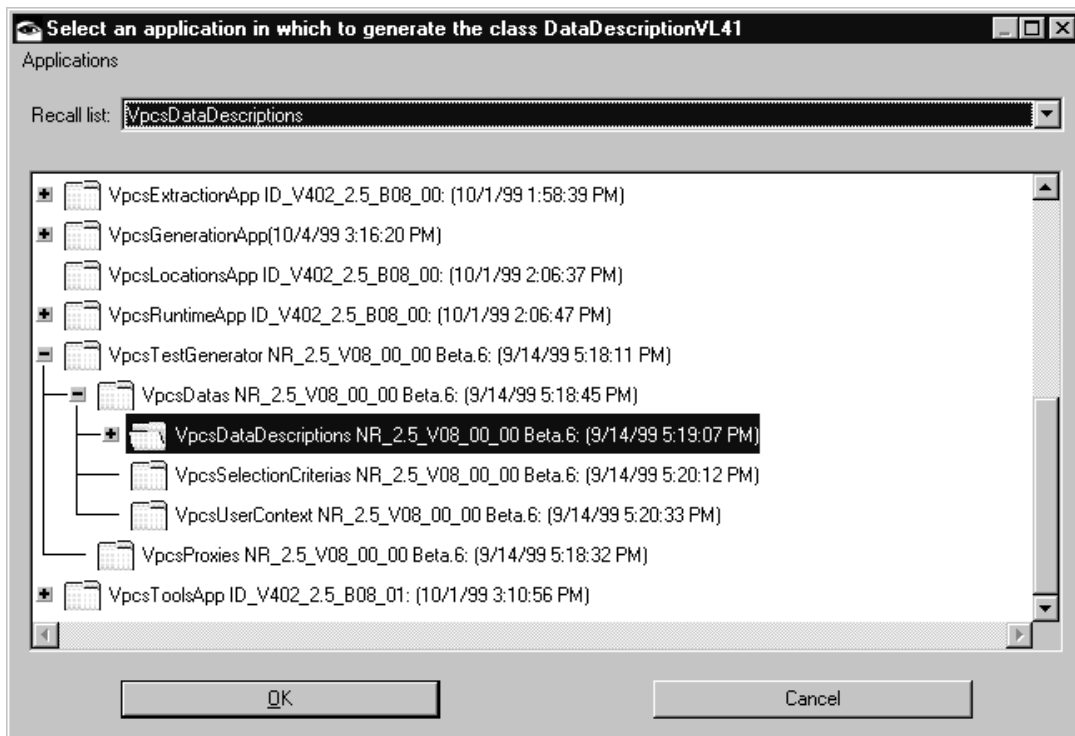


See below the paragraph *Applications Mapping*.

### 3.2.5 Applications Mapping

This window opens up if at least one of the classes to be generated is not in the VisualAge image.

All the **ProxyLv** classes of a Folder are generated in the same application as the Root Proxy whereas the **data** classes of a Folder can be mapped to several applications.

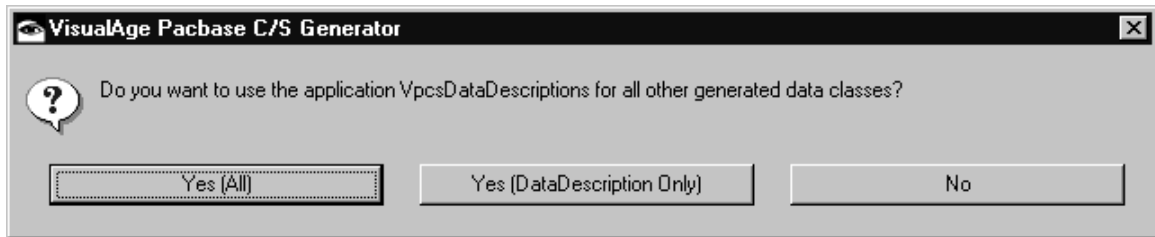


This window allows you to specify the VisualAge application in which the class whose name is displayed in the title bar will be generated.

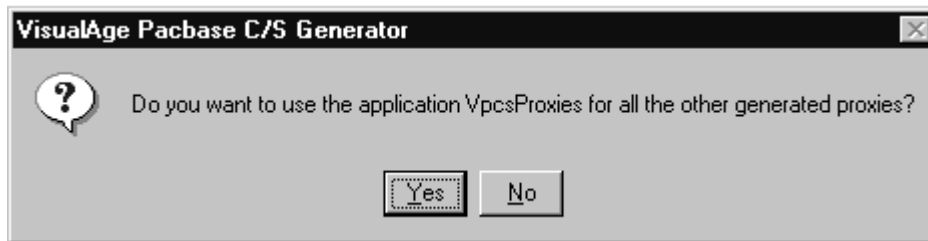
- Select the application:
  - by choosing it in the list of applications or in the recall list,
  - or by creating it as follows:
    - ♦ From the popup menu or the **Applications** menu, select the **New...** option
    - ♦ enter the new application name and validate.
- Validate your selection by clicking OK.

A dialog box opens up:

- For the **data** classes:



- ♦ **Yes (All)**: All the **data** classes that are not mapped to an application yet will be generated in the application previously selected or created.
  - ♦ **Yes (type Only)**: All the **data** classes **of the same type** that are not mapped to an application yet will be generated in this application.
  - ♦ **No**: This application will not be mapped to the other **data** classes to be generated.
- For the **ProxyLv** classes:



- ♦ **Yes**: All the **ProxyLv** classes that are not mapped to an application yet will be generated in the application previously selected or created.
- ♦ **No**: The application will not be mapped to the other **ProxyLv** classes to be generated.

☞ As the entity used for application mapping is the Folder View Proxy, all the **ProxyLv** classes making up this Folder View Proxy are automatically mapped to one application.

**Reminder** The generated **ProxyLv** classes correspond to the Folder View Proxy or Elementary Proxies. The **data** classes designate the generated classes corresponding to the Logical View data: **DataDescription**, **SelectionCriteria**, **UserContext**, **UserDataDescription**, **UpdatedDataDescription**, **ValuesOf** et **ValuesOf(...)**Converter.

### 3.2.6 Generation Report

The report resulting from the generation is automatically displayed if the **View Log After Generation** option has been enabled. If not, it is possible to view the generation report(s) using the **View Log** option.

The report describes the various steps of the generation.

```

VisualAge Pacbase C/S Generator cumulative log
File Edit

Create applications...

Create application editions...

Generate classes...

----- Generation of GVC file D:\usr\GVC\Tests\GVC127.txt -----

Date of generation is: 10/6/99
Start generation at: 7:53:43 AM

===== Start generation of Class DataDescriptionVL41 =====

DataDescriptionVL41 class>>#IS_codpri created.
DataDescriptionVL41 class>>#IS_datfax created.
Info: 42 DataDescriptionVL41 class overriding #IS_instanceInterfaceSpec
DataDescriptionVL41 class>>#IS_instanceInterfaceSpec created.

*** Begin Generate Selectors ***
DataDescriptionVL41 class>>#codpri created.
DataDescriptionVL41 class>>#codpri: created.
DataDescriptionVL41 class>>#datfax created.
DataDescriptionVL41 class>>#datfax: created.

*** End Generate Selectors ***
DataDescriptionVL41 class versionned in NR_2.5_V08_00
DataDescriptionVL41 class released in VpcsDatas

Elapsed time: 00:09.8

===== End generation of Class DataDescriptionVL41 =====

===== Start generation of Class UserDataDescriptionVL41 =====

```

In the step of classes generation, for each generated file, the report presents:

- The access path to the generated file:

----- Generation of GVC file D:\usr\GVC\Tests\GVC127.txt -----

- The generation starting date and time

Date of generation is: 10/6/99

Start generation at: 07:53:43 AM

- the generation description of each class:

- the name of the methods that have been created, replaced, deleted, reloaded or that have not been changed:

DataDescriptionVL41 class>>#IS\_codpri created.

- the version code, and possibly the name of the application in which it has been released:

DataDescriptionVL41 class versionned in NR\_2.5\_V08\_00\_00a

DataDescriptionVL41 class released in VpcsDatas



- the generation time for each class

*Elapsed time: 00:11.5*

- the generation completion message for each class:

*==== End generation of Class DataDescriptionVL41 =====*



The coding and contents of the generated elements are described in subchapter 3.3, *Generation Results*

## 3.3 Generation Results

### 3.3.1 Introduction

The generated elements always depend on the type of service carried out by the Business Component. They will not be the same depending on whether the Business Component updates or just reads.

### 3.3.2 Generated Classes

#### 3.3.2.1 Generated Classes Codification

In VisualAge, the elements generated by Pacbench C/S correspond to classes whose coding depends on the parameters set in the Specifications Database. In this chapter, these parameters are represented by the following elements:

- **Prefix** Folder View code (8 char. max.) or Business Component code in a single-view development (if the option **PREFIX** is not entered).
- **ClassCode** Code of the Elementary Proxies' class specified by parameter **CLASSCODE** (20 char. max.) on the Folder or Business Component General Documentation screen in a single-view development.
- **LogicalViewCode** Logical View code (4 char.)
- **UserBufferCode** User buffer code (4 char.)
- **Suffix** Suffix of the Logical View or user buffer specified
  - by parameter **SUFF** (20 char. max.) on the Logical View General Documentation screen or in the user buffer Segment for the following classes: **DataDescription**, **SelectionCriteria**, **UserContext**, **UpdatedDataDescription**, **UserDataDescription**, **ValuesOf** et **ValuesOf(... )Converter**.
  - by parameter **PROXYSUF** (20 car. max.) on the Business Component General Documentation screen (default value: **ProxyLv**).
- **DataElementCode** Data Element code

### 3.3.2.2 Contents of Generated Classes

The parameters between [...] are variable.

- Class **[Prefix][ClassCode][ProxyLv]**

This class represents a Root, Depending or Reference Elementary Proxy.



In the case of a single-view development, it is necessarily a Root Proxy.

- Class **DataDescription[LogicalViewCode][Suffix]**

This class represents the description of a Logical View instance. It holds a set of attributes that correspond to the Logical View Data Elements.

- Class **SelectionCriteria[LogicalViewCode][Suffix]**

This class represents the description of the selection criteria. It holds a set of attributes that correspond to identifier Data Elements and extraction parameters associated with the Logical View.

- Class **UserContext[UserBufferCode][Suffix]**

This class represents the description of contextual information. It holds a set of attributes that correspond to Data Elements in the user buffer.

- Class **UpdatedDataDescription[LogicalViewCode][Suffix]**

This class inherits from the **DataDescription[LogicalViewCode][Suffix]** class. Compared to its parent class, it holds 2 additional attributes whose values vary according to the modifications in progress. These two attributes are:

- **action**: update action which can be **Read**, **Modified**, **Created** or **Deleted**. This attribute is only visible in the list of the **UpdatedFolders** attribute.
- **updatedInstancesCount**: number of useful server updates associated with the instance of the concerned Folder and which can have a value of 1 to n.

- Class **UserDataDescription[LogicalViewCode][Suffix]**

This class inherits from the **DataDescription[LogicalViewCode][Suffix]** class. It contains one additional attribute that corresponds to the key Data Element of the parent instances.

- Class **ValuesOf[DataElementCode][Suffix]**

This class is generated for each Data Element that contains value tables. It inherits from the **VpcsValues** generic class.

- Class **ValuesOf[DataElementCode][Suffix]Converter**

Associated with the previous class, this class is used to provide input help by enabling the conversion and formatting of values which the user enters. It inherits from the **VpcsValuesConverter** generic class.

### 3.3.2.3 Customizing Classes

All generated classes inherit from generic classes.

Generic classes are loaded upon the installation of the product.

If you want to implement new functions in the generated Proxy Components, you must follow this inheritance system to be able to create new classes allocated to these functions. Do not modify parent and generated classes.

- To modify the behavior of a Proxy:

We strongly advise you against directly modifying a Proxy. You must create a new class inheriting from the generated **ProxyLv** class and redefine the methods of this class.

- To modify the behavior of several Proxies and define a new common behavior for them:

You must create a new class per component to be modified. This class must inherit from the corresponding **ProxyLv** generic class:

- **VpcsFolder**, for a Root Proxy
- **VpcsDependentProxyLv**, for a Depending Proxy
- **VpcsReferenceProxyLv**, for a Reference Proxy

You now simply reload the methods of each newly created parent class.



For more details on the inheritance of **ProxyLv** generated classes, refer to the *Graphic Client Reference Manual: Interface Public Interface Generated Components*.



## 4 Generation for COM Targets

A COM Proxy is a COM object, which can relay services implemented on a remote server. This object can be called by most tools which build graphic clients and support COM objects.



Before generating Proxy objects, you first have to extract the associated Business Components from the Repository. For more information, refer to the *User's Guide, Vol. II – Business Logic*.



This manual does not explain the generation of Logical View Proxy objects. If you want details on this generation mode, refer to the previous *Pacbench C/S User's Guide: Graphic Clients 2.5*.

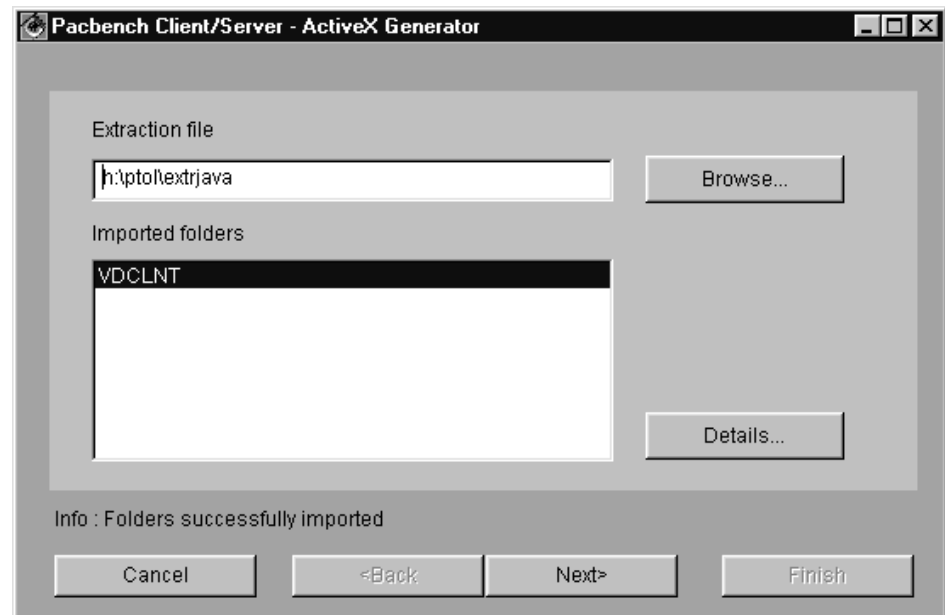
### 4.1 Starting the Generator in Graphical Mode

The ActiveX generator runs under Windows/NT, Windows 95 and Windows 98. It is used to generate COM Proxy objects.

To start the generator, execute the **olegen.exe** program from the specific program group created during the installation.

On the **Command Line Argument**, enter the chosen language option as follows: **-lang en** ou **-lang fr**. If the field is left blank, the generator interface will be in French.

The main window of the generator opens:



- **Selecting the Folder Views to be generated**

You need to enter the access path to the extraction file in the **Extraction file** field. To do this, you can:

- Enter the access path in the edit box, or
- Click on the **browse...** button. A window is displayed in which you can choose the complete file path by using the different list boxes.

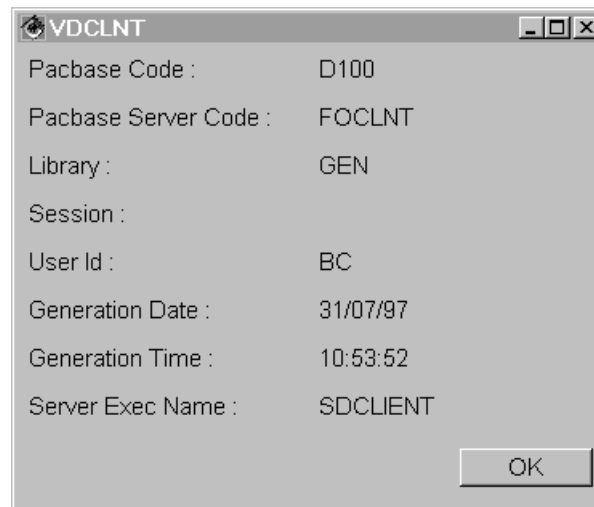
Once you have entered the file name, the list of Folder View(s) contained in the extraction file is displayed in the **Imported Folders** field.

If there are several Folder Views in the list, they are all automatically selected. In this case, you can deselect those that you don't want to generate.

☞ The **Folders successfully imported** message means that the extraction file is correct and that the Folder Views it contains have been identified and are ready for generation.

- **Other functionalities available in the detail**

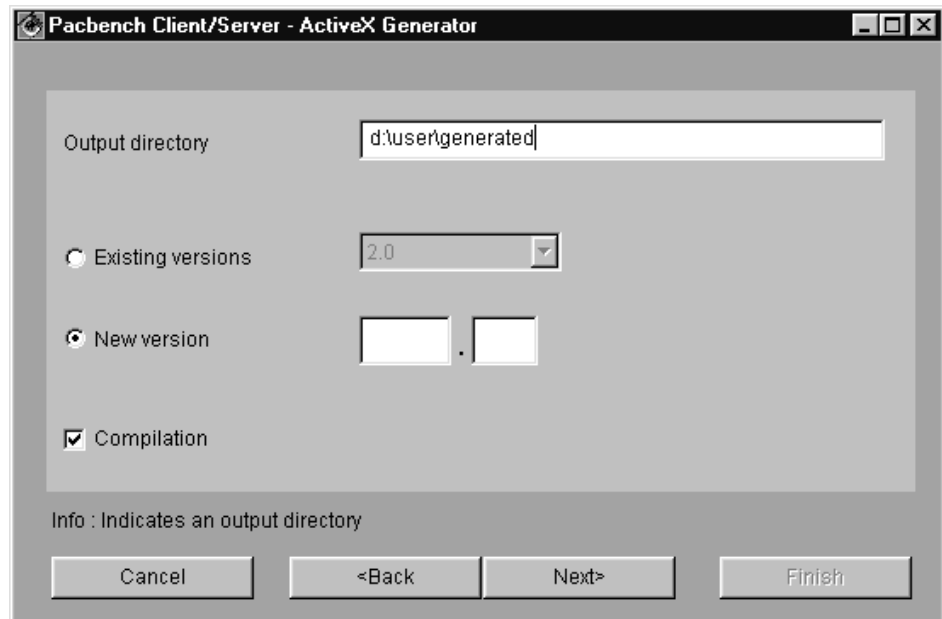
- The **Details...** button is used to display the **GVC** generation context of the selected Folder View.



- Clicking the **Next** button displays a detail in which you will specify generation parameters. For more details on the default parameters, refer to the following paragraph.
- If you want to generate immediately all selected the Folder Views, by using the default parameters, click the **Finish** button. A message box is displayed, you are asked to confirm the creation of an output file if it does not exist yet. Click OK. Generation begins.

- **Generation options**

If you have clicked on **Next** in the main detail, the following detail is displayed:



This detail enables you to specify generation parameters:

- In the **Output directory** field, enter the path of the output directory where the generated Proxy object will be stored.
- In the **existing versions** field, you can select the version to be generated, among all those which already exist.
- In the **New version** field, you can enter a new generation version of the classes. The first input field comprises 5 digits maximum and the second one comprises 3 digits maximum.
- The **Compilation** check box enables you to start the compilation of the C++ sources right after their generation.

This detail also contains the following push buttons:

- The **Back** button displays the main detail.
- The **Next** button displays the detail in which the generated classes prefixes can be modified.
- If you want to generate immediately all the selected Folder Views, by using the default classes name, click the **Finish** button. A message box is displayed, you are asked to confirm the creation of the output directory if it has not been created yet. Click **OK**; generation begins.

## 4.2 Starting the Generator in Batch Mode

To start the generator in batch mode, enter the command:

```
olegenbatch -i<extractfile> -o<outputdir.> [-v<version>] [-c]
```

Parameter **-i** contains the entire path of the extraction file.

Parameter **-o** contains the output directory of the generated classes.

Parameter **-v** (optional) contains the version number (divided into two parts, separated by a period. The first part comprises 5 digits maximum and the second one comprises 3 digits maximum). By default, the host version, if any, specified in VisualAge Pacbase is taken into account. Otherwise the last generated version, if any, is taken into account. If none of them exists, the default value is **1.0**.

Parameter **-c** (optional) indicates a request for compilation.

## 4.3 Generation Results

The generated elements always depend on the type of service carried out by the Business Component. They will not be the same depending on whether the Business Component updates or just reads.

The generated file only contains the classes which depend on the characteristics of the processed Logical View.

### 4.3.1 Introduction

Once generation is over, the following files are created:

- The **VAPLOCAT.INI** location file is created in the generation output directory.



This file must be completed manually. For more information on how to proceed, refer to section **7.3.3**.

This file must be entered at the gateway start.

- The source files of generated classes and resources required for editing them in the directory **<Foldername><Version>** (in our example: **VDCLNT2.0**).

### 4.3.2 Generated Classes

In VisualAge, the elements generated by Pacbench C/S correspond to classes whose coding consists of a prefix and a hard-coded part assigned by the generator. The prefix corresponds to the Logical View's name, whose maximum length is 36 characters.

- **[Prefix]Data** Class  
This class represents the description of a Logical View instance. It contains a set of properties corresponding to the Logical View Data Elements.
- **[Prefix]SelectionCriteria** class  
This class represents the description of the selection criteria. It contains a set of attributes corresponding to identifier data elements and extraction parameters associated with the Logical View.
- **[Prefix]Buffer** Class  
This class represents the description of the contextual information. It contains a set of attributes corresponding to the user buffer Data Elements.
- **[Prefix]DataUpdate** class



This class inherits from the `[Prefix]Data` class. Compared to its parent class, it holds 2 additional attributes whose values vary according to the modifications in progress. These two attributes are the following ones:

- ♦ **action** : update action which can be **Read**, **Modified**, **Created** or **Deleted**. This attribute is visible only in the list of the **UpdatedFolders** attribute.
  - ♦ **updatedInstancesCount** : number of useful server updates associated with the Folder instance concerned and whose value can be 1 to n.
- **[Prefix]UserData** class  
This class inherits from the `[Prefix]Data` class. It contains an additional attribute that corresponds to the key Data Element of the parent instance.
  - **[Prefix]VapError** class  
This class groups information about the problems which may occur during the execution of the program: error key, error message and gravity.



For more information, refer to the Reference Manual *Graphic Client: Public Interface of Generated Components*.



A brief description of the public interface is available in on-line mode. You can view it if your client workstation enables it.

## 4.4 Compilation Results

If compilation ended normally, the compiled elements are located in the `<Foldername><version>\Release` directory (in our example `VDCLNT2.0\Release`).

A report is available in the `<Foldername><version>\Resume.txt` file.



You can also start compilation from Visual C++. You will find information on how to proceed in subchapter 4.5, *Compiling with Visual C++ Version 5.0 and 6.0*.

Once generation and compilation are completed, the Proxy can be immediately used on the workstation where the generator is installed.

## 4.5 Compiling with Visual C++ Version 5.0 and 6.0

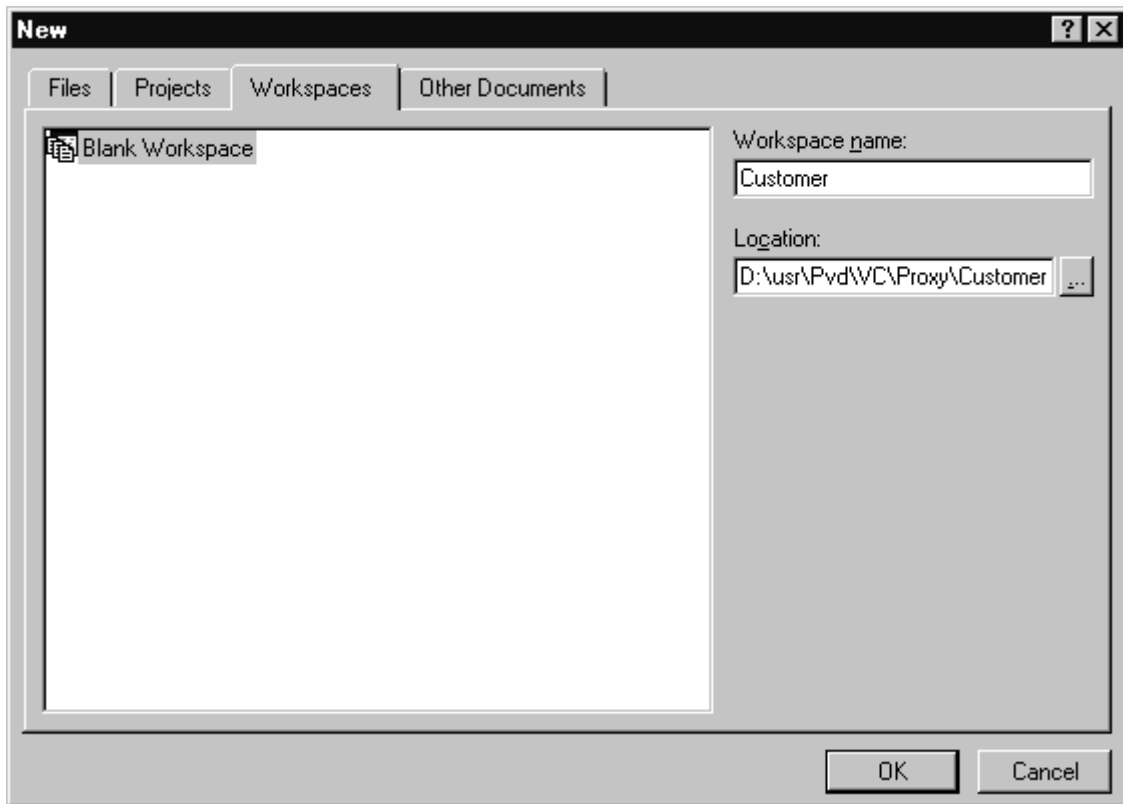
At the end of the generation, you get pieces of code to be used to compile your Proxy.

From Visual C++, you must first create a new Workspace that contains two projects, the first one corresponding to the static library of the Proxy and the second one, to the Proxy itself. To do so, follow the steps below.

### 4.5.1 Creating a New Workspace

- Click the **File** menu in the toolbar and select the **New** sub-menu.

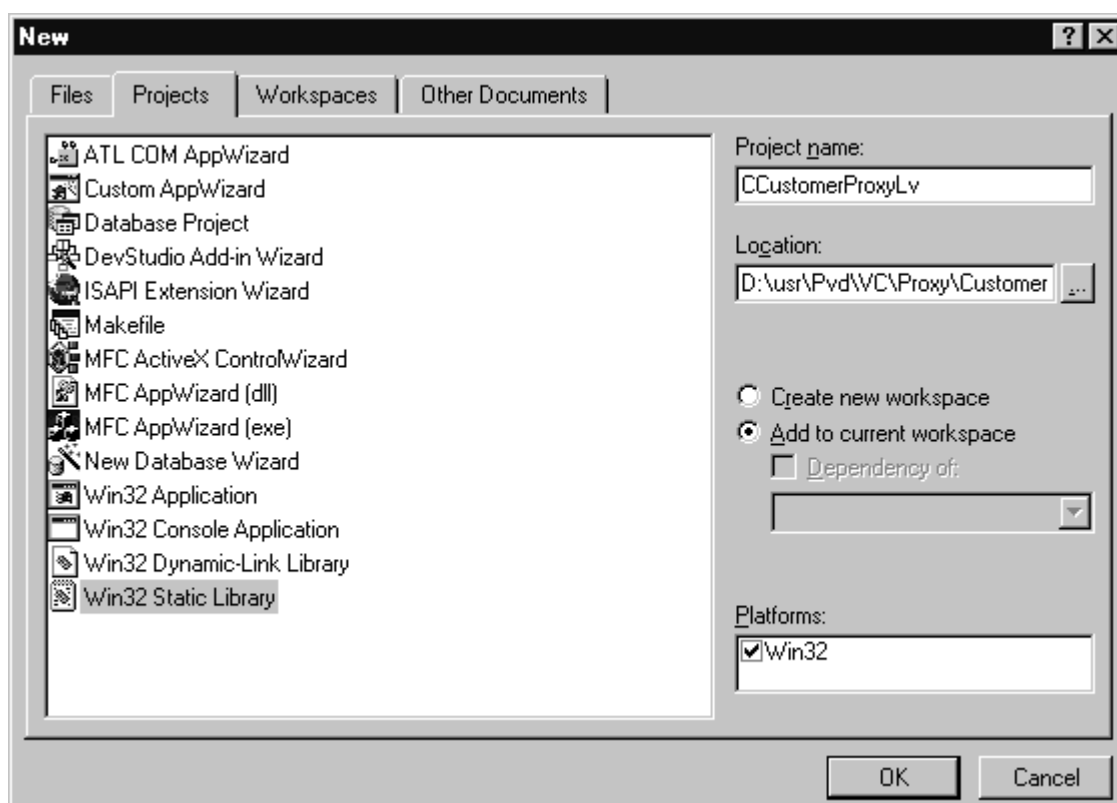
- Click on the **Workspace** tab in the **New** dialog box, then select **Blank Workspace** to create a blank project. In the **Workspace Name** field, enter the name you want for your project and in the **Location** field, the name of the directory in which you want to create it, then click **OK**.



Once this step is completed, a sub-directory is created under the directory you selected in the **Location** field. The name of this sub-directory corresponds to the name of the project you entered in the **Workspace name** field.

#### 4.5.2 Creating the Static Library Project

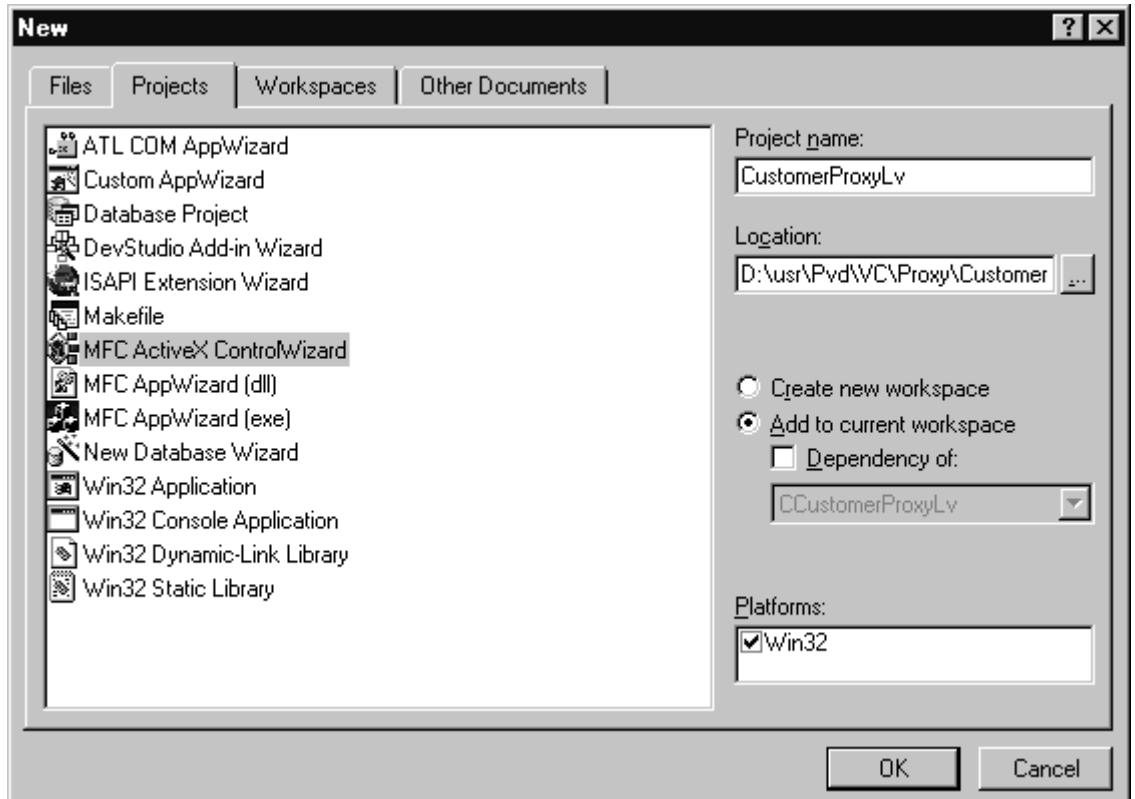
- Click the **File** menu in the toolbar and select the **New** sub-menu.
- Click on the **Projects** tab in the **New** dialog box, then select **Win32 Static Library** in order to add into the workspace previously created a new project that will correspond to the static library used by the Proxy.
- In the **Project Name** field, enter the name of the project according to the following syntax **C<rootNodeName>ProxyLv**.
- In the **Location** field, select the sub-directory that has been automatically created during the creation of the blank workspace.
- Select the **Add to current workspace** radio-button, then click **OK**.



- ☞ In Visual C++ 6.0, a window opens up, proposing options. Do not select anything, just click **Finish** then **OK**.

### 4.5.3 Creating COM Proxy Project

- Click on the **File** menu in the toolbar and select the **New** sub-menu.
- Click on the **Projects** tab in the **New** dialog box, select **MFC ActiveX Control Wizard** in order to add into the workspace previously created a new project that will correspond to the Proxy.
- In the **Project Name** field, enter the name of the project according to the following syntax **<rootNodeName>ProxyLv**.
- In the **Location** field, select the sub-directory that has been automatically created during the creation of the blank workspace.
- Check the **Add to current workspace** radio-button, then select **OK**.
- A new window opens up. Click **Finish** then **OK**.



#### 4.5.4 Transferring files

Once the three previous steps are completed, you get the following directories. To make the next steps clearer, each directory is assigned a name as follows:

- *Folder1* : ...\<<workspaceName>
- *Folder2* : ...\<<workspaceName>\C<rootNodeName>ProxyLv
- *Folder3* : ...\<<workspaceName>\<rootNodeName>ProxyLv

To fill in respectively the two created projects, you must copy some of the files that are stored in the Proxy generation directory using a File Manager.

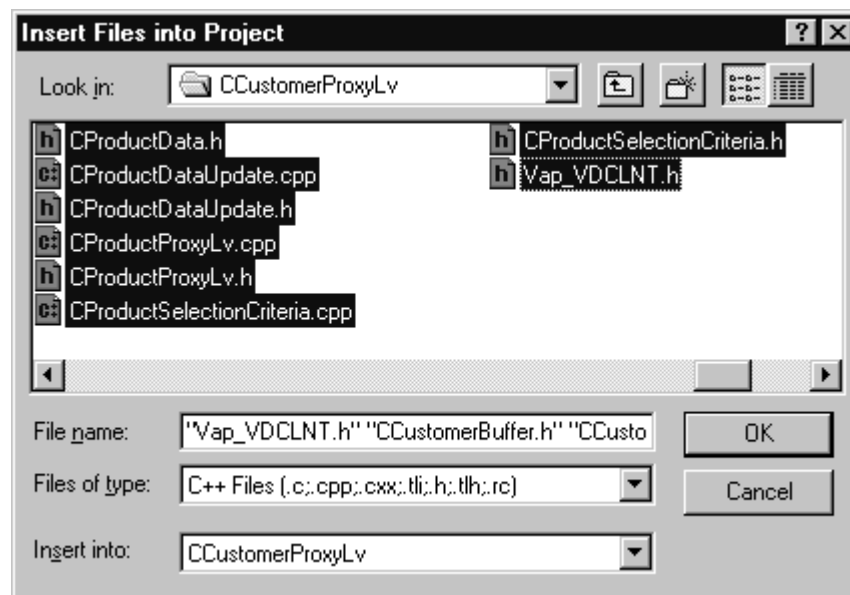
- Select all the files prefixed with **C<nodeName>...** as well as the file **VAP\_<folderName>.h**
- Copy the selected files into the *Folder2* directory.
- Copy all the remaining files in the *Folder3* directory.

#### 4.5.5 Adding the Files to the Projects

You must now add the files you copied into the static library project.

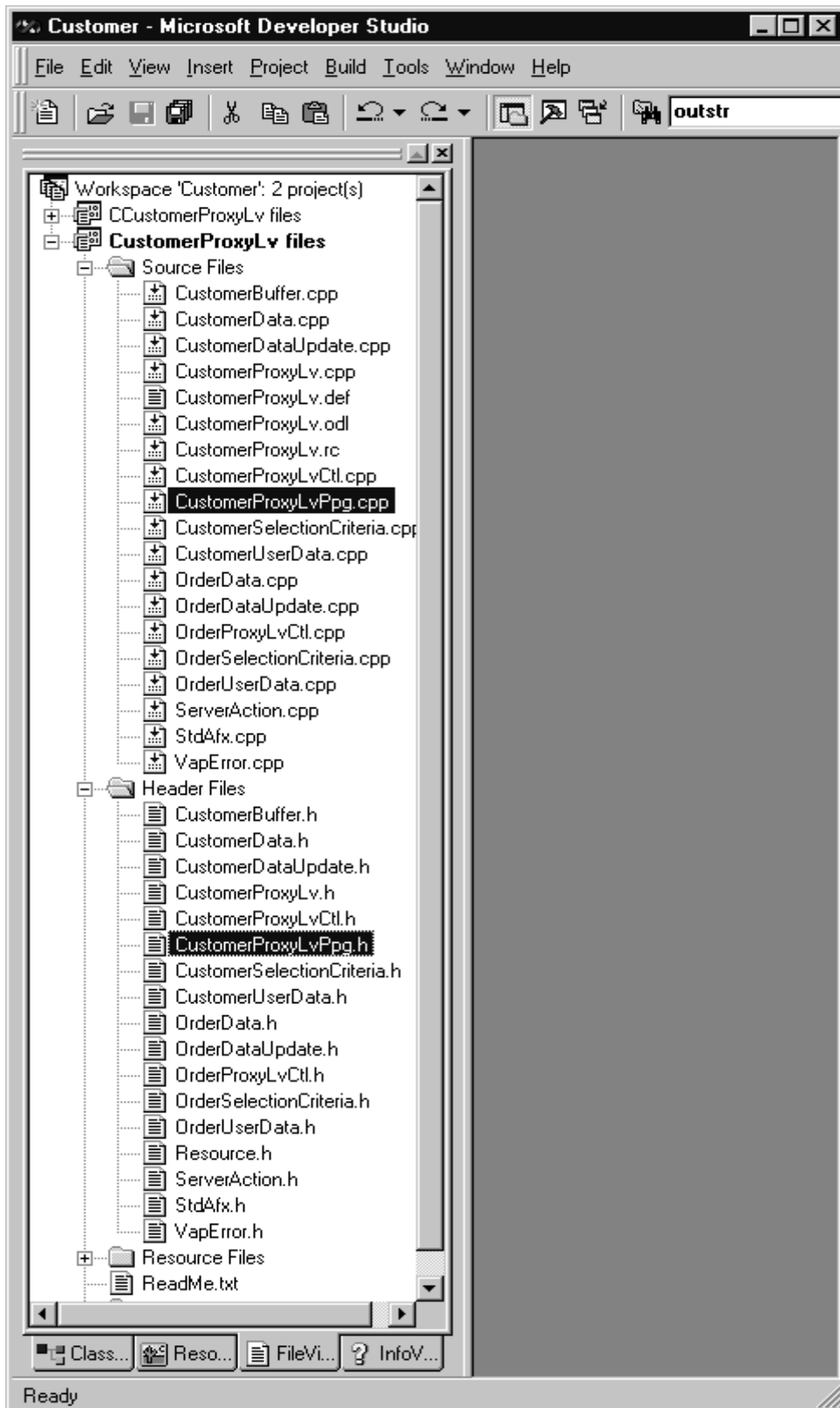
- Click on the **Project** menu in the toolbar, then select the **Set Active Project** sub-menu.
- Select the static library project **C<rootNodeName>ProxyLv**
- Click on the **Project** menu in the toolbar, then select the **Add To Project** then **Files...**

- Select all the files from the *Folder2* directory, then click **OK**.



You must now carry out the same steps for the Proxy project.

- Click on the **Project** menu in the toolbar, then select the **Set Active Project** sub-menu.
- Select the static library project `<rootNodeName>ProxyLv`
- Click on the **Project** menu in the toolbar, then select the **Add To Project** sub-menu, then **Files...**
- Select all the files in the project, then click **OK**.
- Then, delete the two files ending with `<rootNodeName>Ppg.h` and `<rootNodeName>Ppg.cpp` from the project.



#### 4.5.6 Specifying Paths to Compile the Proxy

You must now specify the access paths to the include files, libraries, and dlls.

- Click on the **Tools** menu in the toolbar, then select the **Options** sub-menu.

- In the notebook, click the **Directories** tab.
- In the **Show directories for** drop-down list, select **Include files**, then go down to the bottom of the **Directories** list.
- Enter the complete access path to the files to be included (default: **d:\vapb\ActiveX\Include**)
- Enter the access path to the static library project: **...<workspaceName>\C<rootNodeName>ProxyLv**
- In the **Show directories for** drop-down list, select **Library files**, then go down to the bottom of the **Directories** list.
- Enter the complete access path to the link files (default: **d:\vapb\ActiveX\Lib**).
- In the **Show directories for** drop-down list, select **Executable files**, then go down to the bottom of the **Directories** list.
- Enter the access paths to the required dlls (default **d:\vapb\ActiveX\Bin**).

#### 4.5.7 Compiling the Static Library

The resulting project can now be compiled to get the output library required for the COM Proxy.

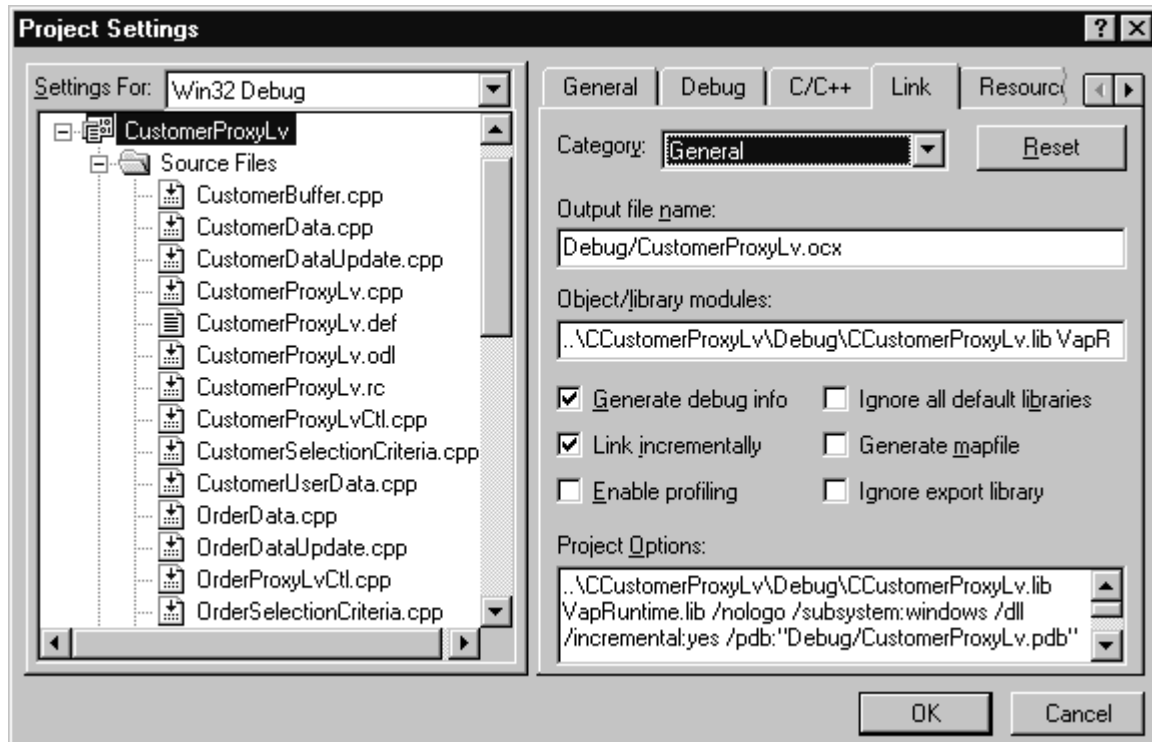
- Click on the **Project** menu in the toolbar and select the **Set Active Project** sub-menu.
- Select the Proxy project **C<rootNodeName>ProxyLv**
- Click on the **Build** menu in the toolbar, then select **Build C<rootNodeName.lib>** from the drop-down menu, or use the function key F7. By default, the WIN32 Debug mode is used for compiling and the output directory is under **\<WorkspaceName>\C<rootNodeName>ProxyLv\Debug**.

#### 4.5.8 Setting Parameters for Link-Edit

You must now specify the links between the object and the **.lib** files of the libraries in use.

- Click on the **Project** menu in the toolbar and select the **Set Active Project** sub-menu.
- Select the Proxy project **<nrootNodeName>ProxyLv**.
- Click on the **Project** menu in the toolbar and select the **Settings** sub-menu.
- In the notebook, click on the **Link** tab.

- In the **Object/library modules** field, enter the name of the libraries to link with the Proxy as well as the library previously obtained by compiling the first project, i.e.:  
`..\CCustomerProxyLv\Debug\CCustomerProxyLv.lib VapRuntime.lib`
- Click **OK** to validate.



#### 4.5.9 Compilation de la Proxy

Click on the **Build** menu in the toolbar, then on **Build <rootNodeName.ocx>** from the drop-down menu, or use the function key F7. The output directory is under `\<WorkspaceName>\<rootNodeName>ProxyLv\Debug`.



## 5 Developing a Client with VisualAge for Java

Once you have generated and imported the Proxy objects in the VisualAge workstation, you just need to integrate them into the graphic application.

After introducing you to the basic concepts of development, this chapter gives you a detailed description of a client development, including the following steps: insertion of Proxy objects with programming links involving methods, properties and events, error management, communication management and test on the application.



To facilitate the development and reusability of clients implemented with VisualAge, we advise you to use a project for each functional application. The project must contain one or more graphic class packages and a generated class package.

### 5.1 General Principles

- If your Proxy contains only one Elementary Proxy (a Root Proxy), the properties, methods and events associated with large reading, reference or depending nodes are not available.

#### 5.1.1 Visual Representation of Proxy Objects in the Composition Editor

Once it has been imported in the VisualAge workstation, a Folder View Proxy can be handled through a graphic bean.

You will find below a presentation of the icons relating to the Proxy objects provided at the installation.

| Icons | Types of Elementary Proxy objects |
|-------|-----------------------------------|
|       | Root Proxy                        |
|       | Depending Proxy 0,N               |
|       | Depending Proxy 0,1               |
|       | Depending Proxy 1,N               |
|       | Depending Proxy 1,1               |
|       | Reference Proxy 0,1               |
|       | Reference Proxy 1,1               |

## 5.1.2 Use of Properties

A property corresponds to a piece of information handled by a Proxy object. This piece of information defines an elementary data, a list of elementary data or a list of composite data instances. A property corresponds either to a constant, a parameter or an action result. According to the context, it is initialized by the GUI application or the Proxy.

Two kinds of properties are found:

- those standing for technological variables. They enable to adjust the behavior of the Proxy objects in VisualAge
- Those corresponding to the Logical View data



The availability of a property depends on the Proxy type. All the public interface properties are documented in the *Graphic Clients Reference Manual: Public Interface of Generated Components*.

### 5.1.2.1 Local Checks

The Elementary Proxy automatically calls the local checks during the creation and modification of an instance via the `createInstance` and `modifyInstance` methods. Each Data Element belonging to the Logical View is checked.

The following checks are made:

- Checks on value lists defined in the Data Element description
- Checks on value ranges defined in the Data Element description
- Checks on compulsory presence defined when calling Data Elements in a Logical View. The presence of identifier-type Data Elements or foreign key-type Data Elements is automatically checked for a reference relation with a minimum cardinal value of 1.

If local checks detect an error, an error message is set via the Error Manager.

These checks can be selectively triggered for each root or depending-type node in the Folder concerned. The property used to activate or deactivate the check of Data Elements on the server is `ServerCheckOption`.

Whether the message is sent or received, the detection of an empty Data Element is automatic.

However, the Elementary Proxy does not perform numeric or date checks; these are performed by the graphic controls.

### 5.1.2.2 Check of the Length of the `detail` Property Fields

For each field of the `detail` property, the checks performed upon the local creation or modification of an instance ensure that the length of the value contained in the property does not exceed the maximum length of the value for this property.

The length is checked systematically (except if the property does not belong to the current sub-schema), even if the property has been defined as 'not to be checked in the client'.

A local error is sent if the length is excessive.

- ☞ The length of the fields included in user buffers is not checked. If it is excessive, the length is truncated to its maximum value.

### 5.1.2.3 Selecting the Local or Server Sort Criterion on a List of Instances

The `localSort` property enables you to specify whether the Proxy sorts the instances of the `rows` property according to the local sort (`true`) or keeps the instances in the order they were sent by the server (`false`).

You can change the sort type at any moment.

- ☞ This attribute is not effective in user services.

#### 5.1.2.3.1 Local Sort

In standard, the instances of the `rows` property are sorted according to the local criterion if the parameter has not been changed after the generation. In this context, two sort types exist:

- If no sort criterion has been locally defined (see paragraph 5.1.2.4), the Proxy implicitly sorts the instances in the increasing order of the identifiers defined on the Logical View.
- If a local criterion has been locally defined, the Proxy sorts the instances in the order defined by this criterion.

In all cases, when an instance is created locally, it is inserted according to the current sort criterion applied in the `rows` property.

If you change dynamically or cancel the local sort criterion, the instances contained in the `rows` property are immediately sorted according to the new criterion.

#### 5.1.2.3.2 Server Sort

The instances of the `rows` property are sorted according to the server criterion if the `localSort` property is set to `false`. In this context, the instances contained in the `rows` property are displayed in the order sent by the server, without taking the local sort criterion into account.

If collections are managed manually or in the case of a paging in extend mode, the instances sent by the server are added at the end of the existing collection in the `rows` property.

All the locally-created instances are added at the end of the existing collection in the `rows` property. In this context, an instance which is not positioned at the end of a collection but which is deleted and created again locally is transferred to the end of the collection contained in the `rows` property.

### 5.1.2.4 Specification of the Local Sort Criterion

You can dynamically change the sort criterion used to present instances in the `rows` property. To do this, use the `dataComparator` property of each Proxy: `void setDataComparator(Comparator c)`.

The required parameter is an instance of the class implementing the following interface `com.ibm.vap.generic.Comparator`.

This interface consists of a method representing the following relation:

```
int compare(Object a, Object b).
```

This method must return:

- a negative number if  $a < b$
- 0 if  $a = b$
- and a positive number if  $a > b$ .

For example:

```
import com.ibm.vap.generic.Comparator ;
public final class CustomerComparator implements Comparator {
public static final int NAME = 0 ;
public static final int COMPANY = 1 ;
public int criteria ;
public int compare(Object a, Object b) {
try {
switch(criteria) {
case NAME:
return ((CustomerData)a).getName().compareTo(
(CustomerData)a).getName());
break;
case COMPANY:
return ((CustomerData)a).getComp().compareTo(
(CustomerData)a).getComp());
break;
}
} catch (IllegalCastException ice) {
return 0;
}
}
}
```

#### 5.1.2.5 Table Model

This property is available in read/write mode on all types of nodes, provided you chose the generation option **Use Swing** (available from VisualAge Java version 2.0).

This property enables you to insert a JTable in your application (a JTable is a swing component made up of rows and columns).

By default this property is initialized with a new instance of a generated TableModel.

#### 5.1.2.6 Sub-schema Management

The sub-schemas specified in the Logical View's description can be taken into account when selection, read or update methods are executed, provided Business Components manage the presence of Data Elements (**VECTPRES=YES** or **CHECKSER=YES** options).

Each node has two properties:

- **subSchema**, via which you can assign the desired sub-schema when a selection, read or update method is executed by the Business Component of the node. The value of this property can be assigned via the **subSchemaList** property.

- **subSchemaList**, via which all the sub-schemas available on a node are listed. Since a sub-schema cannot be given a name in the VisualAge Pacbase Referential, each sub-schema is designated by **SubSchema<n>** (with **n** = 01 to 10).

### 5.1.3 Use of Methods

#### 5.1.3.1 Implementation

A method corresponds to a process which can be executed by a Proxy objet. It is triggered using a connection between an event in the GUI application and the method code of a Proxy.



The availability of a method depends on the Proxy type. All the methods of the public interface are documented in the *Graphic Clients: Public Interface of Generated Components Reference Manual*.

#### 5.1.3.2 The Different Types of Server Methods

The server methods execute procedures implemented in one or more Business Components associated with the Folder. These methods send a query to the Business Components which send back a result on the workstation. The queries and responses generally contain technical parameters, Logical View instances associated with one or more nodes and contextual data defined in a user buffer.

You must distinguish two types of server methods

- Those which systematically access the server:
  - ♦ **selectInstances**
  - ♦ **readInstance**
  - ♦ **readInstanceAndLock**
  - ♦ **readInstanceWithFirstChildren**
  - ♦ **readInstanceWithAllChildren**
  - ♦ **readInstanceWithFirstChildrenAndLock**
  - ♦ **readInstanceWithAllChildrenAndLock**
  - ♦ **readAllChildrenFromDetail.**
  - ♦ **readAllChildrenFrom.**
- Those which do not systematically access the server:
  - ♦ **readNextPage**: The server is accessed except if the **noPageAfter** event was sent back during the previous selection
  - ♦ **readPreviousPage**: The server is accessed except if the **noPageBefore** event was sent back during the previous selection.
  - ♦ **readFirstChildrenFromDetail** : the server is accessed, except if the **maximumNumberOfRequestedInstances** properties of the Depending Proxy objects are set to 0 and if their **globalSelection** properties are set to false.
  - ♦ **checkExistenceOfDependentInstances**: The server is accessed except if the existence of depending instances can be checked locally.
  - ♦ **updateFolder**: The server is only accessed if there is at least one instance of the node concerned, modified in its **updatedFolders** property.

### 5.1.3.3 Managing Folder Reading

Large reading of a Folder root enables to read from a client component all the instances of the Folder's root node existing in the database. The methods concerned are `selectInstances` and `readNextPage`.

#### 5.1.3.3.1 Provisional Large Reading of Depending Nodes

In the context of client/server architectures, a GUI application handling a Folder strives to anticipate data reading to minimize exchanges with the servers.

In a hierarchical network, various provisional reading methods are possible:

- The '`allChildren`' type method reads *all* the depending instances of the instance selected in the `detail` property of the parent Elementary Proxy.
- The '`firstChildren`' type method only reads the instances which are *immediately* dependent of the selected instance in the `detail` property of the parent Elementary Proxy.

The first provisional large reading method is available on the Root Proxy only.

The second provisional large reading method is available on the Root Proxy or on the Depending Proxy which themselves hold Depending Proxy objects.

#### 5.1.3.3.2 Transferring an Instance Between the Rows and Detail Properties

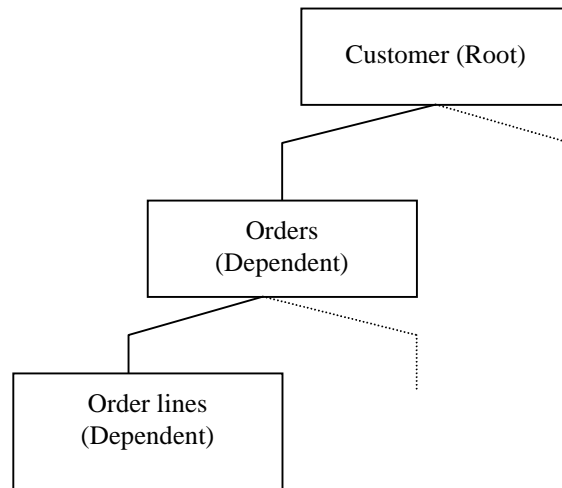
The transfer of an instance between the `rows` and the `detail` properties enables the `detail` property to be loaded with an instance initially retrieved by a large reading method.

This transfer is only available for Root and Depending Proxy. It corresponds to a local reading method which also loads all the local instances of Depending Proxy known by the Folder View Proxy. The transfer is made using the `getDetailFromDataDescription` method.

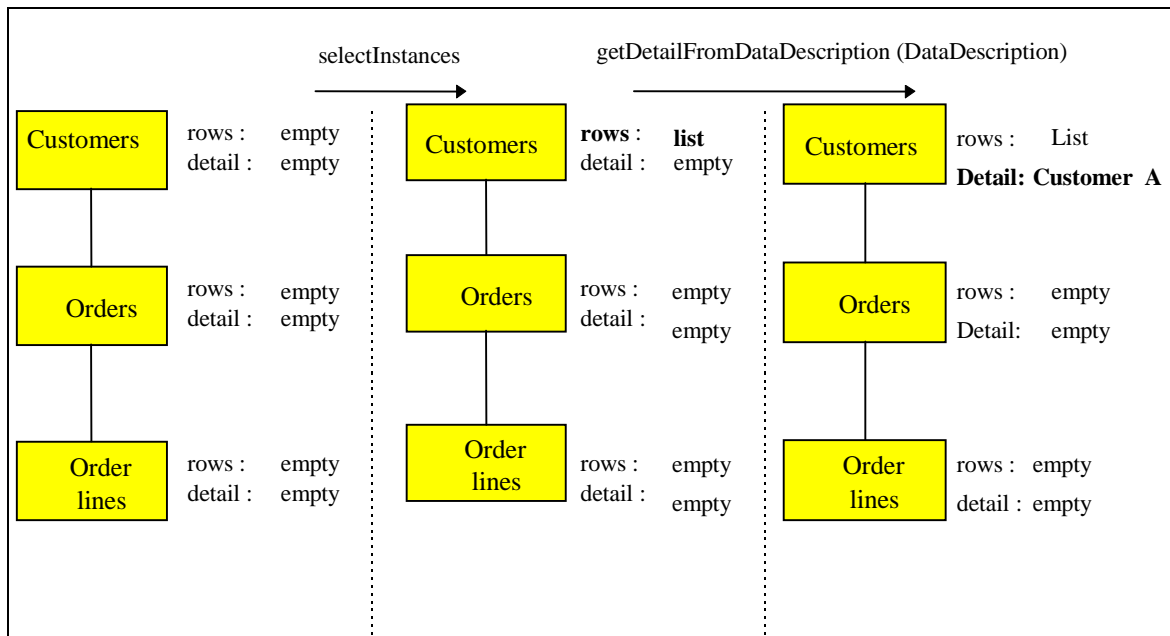
#### 5.1.3.3.3 Large Reading and Transferring an Instance Between Rows and Detail Properties: Working Mechanism

This example illustrates the loading of the `detail` property with an instance previously retrieved in the `rows` property by a large reading method on a root or depending node, and the principle of the provisional large reading of depending nodes.

It is based on a Folder View Proxy, which consists on three Elementary Proxy objects:

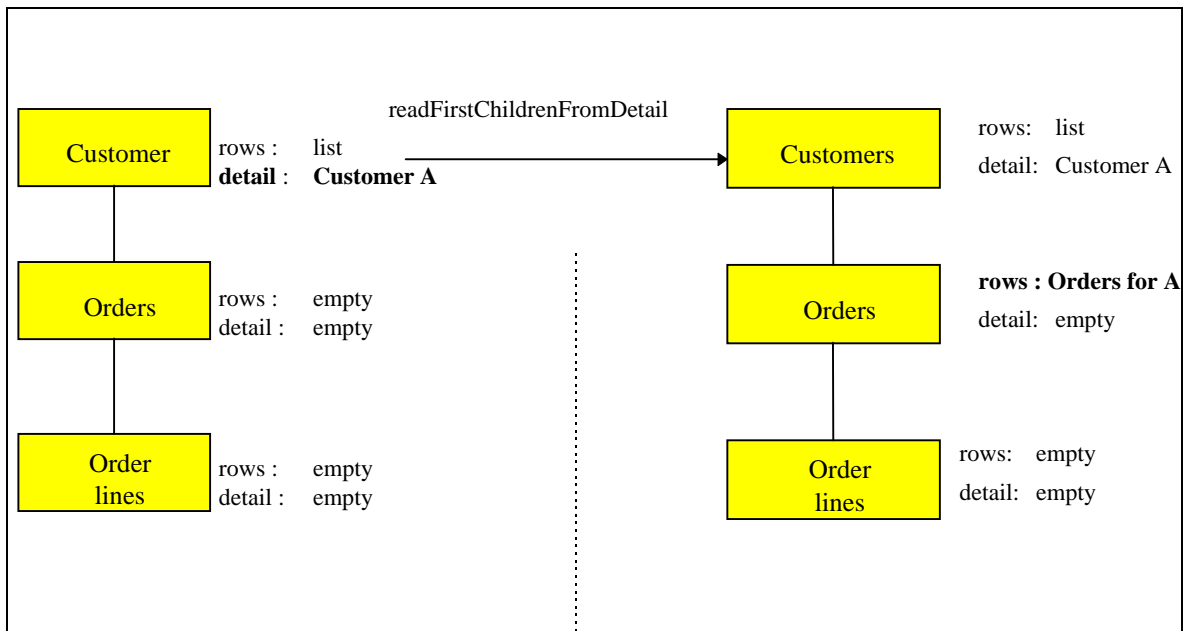


The schemas below introduce the working mechanism based on this principle.



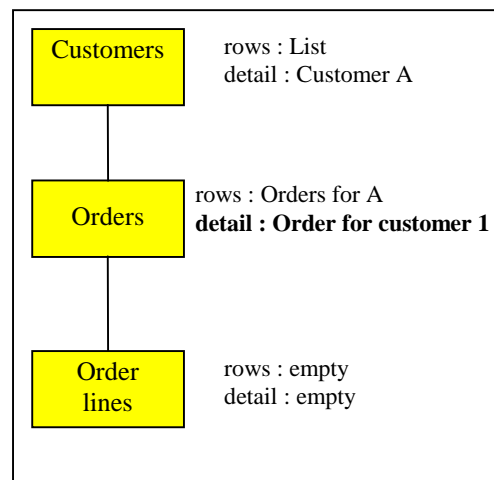
The **detail** property of the customer node now contains Customer A. There are three solutions to read Customer A's depending instances (i.e. his associated orders).

## SOLUTION 1



The `readFirstChildrenFromDetail` method on the Customers Root Proxy not only reads the `rows` property of Order Lines Depending Proxy for Customer A, but it also reads the `rows` property of other possible Elementary Proxy objects directly dependent of the Root Proxy.

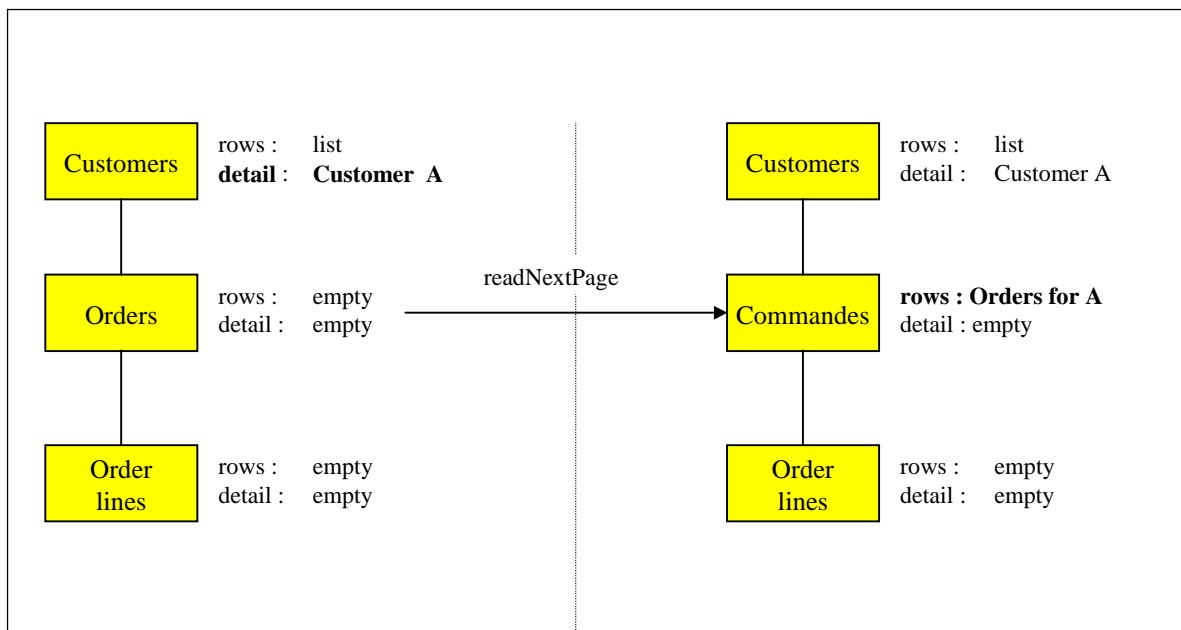
In this context, the `getDetailFromDataDescription (DataDescription)` method on the Orders Depending Proxy initiates:



Then, to read the order lines of the Order 1 of Customer A, use the `readFirstChildrenFromDetail` method on Orders or `readNextPage` method on Order Lines.



### SOLUTION 2

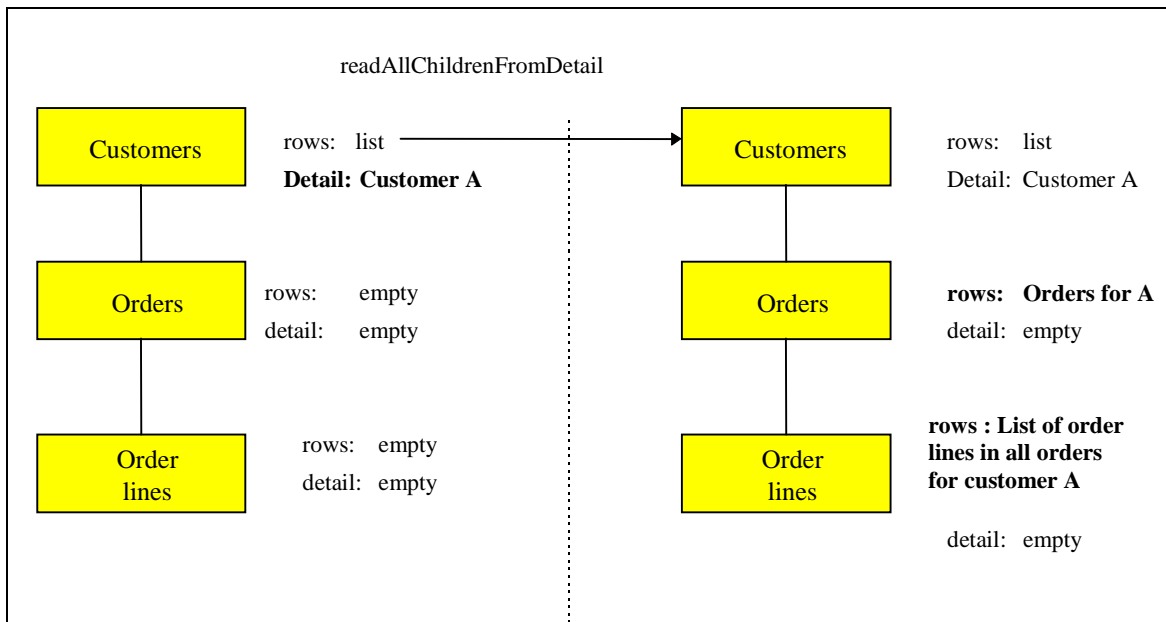


The `readNextPage` method on the Orders Depending Proxy loads its `rows` property. The instances of other possible Elementary Proxy object, which are directly dependent of the Root Proxy, are not read.

In this context, the `getDetailFromDataDescription (DataDescription)` method on the Orders Depending Proxy initiates the same result as in solution 1.

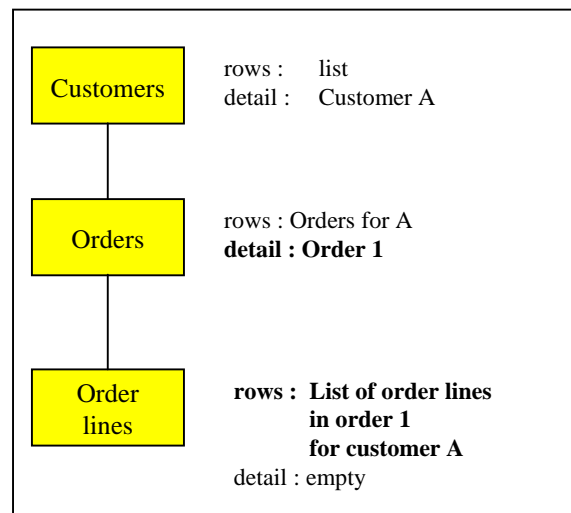
To read subsequently the order lines of the Order 1 of Customer A, proceed as in solution 1.

### SOLUTION 3



The `readAllChildrenFromDetail` method on the Customers Root Proxy reads not only all the orders for A but also all the other possible instances dependent of A whatever their hierarchical level can be. In our example, all the order lines for all the orders of A are therefore read.

In this context, the `getDetailFromDataDescription (DataDescription)` method on the Orders Depending Proxy initiates:



The `getDetailFromDataDescription (DataDescription)` method automatically loads the `rows` property of the Order lines Depending Proxy with the order lines contained in Order 1 for Customer A. These order lines have been previously transferred to the workstation using the `readAllChildrenFromDetail` method.

#### 5.1.3.3.4 Large Reading of Reference Nodes

The reading of a reference node is considered as aid on criteria. It shows the end user a list of information which can be referred to for a depending node.

The information presented to the end user is both necessary and sufficient to assist him in making a choice.

To optimize the volume of characters sent for this type of service, the Logical Views have a « aid on criteria »-type subschema used to select the concerned Data Elements at the server level.

The large reading of reference nodes is executed on request and cannot be involved in provisional large reading. The methods concerned are `selectInstances` and `readNextPage`.

#### 5.1.3.3.5 Principle of Paging in a Folder's Nodes

Two types of paging are offered on a Folder's nodes:

- The first, called *non-extend* paging, is used to paginate forwards and backwards on a predefined collection through specific methods. Each method executes a read request to the server and its result overwrites that of the previous read. This type of paging is available only on root or reference nodes.
- The second type of paging, called *extend* paging, is used to gradually retrieve the instances of a defined collection as read requests for following pages are made. In this context, the backwards paging function disappears and is performed locally by the scroll box of the graphic control which presents the list of instances. This type of paging is available to all the nodes of a Folder.

#### 5.1.3.3.6 Selection Criteria Associated with Large Reading Methods

The selection criteria associated with large reading methods are elementary or composite properties associated with each node of a Folder. They are split into two types:

- Functional selection criteria corresponding to the identifier and to elements required for defining extraction methods for the Logical View associated with the node. These criteria correspond to the following properties:
  - ♦ `selectionCriteria` which defines identifier Data Elements and parameters by value.
  - ♦ `extractMethodCode` which defines the code for the extraction method desired.
  - ♦ `extractMethodCodes` contains the list of the available extraction methods.
- Organic selection criteria corresponding to information used to control the volume of instances selected for each node.
  - ♦ `globalSelection` is a Boolean property which, when set to true, is used to retrieve all instances of the node via a selection query.
  - ♦ `maximumNumberOfRequestedInstances` is a numerical property which specifies, when the `globalSelection` property is set to false, the number of instances to be read for a node via a selection query. This property can hold the value 0. In this case, the concerned part of the tree structure is not read during a provisional large reading.

#### **5.1.3.3.7 Limitation of the Scope of Large Reading**

During a large reading using selection criteria, only the instances corresponding to these criteria are read.

However, in the case of a large reading ordered by an **allChildren** provisional reading type, the **globalSelection** property is considered to be set to true on each node.

#### **5.1.3.3.8 Reading of a Root Node or Depending Instance**

The reading of a depending or root node instance enables you to retrieve an instance of the node without previously making a large selection of a collection of instances for this node. It directly loads the node's **detail** property.

This type of reading is considered as a collection selection and therefore cancels the previous selection even if it was the result of a large reading method. The loading of the **detail** property therefore initializes the **rows** property.

The methods which implement this selection function are used to:

- Retrieve the instance without its dependencies (**readInstance**).
- Retrieve the instance with its first level dependencies(**readInstanceWithFirstChild**).
- Retrieve the instance with all its dependencies(**readInstanceWithAllChildren**).

#### **5.1.3.3.9 Reading of a Reference Node Instance**

The reading of a reference node instance cannot activate the provisional large reading process. It is used to retrieve the entire description of the instance of the node called in its **detail** property.

Only the **readInstance** method is therefore available on a reference node.

Update methods are not available for reference nodes. Their **rows** and **detail** properties are independent. The result of the **readInstance** method does not therefore initialize the **rows** property.

The **rows** property is used to display sufficient information to assign one of the reference node instances to the referencing node instance.

The **detail** property is used to view the entire description of a referenced instance.

The structure of these two properties can thus be different.

#### **5.1.3.3.10 Selection Criteria Associated with Instance Reading**

The identifier of the node instance to be read is specified in the functional selection criteria associated with the node.

For depending nodes, the identifier of the node corresponds to the identifier of the Logical View associated with the node, discarded from the Data Elements which are the identifiers of Logical Views higher in the hierarchy. These hierarchical identifiers are automatically initialized by the Folder View Proxy according to the navigation in the Folder.

### 5.1.3.4 Folder Update Management

#### 5.1.3.4.1 Local Updates

Local update services are available on each root or depending type node in the Folder.

- Create a node instance
- Modify a node instance
- Cancel a node instance

However, there are certain rules specific to Folder management:

- The creation of a depending node instance is only authorized if the hierarchy of instances contained in the `detail` properties of higher nodes exists.
- Canceling a node instance initiates the recursive canceling of local instances of depending nodes.

To allow the developer to manage messages which can warn users of the impact of a cascade cancellation, a method for verifying the existence of depending instances is available on root or depending type nodes.

This method (`checkExistenceOfDependentInstances`) sends a Boolean result which is either true or false. If no dependency is found in the Folder's local cache and the instance concerned is not created locally, this method sends a verification request to the server.

So that a user can undo local manipulations of a Folder instance, an `undoAllLocalFolderUpdates` method can be used to eliminate all local updates on all Folder nodes applied since the last server update method. This method is only available on the Folder's root node. Another method `undoLocalFolderUpdates` can be used to eliminate all updates associated with the Folder's Root node you have parameterized.

#### 5.1.3.4.2 Server Updates

Only the Root Proxy provides server update methods.

Server updates correspond to methods which enable a client component to send all local updates made since the last server update method.

These updates concern all the modified depending instances. They can concern several Folder instances.

When a server update method sends back errors, the Folder remains with the « Modified Locally » status; new collection selections can be made only by correcting the errors and sending back the updates, or by using the `undoAllLocalFolderUpdates` method or the `undoLocalFolderUpdates` method.

Before the request is sent to the server, the server update methods check the Folder integrity for locally created instances. For each locally created instance of the node, this method checks the minimum cardinal values of each link and sends an error if the number of depending instances does not respect the properties of the associated links.

A server update method can be accompanied by a request to refresh the updated instances if some of their Data Elements, such as the identifiers, are calculated by the server. This refreshment request is made using the `refreshOption` property.

#### **5.1.3.4.3 Management of Effective Transactions**

The management of effective transactions is automatically carried out by the local cache.

It consists in calculating the resulting update of various local updates made on the same instance of the Folder's node. It controls the creation of duplicate instances. If several local updates have been made on the same instance of the node, only the last one will be sent to the server.

#### **5.1.3.4.4 Changing the Collection Selection**

The `aboutToChangeSelection` event is returned to the Client by the Root Proxy when a server selection method on a Folder node overwrites its current local collection and when this collection or the depending collections contain local modifications which are candidates for a server update. If the event is not intercepted during programming, the updates will be lost.

#### **5.1.3.4.5 Re-initializing instances in the local cache**

The `resetCollection` method is used to remove all the instances from the cache of a Folder View Proxy before initializing a new collection of instances. This method can be executed by all types of nodes in a Folder View Proxy containing a `rows` property.

#### **5.1.3.4.6 Managing collections of instances**

The management of collections of instances can be carried out automatically, or manually by positioning the `manualCollectionReset` Boolean property which is available for all types of nodes in a Folder View Proxy. The manual management mode is used to create heterogeneous collections through a series of selection and paging methods.

#### **5.1.3.4.7 Load of the local cache with no server access**

The `initializeInstance` method allows to store in the local cache an instance of Logical View that has not been either read from the server or created locally. It allows the update of the Logical View instance though it has not been previously read from the server. This method is available for all types of nodes and is valid when the Logical View instance does not exist locally.

### **5.1.3.5 Asynchronous Methods**

#### **5.1.3.5.1 Principles**

The asynchronous programming is used to dissociate the method used to send a request from the method used to retrieve its response. You can use this type of programming whether you use an asynchronous communication protocol or not.

You can use the Proxy components in asynchronous mode independently of the middleware used. As for the Proxy, you can work with an asynchronous mode, by using a *location* whose middleware is synchronous, and the other way round.

In this context, the end user will be more efficient as he can send a request in advance, and retrieve the response when he needs it. This method allows you to optimize your working time.

Furthermore, the communication protocols are used to make the requests or the responses in the local messages threads more secure by allowing the message to convey independently of the network situation.

The communication mode is defined by the `Asynchronous` Boolean property at the Folder level. Thus, the following code line produces a change in the asynchronous mode:

```
myFolder.setAsynchronous(true);
```

Subsequently, the methods which enable to access a server produces a `com.ibm.vap.generic.AsynchronousRequestException` exception.

This exception holds a public method only:

```
com.ibm.vap.generic.ServerActionContext getContext();
```

The instance of `ServerActionContext` returned must be stored. It constitutes the key that will be used to process the server's response with the `getReply()` method:

```
boolean b = myFolder.getReply(context);
```

It is not always possible to process the response associated to an asynchronous method context, for two reasons:

- The response may not be returned (the `getReply()` method returns false)
- The context is no longer valid, because it is associated to an instance, which is no longer in the local cache (a local exception has been raised).

In the case that no error has come and the response has been processed, `getReply(context)` returns true.

#### **5.1.3.5.2 Global Methods or Methods Associated with an Instance**

Some server methods, labeled as global methods, are independent of any selection, whereas others depend on any instance included in the local cache. In asynchronous mode, global methods store the response identifiers in a collection and each executed request adds its identifier to this collection. The methods associated with a Logical View instance store their response identifiers in a collection associated with the concerned instance. The collections of response identifiers associated with the global methods are lost when the application using the Proxy is closed. A collection of response identifiers associated with an instance contained in the local cache is lost when this instance is locally deleted or after a change of collection.

- **global methods**

The responses associated with the following methods can be executed independently of the current collection:

- `executeUserService()`
- `readInstance()` (ROOT)
- `readInstanceWithFirstChildren()` (ROOT)
- `readInstanceWithAllChildren()` (ROOT)
- `readInstanceAndLock()` (ROOT)
- `readInstanceWithFirstChildrenAndLock()` (ROOT)
- `readInstanceWithAllChildrenAndLock()` (ROOT)
- `readNextPage()` (ROOT)
- `selectInstances()`

- `lock()`
- `unlock()`
- `readPreviousPage()`

- **Methods associated with an instance**

The following methods depend either on the `detail` instance, or on the instance passed as a parameter, or on the `detail` parent instance for depending nodes:

- `checkExistenceOfDependentInstances()`
- `readAllChildren(data)`
- `readFirstChildren(data)`
- `readAllChildrenFromCurrentInstance()`
- `readAllChildrenFrom()(DEP)`
- `readFirstChildrenFromCurrentInstance()`
- `readFirstChildrenFrom() (DEP)`
- `readInstance()(DEP)`
- `readInstanceWithFirstChildren() (DEP)`
- `readInstanceWithAllChildren() (DEP)`
- `readInstanceAndLock() (DEP)`
- `readInstanceWithFirstChildrenAndLock() (DEP)`
- `readInstanceWithAllChildrenAndLock() (DEP)`

### 5.1.3.5.3 Examples

There are two ways to use asynchronous methods:

- **Polling**

This system consists in 'watching for' an answer in the *thread* with no risk of blocking the application while waiting for the information display. The response code is stored in a *thread*, which is different from the one in the main application, but it points on the Root Proxy. We also assume that it is familiar with a context associated with an asynchronous method.

```
while (!myFolder().getReply(context)) {
    wait(1000);
}
```

- **Background job access**

The following example describes how to store information on the depending instances before the end user explicitly requests it, and with no risk to block the application.



- When the user chooses a collection of radical instances:

```
myFolder.setAsynchronous(false);
myFolder.selectInstances();
```

☞ As its result is required for the continuation of the operation, the **selectInstances** method is used with a synchronous mode.

- Then, **rows** will be browsed so as to find all the depending instances of each instance read by the **selectInstances** method.

You need first to switch to the asynchronous mode:

```
myFolder.setAsynchronous(true);
Enumeration rows = myFolder.rows.elements();
Vector contexts = new Vector();
while (rows.hasMoreElements()) {
    Data currentData = (Data)rows.nextElement();
    try {
        myFolder.readAllChildren(currentData);
    } catch (VapException ve) {
    } catch (AsynchronousRequestException are) {
        contexts.addElement(are.getContext());
    }
}
myFolder.setAsynchronous(false);
```

- Then, when the user wants to work in a data:

```
try {
    int index = myFolder.rows().indexOf(data);
    myFolder.getReply(contexts.elementAt(index));
} catch (VapException) {} //Everything is OK //
myFolder.getDetailFromDataDescription(data);
```

By this way, the depending instances are immediatly displayed in the selection tree. Note that the methods' responses are processed at the last moment, but it is not compulsory.

### 5.1.3.6 User Service

Each root or depending type node contains the following elements to implement a user service:

- A property used to obtain the list of user services available on this node plus **nil**. This property is **userServiceCodes**.
- A property used to initialize the user service code to be executed. This property is **userServiceCode**.
- A property used to locally store Logical View instances to be processed for the next user service. This property is **userServiceInputRows**.
- A property used to present various Logical View instances sent back by a user service. This property is **userServiceOutputRows**.
- A property which presents the Logical View instances which are candidates for the execution of the next user service. This property is **userDetail**.

- Local methods used to memorize each Logical View instance to be sent to the server to execute a user service. These methods are `createUserInstance`, `modifyUserInstance` and `deleteUserInstance`.

The root node of the Folder also has the following elements:

- A method used to execute all the user services parameterized on each Folder node. This method is `executeUserServices`.
- A method used to delete all the local instances stored for all the Folder nodes. This method is `resetUserServiceInputInstances`.
- A method used to delete the current instance stored locally. This method is `resetUserServiceCodes`.

This principle means that 1 to n user services can be executed, in the same query, distributed on the different nodes. The execution sequence of these services corresponds to the hierarchical order of nodes, browsing the tree from top to bottom and from left to right.

### 5.1.3.7 Database Logical Lock

The upload-download mechanisms associated with a Folder increase the elapsed time between reading the initial Folder image and displaying the result of an update.

In this context, with no lock mechanism, two users can modify the same Folder instance. The result of accumulated updates are therefore difficult to manage.

To enable the user to use a Folder in an exclusive appropriation mode, two types of locks for a node instance are available:

- The optimistic lock which works on the principle of verifying the change of a `TimeStamp` before executing the update procedure.
- The pessimistic lock which uses an entity for exclusive update by recording a specific resource. In this case, the Folder update procedure is carried out before freeing up the exclusive resource.

The server lock procedure is triggered by the explicit execution of a specific method available on the Root Proxy. This method is `lock`.

The server unlock procedure is triggered automatically with the execution of a server update method or explicitly by the execution of a specific method available on the Root Proxy. This specific method is `unLock`.

The developer is responsible for writing the lock processing in the root Business Component.

This processing receives the identifier of the Logical View instance to be locked as well as the request type (lock or unlock) to be executed.

In return, it must set a status used to grant or refuse the lock and the `TimeStamp` or the name of the resource used.

If the lock is refused, the Root Proxy sends a `lockFailed` event whereby all subsequently executed local or server update methods are disabled for the specified instance of the Folder. In this case, the Folder changes to the « Read only » status.

The concept of the logical lock is defined in the Folder entity or in the Business Component for a single-view development.

When the logical lock method is active on a Folder, all read requests for the **detail** property of the Root Proxy can be accompanied by a logical lock request on the server.

### 5.1.3.8 Customization of the Columns of a JTable

A JTable is a swing component available from VisualAge Java version 2.0 onwards.

If you insert this component as is, it will display, when the application is executed, all the columns which correspond to all the Data Elements of the Logical View, with the clear names defined in the Logical View.

To select the columns to be displayed, to change their heading or to create a new column which will display data locally computed, you must customize the JTable.

To do so, you must first create a new public class which inherits either from the generated TableModel (this is useful to retrieve part of its implementation) or directly from **PacbaseTableModel** (**com.ibm.vap.beans.swing** package).

Then you simply have to customize the following methods:

- **public Int getColumnCount()** : retrieves the number of columns to be displayed.
- **public String getColumnName(int col)** retrieves the heading of the **col** column (starting with column 0).
- **public Object getValueAt(int row, int column)** : retrieves the object (generally String) to be displayed on row **row**, column **column**.

The following example inherits from a generated TableModel. It reduces the number of columns to be displayed from 7 to 3. The first two columns represent two standard Data Elements (Client's Id and name). The third Data Element represents the client's address, i.e. the concatenation of the street, zip code and town.

```
package test.swing;

import com.ibm.vap.generated.reuse.CustomerData;
public class NewCustomerTableModel extends
com.ibm.vap.generated.reuse.CustomerTableModel {

    public int getColumnCount (){
        return 3;
    }

    public String getColumnName (int i){
        if (i == 0) return "Id";
        if (i == 1) return "Name";
        if (i == 2) return "Address";
        return "";
    }

    public Object getValueAt(int row, int column){
        try {
            CustomerData data = (CustomerData) getRows().elementAt(row);
            if (column == 0) return data.getCusId();
```

```

        if (column == 1) return data.getCusNam();
        if (column == 2) {
            String result = "" +
                data.getStreet() + "." +
                data.getZipcod() + "-" +
                data.getTown();
            return result;
        }
    } catch (Throwable t) {}
    return null;
}
}

```

### 5.1.3.9 Management of Data Element Presence

The two following methods enable you to manage the presence of the Logical View's Data Elements at the Proxy level.

The `is<delco>Present` method enables you to test the presence or absence of the `delco` Data element. It is generated for all `DataDescription` and `UserDataDescription` classes.

The `setNull<delco>Present(boolean aBoolean)` method enables you to specify the presence or absence of the `delco` Data element before a local update method. It is generated for all `DataDescription` and `UserDataDescription` classes.

By default, all the Data Elements are considered to be absent, except if a default value has been indicated in the VisualAge Pacbase description..

### 5.1.3.10 Management of Data Element Check

The three following actions enable you to manage the check of the Logical View's Data Elements at the Proxy level.

The `get<delco>Index` method indicates the index of the `delco` Data element in the `DataDescription` class. This index is used in the activation of the server check on the `delco` Data element.

The `setCheck(int index, boolean aBoolean)` method enables you to activate or inhibit the server checks on a Data Element (pointed by the index) before any local update method. It is generated for all the `DataDescription` classes of the root or depending nodes whose Business Components include the `NULLMNGT=YES` and `CHEKSER=YES` options and an update service.

By default, all the Data Elements are to be checked (if the `serverCheckOption` property is set to `true`).

### 5.1.3.11 Sub-Schema Management

The server selection or read methods take into account the sub-schema present in the `subSchema` property and return the values of the Data Elements belonging to the sub-schema. If a selection method is followed by a paging method, the sub-schema taken into account is that associated with the selection method.

The local creation methods do not refer to any sub-schema.

The local modification/deletion methods refer to the sub-schema associated with the instance, that is:

- if the modification/deletion is performed on an instance which was created locally, the sub-schema is empty.
- if the modification/deletion is performed on a read instance, the sub-schema is that associated with the selection of this instance.

Moreover the following methods are specific to the sub-schema management.

The `resetSubSchema` method enables you to reset the `subSchema` property, that is to select no sub-schema.

The `completeInstance` method enables you to retrieve the values of the Data Elements which do not belong to the sub-schema, by calling the Business Component associated with the Logical View.

The `belongsToSubSchema` method enables you to know whether the Data Element passed as a parameter belongs to the sub-schema associated with the `detail` attribute.

## 5.1.4 Use of Events

The events sent by an Elementary Proxy are used to trigger application methods belonging to the GUI application. This processing is performed by connecting a Proxy event to one or more methods in the GUI application. The conditional execution of methods is facilitated by the fact that an event is always accompanied by its opposite event; both events cannot be sent at the same time.



The availability of an event depends on the type of Proxy. All the Public Interface events are documented in the *Graphic Clients: Public Interface of Generated Components Reference Manual*.

### 5.1.4.1 Event-driven Management of Large Reading

Event-driven management of large reading provides the developer with information on the state of the collection of instances contained in a node. Each available paging action offers its own event-driven paging system.

The paging action in non-extend mode can send the following four events:

- **noPageBefore**: This event is sent by a root or reference node at the end of the execution of a collection selection or paging action when it does not return any error and when the read page is the first in the current collection.
- **pageBefore**: This event is sent by a root or reference node at the end of the execution of a collection selection or paging action when it does not return any error and when the read page is not the first in the current collection.
- **noPageAfter**: This event is sent by a root or reference node at the end of the execution of a collection selection or paging action when it does not return any error and when the read page is the last in the current collection.
- **pageAfter**: This event is sent by a root or reference node at the end of the execution of a collection selection or paging action when it does not return any error and when the read page is not the last in the current collection.

The paging method in extend mode can send the following two events:

- **pageAfter**: This event is sent by any type of node at the end of the execution of a collection selection or forwards paging action when it does not return any error and when the number of instances contained in the node is not the total number of instances contained in the database.
- **noPageAfter**: This event is sent by any type of node at the end of the execution of a collection selection or forwards paging action when it does not return any error and when the number of instances contained in the node is the total number of instances contained in the database when the query is made.

#### 5.1.4.2 Event-driven Management of Instance Reading

A Logical View can be mapped on one or more physical storage entities. In this context, the event-driven management of a Logical View instance reading can send two events:

- **notFound** when the instance searched for is not found in the database. This event can be sent when the Logical View is mapped on one or more tables.
- **notComplete** when the instance searched for is incomplete. This means that at least one table has satisfied the selection criterion and at least one table has not satisfied this criterion. This event cannot be sent in a multimapping context.

## 5.2 Example of an Applet

### 5.2.1 Introduction

This example describes a VisualAge Java applet, i.e. a program meant to be used in a browser.

This applet is developed first with VisualAge Java version 1 and then with VisualAge Java version 2 to illustrate the specificities of both versions.

The Proxy components inserted in the applet have been generated with the **Generate Beans** option and:

- The **Use IBM Enterprise Access Builder classes** option for version 1.
- The **Use Swing** option for version 2.

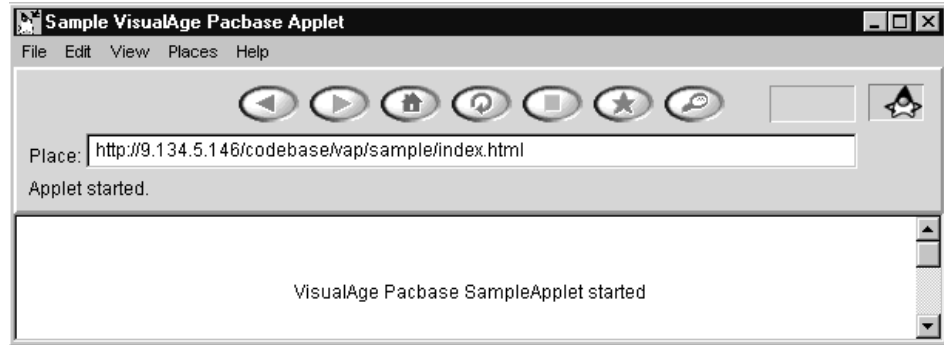
In the example, three Elementary Proxy of the FVP are used:

- The Root Proxy corresponding to the **Customers** node which manages the customers in the information system described by the Folder.
- The Depending Proxy corresponding to the **Orders** node which manages the orders in the information system described by the Folder.
- The Depending Proxy corresponding to the **Order lines** node which manages the order lines in the information system described by the Folder.

### 5.2.2 Presentation of the End User Interface

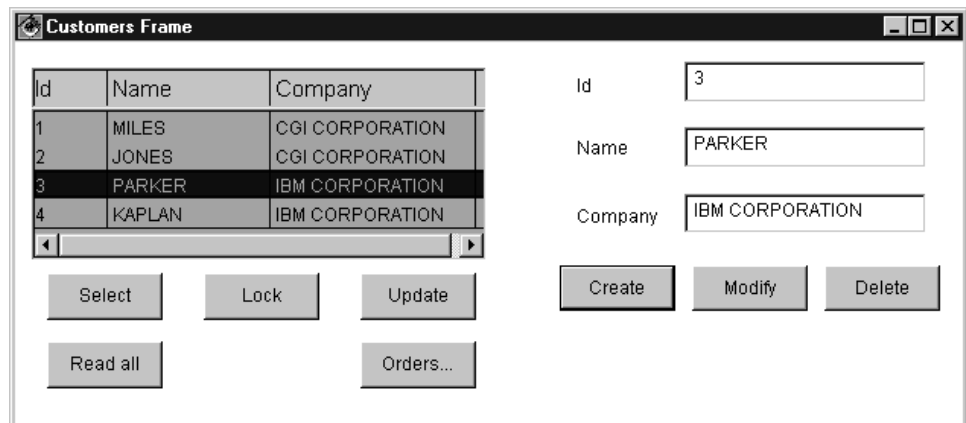
The graphic user interface consists of the applet itself and two windows.

- **The applet**



The applet does not have any function in the user application. It is the starting point and enables the application to be accessed via the web.

- **The Customers Window**

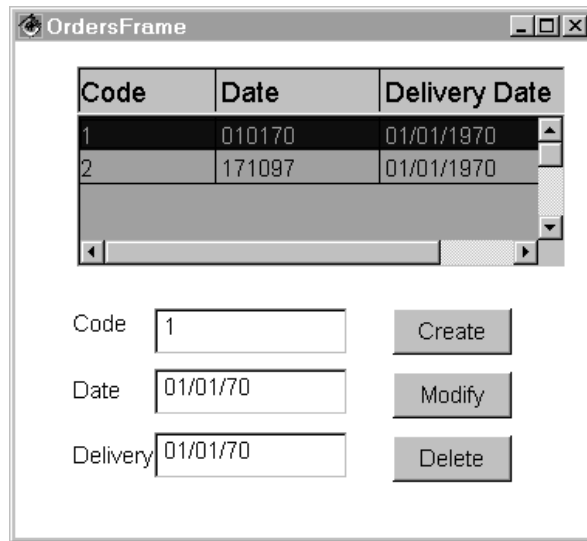


This window opens automatically at the applet start. It contains the following functionalities:

- The **Select** button is used to display a list of customers.
- The **Read all button** is used to submit a reading request of the selected customer 's orders from this window.
- As a lock option has been specified in the Folder, the user must click on the **Lock** button to lock the instance selected in the list before starting any updating process.
- The **Update** button is used to update the database.
- The **Orders...** button is used to display the Orders Frame window.
- The **Modify** button is used to modify customers in the detail after locking the instance.

When a customer is selected in the list, the user must click on **Lock** to appropriate this customer for a short time. Then, he can enter modifications and click on **Modify**.

- **The Orders Window**



The **Orders** window opens when the user clicks on the **Orders...** button in the **Customers** window.

For each customer selected in the detail of the **Customers** window, this window allows you to:

- view this customer's list of orders.
- select an order in the list and display it in the detail.
- create, modify, delete orders.

### 5.2.3 Developing the End User Interface with VisualAge Java V1

Developing the graphic user interface consists in programming the applet and in building the **Customers** and **Orders** windows.

You will not find here detailed information on how to use VisualAge tools and functions. If you are not familiar with them, refer to the appropriate documentation.

We try as much as possible to describe the development steps sequentially, but we sometimes need to organize the description of the windows programming into different consistent groups of functionalities.

At the end of each programming step, the Composition Editor is shown as it should appear on your screen, sometimes the connections previously developed are hidden so that you can see the newly created ones.

#### 5.2.3.1 Implementing the Example and Creating the Applet

- In a project, create a **vap.sample** package meant to contain the application components.
- In this package, create a **SampleApplet** applet. In the **SmartGuide - Create Applet** window, check the **Design the applet visually** option, so that the classes browser opens directly on the **Visual Composition** tab.



### • **Displaying the Text**

In the Visual Composition Editor, execute the following operations:

- Resize the applet.
- Place a **Label** bean (**Data Entry** category) in the applet. Enter **VisualAge Pacbase SampleApplet started** in the **Properties** window of the bean, in the **text** field.

### • **Integrating the Root Proxy**

To place the Root Proxy on the Free Form Surface, execute the following operations:

- In the **Options** menu, select the **Add Bean** choice.
- In the **Add Bean** window, which is opening, you must enter **com.ibm.vap.generated.proxies.CustomerProxyLv** in the appropriate field (you can use the **Browse...** Button).

The class name (**CustomerProxyLv** in our example) must be preceded by the package name selected for the generation. In our example, the package name is **com.ibm.vap.generated.proxies**.

### • **Defining the Communication with the Gateway**



For more information on this subject, refer to **5.5.2.1** paragraph, *Direct Access to the Middleware*.

In this example, we assume that the Folder View Proxy communicates with its host, that is to say, with the HTTP server where the applet is stored. To enable the communication, follow these steps:

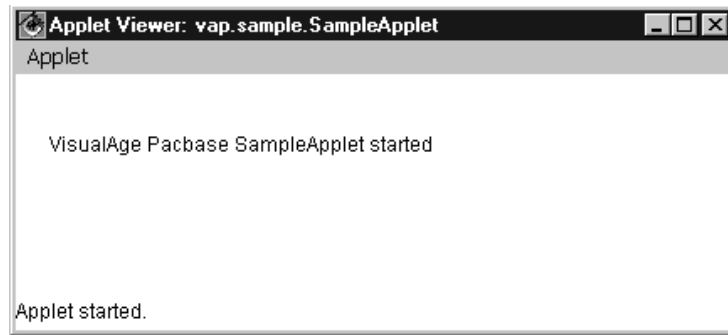
- Open the pop-up menu from the Free Form Surface.
- Select **Tear-Off Property**, then **codeBase (URL)**.
- Click on the Free Form Surface. A variable bean named **codeBase1** is displayed.
- Open the pop-up menu of the **codeBase1** bean, select **Connect**, and then **All Features...**.
- In the **Start connection from** window, select the **host** property. A dotted link is displayed.
- Click the Root Proxy. The pop-up menu is displayed.
- Select **All Features...**. The **Connect property named:** window opens.
- Display all the properties available for the Root Proxy by checking the **Show expert features** box.
- Select the **Host** property, then click on **OK**.

The **host** property of **codeBase1** is now connected to the **Host** property of the Root Proxy.



These connections are equivalent to the following code line in the applet:  
`getCustomerProxyLv1().setHost(getCodeBase().getHost());`

Now, you can test your applet. The **Applet Viewer** windows opens:



- **Programming the Opening of the CustomersFrame Window from the Applet**

This phase consists of two parts ; both must be done after the operations described in the *Creating the Window and Integrating the FVP in the Composition Editor* and *Promoting the Root Proxy* paragraphs:

- Calling the Root Proxy in the applet
  - ♦ In the applet's Composition Editor, place a constant bean of **CustomersFrame** type on the Free Form Surface.
  - ♦ Connect the **this** property of the Root Proxy to the **customerProxyLv1This** property of the **CustomersFrame** bean.
- Programming the opening of the window from the applet.

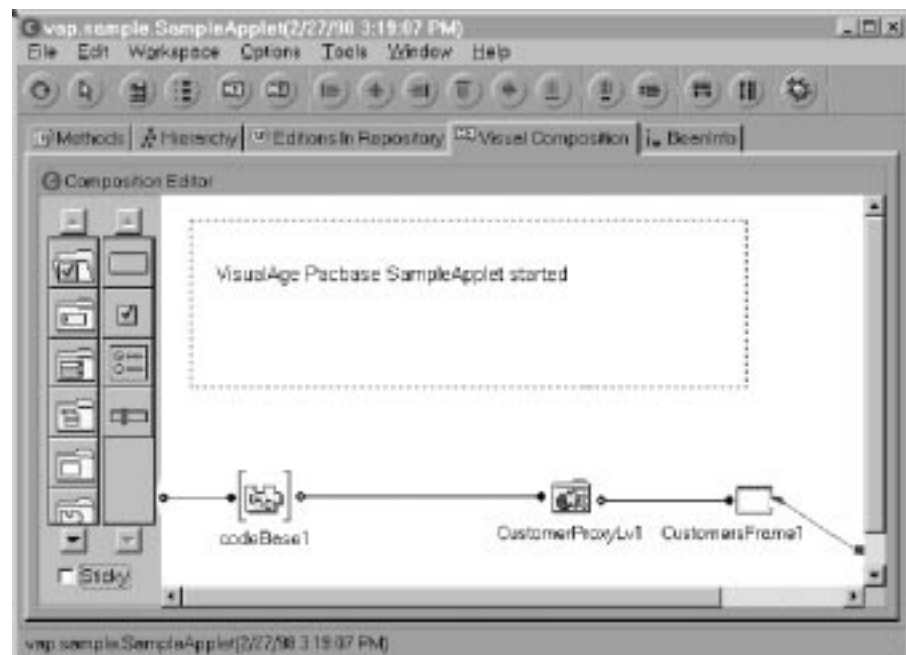
Now we want the **CustomersFrame** window to open immediately after the applet starts.

To do so, connect the **componentShown** event of the applet to the **show** method of the **CustomersFrame** bean.

- **Result in the Composition Editor**

The applet is now completed.

The Composition Editor should look like this at this stage:



### 5.2.3.2 Developing the Customers Window

#### 5.2.3.2.1 Mapping Rows and Detail

Once the Root Proxy has been inserted and promoted, this phase describes the mapping of its **rows** and **detail** properties.

- **Creating the Window and Integrating the FVP in the Composition Editor**

Creating a window for the management of customers consists in creating the corresponding class in the Workbench.

- In the Workbench, create a **CustomersFrame** class in the **vap.sample** package.
- In the **SmartGuide -Create Class or Interface** window which opens then:
  - ♦ Enter **java.awt.Frame** in the **Superclass** field. You specify this way that the **CustomersFrame** class inherits from the **java.awt.Frame** class.
  - ♦ Check the **Design the class visually** option so that the class browser directly opens on the **Visual Composition** tab.
- To integrate the Folder View Proxy in the Visual Composition Editor, execute the steps detailed in the *Creating the Window and Integrating the FVP in the Composition Editor* paragraph. But this time, you must select the **Variable** option in the **Bean Type** field of the **Add Bean** window.
  - ☞ The Root Proxy placed here is a Proxy of a variable type because it must represent the same Proxy instance as the one called in the applet.

- **Promoting the Root Proxy**

Now the purpose is to ensure the permanent identity of this variable Proxy and the constant Proxy instantiated in the applet. To do so, the variable Proxy must be public outside the **CustomersFrame** class, so that a property connection – property between the constant and variable Proxy objects can be performed in the applet.

Proceed as follows:

- From the Proxy's pop-up menu, choose **Promote Bean feature**.
- In the **Property** column, choose **this** and click **Promote**.  
The **CustomersFrame** class now has a readable/writable property named **customerProxyLv1This** and typed Root Proxy.
- Save the window (**File** menu, **Save Bean** or **CTRL-F2** choice).

- **Mapping the Rows Property**

☞ This mapping uses an EAB container; so it is possible only if you checked the **Use the IBM Enterprise Access Builder classes** for the generation.

This step consists in the following operations:

- In the **CustomersFrame** bean, place a **Multi Column List Box** bean, located in the **Access Enterprise** category. Resize the bean.
- You have now to map, in the container, the columns corresponding to the Data Elements associated with the root node.

Open the **Properties** window of the container. From the **columns** field of this window, add a column for each Data Element of the Logical View, in the order they are called in the Repository. You can map as many columns as properties held by the Proxy **DataDescription** or the first ones only.

- ☞ The columns to be mapped correspond to the values returned by the **getAttributeStrings()** method of the **DataDescription**.
- Once the columns have been created, connect the **IRows** property of the Proxy to the **elements** property of the container.
  - ☞ **IRows** is identical to **Rows** but it returns an instance of the **IVector** class, which, contrary to the **DataDescriptionVector** class returned by **Rows**, is compatible with the **elements** property of the **Multi Column List Box** bean.

This operation is equivalent to the mapping of the **Rows** property of the Proxy.

#### • Mapping the Detail Property

This step requires the following operations:

- From the Root Proxy, tear-off the **detail** property.
- Insert a **label** bean for each property to be mapped.
- You must now map an input field for each property to be displayed.

To do this, use the **Pacbase Text Field**, **Pacbase Integer Field**, **Pacbase Decimal Field**, **Pacbase Date Field** and **Pacbase Time Field** beans provided by Pacbench C/S when the runtime is imported.

These beans are located in the **VisualAge Pacbase** category, in the palette.

You can also insert these beans by selecting the **Add bean** choice in the **Options** menu; enter the full name of the package followed by the bean name. For example: **com.ibm.vap.beans.PacbaseDateField**.

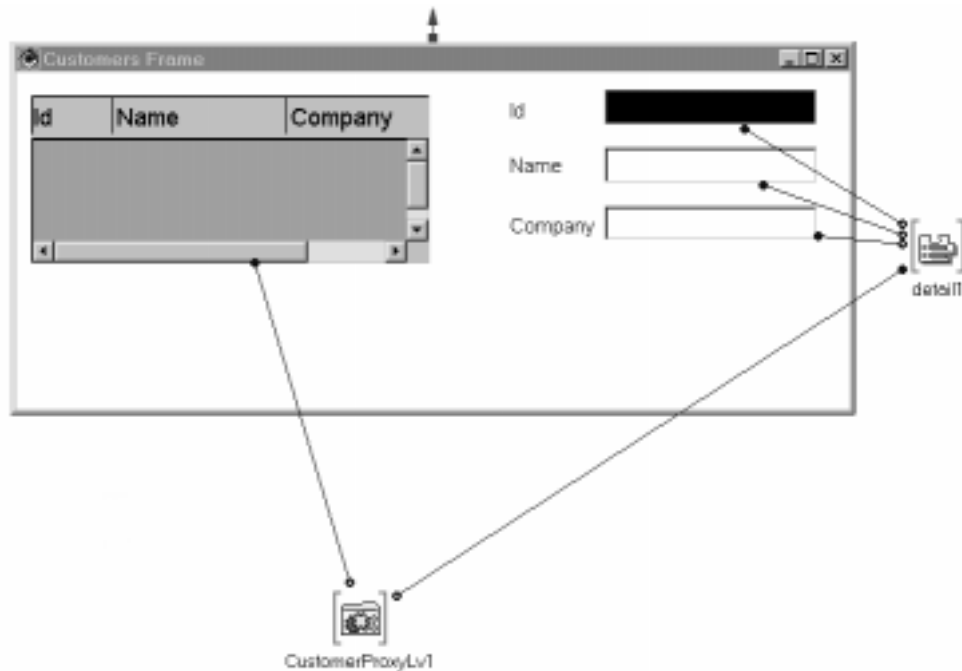
☞ For more details on these beans, refer to the on-line documentation of the generic classes.

- Insert the type of bean required for each property to be mapped.
- Then, for each mapped field, connect the property of the **detail** bean to its corresponding **String**, **Int**, **Decimal**, **Date** or **Time** property.

In our example, for the input field corresponding to the customer number, you must connect the **Customer Number** property of the **detail** bean to the **Int** property of the field.

- **Result in the Composition Editor**

After all these steps, the Composition Editor should look like this:



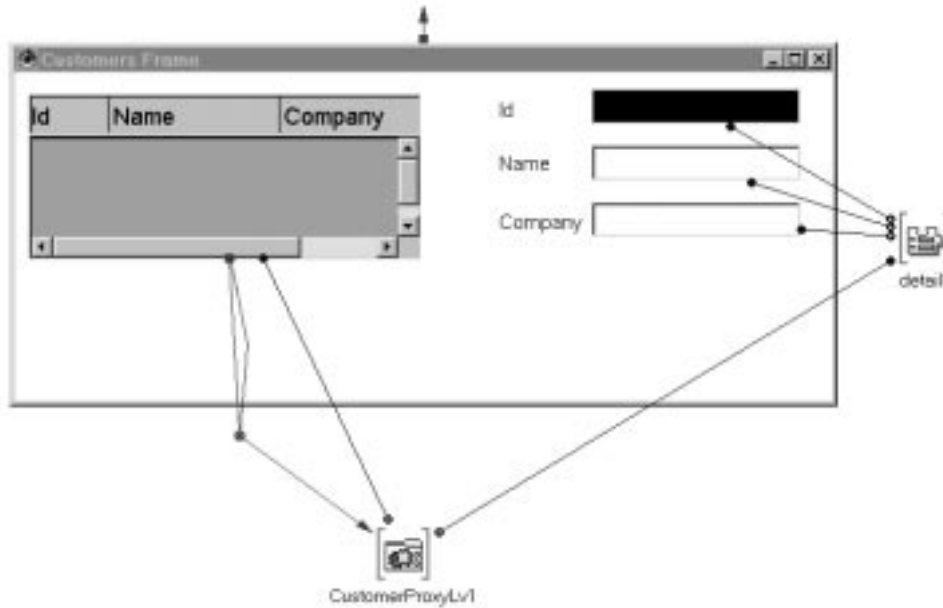
#### 5.2.3.2.2 Selection of an Instance in Rows and Transfer in Detail

Now the purpose is to program the transfer of the instance selected by the user, from the container into the **detail** property of the Proxy.

To do this, proceed as follows:

- Connect the **itemStateChanged** event of the MultiColumnListBox to the Root Proxy's **Get Detail From DataDescription** method.
- The link appears as a dotted line because the **Get Detail From DataDescription** method requires a parameter.
- To define this parameter, connect the **selectedObject** property of the MultiColumnListBox to the data-type parameter of the link (**customerData** in our example). This parameter is available when the cursor is placed in the middle of the connection line.

The Composition Editor should look like this:



### 5.2.3.2.3 Activation of the Proxy's Methods and Navigation towards the Orders Window

- **Activation of the Proxy's Methods**

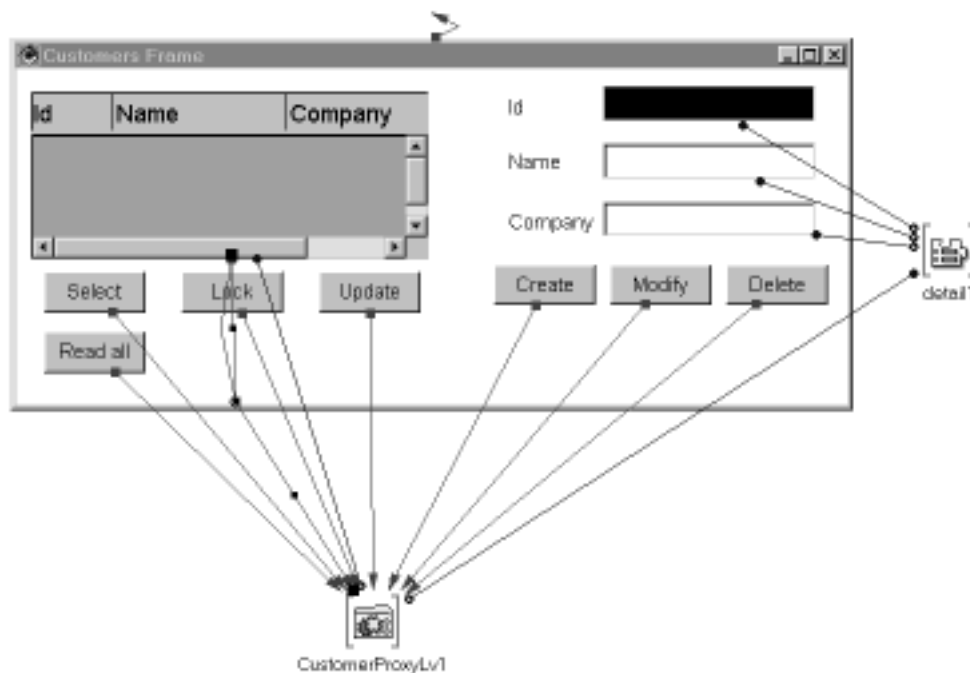
For each method you wish programming, place a push-button in the window. You can modify the label in the **Properties** window of the button.

To activate a method, connect the **actionPerformed** event of a given button to the appropriate method of the Proxy. If this method is not available, check the **Show expert features** option.

For example:

- To program the activation of the **Update** button, you must connect its **actionPerformed** event to the **Update Folder** method of the Proxy.
- To program the activation of the **Read all** button, you must connect its **actionPerformed** event to the **Read All Children From Detail** method of the Proxy.

After these steps, the Composition Editor should look like this:

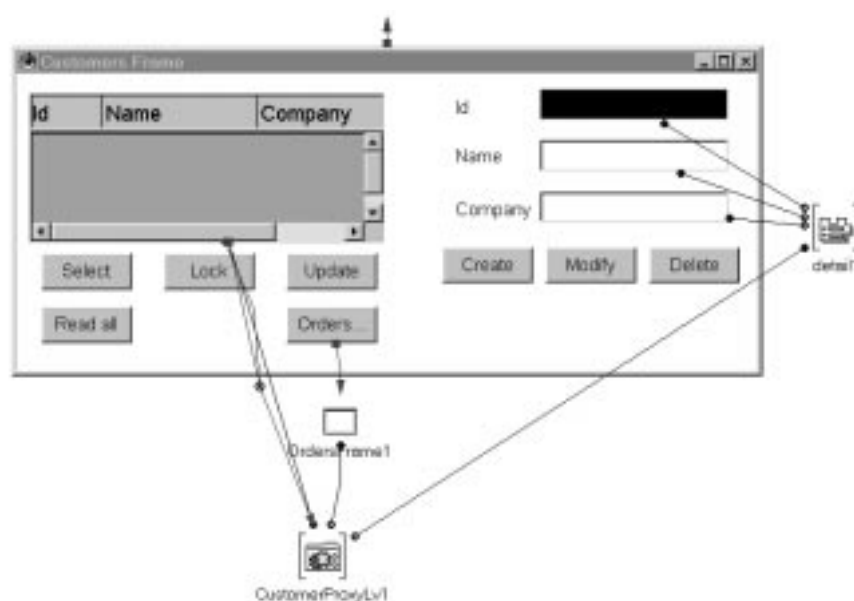


- **Navigation management**

This step must be executed after promoting the Orders Depending Proxy.

- On the Free Form Surface, place an **OrdersFrame** constant bean.
- Connect the **Order Proxy** of the Root Proxy to the **orderProxyLv1This** property of the **OrdersFrame** bean. This connection ensures the link of the two Proxy objects between the two windows.
- Place an **Orders...** button in the **CustomersFrame** bean.
- Connect the **actionPerformed** event of the button to the **show** method of the **OrdersFrame** bean.

At the end of this step, the Composition Editor should look like this:



For better readability, the links between the other buttons and the Root Proxy are hidden.

### 5.2.3.3 Developing the Orders Window


This window gives information on the orders. It opens when the user clicks on the **Orders...** button in the **Customers** window.

The container automatically displays the orders of the selected client in the **Customers** window if the user clicks **Read all** before clicking **Orders...**

The development of this window using the **OrderProxyLv** Depending Proxy consists of the stages described below.


#### 5.2.3.3.1 Creating the Orders Window

In the Workbench, create an **OrdersFrame** class which inherits from **java.awt.Frame** in the **vap.sample** package, as you did when you created the **Customers** window.

 For more details, refer to the paragraph [5.2.3.2](#), point *Creating the Window and Integrating the FVP in the Composition Editor*.

#### 5.2.3.3.2 Integrating and Promoting the Depending Proxy

In the Visual Composition tab, place an **OrderProxyLv** variable bean on the Free Form Surface.

 For more details on the proxy integration, refer to paragraph [5.2.3.2](#), point *Creating the Window and Integrating the FVP in the Composition Editor*.

Now we have to promote this variable Proxy so that it can be public from the outside.


To do this, proceed as follow:

- From the Proxy's pop-up menu, choose **Promote Bean feature**.
- In the **Property** column, choose **this** and click on **Promote**.  
The **OrdersFrame** class has a public property now, in a read/write mode, named **OrderProxyLv1This** and typed Depending Proxy.
- Save the window (**File** menu, **Save Bean** or **CTRL-F2** choice).

#### 5.2.3.3.3 Mapping the Depending Proxy's Rows and Detail


The operations to be executed here are identical to those required for the mapping of the **rows** and **detail** properties of the Root Proxy.

The same EAB container is used for the mapping of **rows**.

 For more details, refer to the point [5.2.3.2.1](#).

#### 5.2.3.3.4 Selection of an Instance in Rows and Transfer in Detail

The operations to be executed here are identical to those required for the **rows** and **detail** properties of the Root Proxy.

 For more details, refer to the point [5.2.3.2.2](#).

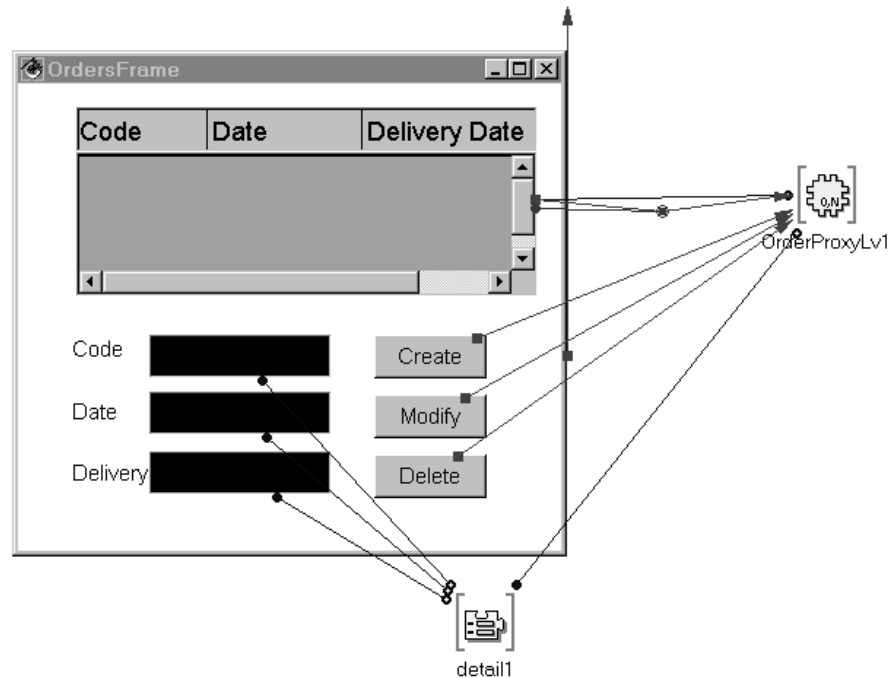
#### 5.2.3.3.5 Activation of the Depending Proxy's Methods

Use the same principles as those described in the point [5.2.3.2.3](#).



### 5.2.3.3.6 Result in the Composition Editor

After all these steps, the Composition Editor should look like this:



## 5.2.4 Developing the End User Interface with VisualAge Java V2

This section details the development, with VisualAge Java V2, of the application whose end-user interface is presented in section 5.2.2.

The Proxy components inserted in the application have been generated with the **Generate Beans** et **Use Swing** options. All the components inserted here are Swing beans.

We develop the application from scratch, without using the application developed with VisualAge Java V1.



If you have already developed an application with VisualAge Java V1, you can save it and resume its development with VisualAge Java V2. However if you want to insert Swing components, you must re-generate the Proxy components with the **Use Swing** option to ensure a correct communication between the Proxy components and the Swing beans.



VisualAge Java V2 does not recognize EAB containers, which are components specific to VisualAge Java V1. If the application developed with the Version 1 includes such components, you must replace them with other components (swing JTable component for example).

### 5.2.4.1 Implementing the Example and Creating the Applet

- In a project, create an **example.swing** package meant to contain the application components.
- In this package, create a **SwingApplet** applet. In the **SmartGuide - Create Applet** window, select **JApplet** in the **SuperClass** field, via the **Browse** button, and check the **Compose the applet visually** option, so that the class browser opens directly on the **Visual Composition** tab.

- **Displaying the Text**

In the Visual Composition Editor, execute the following operations:

- Resize the applet.
- Place a `JLabel` bean in the applet. Enter `VisualAge Pacbase SwingSampleApplet started` in the `Properties` window of the bean, in the `text` field.

- **Integrating the Root Proxy**

To place the Root Proxy on the Free Form Surface, execute the following operations:

- Click the `Choose Bean` icon located above the palette.
- Select the type of bean `Class`.
- With the Browse button, select class name `CustomerProxyLv`.

- **Defining the Communication with the Gateway**



For more information on this subject, refer to paragraph [5.5.2.1](#), *Direct Access to the Middleware* .

In this example, we assume that the Folder View Proxy communicates with its host, that is to say with the HTTP server where the applet is stored. To enable this communication, follow these steps:

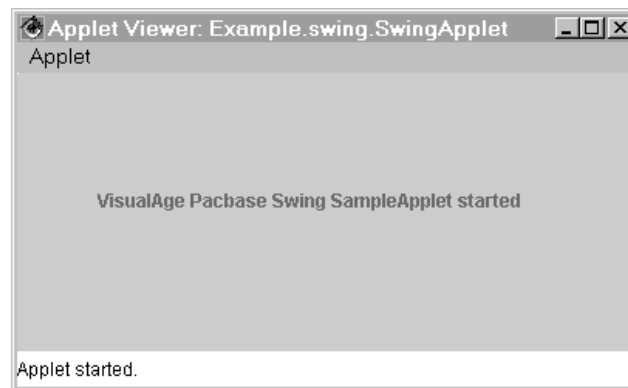
- Open the pop-up menu in the Free Form Surface.
- Select `Tear-Off Property`, then `codeBase (URL)`.
- Click in the Free Form Surface. A variable bean named `codeBase1` is displayed.
- Open the pop-up menu of the `codeBase1` bean, select `Connect`, and then `Connectable Features...`
- In the `start connection from` window, select the `host` property. A dotted link is displayed.
- Click on the Root Proxy. The pop-up menu is displayed.
- Select `Connectable Features...`. The `End Connection to` window opens.
- Display all the properties available for the Root Proxy by checking `Show expert features`.
- Select the `Host` property, then click `OK`.

The `host` property of `codeBase1` is now connected to the `Host` property of the Root Proxy.



These connections are equivalent to the following code line in the applet:  
`getCustomerProxyLv1().setHost(getCodeBase().getHost());`

Now you can test your applet. The **Applet Viewer** window opens:



- **Programming the opening of the CustomersFrame Window from the applet**

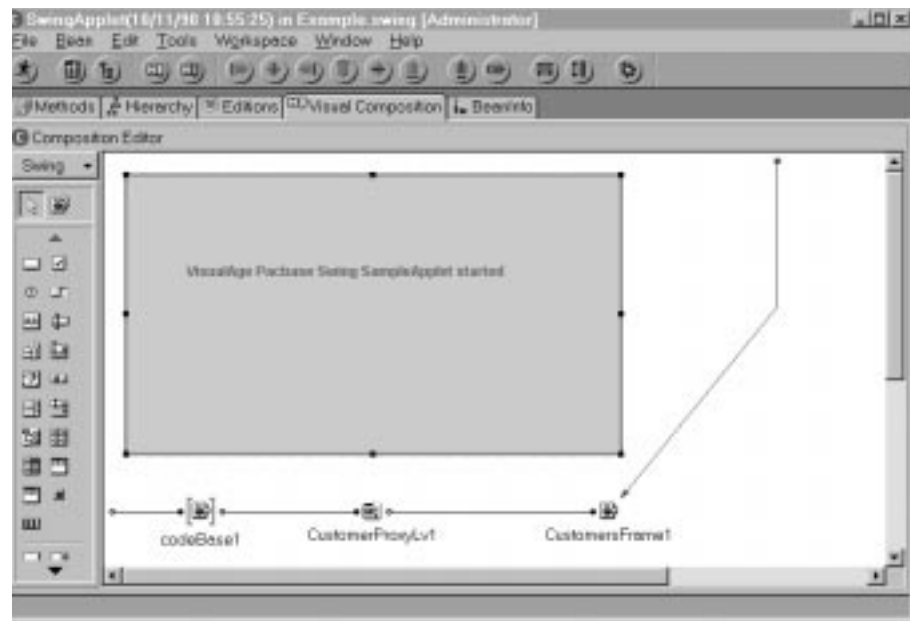
This step consists of two parts. Both must be done once the operations described in paragraphs *Creating the Window* and *Integrating the FVP in the Composition Editor* and *Promoting the Root Proxy* have been executed :

- Call of the Root Proxy in the applet
  - ♦ In the applet's Composition Editor, place a constant bean with the **CustomersFrame** type on the Free Form Surface.  
To do so, select the **Choose Bean** icon located above the palette, select the **Class** type of bean and select, with the Browse button, the **CustomersFrame** class.
  - ♦ Connect the **this** property of the Root Proxy to the **customerProxyLvl1This** property of the **CustomersFrame** bean.
- Programming the opening of the window from the applet  
Now we want the **CustomersFrame** window to open immediately after the applet starts.  
To do so, connect the **componentShown** event of the applet to the **show** method of the **CustomersFrame** bean.

- **Result in the Composition Editor**

The applet is now completed.

The Composition Editor should look like this as this stage:



## 5.2.4.2 Developing the Customers Window

### 5.2.4.2.1 Mapping Rows and Detail

Once the Root Proxy has been inserted and promoted, this phase describes the mapping of its **rows** and **detail** properties.

- **Creating the Window and Integrating the FVP in the Composition Editor**

To create a window for the management of customers consists in creating the corresponding class in the Workbench.

- In the Workbench, create a **CustomersFrame** class in the **example.swing** package.
- In the **SmartGuide -Create Class or Interface** window which opens then:
  - ♦ select **JFrame**, with the **Browse** button, in the **Superclass** field: you specify this way that the **CustomersFrame** class inherits from the **com.sun.java.swing.JFrame** class.
  - ♦ Check the option **Compose the class visually** so that the class browser opens directly on the **Visual Composition** tab.
- To insert the Folder View Proxy in the Visual Composition Editor:
- Click on the **Choose Bean** icon located above the palette.
- Select the **Variable** type of bean.
- With the Browse button, select the **CustomerProxyLv** class.

☞ The Root Proxy inserted here is of a variable type because it must represent the same Proxy instance as the one called in the applet.

- **Promoting the Root Proxy**

Now the purpose is to ensure the permanent identity of this variable Proxy and the constant Proxy instantiated in the applet. To do so, the variable Proxy must be public outside the `CustomersFrame` class, so that a property – property connection between the constant Proxy and the variable Proxy can be performed in the applet.

Proceed as follows:

- Open the Proxy's pop-up menu. Select **Promote Bean feature**.
- In the **Property** column, choose **this** then click **>>**.

The `CustomersFrame` class now has a readable/writable public property named `customerProxyLvlThis` and typed Root Proxy.

- Save the window (**Bean** menu, **Save Bean** choice).

- **Mapping the Rows Property**

In the `CustomersFrame` bean, place a `JTable` bean and resize it.

To see the columns of the `JTable`, you must run the `JFrame`. The columns of the `JTable` are initialized with the Data Elements of the Logical View and the lines represent the instances included in **Rows**. The content of the `JTable` is refreshed each time **Rows** is updated (selections, creations...).

Two mappings are possible:

- If you want to display all the columns which correspond to all the Data Elements of the Logical View with the clear names defined in the Proxy, you simply have to connect the **Table model** property of the Proxy to the **model** property of the `JTable`.
- However if you want to select the columns to be displayed, modify their heading or create a new column to display data locally computed, you must customize the `JTable`.

To do so, you must create a new `TableModel` class and customize its methods.

This new class must be public and must inherit:

- ♦ either from `CustomerTableModel`, located in the `com.ibm.vap.generated.reuse` package. This way, this method automatically inherits all the existing methods of `CustomerTableModel` and you will have to modify only the methods that do not suit your application.
- ♦ or directly from `PacbaseTableModel`, located in the `com.ibm.vap.beans.swing` package. In this case, you will have to re-write all the methods you want to use since they are not retrieved automatically.

To create the new class, select the **Add Class** choice in the pop-up menu of the package (`com.ibm.vap.generated.reuse` or `com.ibm.vap.beans.swing`). Name it (`NewCustomerTableModel` for example) and, in the **Superclass** field, select the class (`CustomerTableModel` or `PacbaseTableModel`) from which the new class will inherit.


You must then customize the methods of this new class by inserting code directly in the **source** part. In our example, we chose to make the new `NewCustomerTableModel` class inherit from the `CustomerTableModel` class.

The `public int getColumnCount()` method must be customized to limit the number of columns to 3 in the JTable, whereas the Logical View contains 7 Data Elements.

```
public class NewCustomerTableModel extends
com.ibm.vap.generated.reuse.CustomerTableModel {

    public int getColumnCount (){
        return 3;
    }

}
```

 The other methods via which the columns of the JTable can be customized are documented in the paragraph [5.1.3.8](#).

In the Composition Editor, you simply have to:


- Place an instance of the new `TableModel`,
- Connect this instance, via its `this` property, to the `TableModel` property of the Proxy,
- Connect this instance, via its `this` property, to the `model` property of the JTable.

#### • Mapping the Detail Property

This step requires the following operations:

- From the Root Proxy, tear-off the `detail` property.
- Insert a `JLabel` bean for each property to be mapped.
- you must now map an input field for each property to be displayed.  
To do so, use the beans `Pacbase Swing Text Field` and `Pacbase Swing Integer Field` provided by Pacbench C/S when the runtime is imported.

You can also insert these beans by clicking on the `Choose Beans` icon located above the palette. Select the type `Class` or `Variable`. Then select, with the Browse button, the name of the bean: for example: `PacbaseJTextField`.

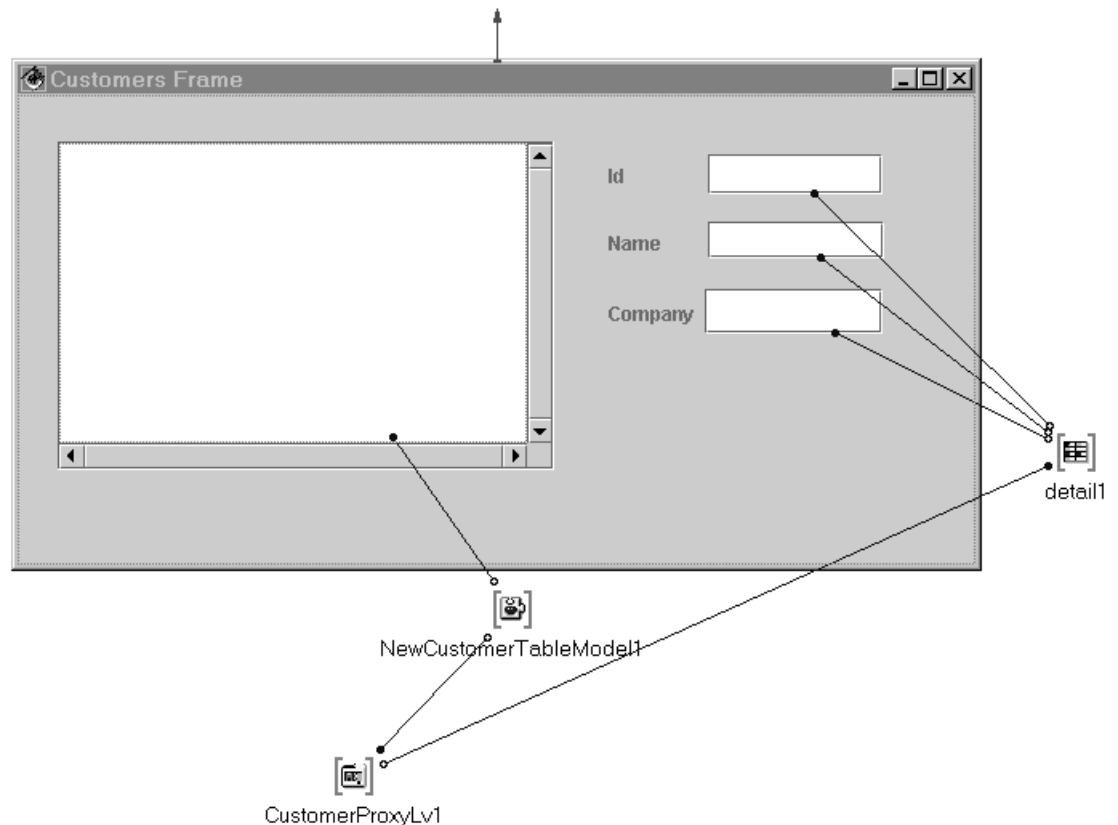
 For more details on these beans, refer to the on-line documentation of the generic classes.

- Insert the type of bean required for each property to be mapped.
- then, for each mapped field, connect the property of the `detail` bean to the corresponding `String` or `Int` type property.

In our example, for the input field which corresponds to the customer number, you must connect the `Customer Number` property of the `detail` bean to the `Int` property of the field.

- **Result in the Composition Editor**

After all these steps, the Composition Editor should look like this:



We chose here to customize the JTable. This is the reason why a NewCustomerTableModel1 bean is inserted.

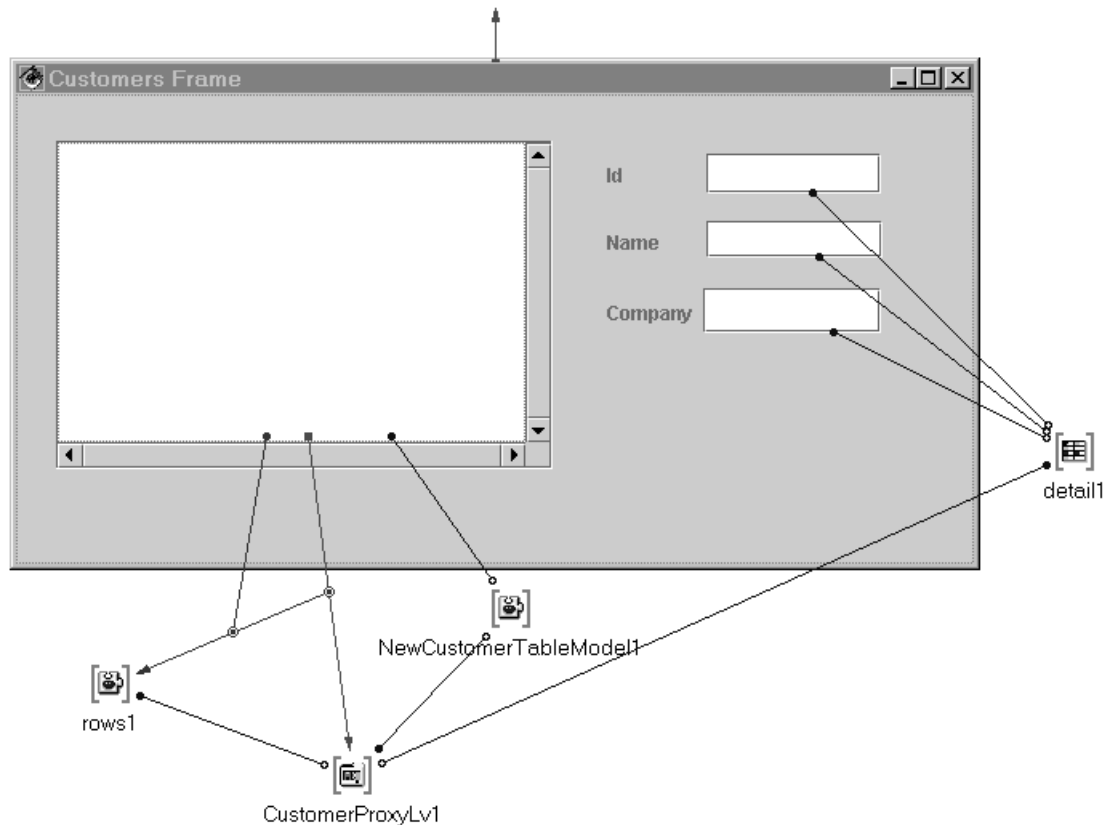
#### 5.2.4.2.2 Selection of an Instance in Rows and Transfer in Detail

Now the purpose is to program the transfer of the instance selected by the user from the table into the **detail** property of the Proxy.

To do so, proceed as follows:

- Connect the **keyEvents** and **mouseEvents** events of the JTable to the **Get Detail From DataDescription** method of the Root Proxy.
- Two links appear in dotted lines since the **Get Detail From DataDescription** method requires a parameter.
- Make a **Tear-Off** of the **Rows** property of the Root Proxy. You obtain a variable bean named **rows1**.
- Select the dotted links one at a time. To specify the parameter required by the **Get Detail From DataDescription** method, connect the data-type parameter of the link (here **customerData**) to the **elementAt(int)** method of **rows1**.
- Then connect the **SelectedRow** property of the JTable to the **Arg1** parameter of the link created in the preceding step. This parameter is available when the cursor is placed in the middle of the connection link.

The Composition Editor should look like this:



To avoid complexity, we display here the connection of only one of the two events sent by the JTable.

#### 5.2.4.2.3 Activation of the Proxy's Methods and Navigation towards the Orders Window

- **Activation of the Proxy's Methods**

Methods are activated in the same way as in the Root Proxy of the VisualAge Java V1 example.



For more details, refer to point [5.2.3.2.3](#).

- **Navigation Management**

Navigation is managed in the same way as in the Root Proxy of the VisualAge Java V1 example.



For more details, refer to point [5.2.3.2.3](#).

#### 5.2.4.3 Developing the Orders Window

The Orders window is identical to that developed in the VisualAge Java V1 example.



For more details, refer to paragraph [5.2.3.3](#).

##### 5.2.4.3.1 Creating the Orders Window

In the Workbench, create an `OrdersFrame` class which inherits from `JFrame` in the `example.swing` package, as you did when you created the `Customers` window.



For more details, refer to paragraph [5.2.4.2](#), point *Creating the Window and Integrating the FVP in the Composition Editor*.

#### 5.2.4.3.2 Integrating and Promoting the Depending Proxy

In the Visual Composition tab, place an `OrderProxyLv` variable bean on the Free Form Surface.

For more details on the integration of the Proxy, refer to paragraph [5.2.4.2](#), point *Creating the Window and Integrating the FVP in the Composition Editor*.

You must then promote this variable Proxy so that it can be public from the outside.

To do so, proceed as follows:

- From the Proxy's pop-up menu, choose **Promote Bean feature**.
- In the **Property** column, choose **this** and click on **>>**.  
The `OrdersFrame` class has a public property now, in read/write mode, named `OrderProxyLv1This` and typed Depending Proxy.
- Save the window (**Bean** menu, **Save Bean** choice).

#### 5.2.4.3.3 Mapping the Depending Proxy's rows and detail

The operations to be executed are identical to those required for the Root Proxy.

For more details, refer to point [5.2.4.2.1](#).

#### 5.2.4.3.4 Selection of an Instance in rows and Transfer in detail

The operations to be executed are identical to those required for the `rows` and `detail` properties of the Root Proxy.

For more details, refer to point [5.2.4.2.2](#).

#### 5.2.4.3.5 Activation of the Proxy's Methods

Use the same principles as those described in the point [5.2.3.2.3](#).

## 5.3 Specificities of the Development of a Standalone Application

### 5.3.1 Introduction

This subchapter puts the emphasis on the differences between the development of an applet and the development of a *standalone* application.

A *standalone* application, unlike an applet application, is not meant to be executed by a Web browser.

The main differences between the two types of applications are the following ones:

- For the development of a *standalone* application, the base class in use has a `Frame` type (for awt) or a `JFrame` type (for swing). This class inherits from `java.awt.Frame` (for awt) or `com.sun.java.swing.JFrame` (for swing) whereas an applet inherits from `java.applet.Applet` (for awt) or from `java.applet.JApplet` (for swing).

- As for the development of a *standalone* application, the instantiation of the Root Proxy is conditioned by the use of the `setLocationsFile` method, and not by the use of the `Host` and `Port` properties. This method is used to position the `VAPLOCAT.INI` *locations* file so that the application can directly get to the middleware without passing through the gateway.

### 5.3.2 Example

This example shows you how to transform an applet previously developed in this subchapter into a *standalone* application. In the example, the `CustomersFrame` and `OrdersFrame` windows are re-used. Now, the `CustomersFrame` window is the starting point of the application.

☞ For the windows developed with the Swing option of VisualAge Java V2, you simply have to replace the awt classes present in the code by the corresponding Swing classes.

No changes are required in the Composition Editor.

You just have to insert specific code in the `main` method of the `CustomersFrame` class. The `main` method is the starting point of a *standalone* application.

The specific code is presented below, between the `//begin` and `//end` comment lines.

```
/**
 * main entrypoint - starts the part when it is run as an application
 * @param args java.lang.String[]
 */
public static void main(java.lang.String[] args) {
    try {
        vap.sample.CustomersFrame aCustomersFrame = new vap.sample.CustomersFrame();
        try {
            Class aClosorClass = Class.forName(« uvm.abt.edit.WindowClosor »);
            Class parmTypes[] = { java.awt.Window.class };
            Object parms[] = { aCustomersFrame };
            java.lang.reflect.Constructor aCtor = aClosorClass.getConstructor(parmTypes);
            aCtor.newInstance(parms);
        } catch (java.lang.Throwable exc) {};
        //begin
        aCustomersFrame.setCustomerProxyLvl(new
        com.ibm.vap.generated.proxies.CustomerProxyLv());
        aCustomersFrame.getCustomerProxyLvl().setLocationsFile(« d:\\user\\vapb\\vaplocat.ini
        »);
        //end
        aCustomersFrame.setVisible(true);
        catch (Throwable exception) {
            System.err.println(« Exception occurred in main() of java.awt.Frame »);
        }
    }
}
```

The first instruction specifies the creation of a new Root Proxy. The former Proxy has already been instantiated in the applet and it cannot be re-used here as a constant bean.

The next instruction positions the *locations* file.

## 5.4 Error Management

There is no error message displayed during the extraction and generation phases.

To be extracted, a Business Component must have been compiled correctly, because:

- the GVC command in the GPRT procedure extracts the required data without displaying any error messages, even if the Business Component does not contain any Logical View,
- The generator does not control the input file.

However, some error messages can be displayed during the development, test and execution phases of the application. These messages result from local, server or communication errors.

### 5.4.1 Principles

#### 5.4.1.1 Introduction

The management of the errors associated with the handling of VisualAge Pacbase Proxies is based on the the raise of exceptions principle specific to the Java language.

The Proxy objects can raise four types of Pacbench C/S errors and also the whole set of Java errors.

For the management of the Pacbench C/S errors, the user is provided with four generic classes whose methods enable to convey the errors raised by the Proxy objects. Each class corresponds to a type of errors:


- Local errors
- Server errors
- System errors
- and Communication errors


All these classes inherit from `java.lang.Throwable` and are stored in the `com.ibm.vap.generic` package.

#### 5.4.1.2 Programming

The exceptions must be intercepted by the programming in the Client component.

Programming errors requires writing in Java on the one hand, and on the other hand, creating the window used to display these errors.

 An example of error management is provided in section [5.4.6](#).

 For more information on the exceptions that can be possibly raised by the Proxies, refer to the *Reference Manual Graphic Clients: Public Interface of Generated Components* or, in your VisualAge station, the corresponding method signature.

## 5.4.2 Local Errors

The local errors produce the `com.ibm.vap.generic.LocalException` exception. This exception holds a property of `int` type that enables the error identification.

☞ The errors which are responsible for this exception are also described in the HTML documentation associated with the generic classes: Package `com.ibm.vap.generic`.

### 5.4.2.1 List of Errors

- **ASYNCHRONOUS\_VIOLATION**  
This error is produced when a lock, unlock or existence check of depending instances in asynchronous mode is requested.
- **CARDINALITY\_VIOLATION**  
This error is produced if the cardinalities are not respected when an update method is activated.
- **CURRENT\_INSTANCE\_MISSING**  
This error is produced when a method is applied to the `detail` property whereas the latter does not contain any instance.
- **CURRENT\_USER\_INSTANCE\_MISSING**  
This error is produced when a user method is applied to the `userDetail` property whereas this property does not contain any instance.
- **FOLDER\_USER\_CONTEXT\_LENGTH\_ERROR**  
This error is produced when the length of the value of a Data Element belonging to the user buffer exceeds the authorized length for this Data Element.
- **INSTANCE\_ALREADY\_LOCKED**  
This error is produced when a lock action is requested on an instance, which has already been locked on the server.
- **INSTANCE\_NOT\_LOCKED**  
This error is produced when a lock action is requested on an unlocked instance on the server.
- **INVALID\_CHANGE**  
This error is produced when the instance to be modified does not exist in the local cache.
- **INVALID\_CREATION**  
This error is produced when an instance is created though it already exists in the local cache.
- **INVALID\_DELETION**  
This error is produced when the instance to be deleted does not exist in the local cache.
- **INVALID\_INITIALIZATION**  
This error occurs when there is an attempt to initialize an instance which is already known by the local cache, whatever the instance status may be (READ, CREATED, MODIFIED ; DELETED).

- **INVALID\_INSTANCE**  
This error is produced when a primary key of the current instance is not valid.
- **LENGTH\_ERROR**  
This error is produced when the length of the value of a Data Element belonging to the current instance exceeds the authorized length for this Data Element.
- **PARENT\_INSTANCE\_MISSING**  
This error is produced when a depending node instance is selected though the parent instance does not exist.
- **REFERENCE\_USER\_CONTEXT\_LENGTH\_ERROR**  
This error occurs when the length of the content of a user buffer Data Element associated with a reference node exceeds the authorized length for this Data Element.
- **REFERING\_INSTANCE\_MISSING**  
This error is produced when the `transferReference` method does not find any instance in the `detail` of the referring node.
- **SERVER\_UPDATE\_REQUIRED**  
This error is produced when a method is applied to an instance whereas the parent instance created locally does not exist in the database yet. A server update is previously required for the parent instance.
- **SUBSCHEMA\_ERROR**  
This error is produced when a Data Element belonging to the current instance is updated whereas it was not filled in for this instance on a server access parameterized with a sub-schema.
- **UNKNOWN\_ASYNCHRONOUS\_REQUEST\_ID**  
This error occurs when a response retrieval request is sent and when the response identifier is not recognized or has expired.
- **UNKNOWN\_INSTANCE**  
This error is produced when the selected instance is not known by the local cache.
- **VALUE\_ERROR**  
This error is produced when the contents of a Data Element belonging to the current instance are not valid.
- **VALUE\_REQUIRED**  
This error is produced when the contents of a Data Element belonging to the current instance are considered to be absent whereas they are required.

☞ These error messages correspond to constants of the `com.ibm.vap.generic.LocalException` class and represent types of errors. Prefixed with `LOCAL_`, each constant makes up an error key. This error key allows to identify the associated label in the local error labels file `vaperror.properties`.


### 5.4.3 Server Errors

Server errors produce the `com.ibm.vap.generic.ServerException` exception.

This exception is raised upon the receiving of a logical error message detected by the server. The exception holds the key and associated message.

#### 5.4.3.1 Structure of the Error Key

You must know the error key if you wish to display customized labels in the Client component.

 For details on how the access keys to the local labels of server errors are obtained, refer to the *Reference Manual - Graphic Clients: Public Interface of Generated Components*.

| Col 1-3 | Col 4-9 | Col 10-13 | Col 14-19 | Col 20 | Col 21 | Col 22-25  |                               |
|---------|---------|-----------|-----------|--------|--------|------------|-------------------------------|
| lib     | ser     | seg       |           |        | E      | DUPL       | Invalid creation              |
| lib     | ser     | seg       |           |        | E      | NFND       | Invalid deletion/modification |
| lib     | ser     |           |           |        | E      | Error code | User error                    |
| lib     | ser     | vie       | dte       | 2      | E      |            | Required Data Element         |
| lib     | ser     | vie       | dte       | 5      | E      |            | Value error                   |
| lib     | ser     |           | LOCKED    |        | E      |            | Already locked instance       |
| lib     | ser     |           | NTLOCK    |        | E      |            | Instance not locked           |

#### Legend:


lib = library code                      vie = Logical View code                      ser = server code  
dte = Data Element code                seg = physical access segment code  
E = Exception (Server error)

### 5.4.4 System Errors

The system errors (physical errors) produce a `com.ibm.vap.generic.SystemError` error. This type of error represents an internal and irretrievable error.

#### 5.4.4.1 Structure of the Error Key

You must know the error key if you wish to display customized labels in the Client component.

 For details on how the access keys to the local labels of system errors are obtained, refer to the *Reference Manual - Graphic Clients: Public Interface of Generated Components*.

**5.4.4.1.1 System Errors Received from the Business Component**

| Col 1-3 | Col 4-9 | Col 10-13 | Col 14-19 | Col 20 | Col 21 | Col 22-25 |                          |
|---------|---------|-----------|-----------|--------|--------|-----------|--------------------------|
| lib     | ser     |           | LKABSC    |        | S      |           | No timestamp set on Lock |
| lib     | ser     | vie       | ACCESS    |        | S      |           | Data access error        |
| lib     | ser     | STRU      |           |        | S      |           | View structure error     |
| lib     | ser     | VERS      |           |        | S      |           | Version error            |
| lib     | ser     | VIEW      |           |        | S      |           | Unknown View             |
| lib     | ser     | SERV      |           |        | S      |           | Unknown Service          |
| lib     | ser     | LTH       |           |        | S      |           | View length error        |
|         |         |           | MISPCV    |        | S      |           | Components out of phase  |

*Legend:*

lib = library code                      vie = Logical View code                      ser = server code  
 dte = Data Element code              seg = physical access segment code  
 [S] = system error

The type of error **Unknown service** is displayed when the service requested by the Proxy is not recognized by the Business Component.

The type of error **View length error** is displayed when the format of a Logical View associated with the Proxy changes and when this Proxy cannot be regenerated.

To solve this problem, you must regenerate the Proxy.

The type of error **Components out of phase** occurs when the Client and Business Components are out of phase.

To solve this problem, you must regenerate the Proxy.

**5.4.4.1.2 System Errors Received from the Communications Monitor**

| Col 1-3 | Col 4-9 | Col 10-13 | Col 21 |   |
|---------|---------|-----------|--------|---|
| lib     | mon     | LSRV      | S      | Erroneous length of the message received                          |
| lib     | mon     | PNUM      | S      | Erroneous structure of the "service number" parameter             |
| lib     | mon     | PCOD      | S      | Erroneous structure of the "service code" parameter               |
| lib     | mon     | PNOD      | S      | Unknown Folder code   |
| lib     | mon     | PCVS      | S      | Erroneous structure of a service request from the client          |
| lib     | mon     | PCVF      | S      | Erroneous structure of the message from the client                |
| lib     | mon     | TAND      | S      | Tandem/Pathway error  |
| lib     | mon     | WF00      | S      | Erroneous access to the work file or to the database (open/close) |

*Legend*    lib = library code                      mon = Communications Monitor code                      S = System error

### 5.4.4.1.3 System Errors Received from the Services Manager

Most of these errors are internal errors that you can solve only by contacting the VisualAge Pacbase support.

| Col 1-3 | Col 4-9 | Col 10-13 | Col 21 |   |
|---------|---------|-----------|--------|---|
| lib     | serv    | SRV1      | S      | Service not found in the work file  |
| lib     | serv    | LNG1      | S      | Erroneous conversion of service length on a multi-messages request                    |
| lib     | serv    | LNG2      | S      | Erroneous conversion of service length on a single-message request                    |
| lib     | serv    | NOS1      | S      | Erroneous structure of the "service number" parameter                                 |
| lib     | serv    | NOS2      | S      | Erroneous conversion of the "service number" parameter                                |
| lib     | serv    | SRV1      | S      | Erroneous structure of the "service code" parameter                                   |
| lib     | serv    | SRV2      | S      | Unknown service code in the Folder  |
| lib     | serv    | DOS1      | S      | Missing "Folder name" parameter   |
| lib     | serv    | DOS2      | S      | Erroneous length of the "Folder name" parameter                                       |
| lib     | serv    | VER2      | S      | Erroneous length of the "version number" parameter                                    |
| lib     | serv    | NOD1      | S      | Missing "node name" parameter   |
| lib     | serv    | NOD2      | S      | Erroneous length of the "node name" parameter   |
| lib     | serv    | NOD3      | S      | Unknown node name in the Folder   |
| lib     | serv    | TYNO      | S      | Unauthorized service on the node  |
| lib     | serv    | SCH1      | S      | Erroneous structure of the "sub-schema code" parameter                                |
| lib     | serv    | SCH2      | S      | Unknown sub-schema code in the Folder   |
| lib     | serv    | NOC1      | S      | Erroneous structure of the "number of occurrences" parameter                          |
| lib     | serv    | NOC2      | S      | Erroneous length of the " number of occurrences" parameter                            |
| lib     | serv    | NOC3      | S      | Erroneous conversion of the "number of occurrences" parameter "                       |
| lib     | serv    | EXT1      | S      | Erroneous structure of the "extraction method" parameter                              |
| lib     | serv    | EXT2      | S      | Unknown extraction method in the Folder   |
| lib     | serv    | USR1      | S      | Missing "User service" parameter  |
| lib     | serv    | USR2      | S      | Erroneous length of the "user service" parameter                                      |
| lib     | serv    | USR3      | S      | Unknown user service in the Folder  |
| lib     | serv    | CHK1      | S      | Missing "Check Option" parameter  |
| lib     | serv    | CHK2      | S      | Erroneous length of the "Check Option" parameter                                      |
| lib     | serv    | RFH1      | S      | Missing "Refresh Option" parameter  |
| lib     | serv    | RFH2      | S      | Erroneous length of the "Refresh Option" parameter                                    |
| lib     | serv    | LCK1      | S      | Missing "Lock Timestamp" parameter  |
| lib     | serv    | LCK2      | S      | Erroneous length of the "Lock Timestamp" parameter                                    |
| lib     | serv    | PCV1      | S      | Erroneous structure of the "Selection Criteria" parameter                             |
| lib     | serv    | PC01      | S      | Erroneous conversion of the "Selection Criteria" parameter                            |
| lib     | serv    | PILO      | S      | Erroneous access to the main record of the work file                                  |
| lib     | serv    | BUF1      | S      | Erroneous structure of user buffer  |
| lib     | serv    | PC02      | S      | Erroneous conversion of a field of the user buffer                                    |
| lib     | serv    | FRWR      | S      | Erroneous write access to the work file   |
| lib     | serv    | FRW2      | S      | Erroneous writing of the last record of the work file                                 |
| lib     | serv    | FRRE      | S      | Erroneous read access to the work file before update                                  |
| lib     | serv    | FRRD      | S      | Erroneous read access to the work file  |
| lib     | serv    | FRRW      | S      | Erroneous update access to the work file  |
| lib     | serv    | CP01      | S      | Erroneous structure of a "Selection Criteria" field from the Business Component       |
| lib     | serv    | CP02      | S      | Erroneous structure of a field of a Logical View instance from the Business Component |



| Col 1-3 | Col 4-9 | Col 10-13 | Col 21 |   |
|---------|---------|-----------|--------|---|
| lib     | serv    | ERKY      | S      | Erroneous structure of the "Selt Message" key from the Business Component                         |
| lib     | serv    | ERLA      | S      | Erroneous structure of the "Selt Message" label from the Business Component                       |
| lib     | serv    | PCV!      | S      | Erroneous structure of a field of the user buffer from the Business Component                     |
| lib     | serv    | PCV3      | S      | Erroneous structure of a field of a Logical View instance from the client                         |
| lib     | serv    | PC03      | S      | Erroneous conversion of a field of a Logical View instance from the client                        |
| lib     | serv    | PCV4      | S      | Erroneous structure of a "Selection Criteria" field from the client                               |
| lib     | serv    | PC05      | S      | Erroneous conversion of a "Selection Criteria" field from the client                              |
| lib     | serv    | NUVE      | S      | Erroneous "version number" parameter in the elementary Business Component                         |
| lib     | serv    | VIEW      | S      | Erroneous "Logical View code" parameter in the elementary Business Component                      |
| lib     | serv    | SERV      | S      | Erroneous "operation code" parameter in the elementary Business Component                         |
| lib     | serv    | LTH       | S      | Erroneous "length" parameter in the elementary Business Component                                 |
| lib     | serv    | PCV5      | S      | Erroneous structure of the action code of a Logical View instance to be updated                   |
| lib     | serv    | PCV6      | S      | Erroneous structure of a field of a Logical View instance to be updated from the client           |
| lib     | serv    | PCV7      | S      | Erroneous structure of the presence indicator of a field of a Logical View instance to be updated |
| lib     | serv    | PC06      | S      | Erroneous conversion of a field of a Logical View instance to be updated                          |
| lib     | serv    | ERK1      | S      | Erroneous structure of the user error message key   |
| lib     | serv    | ERL1      | S      | Erroneous structure of the user error message label   |
| lib     | serv    | OCNB      | S      | Erroneous structure, in the user error message, of the field code which bears the error           |
| lib     | serv    | DANA      | S      | Erroneous structure of the erroneous field code in the user error message                         |
| lib     | serv    | PCVS      | S      | Erroneous structure of a service request from the client  |
| lib     | serv    | PCVF      | S      | Erroneous structure of the message from the client  |
| lib     | serv    | WF00      | S      | Erroneous access to the work file   |
| lib     | ges     | TAND      | S      | Tandem/Pathway error  |
| lib     | serv    | STRU      | S      | Erroneous "structure" parameter in the elementary Business Component                              |

*Legend*    lib = library code                      serv = Services Manager code                      S = system error

### 5.4.5 Communication Errors

The communication errors with the Server produces the `com.ibm.vap.generic.CommunicationError` error. Like any Java error, VisualAge Pacbase communication error returns no key. To know the cause of the error, you must retrieve the message associated with the error (`getMessage()` method of the class).

#### 5.4.5.1 List of Errors

If a communication error message is displayed, inform the person in charge of the communication because the line may be blocked or defective, or a Server may be busy, etc.

Three communication error messages are likely to be displayed:

- `Open server error`
- `Call server error`
- `Close server error`

## 5.4.6 Example of Error Management

### 5.4.6.1 Introduction

The Java code of this example is provided with the Java version of Pacbench C/S. It shows a method allowing to manage all the types of errors that are likely to be raised by the Proxies. To do so, three classes have been defined:

- **StandardErrorMessageFactory**: class allowing to create a vector of **StandardErrorMessage** from the exception raised
- **StandardErrorMessage**: class unifying the characteristics of the various types of errors
- **ErrorManagerExample**: graphic class used to display the errors

### 5.4.6.2 Presentation of the non-visual classes in use

- **StandardErrorMessageFactory** class

This class provides the **public static java.util.Vector getStandardErrorMessages (Throwable th)** method allowing to create a vector of **StandardErrorMessage** from a given exception whatever its type may be.

- **StandardErrorMessage** class

Each **StandardErrorMessage** object has a number of properties that are initialized or not according to the error type (local, server, system, communication error, etc.) :

- the **hierarchical Proxy associated with the error** if the error is related to a particular instance of Logical View.
- the **error key** ( for local, server or system errors):  
It is a character string returned by the server, in the case of a server or system error.  
In the case of a local error, it is a character string corresponding to the name of the constant associated with the error type and defined in the **LocalException** class, prefixed with **LOCAL\_**.  
  - ☞ For the list of all local error types, refer to section **5.4.2**.
  - ☞ To know the error keys corresponding to the server or system errors, refer respectively to the sections **5.4.3, Server Errors** and **5.4.4, System Errors**.
- the **local label of the error**:
  - ♦ in the case of a local, server, or system error, the error key is interpreted by the Client component to get the local label. In the **vaperror.properties** file, a correspondence table allows to determine this label from a given error key.  
    - ☞ For information on the **vaperror.properties** file, refer to the *Graphic Clients Reference Manual: Public Interface of Generated Components*.
  - ♦ for the other error types, the local label corresponds directly to the message associated with the Java exception..
- the **server label of the error**, for the server and system errors, if a error label server has been coded using the Business Logic function.
- the boolean property **Restorable**. This property is true:
  - ♦ when a Proxy is associated with the error.

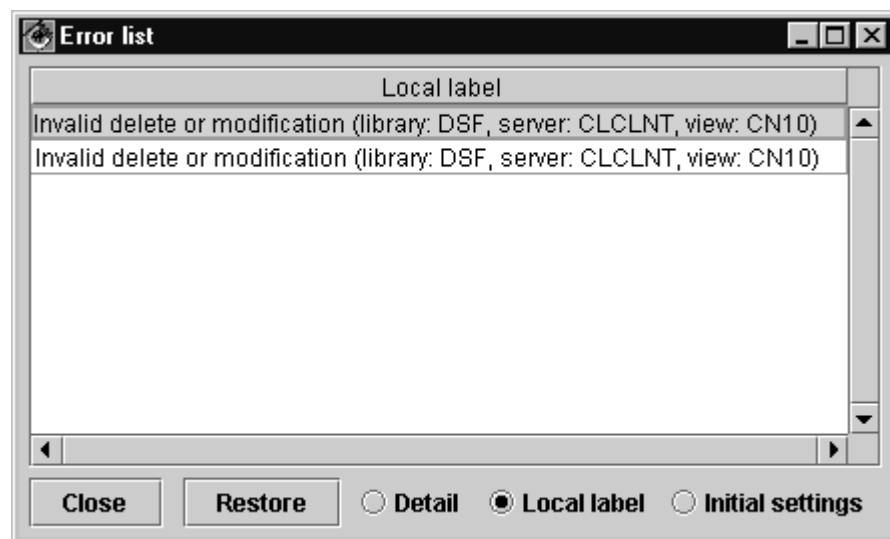
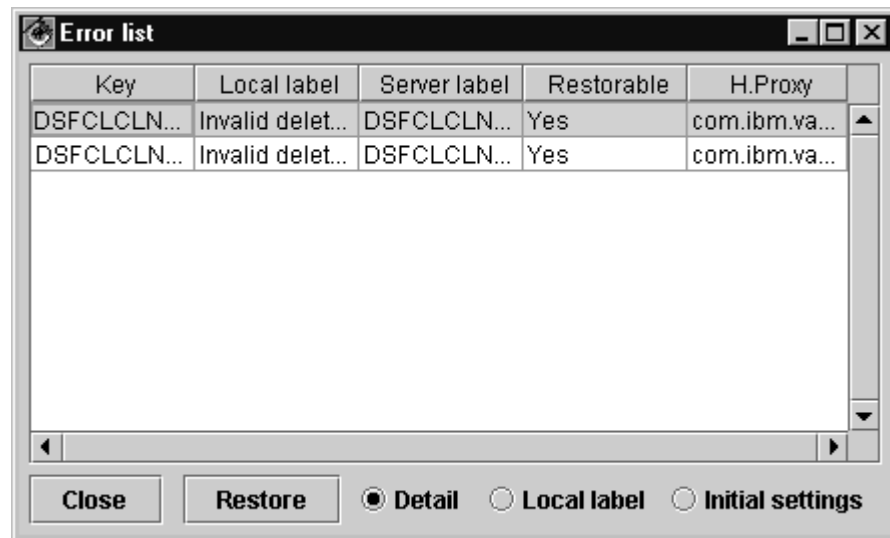
- ◆ when it is possible to display again the instance that caused the error in the detail of the window calling the Proxy.



For additional details on error management, refer to the *Graphic Clients Reference Manual: Public Interface of Generated Components*.

### 5.4.6.3 Presentation of the ErrorM anagerExample visual class

#### 5.4.6.3.1 Graphic interface



#### 5.4.6.3.2 Class functionalities

This class is used to display a list of error messages corresponding to instances of the `StandardErrorMessage` class.

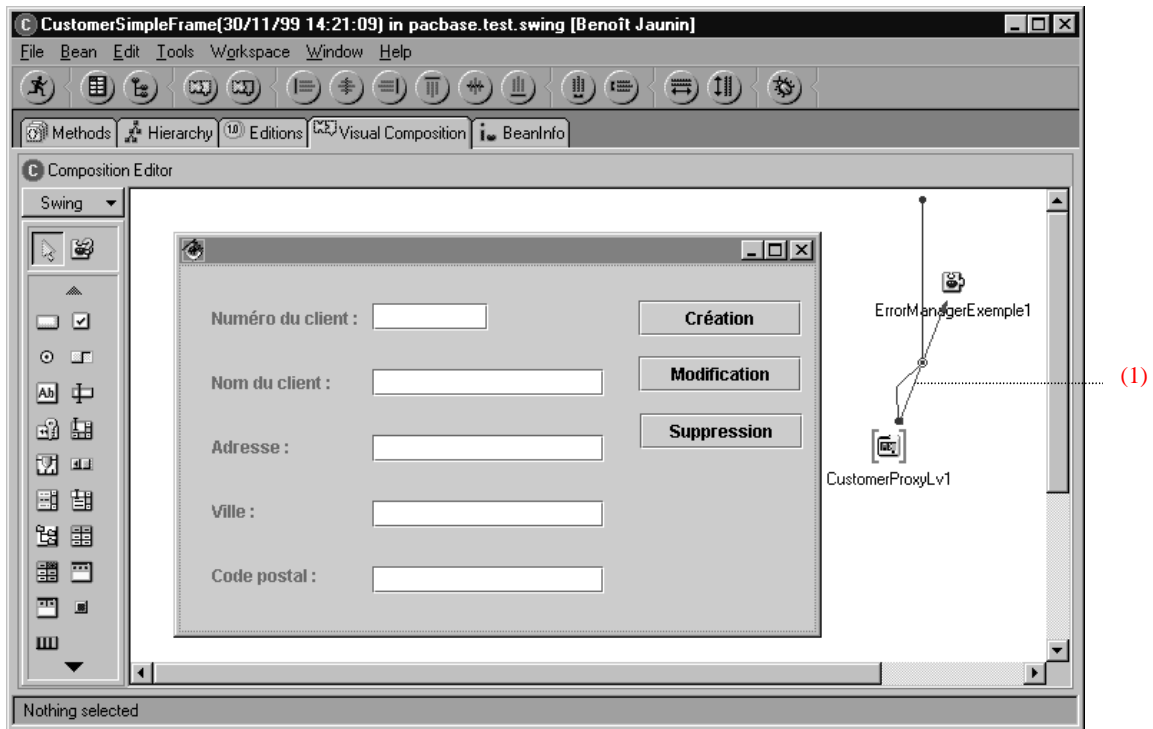
All the graphic controls used by this window are proportional to its size.

The window's functionalities are the following:

- The instances of `StandardErrorMessage` inferred by the `StandardErrorMessageFactory` class are transferred to this window in a vector using the `addStandardErrorMessages(Vector)` method. As long as the window is not closed, the transferred messages are added to the messages already displayed. The messages displayed can be deleted using the `resetCurrentStandardErrorMessages()` method.
- The properties of `StandardErrorMessage` instances that can be viewed are defined in the `visibleColumns` property of the `ErrorManagerExample` window. This property corresponds to a table of `int` whose values can be the following:
  - `1` to display the error message key
  - `2` to display the local label of the error message
  - `3` to display the server label of the error message
  - `4` to display the restoration status of the error
  - `5` to display the name of the class associated with the hierarchical Proxy
- Three radio-buttons have been developed to select dynamically different presentations for the list of `StandardErrorMessage` instances:
  - `Detail` is used to display all the properties of each `StandardErrorMessage` instance.
  - `Local label` is used to display only the local label of the `StandardErrorMessage` instance.
  - `Initial settings` is used to display the properties defined in the `visibleColumns` attribute.
- The `Restore` button is enabled when an instance of `StandardErrorMessage` is selected and if the error context can be restored. When the user clicks on the button, the error context is restored and the window that manages the Logical View instance that caused the error is displayed.

To implement this functionality, the `ErrorManagerExample` window must know each window managing the `detail` property of a Proxy node using the `addProxyManagingbyWindow` method. This method takes as parameters the Proxy node (`HierarchicalProxyLv`) and the error management window (`Jframe`).

Example



This example illustrates the principle allowing to record the association of a hierarchical Proxy and the window that calls it, and inform the error management window of this association.

The link (1) connects the event **this** of the Proxy to the method **addProxyManagingbyWindow** of the error management window (**ErrorManagerExemple1**). In this context, this method will be executed on the Proxy instantiation.

The two other links allow to transfer the hierarchical Proxy instance (**this** paramater of the Proxy to the **hp** parameter of the connection) and the window instance (**this** parameter of **CustomerSimpleFrame** to the **wi** parameter of the connection).

#### 5.4.6.4 Code for displaying the error window

The solution proposed is adapted to applications developed in VisualAge Java with the visual composition editor.

It assumes that the error management window (**ErrorManagerExample** class) has been inserted as a *class*-type or *variable*-type bean in the visual composition editor containing the window that calls the error management window.

The exceptions associated with the calling window are handled by inserting in the method **handleException( Throwable )** of the window, the following:

```
private void handleException(Throwable exception) {
    java.util.Vector standardErrorMessages =
    pacbase.test.swing.ErrorManager.StandardErrorMessageFactory.getStandardErrorMessages(exception);
    if (standardErrorMessages != null) {
        if (standardErrorMessages.size() >= 0) {
            getErrorManagerExemple1().addStandardErrorMessages(standardErrorMessages);
            getErrorManagerExemple1().showStandardErrorMessages();
        }
    }
}
```

## 5.5 Communication Management

### 5.5.1 Middleware Components

Pacbench C/S middleware corresponds to a C and C++ application whose functions are distributed in three types of DLLs:

- The first type groups together the interfacing functions with C and C++ languages and the interpreting functions of the middleware and switching services to libraries of specialized functions for a specific Communication protocol.
- The second type groups together the functions for the execution of Pacbench C/S middleware services for a specific Communication protocol.
- The third type groups together functions for managing resources such as the translation in natural language of the encountered exceptions.

The OS/2 version of Pacbench C/S middleware also uses a run-time materialized by a DLL coded **CSOOM30**.

| <b>DLL name</b> | <b>Function</b> | <b>Platform</b>             | <b>Protocol</b>             |
|-----------------|-----------------|-----------------------------|-----------------------------|
| ixomwarp.dll    | Type-1          | Windows 95/NT<br>OS2        | All                         |
| ixomware.dll    | Type-1          | Windows 95/NT<br>OS2, UNIX  | All                         |
| ixocics.dll     | Type-2          | Windows 95/NT<br>OS2        | CICS-ECI                    |
| ixocpic.dll     | Type-2          | Windows 95/NT<br>OS2        | CPI-C                       |
| ixotux.dll      | Type-2          | Windows 95/NT<br>UNIX       | TUXEDO                      |
| ixosock.dll     | Type-2          | Windows 95/NT<br>OS2, UNIX  | TCP-IP Socket<br>VA Pacbase |
| ixoloc.dll      | Type-2          | Windows 95/NT<br>OS2        | Call of COBOL<br>DLL        |
| ixomqs.dll      | Type-2          | Windows 95/NT<br>OS2, UNIX  | MQSERIES                    |
| ixotmvs.dll     | Type 2          | Windows 95/NT,<br>OS2, UNIX | TCP-IP Socket<br>CICS       |
| ixotcis.dll     | Type 2          | Windows 95/NT,<br>OS2, UNIX | TCP-IP Socket<br>TCIS       |
| ixomsgen.dll    | Type-3          | Windows 95/NT<br>OS2        | All                         |
| csoom30.dll     | runtime C++     | OS2                         | All                         |

Windows(\*) = all Windows platforms

### 5.5.2 Processing a Request

In VisualAge, the middleware services are executed from a set of specific communication classes provided at the installation of the product.

The communication with the Server is executed via the **ServerAdapter** interface. There are two ways of implementing this interface:

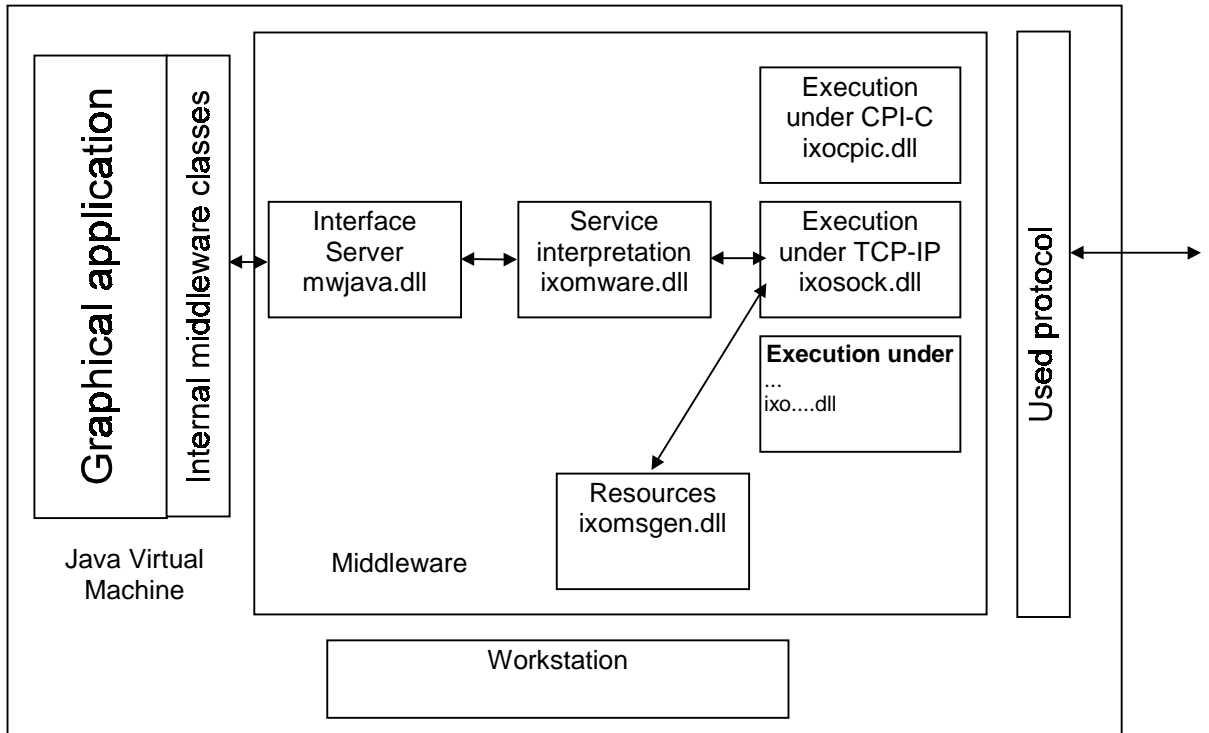
- Implementing the interface so that it allows direct access to the C++ DLL of the middleware. This type of implementation is dedicated to standard applications.

The DLL must be installed in the path.

- Implementing the interface by using a Gateway or a Relay via TCP/IP. This type of implementation is dedicated to applets.

**5.5.2.1 Direct Access to the Middleware or Local Mode**

**5.5.2.1.1 Diagram of the Processing of a Request**



**5.5.2.1.2 Instantiating a Folder in Local Mode**

To instantiate a Folder in local mode, you can use the default builder `CustomerProxyLv()`

An example:

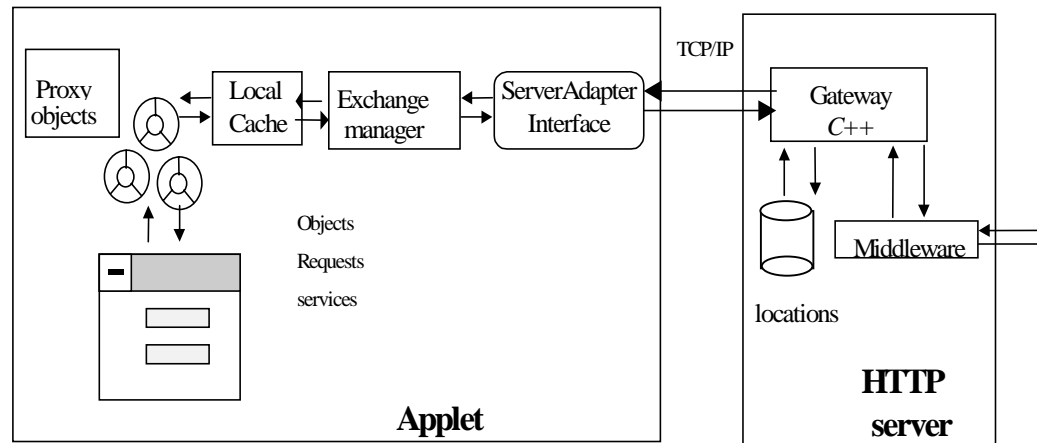
```
myProxy = new CustomerProxyLv() ;
myproxy.setLocationsFile("c:\vap\gen\java\VAPLOCAT.INI") ;
```

The second instruction specifies which location file must be used. By default, we search for a `VAPLOCAT.INI` file located in the current directory.



## 5.5.2.2 Access via a Gateway

### 5.5.2.2.1 Schema of a Request Processing



### 5.5.2.2.2 Using the Gateway and the Relay

The gateway and the relay are used by the Proxy objects to access the Folder from a VisualAge for Java applet.

#### • Using the Gateway

The gateway program is used to create accesses to the middleware.

It is started with the `gateway.exe` command with the following syntax:

```
gateway [-h] -s|i|d TCP_PORT_NUM [-l LOCATION_FILE] [-t[TRACE_LEVEL]]
[-c CHAR_CONV_FILE] [-pcv]
```

where:

- `-h[e|p]` : option used to display the syntax
- `-s` : starts the gateway
- `-i` : installs the gateway as a Windows NT service

By default, the gateway is installed in automatic startup mode.

- `-d` : uninstalls the gateway from the Windows NT services
- `TCP_PORT_NUM`: sets the communication port used by the Proxy objects.
- `-l LOCATION_FILE` : indicates the *locations file*.  
Default: « `.\VAPLOCAT.INI` ».
- `-t[TRACE_LEVEL]`: set the trace level.
  - ♦ 0: mute (default)
  - ♦ 1: errors
  - ♦ 2: standard
  - ♦ >2: debug
- `-c CHAR_CONV_FILE` : specifies the page code conversion file
- `-pcv`: Optimization of the messages length for the PCV. Through the gateway, this option is required.

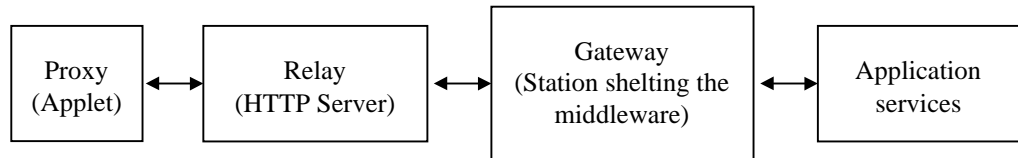
For example :

```
gateway -s 6001 -l c:\vap\generated\java\VAPLOCAT.INI -t1 -pcv
```

- **Using the Relay**

The relay is used to relay the requests arriving to an HTTP server to another station on which the middleware and the gateway are installed.

The following diagram is a standard use:



It can also be used for the Internet. You must execute it with the **relay** command with the following syntax:

```
relay [-h] [CLIENT_PORT] [-t[TRACE_LEVEL]] -server HOST[:PORT]
```

where:

- **-h[elp]**: is the option used to display the syntax
- **[CLIENT\_PORT]**: sets the communication port used by the Proxy objects. Default 5647
- **-t[TRACE\_LEVEL]**: trace level (these values are the same as those of the gateway)
- **-server HOST[:PORT]**: the gateway/relay address to be relayed.

For example :

```
relay 5647 -t1 -server pc12:6001
```

### 5.5.2.2.3 Instantiation of a Folder via the Gateway

Two builders can be used in this case:

- **CustomerProxyLv(String host)**

For example: `new CustomerProxyLv("9.134.5.146")`

- **CustomerProxyLv(String host, int port)**

For example: `new CustomerProxyLv("9.134.5.146", 6001)`

☞ If **host** is equivalent to null, we consider that it is a local middleware.

### 5.5.2.3 Dynamic Change of the Middleware Access Parameters

Different methods of the Root Proxy allow dynamic changes of these parameters:

- **void setHost(String host)**

☞ If **host** is equivalent to null, we consider that it is a local middleware.

- **void setPort(int port)**
- **void setLocationsFile(String filename)** (local mode)
- **void setTraceFile(String filename)**
- **void setTraceLevel(int traceLevel)** (local mode)

### 5.5.3 Definition of the Use Context

The communication management requires the definition of the middleware use context. This context is defined in a specific file where one or more locations are specified for each generated Folder.

A location is used to specify the elements which enable data exchanges between the client applications and the services managers, during the development phase of the client application or during the operation phase. For example, a client application can address a server without any tracepoint for an operating version.

For a folder, a location consists of:

- The location identifier. This identifier is case-sensitive
- The external name of the communications monitor
- The maximum physical length of an exchange message
- A variable number of parameters which make the *middleware* work (IP address, trace, etc.)

The locations are specified in the **VAPLOCAT.INI** file (this file name in upper case is reserved).

#### 5.5.3.1 Structure of the VAPLOCAT.INI File

The locations file is structured in sections and subsections.

Each section corresponds to a Folder and is identified by the Folder code, delimited by [...].

Each subsection corresponds with a location and is identified by the environment name specified for the **LOCATION** option on the Folder **General Documentation** screen. The subsection identifier is delimited by <...>.

Within a subsection, the parameters are identified by keywords.

Recognized keywords are as follows:

| <b>keyword</b>  | <b>Meaning</b>  |
|-----------------|---|
| COMMENT         | Comments  |
| LENGTH          | Physical message length (generated)   |
| MONITOR         | Communications monitor external name (generated)  |
| MWADDRESS       | Server's address (for SOCKET, TCPMVS, TCIS, TCPIMSI and TCPIMSE).<br>Required for these protocols<br><br>The following formats are possible:<br>- Decimal coding:<br>* IP address<br>* Space<br>* Port number, in decimal coding<br>- Hexadecimal coding:<br>* [1, 2]: '0x'<br>* [3, 6]: AF_INET domain<br>* [7,10]: port number<br>* [11, 18]: IP address, in hexadecimal coding                           |
| MWARE           | Communication protocol<br><b>Values :</b><br>TUXEDO<br>CICS<br>LOCAL<br>CPIC<br>SOCKET<br>TCIS<br>MQSERIES<br>TCPMVS<br>CPICXCP2<br><b>Protocols</b><br>TUXEDO XA or non XA<br>CICS ECI extend or nonextend<br>Server on the Client workstation<br>CICS CPI-C/APPC ou IMS CPI-C/APPC<br>TCP-IP SOCKET for UNIX or Windows/NT<br>TCIS<br>MQSERIES<br>TCP-IP SOCKET for MVS/CICS<br>CPI-C/XCP2 for TDS or TP8 |
| MWBUFFERTYPE    | Buffer type ( for TUXEDO)<br><b>Values:</b><br>- CARRAY (default)<br>- FML  |
| MWCODEPAGE      | Server page code<br>- Optional  |
| MWFMLNAME       | FML file name ( for TUXEDO)<br>Caution: you need also to state the path in the FLDTBLDIR environment variable and the file name in the FIELDTBLS environment variable.  |
| MWMAXREPLY      | Maximum number of asynchronous answer-waiting requests<br>- Optional<br>- Numeric   |
| MWQUEUEMANAGER  | Queue manager name for asynchronous protocol (MQSERIES)<br>- Required for this protocol<br>- Alphanumeric<br>- 48 characters max.   |
| MWREPLYQUEUE    | Name of the reception-message queue manager for asynchronous protocol (MQSERIES)<br>- Required for this protocol<br>- Alphanumeric<br>- 48 characters max.  |
| MWREQUESTEXPIRY | Duration of an asynchronous request (in seconds)<br>- Optional (default: unlimited)<br>- Numeric  |

| keyword        | Meaning  |
|----------------|--|
| MWREQUESTQUEUE | Name of the emission-message queue manager for asynchronous protocol (MQSERIES)<br>- Required for this protocol<br>- Alphanumeric<br>- 48 characters max.  |
| MWTIMEOUT      | Timeout managed by the middleware (in seconds )<br>- Optional<br>- Numeric<br>- Value: [0, 32767]  |
| MWTRANSID      | Code of the transaction CICS on 4 characters:<br>- required for the MVS TCP-IP socket protocol<br>- optional for the CICS/ECI protocol <ul style="list-style-type: none"> <li>• If the MWTRANSID parameter does not exist or is not valued, the CPMI transaction is automatically executed under CICS. In this case, the MONITOR parameter is required and must correspond to a program running under CICS.</li> <li>• If the parameter is valued, the transaction code is executed under CICS for each request using this location. In this case, the value entered for the MONITOR parameter is not taken into account.</li> </ul> |

You only indicate a single parameter per file line.

Example of a location file:

```
[ FOCLNT ]
<Location1>
COMMENT=operation monitor
MONITOR=CLCOMM
LENGTH=8192
MWCODEPAGE=850
MWARE=SOCKET
MWADDRESS=0x00021770C0060A5D
<Location2>
COMMENT=monitor with trace
MONITOR=CLCOM2
LENGTH=8192
MWARE=SOCKET
MWADDRESS=0x00021770C0060A5D
<Location3>
COMMENT=local monitor
MONITOR=CLCOMM
LENGTH=8192
MWARE=LOCAL
```

### 5.5.3.2 Implementation

The `VAPLOCAT.INI` file is required in both development and operations phases.

If the file is missing or if the syntax used is incorrect, the Exchange Manager cannot be instantiated. The incorrect parameter is indicated in the trace file.

In the development phase, the Folder View Proxy generator is responsible for creating or updating the **VAPLOCAT.INI** file.

- If the **VAPLOCAT.INI** file does not exist, the generator creates it in the current VisualAge directory. For each location, the values of the **MONITOR** and **LENGTH** parameters (generated) are automatically set. The developer must specify the other parameters.
- If it already exists (in the VisualAge directory), it is automatically loaded. For each Folder, the generator compares the locations contained in the extraction file with the existing locations. The existing **VAPLOCAT.INI** file is updated accordingly:
  - ♦ the new locations are added to the existing file
  - ♦ the locations that are no longer in the new extraction file are automatically removed from the existing file
  - ♦ the existing locations are modified according to the modifications contained in the new extraction file.
 In case of an update, the parameters set by the developer remain unchanged.

## 5.6 Testing the Application Generated– Packaging

### 5.6.1 Testing the Generated Application

#### 5.6.1.1 Version Compatibility Check

If the version control option detects any discrepancy, it means that the Business Component and the Proxy object were not generated with the same version number. So you must:

- regenerate the Proxy object if you have regenerated only a new version of the Business Component,
- or implement, in VisualAge, the generated graphic application including the new proxy component if it has not already been done,
- or implement the generated Business Component if it has not already been done.



About the version compatibility check, refer to Subchapter [1.2](#).

#### 5.6.1.2 Optimized Middleware or Trace Mode

Pacbench Client/Seurveur middleware is available in optimized version or in trace version. The extensions of the files associated with the middleware's DLLs allow you to distinguish the two versions.

The files with the '001' extension corresponds to the DLLs used in TRACE mode. With the TRACE mode, you obtain a file containing all events, actions and objects handled by the middleware. This mode is activated from the two following environment variables:

- **IXOTRACE** :  
This variable enables to activate (**IXOTRACE=1**) or deactivate (**IXOTRACE=0**) the middleware API trace.
- **IXOTRACE\_FILE** :  
This variable enables to specify, when the trace is active (**IXOTRACE=1**), the API middleware trace.  
*exemple :* `IXOTRACE_FILE=c:\tmp\lixo_err.txt`

The file containing the events, actions and objects is never reinitialized by the middleware functions. We advise you to frequently destroy this file, as the middleware immediately creates a new one.

The files with the '002' extension corresponds to the optimized version of Pacbench Client/Server middleware DLLs. These DLLs do not contain the code allowing to trace the communication process and do not interpret the environment variables dedicated to this use.

At the installation of Pacbench Client/Server on the workstation, the DLLs ready to be interpreted correspond to those of the middleware in optimized version.

## 5.6.2 Packaging

Packaging an applet or an application is the process of implementing a developed application, on a Client workstation once it has been tested. It allows you to use your applet or application in another development environment, and especially in VisualAge. In VisualAge for Java, this process consists on exporting this applet or application.

### 5.6.2.1 Reminder: Prerequisites

In the operating phase, the **VAPLOCAT.INI** file must be located in the same directory as the final applications. The developer responsible for the installation of these applications must make sure of it.

- Applet

For the implementation of a Java applet, the following elements should be installed on the user's station:

- An HTTP server
- A 1.1 *enabled* Java Web browser
- The gateway and the relay are installed on the HTTP server station

- *Standalone* application

For the implementation of a Java *standalone* application, the Java Runtime Environment (JRE) should be installed on the user's workstation.



For more details on the execution environment, refer to the *Pacbench C/S User's guide, Vol. I – Concepts – Architectures – Environments*. Refer to subchapter 5.5 as well.

### 5.6.2.2 Export

#### 5.6.2.2.1 What will You Export ?

You must export all the execution classes used by the application which do not belong to the base classes. The edition classes such as the **BeanInfo** classes or the beans used for a quick mapping of Pacbase Data Element-type properties are optional.

As for a Pacbench C/S application, you must export:

- All the packages of the project which contains the Pacbench C/S runtime, except the packages `com.ibm.vap.beans` and/or `com.ibm.vap.beans.swing` (depending on the package used in your application). For those two packages, export only the classes actually used by your application.
- The project containing the generated Proxy components,
- The applet or application itself, that is to say the whole project containing the applet or the application or the package(s). In our example, it is the `vap.sample` package (for the V1 example) or `example.swing` (for the V2 example).
- possibly, the external beans: in the V1 example, we use the `ImulticolumnListbox` bean. Therefore, it is necessary to export the whole package, which is the `com.ibm.ivj.javabeans` package.

#### 5.6.2.2.2 Implementation

- **For an applet**

Before exporting an applet, you must create a directory in the HTTP server root, in which will be stored the export result. For example `c:\www\html\codebase`. This directory constitutes the Root of Java classes in the HTTP server.

☞ You can export the `class` files directly in the HTTP server tree structure, in `codebase` in our example, or in another directory. In this case, copy these files in the directory before implementing the applet, and respect the packages' tree structure.

- **For a *standalone* application**

In this case, the location of the directory in which the result of export will be stored does not matter. You just have to declare this directory in the `CLASSPATH` variable.

#### 5.6.2.2.3 How to export?

To export, follow these steps:

- In the VisualAge Workbench, select all the elements that you want to export,
- In the **File** menu, select the **Export** choice,
- In the **SmartGuide - Type of Export** window, select the **Class Files** option, and then click on **Next**,
- In the **SmartGuide - Export to files** window, enter the output directory name or select it with the **Browse** button, by using the **Create package subdirectories** option.



For more information, refer to the VisualAge for Java on-line documentation accessible from VisualAge on-line help or from Windows 95 or NT Explorer.

- From VisualAge  
In the **Help** menu, choose **Tasks**. The Internet browser opens: select the *Exporting to the file system* data element.



- From the Windows Explorer, open the **Tasks** directory, then select an HTML file in the **Export** director, the **overview.htm** file being a suggested entry point

#### **5.6.2.2.4 Optimizing the Downloading Time of the .class File**

For this option, the JDK should be installed in the execution environment.

Once the export has been executed in the form of **.class** files, you can optimize the downloading time of classes by saving all these files into a single archived file **.jar**.

In the DOS or OS/2 window, put your cursor in the export output directory, then enter the following command:

```
jar cvf sample.jar com vap
```

where:

- **sample.jar** is the name of the archived file,
- **com** and **vap** represent two directories which contain all the **.class** files required for the execution of the final application.

For an applet, the **.jar** file obtained must be copied in the HTTP server tree structure.

#### **5.6.2.2.5 Writing an HTML File (Applet Only)**

Finally, it is necessary to write an HTML file containing the applet to be able to execute it in a Web browser. This file is used to set some parameters, such as the applet width and length.

To apply this procedure to our example, you must create, in the **c:\www\html\codebase\vap\sample** directory, an **index.html** file containing the following text:

```
<HTML>
<TITLE>
Sample Applet
</TITLE>
<BODY>
<CENTER>
<APPLET code="vap.sample.SampleApplet.class"
WIDTH=1000" HEIGHT=1000
codebase="/codebase"></APPLET>
</CENTER>
</BODY>
</HTML>
```

or the following text, if you have created a **.jar** archived file:

```
<HTML>
<TITLE>
Sample Applet
</TITLE>
<BODY>
<CENTER>
<APPLET code="vap.sample.SampleApplet.class" WIDTH=1000
HEIGHT=1000
archive="/codebase/sample.jar"
codebase="/codebase"></APPLET>
</CENTER>
</BODY>
</HTML>
```

## 5.7 Application Deployment



To deploy an application, follow the operations described in VisualAge Java documentation.

But you must also install some files specific to the use of Pacbench C/S.

- **For an applet:**

The end-user workstation must be equipped with a browser.

- **For a *standalone* application:**

- If no gateway is used, you must install, on the end-user workstation:
  - ♦ **MWJAVA.DLL**,
  - ♦ the middleware DLLs,
  - ♦ **VAPLOCAT.INI**,
  - ♦ **CHARCONV.TXT** (conversion of page code),
  - ♦ **VAPRUN.JAR**,
  - ♦ **VAPSWING.JAR** if the application uses **Swing** or **VAPAWT.JAR** if the application uses **AWT**.
- If a gateway is used, you must install, on the station where the gateway is installed:
  - ♦ **GATEWAY.EXE**,
  - ♦ the middleware DLLs,
  - ♦ **VAPLOCAT.INI**,
  - ♦ **CHARCONV.TXT** (conversion of page code).

In this case, you must not install any of these files on the end-user workstation. However, on this workstation, you must install:

- ♦ **VAPSWING.JAR** if the application uses **Swing** or
- ♦ **VAPAWT.JAR** if the application uses **AWT**.

## 6 Developing a Client with VisualAge for Smalltalk

Once you have generated and imported the Proxy objects in the VisualAge workstation, you just need to integrate them into the graphic application.

After introducing you to the basic concepts of development, this chapter gives you a detailed description of a client development, including the following steps: insertion of Proxy objects with programming links involving actions, attributes and events, error management, communication management and test on the application.



To facilitate the development and reusability of clients implemented with VisualAge, we advise you to make the VisualAge application containing graphic parts (VisualAge Views or windows) coincide with a functional application. The VisualAge application is the packaging entity of a functional application.



For further information on the packaging, refer to section [6.6.2](#).

To avoid errors at the local backup, we advise you:

- To develop your functional application in any VisualAge application other than the application in which you have generated your Proxy objects.
- Before starting the local backup, make sure that the *manager*, *group member*, *owner* and *developer* in the VisualAge station corresponds to the same developer who is in charge of the application that will be saved.
- If you want to export the application to be saved in the Repository, in another environment, choose the *library supervisor* as *manager*, *group member*, *owner* and *developer*.



We advise you against developing a Smalltalk Client with Logical View Proxy objects with the current Pacbench Client/Server version because of its limited functionalities. Consequently, you will not find any information on the use of LVP in this Chapter.

However, if you want details on this development mode, refer to the *Pacbench C/S User's Guide: Graphic Clients 2.0*.

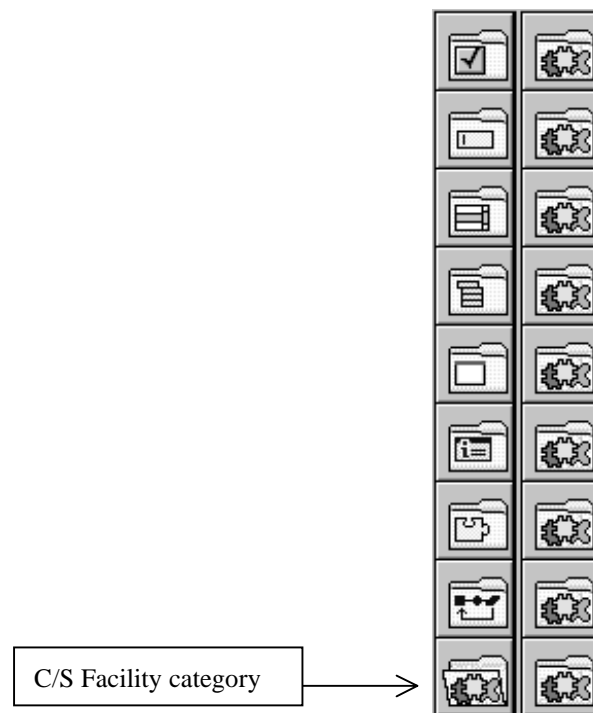
## 6.1 General Principles

- ☞ If your Proxy contains a single Elementary Proxy (a Root Proxy), the attributes, actions and events associated to large readings, reference or depending nodes are not available.

### 6.1.1 Using the Proxy Objects in the Composition Editor

Once imported in VisualAge, the Folder View Proxy can be used by a graphic part.

To place the Folder View Proxy on the Free Form Surface select the part corresponding to the FVP in the **Client/Server Facility** category of the parts palette.



- ☞ Once the Proxy is selected a message is displayed prompting you to confirm if you want to insert it as a variable or constant part.

**Note** In VisualAge, a constant part stands for a real objet. It can be instantiated only once. A variable part acts as a placeholder for this actual objet in another VisualAge View. It thus enables this objet to be reused and shared by several Views.

#### 6.1.1.1 Unfold Option in the Elementary Proxy Pop-up Menu

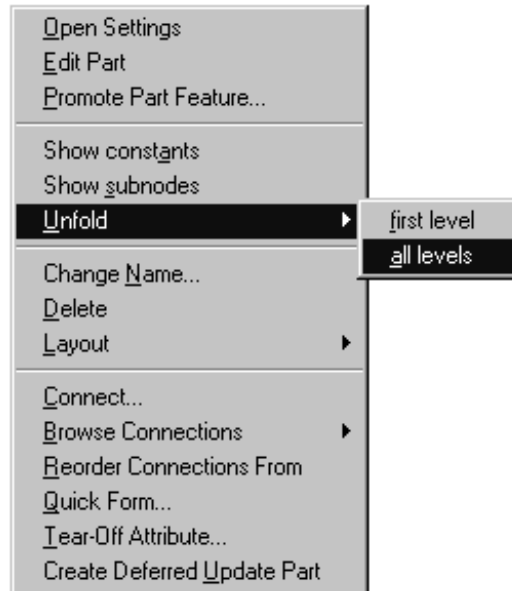
Now that your Folder View Proxy is on the Free Form Surface, you can access public interface of the Root Proxy. However, the child public interface of the Elementary Proxy is not accessible yet.

To make it accessible, you must unfold the child Elementary Proxies to make them visible on the Free Form Surface.

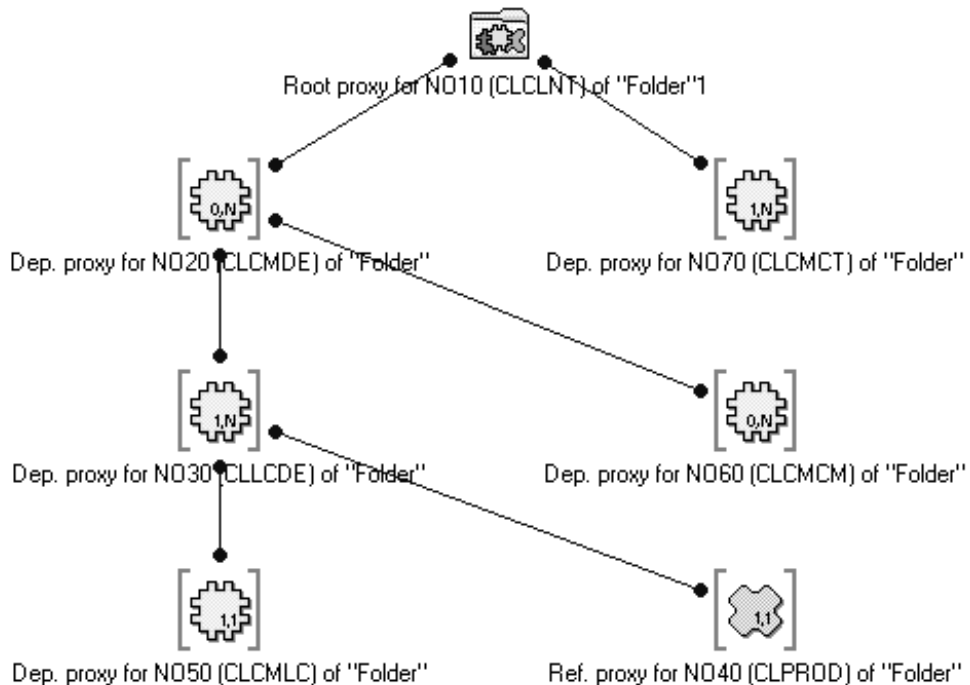
Proceed by displaying the pop-up menu for the Root Proxy and select the **Unfold** submenu which contains the following two choices:

- The **first level** choice is used to display only the level immediately below
- The **all levels** choice is used to unfold all the levels at once.

In the example below, we come back to the Folder View Proxy which we generated and imported in the previous chapter.










initiates:



You can also display all the levels using the **first level** option several times after unfolding each level.

The **Unfold** submenu is also available for any Elementary Proxy other than the Root, provided it has child Proxies.

Each Elementary Proxy in the Folder View Proxy is represented by a specific icon which depends on its type or on the cardinal value of the relation which links it to a parent Elementary Proxy.

| Icons   | Possible Types of Elementary Proxies |
|---|--------------------------------------|
|  | Root Proxy                           |
|  | Depending Proxy 0,N                  |
|  | Depending Proxy 0,1                  |
|  | Depending Proxy 1,N                  |
|  | Depending Proxy 1,1                  |
|  | Reference Proxy 0,1                  |
|  | Reference Proxy 1,1                  |

Each of our 7 Elementary Proxies corresponds, in the server, to a node, that is an elementary Business Component and the Logical View it manages.

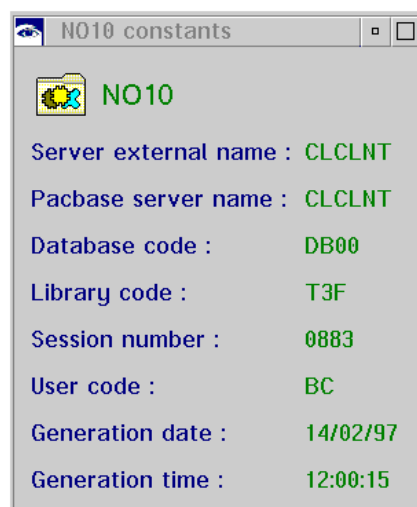
### 6.1.1.2 Generation Context

This generation context is that of the **GVC** generation of the associated Folder View.

You can display the generation context for each Elementary Proxy visible on the Free Form Surface.

To do so, display the pop-up menu for the Elementary Proxy and select the **Show constants** choice.

The following window opens:



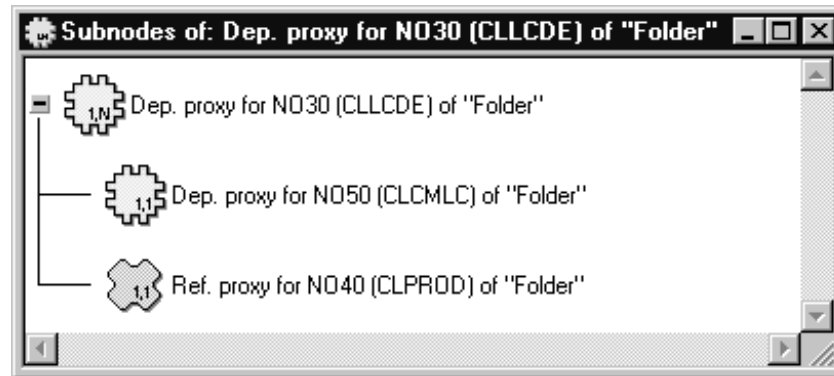
If the **Optimize Generated Code** generation option has been used, this information is not specified (instead « ? ? ? » is displayed).

### 6.1.1.3 Displaying Child Elementary Proxy

For each Elementary Proxy visible on the Free Form Surface, you can display its child Proxies without using the **Unfold** submenu: actually there is no need to unfold them if you do not have to deal with them.

Proceed by selecting the **Show subnodes** choice in the Elementary Proxy's pop-up menu.

The following window is displayed:



The window displays the child Proxy object(s). If one of the child Proxies itself contains other child Proxies, it is preceded by the + sign: click on it to display these other Proxies.

## 6.1.2 Use of Attributes

An attribute corresponds to a piece of information handled by a Proxy object. This piece of information defines an elementary data, a list of elementary data or a list of composite data instances. An attribute corresponds either to a constant, a parameter or an action result. According to the context, it is initialized by the GUI application or the Proxy.

Two kinds of attributes are found:

- those standing for technological variables. They enable to adjust the behavior of the Proxy objects in VisualAge
- Those corresponding to the Logical View data



The availability of an attribute depends on the Proxy type. All the public interface attributes are documented in the *Graphic Clients Reference Manual: Public Interface of Generated Components*.

### 6.1.2.1 Parameter Definition of Attributes: Settings Window

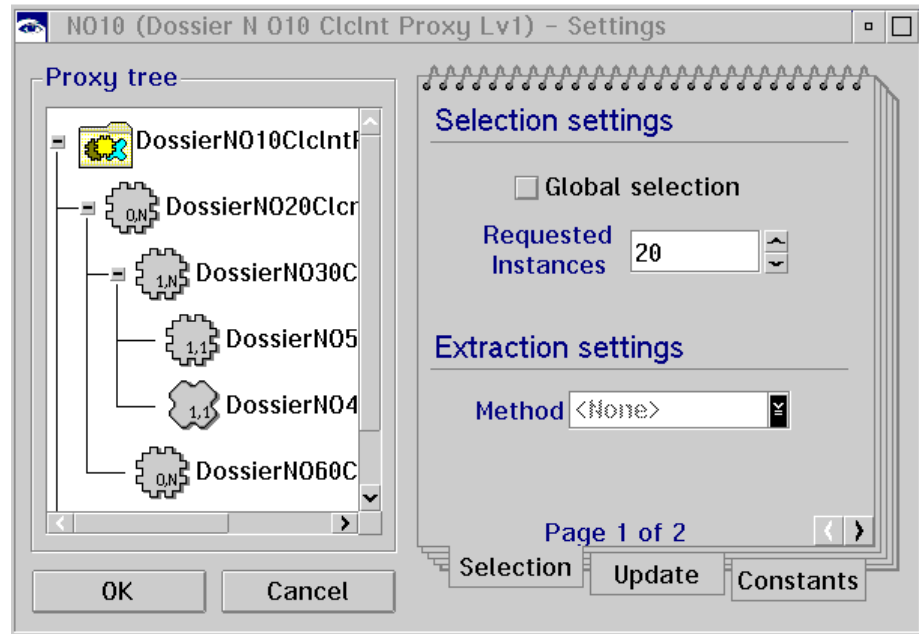
Once the Folder View Proxy has been placed on the Free Form Surface, before using the attributes associated with the Proxy objects, you can set the initial values for some technological attributes via the **Settings** window. These initial values will thus be the default values for these attributes. You will nevertheless be able to dynamically modify these values during development.

This parameter definition is not compulsory since a default parameter definition is provided which you can also dynamically modify later on.

To open the **Settings** window, select the **Open Settings** choice from the Root Proxy pop-up menu or double click on the corresponding icon.



The **Settings** is only available when the Root Proxy is used as a constant-type part.



• **Selecting a Proxy**

The **Proxy Tree** area presents the list of Proxy objects, which make up the Folder View Proxy in the form of a directory structure. It is used to select the Proxy whose default parameters you want to define or to view those parameters already defined.

The tree structure can contain only one Proxy. It corresponds necessarily to a Proxy of root type and as such, it behaves the same way and holds the same properties.

When you open the window, only the Root Proxy is displayed. The + sign which precedes the icon indicates the existence of child Proxies. Click on it or double-click on the parent Proxy to display them. To display the entire directory structure, proceed in the same way until there are no more + signs.

To retain only the levels which you are interested in, click on the - sign(s) or double-click on the desired Elementary Proxy object(s).

Select the Proxy you want.

The **Selection Settings** area will now display the default values for this Elementary Proxy.

• **Defining the Parameters for a Proxy**

The **Selection** and **Update** tabs in the notebook are used to specify the default parameters for the Elementary Proxy selected in the **Proxy Tree** area. Each parameter corresponds to a Proxy attribute.

- In the first page of the **Selection** tab, you:
  - ♦ Specify or view the number of occurrences to be read. The **Requested Instances** field corresponds to the **maximumNumberOfRequestedInstances** attribute. The default value is that specified on the server.
  - ♦ Indicate the desired extraction method. The **Method** field corresponds to the **extractMethodCode** attribute.
  - ♦ Activate or deactivate the global selection indicator for Proxy instances which are the result of a multi-instance reading action. The **Global Selection** field corresponds to the **globalSelectionIndicator** attribute.

The **Global Selection** and **Requested Instances** parameters are mutually exclusive.

- In the second page of the **Selection** tab, you indicate the location of the associated Folder. The **Server location** parameter corresponds to the **location** attribute.
- The **Update** tab includes three options:
  - ♦ **Refresh instances** used in the client to refresh the local image of instances updated by the server when certain Data Elements, such as identifiers, are calculated by the server. This option corresponds to the **refreshOption** attribute.
  - ♦ **Check validity on server** which runs a value range check and a user check on Data Elements for which these options have been set on the server. This option corresponds to the **serverCheckOption** attribute.
  - ♦ **local rows sort** which specifies the sort criterion of lists (**rows** attribute): local or server criterion. By default, the local sort criterion is selected. This option corresponds to the **localsort** attribute.



All of these attributes are dealt with in the *Graphic Clients: Public Interface of Generated Components Reference Manual*.

### • Viewing the Generation Context

The **Settings** window also includes the **Constants** tab used to view the generation context of the selected Elementary Proxy.



To view the generation context of an Elementary Proxy when the **Settings** window is not available, use the **Show constants** choice from its pop-up menu.



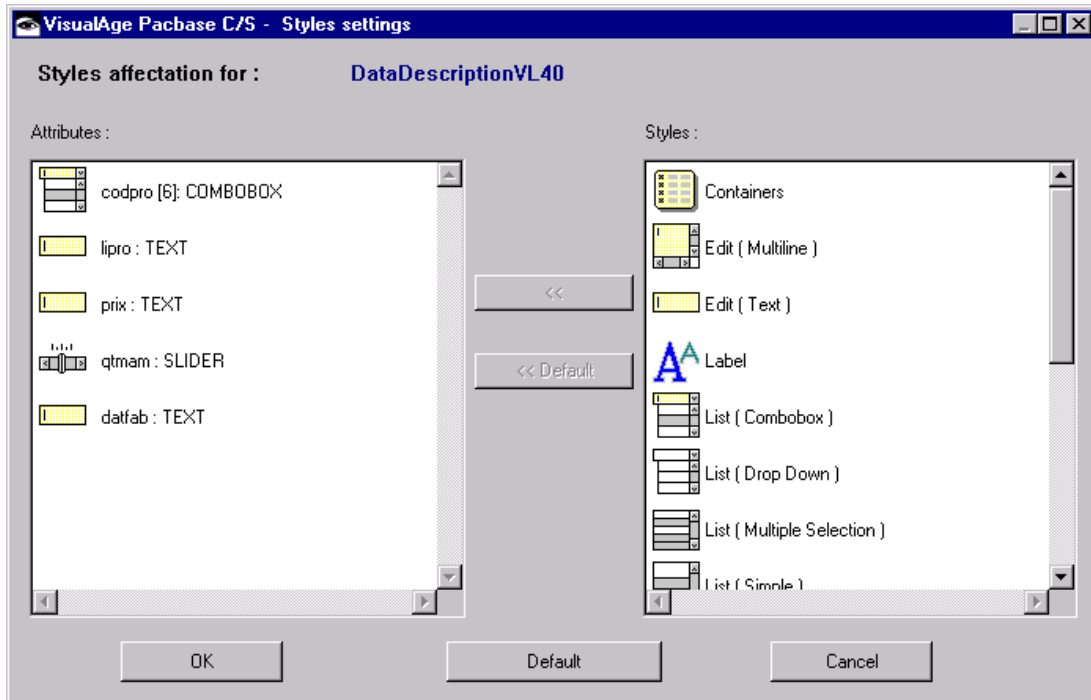
If the **Optimize Generated Code** generation option has been used, this information is not specified (instead « ? ? ? » is displayed).

#### 6.1.2.2 Defining a Graphic Representation of the Data Elements in the Detail Attribute

A default graphic representation can be defined in the Repository for each Data Element in the Logical View. Each Data Element can be presented as an edit box, a drop-down list, etc. The Quick Form choice is used to automatically generate this graphic representation for the Elementary Proxy in the VisualAge Workstation.

If you want to modify this default presentation, proceed as follows:

- From the pop-up menu for the desired Elementary Proxy, tear off its `detail` attribute
  - From the `detail` part pop-up menu, select `Styles Settings`



- Use the button to apply the desired presentation to the selected Data Element(s)
- Use the button to retrieve the initial presentation defined in the Repository for the selected Data Element(s)
- Use the button to retrieve the initial presentation defined in the Repository for all the Data Elements in the Logical View associated with the Proxy
- Once you have defined your styles, click `OK`. These styles will be automatically generated by the Quick Form function.

### 6.1.2.3. Check of the Length of the `detail` Attribute Fields

For each field of the `detail` attribute, the checks performed upon the local creation or modification of an instance ensure that the length of the value contained in the attribute does not exceed the maximum length of the value for this attribute.

The length is checked systematically (except if the attribute does not belong to the current sub-schema), even if the attribute has been defined as 'not to be checked in the client'.

A local error is sent if the length is excessive.



The length of the fields included in user buffers is not checked. If it is excessive, the length is truncated to its maximum value.

#### 6.1.2.4 Selecting the Local or Server Sort Criterion on a List of Instances

The `localSort` attribute enables you to specify whether the Proxy sorts the instances of the `rows` attribute according to the local sort (`true`) or keeps the instances in the order they were sent by the server (`false`).

You can change the sort type at any moment.



You can also specify the sort type by modifying the local parameters of the `Settings` window. See paragraph 6.1.2.1.



This attribute is not effective in user services.

##### 6.1.2.4.1 Local Sort

In standard, the instances of the `rows` attribute are sorted according to the local criterion if the parameter has not been changed after the generation. In this context, two sort types exist:

- If no sort criterion has been locally defined, the Proxy implicitly sorts the instances in the increasing order of the identifiers defined on the Logical View.
- If a local criterion has been locally defined, the Proxy sorts the instances in the order defined by this criterion.

In all cases, when an instance is created locally, it is inserted according to the current sort criterion applied in the `rows` attribute.

If you change dynamically or cancel the local sort criterion, the instances contained in the `rows` attribute are immediately sorted according to the new criterion.

##### 6.1.2.4.2 Server Sort

The instances of the `rows` attribute are sorted according to the server criterion if the `localSort` attribute is set to `false` or if the standard parameter of the node has been modified. In this context, the instances contained in the `rows` attribute are displayed in the order sent by the server, without taking the local sort criterion into account.

If collections are managed manually or in the case of a paging in extend mode, the instances sent by the server are added at the end of the existing collection in the `rows` attribute.

All the locally-created instances are added at the end of the existing collection in the `rows` attribute. In this context, an instance which is not positioned at the end of a collection but which is deleted and created again locally is transferred to the end of the collection contained in the `rows` attribute.

#### 6.1.2.5 Specific Use of the Quick Form

##### 6.1.2.5.1 Automatic Generation of a Column per Data Element of the Rows Attribute

A Quick Form of the `rows` attribute of an Elementary Proxy produces a table with one column per Data Element in the Logical View.

### 6.1.2.5.2 List of Values

When a graphic presentation is associated with a Data Element's list of values in the Repository and managed by the Quick-Form function, the transfer of the code corresponding to a label is automatically managed by the Root Proxy or the Depending Proxy.

### 6.1.2.6 Local Checks

The Elementary Proxy automatically calls the local checks during the creation and modification of an instance via the `createInstance` and `modifyInstance` actions. Each Data Element belonging to the Logical View is checked.

The following checks are made:

- Checks on value lists defined in the Data Element description
- Checks on value ranges defined in the Data Element description
- Checks on compulsory presence defined when calling Data Elements in a Logical View. The presence of identifier-type Data Elements or foreign key-type Data Elements is automatically checked for a reference relation with a minimum cardinal value of 1.

If local checks detect an error, an error message is set via the Error Manager.

These checks can be selectively triggered for each root or depending-type node in the concerned Folder. The attribute used to activate or deactivate the check of Data Elements on the server is `ServerCheckOption`.

Whether the message is sent or received, the detection of an empty Data Element is automatic.

However, the Elementary Proxy does not perform numeric or date checks; these are performed by the graphic controls.

### 6.1.2.7 Sub-schema Management

The sub-schemas specified in the Logical View's description can be taken into account when selection/read actions are performed, provided Business Components manage the presence of Data Elements (`VECTPRES=YES` or `CHECKSER=YES` options).

Each node has two attributes:

- `subSchema`, via which you can assign the desired sub-schema when a selection/read action is performed by the Business Component of the node. The value of this attribute can be assigned via the `subSchemaList` attribute. It is ignored for creation or deletion actions.
- `subSchemaList`, via which all the sub-schemas available on a node are listed. Since a sub-schema cannot be given a name in the VisualAge Pacbase Referential, each sub-schema is designated by `SubSchema<n>` (with `n` = 01 to 10).

## 6.1.3 Use of Actions

### 6.1.3.1 Implementation

An action corresponds to a process which can be executed by a Proxy objet. It is triggered using a connection between an event in the GUI application and the method code of a Proxy.



The availability of an action depends on the Proxy type. All the actions of the public interface are documented in the *Graphic Clients: Public Interface of Generated Components Reference Manual*.

### 6.1.3.2 The Different Types of Server Actions

The server actions execute procedures implemented in one or more Business Components associated with the Folder. These actions send a query to the Business Components which send back a result on the workstation. The queries and responses generally contain technical parameters, Logical View instances associated with one or more nodes and contextual data defined in a user buffer.

You must distinguish two types of server actions:

- Those which systematically get to the server:
  - ♦ `selectInstances`
  - ♦ `readInstance`
  - ♦ `readInstanceAndLock`
  - ♦ `readInstanceWithFirstChildren`
  - ♦ `readInstanceWithAllChildren`
  - ♦ `readInstanceWithFirstChildrenAndLock`
  - ♦ `readInstanceWithAllChildrenAndLock`
  - ♦ `readAllChildrenFromCurrentInstance`.
  - ♦ `readAllChildrenFrom:`
- Those which do not systematically get to the server:
  - ♦ `readNextPage`: The server is accessed except if the `noPageAfter` event was sent back during the previous selection
  - ♦ `readPreviousPage`: The server is accessed except if the `noPageBefore` event was sent back during the previous selection.
  - ♦ `readFirstChildrenFromCurrentInstance`: The server is accessed except if the `maximumNumberOfRequestedInstances` attributes of Depending Proxies are set to 0 and if their `globalSelectionIndicator` attributes are set to false.
  - ♦ `readFirstChildrenFrom`: The server is accessed except if the `maximumNumberOfRequestedInstances` attributes of Depending Proxies are set to 0 and if their `globalSelectionIndicator` attributes are set to false.
  - ♦ `checkExistenceOfDependentInstances`: The server is accessed except if the existence of depending instances can be checked locally.
  - ♦ `updateFolder`: The server is accessed only if there is at least one instance of the node concerned, modified in its `updatedFolders` . attribute.

### 6.1.3.3 Managing Folder Reading

#### 6.1.3.3.1 Large reading of the Root node

Large reading of a Folder root enables to read from a client component all the instances of the Folder's root node existing in the database. The actions concerned are `selectInstances` and `readNextPage`.

#### 6.1.3.3.2 Provisional Large Reading of Depending Nodes

In the context of client/server architectures, a GUI application handling a Folder strives to anticipate data reading to minimize exchanges with the servers.

In a hierarchical network, various provisional reading actions are possible:

- The '**allChildren**' type action reads *all* the depending instances of the selected instance in the **detail** attribute of the parent Elementary Proxy.
- The '**firstChildren**' type action only reads the instances which are *immediately* dependent of the selected instance in the **detail** attribute of the parent Elementary Proxy.

The first provisional large reading action is available on the Root Proxy only.

The second provisional large reading action is available on the Root Proxy or on the Depending Proxies which themselves hold Depending Proxy objects.

#### **6.1.3.3 Transferring an Instance between the Rows and Detail Attributes**

The transfer of an instance between the **rows** and the **detail** attributes enables the **detail** attribute to be loaded with an instance initially retrieved by a large reading action.

It corresponds to a local reading action which also loads all the local instances of Depending Proxies known by the Folder View Proxy. The transfer is made using the **getDetailFromDataDescription** action.

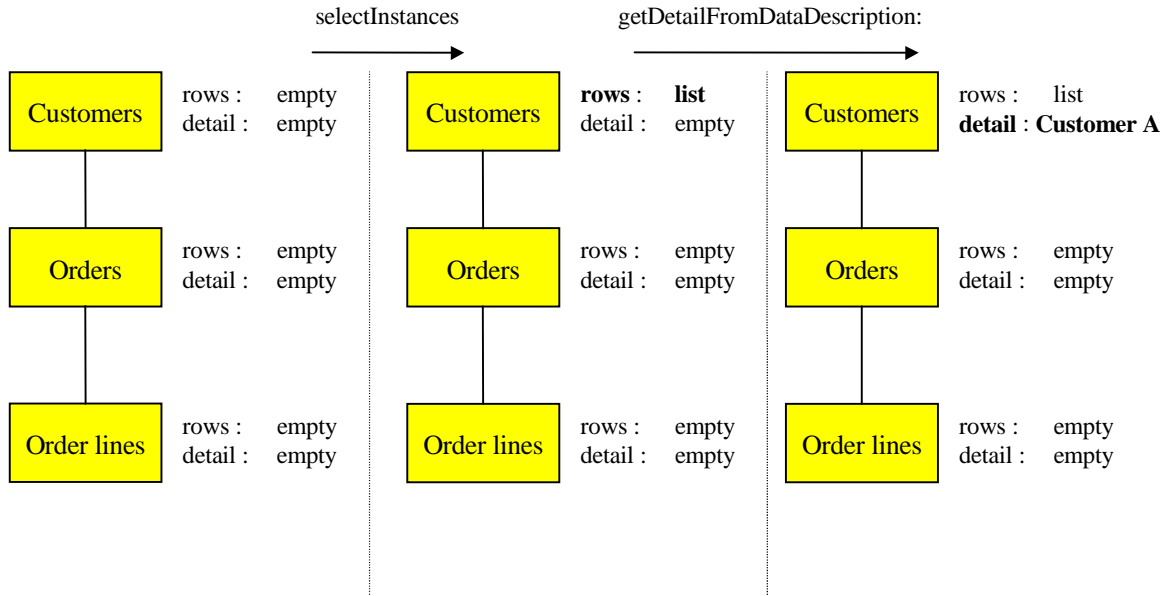
#### **6.1.3.3.4 Large Reading and Transferring an Instance between the Rows and Detail attributes: Operating Mechanism**

This example illustrates the loading of the **detail** attribute with an instance previously retrieved in the **rows** attribute by a large reading action on a root or depending node, and the principle of the provisional large reading of depending nodes.

It is based on the Folder View Proxy generated previously.

For the needs of our example, we will use three Elementary Proxies:

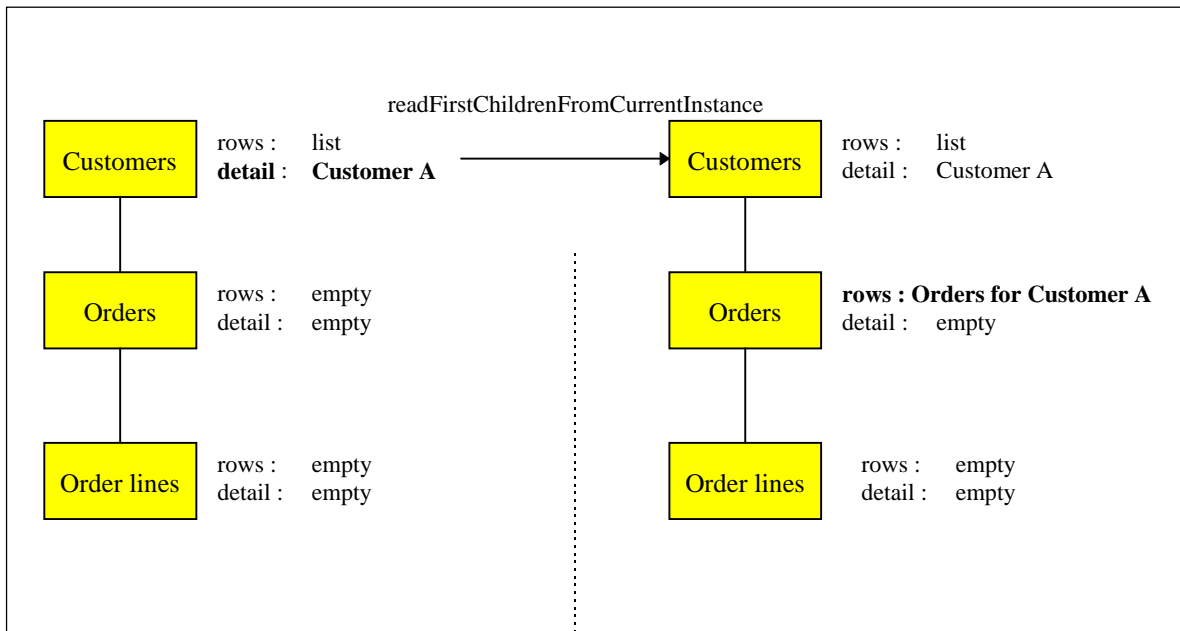
- The **NO10(CLCLNT)** Root Proxy corresponding to the **Customers** node which manages the customers in the information system described by the Folder
- The **NO20(CLCMDE)** Depending Proxy corresponding to the **Orders** node which manages the orders in the information system described by the Folder
- The **NO30(CLLCDE)** Depending Proxy corresponding to the **Order Lines** node which manages the order lines in the information system described by the Folder.



The **detail** attribute of the customer node now contains Customer A. There are three solutions to read Customer A's depending instances (i.e. his associated orders).

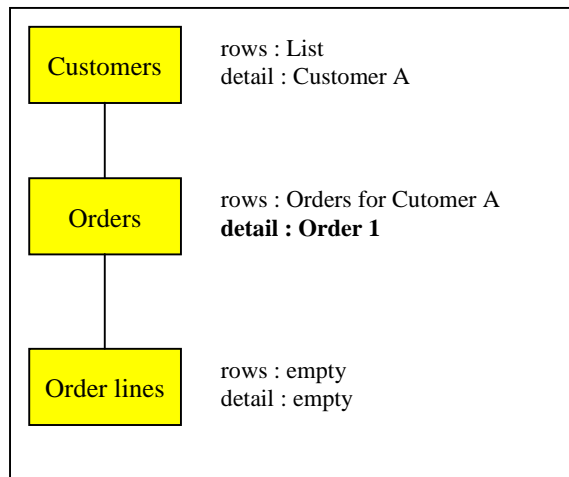


**SOLUTION 1**



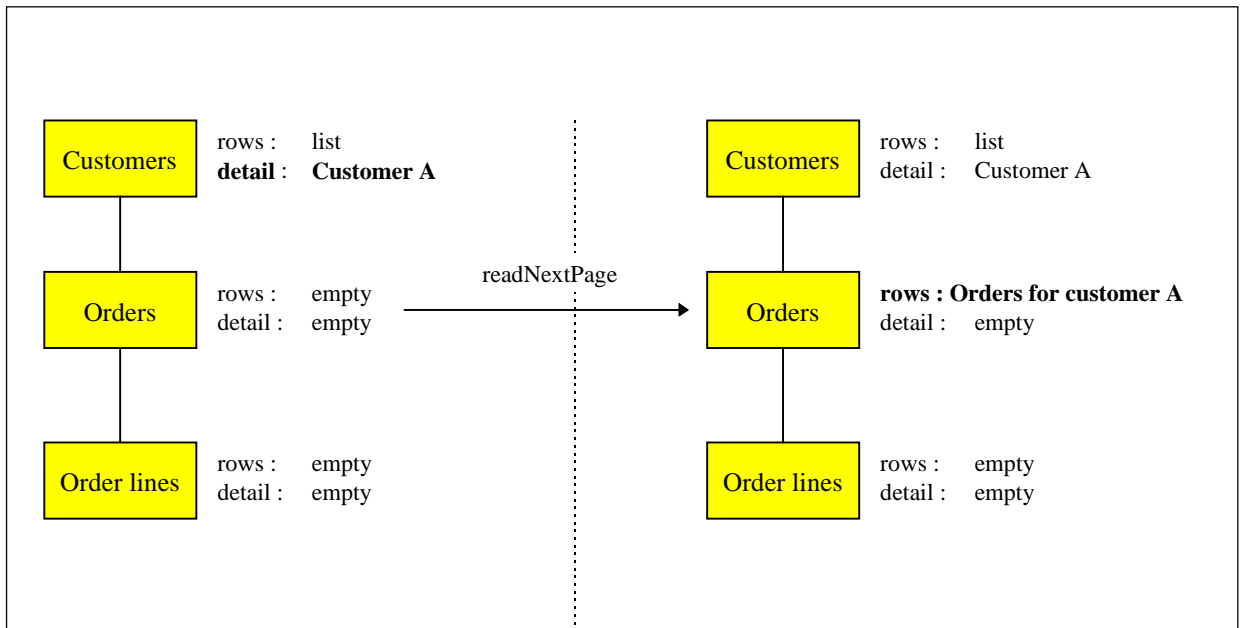
The `readFirstChildrenFromCurrentInstance` action on the Customers Root Proxy not only loads the `rows` attribute of the Orders Depending Proxy for Customer A, but also the `rows` attribute of other possible Elementary Proxies directly dependent of the Root Proxy.

In this context, the `getDetailFromDataDescription:` action on the Orders Depending Proxy initiates:



Then, to read the order lines of the Order 1 for Customer A, use the `readFirstChildrenFromCurrentInstance` action on Orders or the `readNextPage` action on Order lines.

## SOLUTION 2

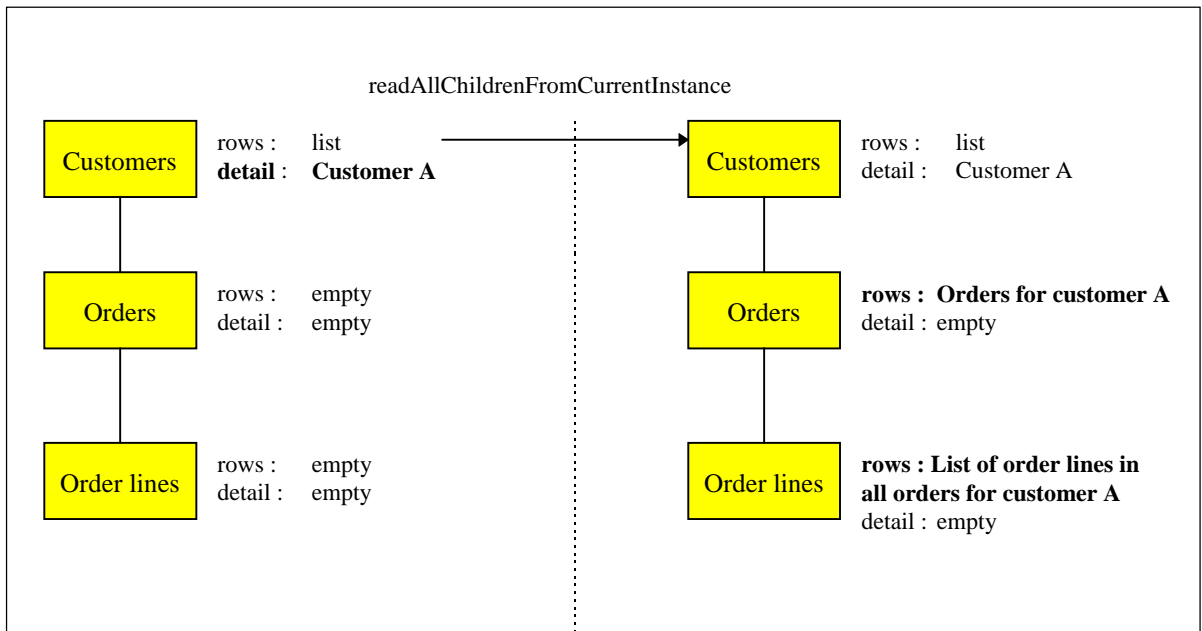


The `readNextPage` action on the Orders Depending Proxy loads its `rows` attribute. The instances of other possible Elementary Proxies directly dependent of the Root Proxy are not read.

In this context, the `getDetailFromDataDescription:` action on the Orders Depending Proxy initiates the same result as in solution 1.

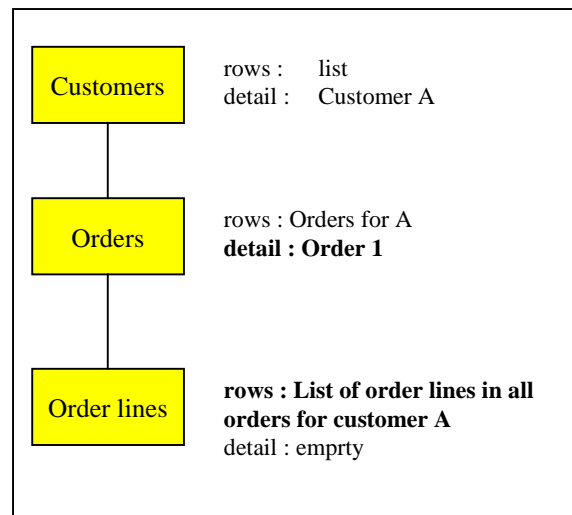
To subsequently read the order lines of the Order 1 of Customer A, proceed as in solution 1.

### SOLUTION 3



The `readAllChildrenFromCurrentInstance` action on the Customers Root Proxy reads not only all the orders for A but also all the other possible instances dependent of A whatever their hierarchical level can be. In our example, all the order lines for all the orders of A are therefore read.

In this context, the `getDetailFromDataDescription` action on the Orders Depending Proxy initiates:



The `getDetailFromDataDescription` action automatically loads the `rows` property of the Order lines Depending Proxy with the order lines contained in Order 1 for Customer A. These order lines have been previously transferred to the workstation using the `readAllChildrenFromCurrentInstance` action.

#### 6.1.3.3.5 Large Reading of Reference Nodes

The reading of a reference node is considered as aid on criteria. It shows the end user a list of information which can be referred to for a depending node.

The information presented to the end user is both necessary and sufficient to assist him in making a choice.

To optimize the volume of characters sent for this type of service, the Logical Views have a « aid on criteria »-type subschema used to select the concerned Data Elements at the server level.

The large reading of reference nodes is executed on request and cannot be involved in provisional large reading. The actions concerned are **selectInstances** and **readNextPage**.

#### 6.1.3.3.6 Principle of Paging in a Folder's Nodes

Two types of paging are offered on a Folder's nodes:

- The first, called *non-extend* paging, is used to paginate forwards and backwards on a predefined collection through specific actions. Each action executes a read request to the server and its result overwrites that of the previous read. This type of paging is available only on root or reference nodes.
- The second type of paging, called *extend* paging, is used to gradually retrieve the instances of a defined collection as read requests for following pages are made. In this context, the backwards paging function disappears and is performed locally by the scroll box of the graphic control which presents the list of instances. This type of paging is available to all the nodes of a Folder.

#### 6.1.3.3.7 Selection Criteria Associated with Large Reading Actions

The selection criteria associated with large reading actions are elementary or composite attributes associated with each node of a Folder. They are split into two types:

- Functional selection criteria corresponding to the identifier and to elements required for defining extraction methods for the Logical View associated with the node. These criteria correspond with the following attributes:
  - ♦ **selectionCriteria** which defines identifier Data Elements and parameters by value.
  - ♦ **extractMethodCode** which defines the code of the desired extraction method.
  - ♦ **extractMethodList** which contains the list of available extraction methods.
- Organic selection criteria corresponding to information used to control the volume of instances selected for each node
  - ♦ **globalSelectionIndicator** is a Boolean attribute which, when set to true, is used to retrieve all the instances of the node via a selection query.

- ♦ **maximumNumberOfRequestedInstances** is a numerical attribute which specifies, when the **globalSelectionIndicator** attribute is set to false, the number of instances to be retrieved for a node via a selection query. This attribute can hold the value 0. In this case, the concerned part of the tree structure is not read during a provisional large reading action.

#### **6.1.3.3.8 Limitation of the Scope of Large Reading**

The result of a large reading action is limited by the selection criteria associated with the node.

However, in the case of a large reading ordered by an **allChildren** type provisional reading, the **globalSelectionIndicator** attribute is considered to be set to true on each node.

#### **6.1.3.3.9 Reading of a Depending or Root Node Instance**

The reading of a depending or root node instance enables you to retrieve an instance of the node without previously making a large selection of a collection of instances for this node. It directly loads the node's **detail** attribute.

This type of reading is considered as a collection selection and therefore cancels the previous selection even if it was the result of a large reading action. The loading of the **detail** attribute therefore initializes the **rows** attribute.

The actions which implement this selection function are used to:

- Retrieve the instance without its dependencies (**readInstance**).
- Retrieve the instance with its first level dependencies (**readInstanceWithFirstChildren**).
- Retrieve the instance with all its dependencies (**readInstanceWithAllChildren**).

#### **6.1.3.3.10 Reading of a Reference Node Instance**

The reading of a reference node instance cannot activate the provisional large reading process. It is used to retrieve the entire description of the instance of the node called in its **detail** attribute.

Only the **readInstance** action is therefore available on a reference node.

Update actions are not available for reference nodes. Their **rows** and **detail** attributes are independent. The result of the **readInstance** action does not therefore initialize the **rows** attribute.

The **rows** attribute is used to display sufficient information to assign one of the reference node instances to the referencing node instance.

The **detail** attribute is used to view the entire description of a referenced instance.

The structure of these two attributes can thus be different.

#### **6.1.3.3.11 Selection Criteria Associated with Instance Reading**

The identifier of the node instance to be read is specified in the functional selection criteria associated with the node.

For depending nodes, the identifier of the node corresponds to the identifier of the Logical View associated with the node, discarded from the Data Elements which are the identifiers of Logical Views higher in the hierarchy. These hierarchical identifiers are automatically initialized by the Folder View Proxy according to the navigation in the Folder.

### **6.1.3.4 Folder Update Management**

#### **6.1.3.4.1 Local Updates**

Local update services are available on each root or depending type node in the Folder.

- Create a node instance
- Modify a node instance
- Cancel a node instance

However, there are certain rules specific to Folder management:

- The creation of a depending node instance is only authorized if the hierarchy of instances contained in the **detail** attributes of higher nodes exists.
- Canceling a node instance initiates the recursive canceling of local instances of depending nodes.

To allow the developer to manage messages which can warn users of the impact of a cascade cancellation, an action for verifying the existence of depending instances is available on root or depending type nodes.

This action (**checkExistenceOfDependentInstances**) sends a Boolean result which is either true or false. If no dependency is found in the Folder's local cache and the instance concerned is not created locally, this action sends a verification request to the server.

So that a user can undo local manipulations of a Folder instance, an **undoAllLocalFolderUpdates** action can be used to eliminate all local updates on all Folder nodes applied since the last server update action. This action is only available on the Folder's root node. Another action **undoLocalFolderUpdates(DataUpdate)** can be used to eliminate all updates associated with the instance set as parameter.

#### **6.1.3.4.2 Server Updates**

Only the Root Proxy provides server update actions.

Server updates correspond to actions which enable a client component to send all local updates made since the last server update action.

These updates concern all the modified depending instances. They can concern several Folder instances.

When a server update action sends back errors, the Folder remains with the « Modified Locally » status; new collection selections can be made only by correcting the errors and sending back the updates, or by using the `undoAllLocalFolderUpdates` action or the `undoLocalFolderUpdatesFor` action.

Before the request is sent to the server, the server update actions check the Folder integrity for locally created instances. For each locally created instance of the node, this action checks the minimum cardinal values of each link and sends an error if the number of depending instances does not respect the properties of the associated links.

A server update action can be accompanied by a request to refresh the updated instances if some of their Data Elements, such as the identifiers, are calculated by the server. This refreshment request is made using the `refreshOption` attribute.

#### **6.1.3.4.3 Management of Effective Transactions**

The management of effective transactions is automatically carried out by the local cache.

It consists in calculating the resulting update of various local updates made on the same instance of the Folder's node. It controls the creation of duplicate instances. If several local updates have been made on the same instance of the node, only the last one will be sent to the server.

#### **6.1.3.4.4 Changing the Collection Selection**

The `aboutToChangeSelection` event is returned to the Client by the Root Proxy when a server selection action on a Folder node overwrites its current local collection and when this collection or the depending collections contain local modifications which are candidates for a server update. If the event is not intercepted during programming, the updates will be lost.

#### **6.1.3.4.5 Re-initializing instances in the local cache**

The `resetCollection` action is used to remove all the instances from the cache of a Folder View Proxy before initializing a new collection of instances. This action can be executed by all types of nodes in a Folder View Proxy containing a `Rows` attribute.

#### **6.1.3.4.6 Managing collections of instances**

The management of collections of instances can be carried out automatically, or manually by positioning the `manualCollectionReset` boolean attribute which is available for all types of nodes in a Folder View Proxy. The manual management mode is used to create collections of heterogeneous collections through a series of selection and paging actions.

### **6.1.3.5 Asynchronous Actions**

#### **6.1.3.5.1 Principles**

The asynchronous programming is used to dissociate the action used to send a request from the action used to retrieve its response. You can use this type of programming whether you use an asynchronous communication protocol or not.

You can use the Proxy components in asynchronous mode independently of the middleware used. As for the Proxy, you can work with an asynchronous mode, by using a *location* whose middleware is synchronous, and the other way round.

In this context, the end user will be more efficient as he can send a request in advance, and retrieve the response when he needs it. This method allows you to optimize your working time.

Furthermore, the communication protocols are used to make the requests or the responses in the local messages threads more secure by allowing the message to convey independently of the network situation.

The communication mode of a Proxy is defined by the `setAsynchronous:aBoolean` attribute.

When the communication is asynchronous, the Proxy sends an `asyncRequest` event after the execution of each server type action. The response identifier used to further retrieve the response associated with the request is returned in the `lastMessageId attribute`. The application should save this identifier.

The `getReplyOf:aReplyId` action explicitly retrieves a response .

When the response is available, the Proxy behavior is the same as the behavior resulting from the associated response received with a synchronous server action.

When the response is unavailable, a `replyPending` event is sent.

#### **6.1.3.5.2 Global Actions or Action Associated with an Instance**

Some server actions, labeled as global actions, are independent of any selection, whereas others depend on any instance included in the local cache. In asynchronous mode, global actions store the response identifiers in a collection and each executed request adds its identifier to this collection. The actions associated with a Logical View instance store their response identifiers in a collection associated with the concerned instance. The response identifier collections associated with the global actions are lost when the application using the Proxy is closed. A response identifier collection associated with an instance included in the local cache is lost when this instance is locally deleted or after a change of collection.

- **global Actions**

The responses associated with the following actions can be executed independently of the current collection:

- `executeUserService`
- `readInstance` (ROOT)
- `readInstanceWithFirstChildren` (ROOT)
- `readInstanceWithAllChildren` (ROOT)
- `readInstanceAndLock` (ROOT)
- `readInstanceWithFirstChildrenAndLock` (ROOT)
- `readInstanceWithAllChildrenAndLock` (ROOT)
- `readNextPage` (ROOT)
- `selectInstances`
- `lock`
- `unlock`



- `readPreviousPage`

- **Actions associated with an instance**

The following actions depend either on the `detail` instance, or on the instance passed as a parameter, or on the `detail` parent instance for depending nodes:

- `checkExistenceOfDependentInstances`
- `readAllChildren(data)`
- `readFirstChildren(data)`
- `readAllChildrenFromCurrentInstance`
- `readAllChildrenFrom` (DEP)
- `readFirstChildrenFromCurrentInstance`
- `readFirstChildrenFrom` (DEP)
- `readInstance` (DEP)
- `readInstanceWithFirstChildren` (DEP)
- `readInstanceWithAllChildren` (DEP)
- `readInstanceAndLock` (DEP)
- `readInstanceWithFirstChildrenAndLock` (DEP)
- `readInstanceWithAllChildrenAndLock` (DEP)

#### 6.1.3.6 User Services

Each root or depending type node contains the following elements to implement a user service:

- An attribute used to obtain the list of user services available on this node plus `nil`. This attribute is `userServiceList`.
- An attribute used to initialize the user service code to be executed. This attribute is `userServiceCode`.
- An attribute used to locally store Logical View instances to be processed for the next user service. This attribute is `userServiceInputRows`.
- An attribute used to present various Logical View instances sent back by a user service. This attribute is `userServiceOutputRows`.
- An attribute which presents the Logical View instances which are candidates for the execution of the next user service. This attribute is `userDetail`.
- Local actions used to memorize each Logical View instance to be sent to the server for the execution of a user service. These actions are `create-UserServiceInstance`, `modifyUserServiceInstance` and `deleteUserServiceInstance`.

The root node of the Folder also has the following elements:

- An action used to execute all the user services parameterized on each Folder node. This action is **executeUserServices**.
- An action used to delete all the local instances stored for all the Folder nodes. This action is **resetUserServiceInputInstances**.
- An action used to delete the current instance stored locally. This action is **resetUserServiceCodes**.

This principle means that 1 to n user services can be executed, in the same query, distributed on the different nodes. The execution sequence of these services corresponds to the hierarchical order of nodes, browsing the tree from top to bottom and from left to right.

### 6.1.3.7 Database Logical Lock

The upload-download mechanisms associated with a Folder increase the elapsed time between reading the initial Folder image and displaying the result of an update.

In this context, with no lock mechanism, two users can modify the same Folder instance. The result of accumulated updates are therefore difficult to manage.

To enable the user to use a Folder in an exclusive appropriation mode, two types of locks for a node instance are available:

- The optimistic lock which works on the principle of verifying the change of a **TimeStamp** before executing the update procedure.
- The pessimistic lock which uses an entity for exclusive update by recording a specific resource. In this case, the Folder update procedure is carried out before freeing up the exclusive resource.

The server lock procedure is triggered by the explicit execution of a specific action available on the Root Proxy. This action is **lock**.

The server unlock procedure is triggered automatically with the execution of a server update action or explicitly by the execution of a specific action available on the Root Proxy. This specific action is **unLock**.

The developer is responsible for writing the lock processing in the root Business Component.

This processing receives the identifier of the Logical View instance to be locked as well as the request type (lock or unlock) to be executed.

In return, it must set a status used to grant or refuse the lock and the **TimeStamp** or the name of the resource used.

If the lock is refused, the Root Proxy sends a **lockFailed** event whereby all subsequently executed local or server update actions are disabled for the specified instance of the Folder. In this case, the Folder changes to the « Read only » status.

The concept of the logical lock is defined in the Folder entity or in the Business Component for a single-view development.

When the logical lock option is active on a Folder, all read requests for the **detail** attribute of the Root Proxy can be accompanied by a logical lock request on the server.

### 6.1.3.8 Management of Data Element Presence

The two following actions enable you to manage the presence of the Logical View's Data Elements at the Proxy level.

The `isNull:<delco>` action enables you to test the presence or absence of the `delco` Data element. It is generated for all the `DataDescription` and `UserDataDescription` classes of the nodes whose Business Components include the `NULLMNGT=YES` option.

The `setNull:aBoolean on:<delco>` action enables you to specify the presence or absence of the `delco` Data element. It is generated for all the `DataDescription` and `UserDataDescription` classes of the nodes whose Business Components include the `NULLMNGT=YES` option and a user service or an update service.

By default, all the Data Elements are considered to be present.

### 6.1.3.9 Management of Data Element Check

You can manage the check of the Logical View's Data Element at the Proxy level with the `setCheck:aBoolean on:<delco>` action. This action enables you to activate or inhibit the server checks on a Data Element before any local update action. It is generated for all the `DataDescription` classes of the root or depending nodes whose Business Components include the `NULLMNGT=YES` and `CHEKSER=YES` options and an update service.

By default, all the Data Elements are to be checked (if the `serverCheckOption` attribute is set to `true`).

### 6.1.3.10 Sub-schema Management

The server selection or read actions take into account the sub-schema present in the `subSchema` attribute and return the values of the Data Elements belonging to the sub-schema. If a selection action is followed by a paging action, the sub-schema taken into account is that associated with the selection action.

The local creation actions do not refer to any sub-schema.

The local modification/deletion actions refer to the sub-schema associated with the instance, that is:

- if the modification/deletion is performed on an instance which was created locally, the sub-schema is empty.
- if the modification/deletion is performed on a read instance, the sub-schema is that associated with the selection of this instance.

Moreover the following actions are specific to the sub-schema management.

The `resetSubSchema` action enables you to reset the `subSchema` attribute, that is to select no sub-schema.

The `completeInstance` action enables you to retrieve the values of the Data Elements which do not belong to the sub-schema, by calling the Business Component associated with the Logical View.

The `belongsToSubSchema` action enables you to know whether the Data Element passed as a parameter belongs to the sub-schema associated with the `detail` attribute.

## 6.1.4 Use of Events

The events sent by an Elementary Proxy are used to trigger application actions belonging to the GUI application. This processing is performed by connecting a Proxy event to one or more actions in the GUI application. The conditional execution of actions is facilitated by the fact that an event is always accompanied by its opposite event; both events cannot be sent at the same time.



The availability of an event depends on the type of Proxy. All the Public Interface events are documented in the *Graphic Clients: Public Interface of Generated Components Reference Manual*.

### 6.1.4.1 Event-driven Management of Large Reading

Event-driven management of large reading provides the developer with information on the state of the collection of instances contained in a node. Each available paging action offers its own event-driven paging system.

The paging action in non-extend mode can send the following four events:

- **noPageBefore**: This event is sent by a root or reference node at the end of the execution of a collection selection or paging action when it does not return any error and when the read page is the first in the current collection.
- **pageBefore**: This event is sent by a root or reference node at the end of the execution of a collection selection or paging action when it does not return any error and when the read page is not the first in the current collection.
- **noPageAfter**: This event is sent by a root or reference node at the end of the execution of a collection selection or paging action when it does not return any error and when the read page is the last in the current collection.
- **pageAfter**: This event is sent by a root or reference node at the end of the execution of a collection selection or paging action when it does not return any error and when the read page is not the last in the current collection.

The paging action in extend mode can send the following two events:

- **pageAfter**: This event is sent by any type of node at the end of the execution of a collection selection or forwards paging action when it does not return any error and when the number of instances contained in the node is not the total number of instances contained in the database.
- **noPageAfter**: This event is sent by any type of node at the end of the execution of a collection selection or forwards paging action when it does not return any error and when the number of instances contained in the node is the total number of instances contained in the database when the query is made.

Event-driven management is characterized by the sending of data at a given moment. In some cases, you need to know the status of a node after a paging event has been sent.

Each node is thus supplied with an action which can be used to retransmit the last paging event sent by this node (`getLastSelectResponseStatus`). In this case, the nodes which did not participate in the last reading action send the 'notRead' event.

### 6.1.4.2 Event-driven Management of Instance Reading

A Logical View can be mapped on one or more physical storage entities. In this context, the event-driven management of a Logical View instance reading can send the following event:

- `notFound` when the instance searched for is not found in the database. This event can be sent when the Logical View is mapped on one or more tables.

## 6.2 Example of the Development of a Standard GUI Application

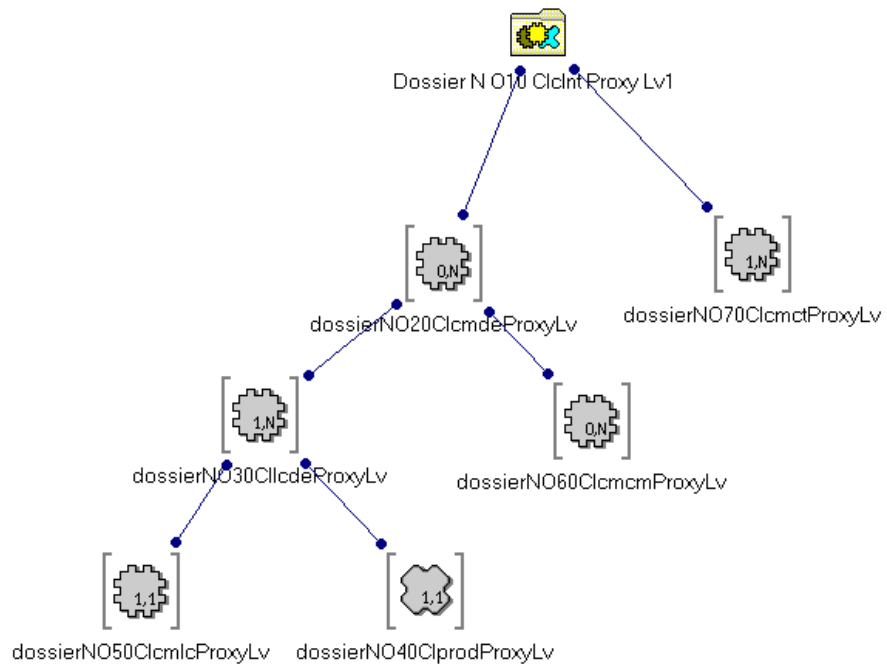
### 6.2.1 Introduction

Supplied for information only, this example illustrates, without attempting to give an exhaustive description of all the possibilities of a folder mode development, the basic implementation principles.



This example is based on a Folder View Proxy, which contains several Elementary Proxy objects. But the implementation principles remain the same for a Folder View Proxy containing a single Elementary Proxy. If the application was just containing the root Proxy, the first window only would be available; and the buttons used to branching to the other windows would be disable.

The example is based on the use of the `dossierNO10ClclntProxyLv` generated in Chapter 3, *VisualAge for Smalltalk*.



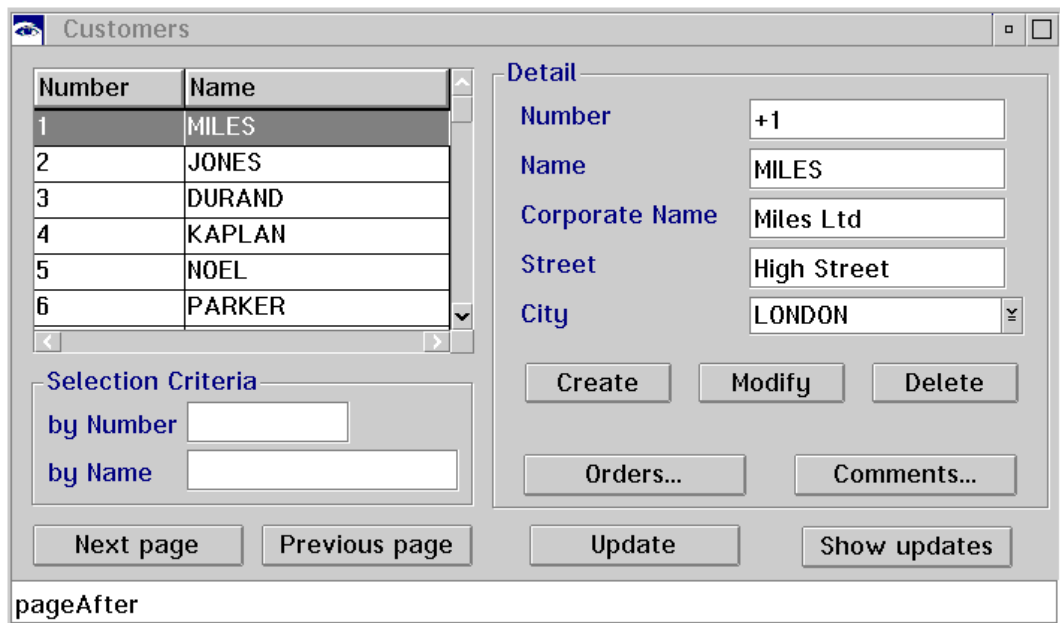
The example deals with only four of the Elementary Proxies of the FVP:

- The **dossierNO10ClclntProxyLv** Root Proxy corresponding to the **Customers** node which manages the customers in the information system described by the Folder
- The **dossierNO20ClcmdeProxyLv** Depending Proxy corresponding to the **Orders** node which manages the customers' orders in the information system described by the Folder
- The **dossierNO30ClldcdeProxyLv** Depending Proxy corresponding to the **Order Lines** node which manages the order lines in the information system described by the Folder
- The **dossierNO40ClprodProxyLv** Reference Proxy corresponding to the **Products** node which manages the products likely to be ordered in the information system described by the Folder.

## 6.2.2 Presentation of the Application - Example

The developed end user interface includes five windows:

- **The Customers window**



This window has the following functionalities:

- Paging forwards and backwards to select a list of customers in sets of 50: **Next Page** and **Previous Page** buttons.
- Selection of a list of customers in sets of 50 but according to the following criteria:
  - ♦ Display in ascending order from a given customer number. The user must enter an identifier in the **By Number** field, then click **Next Page** or **Previous Page**.
  - ♦ Display in alphabetical order from a given customer name. The user must enter a name in the **By Name** field, then click **Next Page** or **Previous Page**.
- Create, modify or delete a customer from a detail.
 

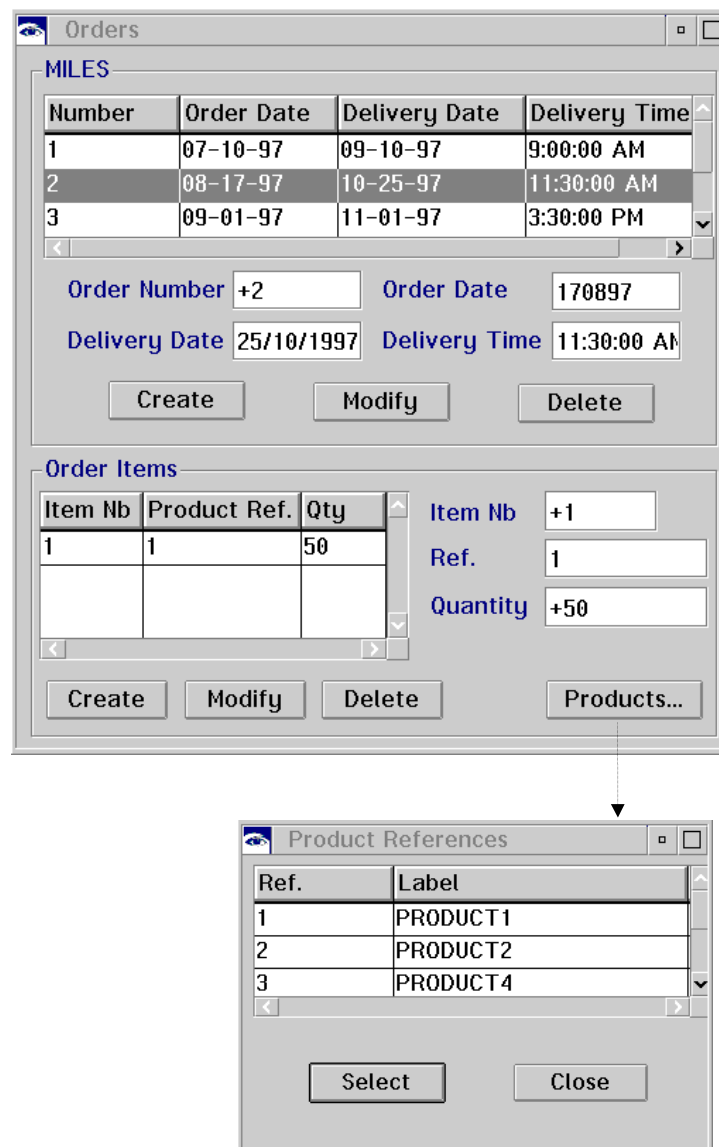
To create a customer, the user must enter the data associated with the new customer in the detail fields, then click **Create**.

To modify a customer, the user must select a customer by double-clicking on a line from the list. The data sections associated with this customer (identifier, name, company name, address) are displayed in the **Detail** area. The user must then click **Delete** or **Modify** after entering the new data.
- Navigation to a window which presents the list of customers affected by local updates: **Show updates** button.
- Update of the database by accepting all the customers affected by local updates: **Update** button.
- Navigation to the Orders window: **Orders...** button.

- Display messages sent back by the server after selecting a new set of customers. These messages are used, for example, to find out whether the selected set of customers is the last.
- The **Comments...** button is not developed. Its existence only indicates the possibility of developing a window which would use the **dossierNO70ClcmctProxyLv** Depending Proxy corresponding to the Comments node. The principles implemented for the development of this window are identical to those used for the **Orders** window.

- **The Orders and Product References Windows**

The functions of these two windows are closely linked since the second of the two is used as a help window by the first.



The **Orders** window is displayed when the user clicks on the **Orders...** button in the **Customers** window.



For the customer selected in the detail area of the **Customers** window, this window is used to:

- View the list of the customer's orders.
- Create, modify or delete an order from the order detail.
- View, for the selected order, the order lines: **Order Items** area.
- Create, modify or delete an order line from the order detail.
- Open the window which presents the list of products: **Products...** button.

The **Product References** window is displayed when the user clicks on the **Products...** button in the **Orders** window.

This window provides help when the user modifies an order line in the **Order Items** area. It contains a list of products which the user cannot modify. It saves the user from manually entering the reference number of the product associated with the order line.

The principle behind the use of this help is the following:

- When creating an order line, in the order line detail area, the user must enter the number of the new order line and the amount desired for this product, then open the **Product References** window. In this window's product table, the user must select the required product and click **Select**. The value contained in the **Ref** column in the product table is automatically transferred to the **Ref** field in the order line detail area. The user then simply clicks **Create**.
- When modifying an order line, the user must select an order line from the list in the **Order Items** area. The details of the selected order line are displayed.

The user must then open the **Product References** window and, from the product table, select the product which is to replace the current product, then click **Select** to load the detail area of the order line with the reference number of the new product. Following further possible modifications, the user clicks **Modify**.

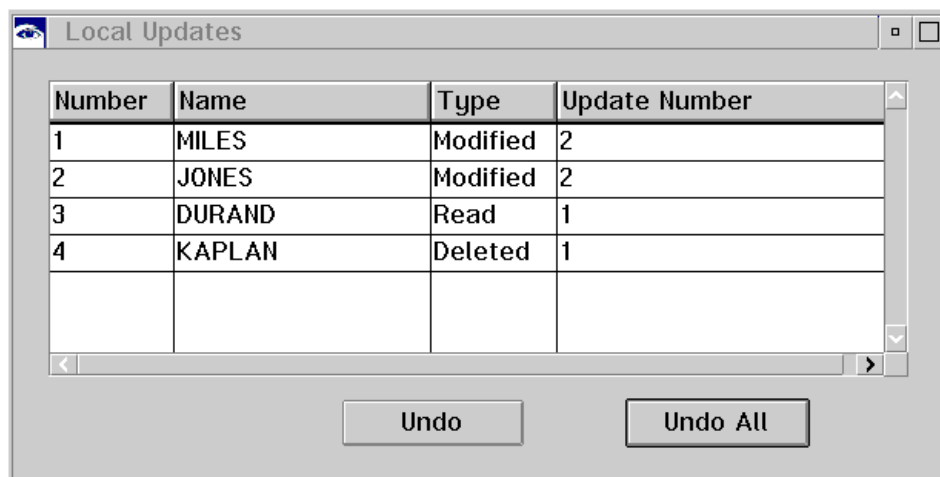


All updates, whether made on customers, orders or order lines, are stored locally. To transfer them to the database, the user must click on **Update** in the **Customers** window.

#### • **The Local Updates Window**

This window is displayed when the user clicks **Show Updates** in the **Customers** window.

It is accessed when the user wants to view the list of updates made since the last database update. The user can delete one or more lines from this list and thus cancel the corresponding operations for the customers concerned before making a renewed server update.



Each line in the table corresponds to a customer for which local updates have been made.

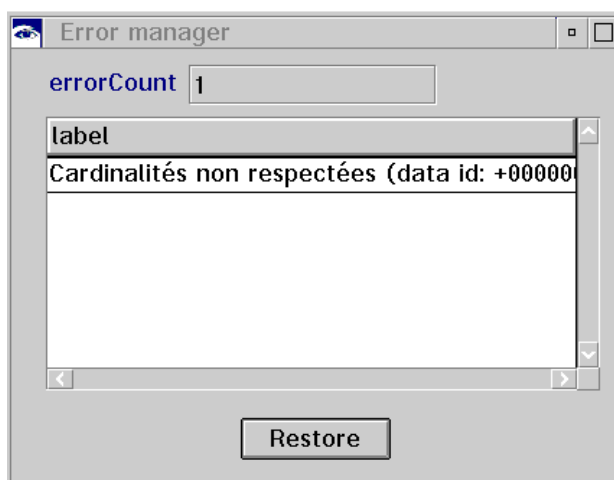
The **Type** column specifies the nature of the update. The possible values are:

- **Modified** indicating that data sections concerning a customer have been modified
- **Deleted** indicating that a customer has been deleted
- **Created** indicating that a customer has been created
- **Read** indicating that an update (creation, modification, deletion) has been made on a customer order or order line

The user can cancel all these modifications by clicking **Undo All** or else delete one or more modifications selected by clicking **Undo**.

#### • The Error Manager Window

This window is opened automatically if the server, the exchange manager or the local cache detects an error following a user action.



The **ErrorCount** field indicates the number of detected errors.

For some errors, the user can automatically restore the context of the error in order to find out the origin of the error and to correct it before a new attempt at making a server update.

**Example** Let us consider a concrete example to illustrate the automatic restoration mechanism of the error context.

Let us imagine that ORDER LINE No. 1 of ORDER No. 2 of CUSTOMER MILES is the source of an error which blocks a server update request. The error messages window opens when the user clicks the **Update** order button. To assist the user in finding the cause of the error, the user can make use of the restore context function.

To do this, the user must select the line corresponding to the error in the list and click **Restore**.

In the **Orders** window, the details of ORDER LINE No. 1 are automatically displayed - as below - in place of the previously selected order line.

| Item Nb | Product Ref. | Qty |
|---------|--------------|-----|
| 1       | 1            | 50  |
|         |              |     |

Item Nb:

Ref.:

Quantity:

Buttons: Create, Modify, Delete, Products...

☞ Since the selection tree is respected, the details of ORDER No. 2 and the details of the CUSTOMER MILES are also displayed in the corresponding windows in place of the previous instances.

👁 The programming of the error message window is detailed in the Chapter, Subchapter 6.4.

## 6.2.3 Development of the Application- Example

The development of the graphical user interface includes the construction of four windows which make up the application in the following order: **Customers, Orders, Product References, Local Updates**.

In the description of these different stages, the use of different tools and functions in VisualAge is not detailed. If you are not familiar with these tools and functions, refer to the appropriate documentation.

Whilst maintaining - as much as possible - a sequential approach to the development, we will divide, where necessary, the programming of windows into several parts according to consistent groups of functions.

Once each part has been dealt with, we will present the Composition Editor as it should appear whilst hiding, where necessary, previously developed links to better isolate those which are newly created.

☞ There can be several development solutions to the same function. In our example we give preference to visual programming.

### 6.2.3.1 Developing the Customers Window

#### 6.2.3.1.1 Initializing the Window and Integrating the FVP in the Composition Editor

👁 The operations which must be carried out to initialize a window are dealt with in the VisualAge documentation.

The integration of the Folder View Proxy in the Composition Editor consists in placing the Root Proxy on the Free Form Surface.

In the parts palette, select the **Client/Server Facility** category then the part corresponding to the Proxy.

**Client/Server facility** category:



Part corresponding to the FVP:



### **6.2.3.1.2 Programming Functions Linked to the List: Rows Attribute**

This stage in the programming includes the following operations:

- **Creation of the list of customers**

To create the list, you must create a Quick Form for the **rows** attribute of the Root Proxy. We obtain a container where each column corresponds to an attribute of the Root Proxy.

In our example, we have deleted some columns to only keep those labeled **Number** and **Name**.

- **Programming the Selection**

We have programmed two types of selections with forwards and backwards paging.

- Selection of a list of customers in sets of 50 customers
- Selection of a list of customers in sets of 50 customers but according to criteria: display in ascending order from a given customer number or display in alphabetical order from a given customer name.



The forwards and backwards paging function is specified in the Folder (or in the Business Component if no Folder has been specified). For more information, confer to paragraph **6.1.3.3**.

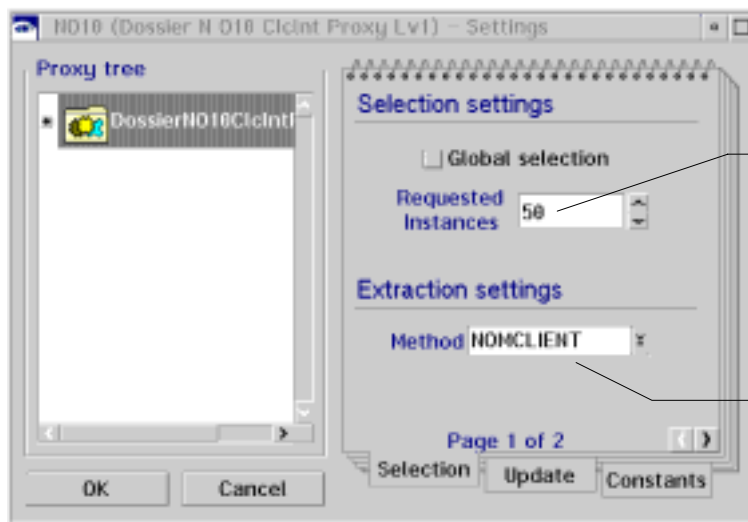
La programmation de ces services de sélection nécessitent les opérations décrites ci-dessous.

The operations described below are required for the programming of these selection services.

- You must firstly specify the selection criteria associated with the reading actions: the number of occurrences to be read and the extraction method to be used for a selection by name. On the server, since an identifier-type Data Element represents an implicit extraction parameter, the corresponding extraction method is implicitly generated.

The most simple solution consists in using the **Settings** window.

This window is accessed via the **Open Settings** choice in the Root Proxy's pop-up menu or by double-clicking on the corresponding part.



In the **Proxy Tree** area, click on the Root Proxy.

In this field, specify the value 50 (the default value is that specified on the server).

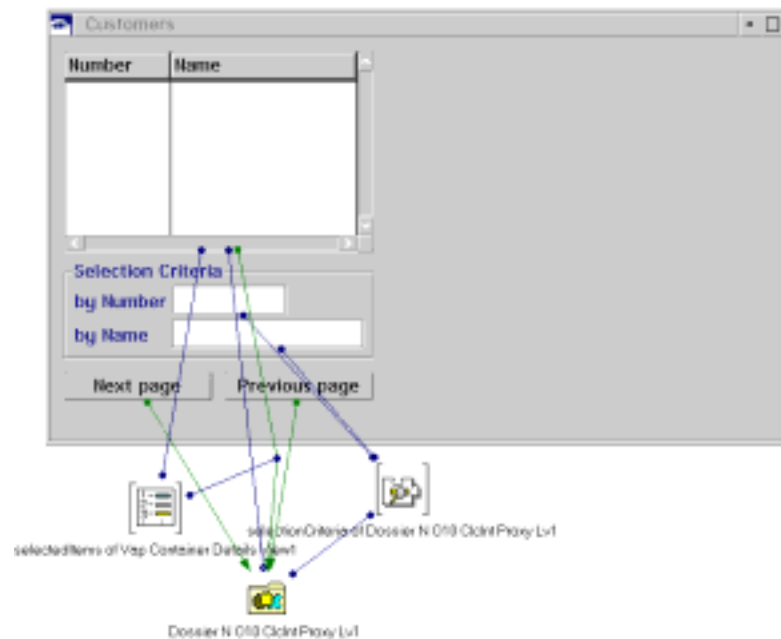
Here you must select the extraction method which will enable the user to make a selection from a customer name (method specified on the server).

Click **ok**.

- Next, you must perform a Quick Form on the **selectionCriteria** attribute of the Root Proxy. You will obtain two entry fields - corresponding to the two selection criteria - with their labels.
- You must now program the **Next Page** and **Previous Page** buttons. To do this, you must connect the **clicked** event of the corresponding buttons with the respective **readNextPage** and **readPreviousPage** actions of the Root Proxy.
- You must now program the transfer of the instance between the list and the detail so that when the user double-clicks on a line in the list, the data associated with the selected customer is displayed in the **Detail** part. To begin with, you must be able to retrieve only one instance from the list since the **rows** attribute which enables this graphic control to be generated is of the Ordered Collection type, thus repeated. To do this, the visual solution consists in tearing off the **selectedItems** attribute from the container. Place the Ordered Collection obtained on the Free Form Surface. At this stage, the Ordered Collection is loaded with several items from the list. You must now retrieve a single instance from the list and transfer it to the **detail** attribute. You must firstly connect the **getDetailFromDataDescription:** action of the Root Proxy to the **defaultActionRequested** event of the container. The link appears as a dotted line because the action requires a parameter: connect the **first** attribute of the Ordered Collection to the **aDataDescription** parameter of the link.

- **Result in the Composition Editor**

Once all the procedures linked to the list have been completed, the Composition Editor should look like this:



### 6.2.3.1.3 Programming Functions Associated with the Detail: detail Attribute

The programming of the different functions linked to the detail consists of the following stages:

- **Creation of the detail**

To begin with, you can redefine the default graphic representation for certain attributes if the representation specified on the server for the corresponding Data Elements of the Logical View does not satisfy your needs.

Proceed by tearing off the Root Proxy's **detail** attribute and place the part obtained on the Free Form Surface.

From this part's pop-up menu, select the **Styles Settings** choice.

In the **Styles Settings** window, assign a new graphic representation to the desired Data Elements.

In our example, a combobox has been assigned to the **City** attribute.

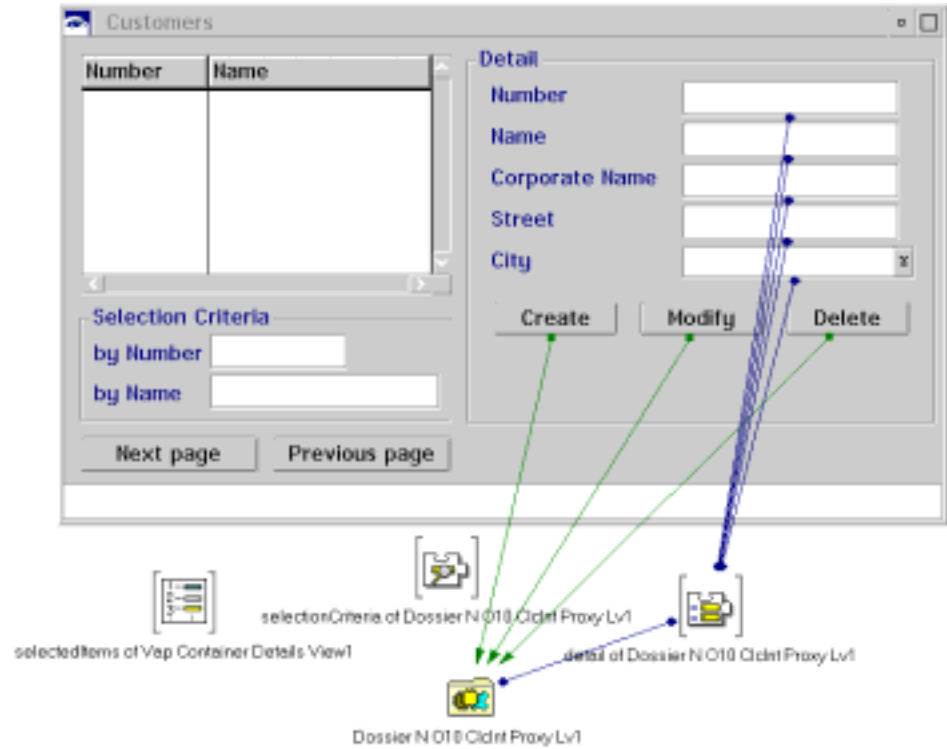
Now, perform a Quick Form on the **detail** part of the Root Proxy to map the fields corresponding to the different attributes.

- **Programming Update Services**

To program the update services, you must connect the `clicked` event of the `Create`, `Modify` and `Delete` buttons to the `createInstance`, `modifyInstance` and `deleteInstance` actions of the Root Proxy, respectively.

- **Result in the Composition Editor**

At the end of this stage, the Composition Editor should look like this:



#### 6.2.3.1.4 Other Functions

Here are the other functions of the window:

- Update of the database by accepting all the customers affected by local updates: `Update` button.
- Display messages sent back by the server after selecting a new set of customers. These messages are used, for example, to find out whether the selected set of customers is the last.
- Navigation to the `Local Updates` window.
- Navigation to the `Orders` window.

- **Programming the Server Update**

You must connect the `clicked` event of the `Update` button to the `updateFolder` action of `dossierNO10CldntProxyLv`, via the `updateFolder` action.

- **Programming the Message Bar**

To program the message bar, you must perform a Quick Form on the `accessInfoLabel` attribute of the Root Proxy.

- **Programming the Navigation to the Local Updates window**



This stage must be carried out once the **Local Updates** window has been built. We presume that this is the case and that in this secondary window, the **self** attribute of the variable Root Proxy has been promoted. The code of the promoted attribute is **dossierNO10ClclntProxyLv1**.

You are now back in the **Customers** window. You must now call the part corresponding to the **Local Updates** window. To do so, you must:

- Insert a View Wrapper part and assign it a name (in the example: **Updates**).
- Connect the **self** attribute of the **dossierNO10ClclntProxyLv1** Root Proxy to the **dossierNO10ClclntProxyLv1** attribute of the View Wrapper.
- Connect the **clicked** event of the **Show Updates** button to the **openWidget** action of the View Wrapper.

- **Programming the navigation towards the Orders window**



This stage must be carried out once the **Orders** window has been built. We presume that this is the case and that in this secondary window, the **self** attribute of the variable Root Proxy has been promoted. The code of the promoted attribute is **dossierNO10ClclntProxyLv1**.

You are now back in the **Customers** window. You must now call the part corresponding to the **Orders** window. To do so, you must:

- Insert a View Wrapper part and assign it a name (in the example: **Orders**).
- Connect the **self** attribute of the **dossierNO10ClclntProxyLv1** Root Proxy to the **dossierNO10ClclntProxyLv1** attribute of the View Wrapper.
- Connect the **clicked** event of the **Orders...** button to the **openWidget** action of the View Wrapper.

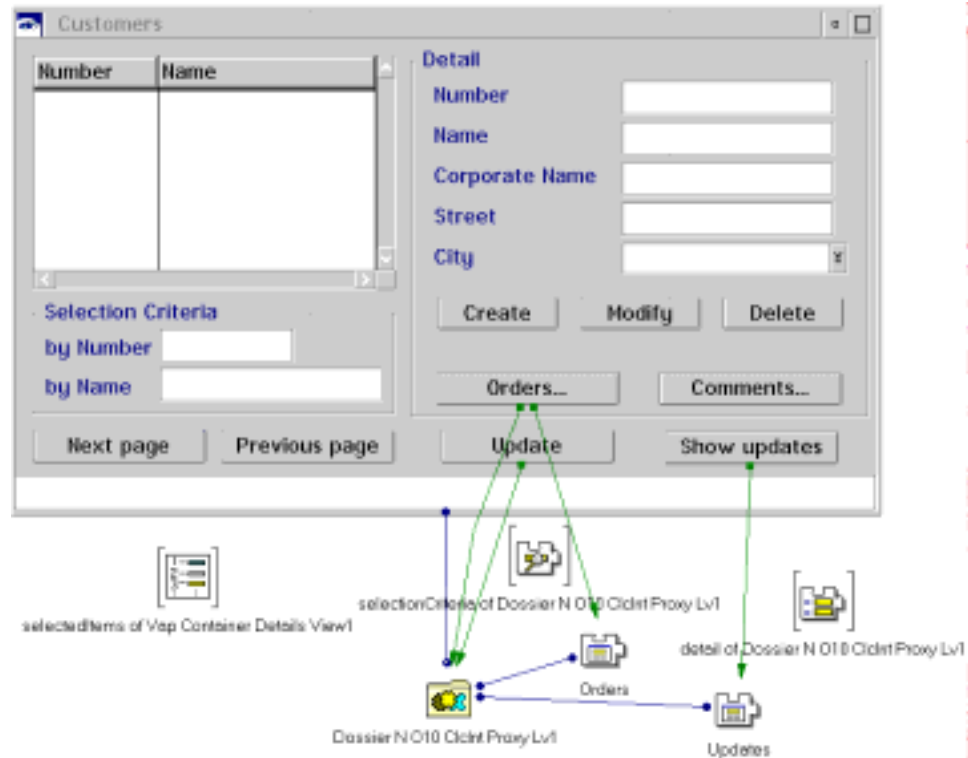
In our example, we have also programmed the following processing: clicking on the **Orders...** button triggers the read - on the server - of the current customer's orders and of the order lines for each of these orders. When the window opens, the two list areas in the **Orders** window will thus be loaded.

To implement this function, you must connect the **clicked** event of the **Orders...** button to the **readAllChildrenFromCurrentInstance** action of the Customers Root Proxy.



• **Result in the Composition Editor**

At the end of these stages, the Composition Editor should look like this:



**6.2.3.2 Development of the Orders Window**

The **Orders** window uses two nodes: **dossierNO20ClcmdeProxyLv** corresponding to the Orders node and **dossierNO30Cl1cdeProxyLv** corresponding to the Order lines node. It is developed in another VisualAge View.

The functions of this window are not presented here: refer to the earlier section **6.2.2**.

The window is split into two parts: the top part handles the Orders node, the bottom part the Order lines node.

**6.2.3.2.1 Integrating the FVP in the Composition Editor**

Having initialized the window, insert the FVP as a variable part on the Free Form Surface. You must then associate the FVP image with this variable part. To do this, you must promote the **self** attribute of the variable part. Once you come back to the **Customers** window, you will use this promoted attribute to connect the View Wrapper part representing the **Orders** window to the **self** attribute of the **dossierNO10ClclntProxyLv1** Root Proxy.

Furthermore, since you have to handle the public interface of the **dossierNO20ClcmdeProxyLv** and **dossierNO30Cl1cdeProxyLv** Depending Elementary Proxies, you must unfold these proxies. In our example, we directly unfold the whole tree structure since we will also use the **dossierNO40ClprodProxyLv** Reference Proxy (Products node) to build a second window in the same VisualAge View.

### 6.2.3.2.2 Programming Functions Associated with the Orders Node

The programming of functions associated with the Orders node includes the following stages:

- **Creation of a group box**

In our example, we have gathered everything associated with the management of orders in a group box: the table displaying the list of orders, the area displaying the detail of an order and push-buttons used to create, modify or delete an order. The label of this group box is **Customer Name**. For the end user, this label is contextual: it corresponds to the name of the customer selected in the detail of the **Customers** window.

Having positioned the group box in the window, you must proceed as follows to implement this behavior:

- Tear off the **detail** attribute of the Root Proxy (Customers node) and place the variable part obtained on the Free Form Surface.
- Connect the variable part using the attribute corresponding to the Customer Name Data Element in the Logical View to the **label** attribute of the group box.

- **Procedures linked to the list**

The implementation of procedures related to the list consists of several stages:

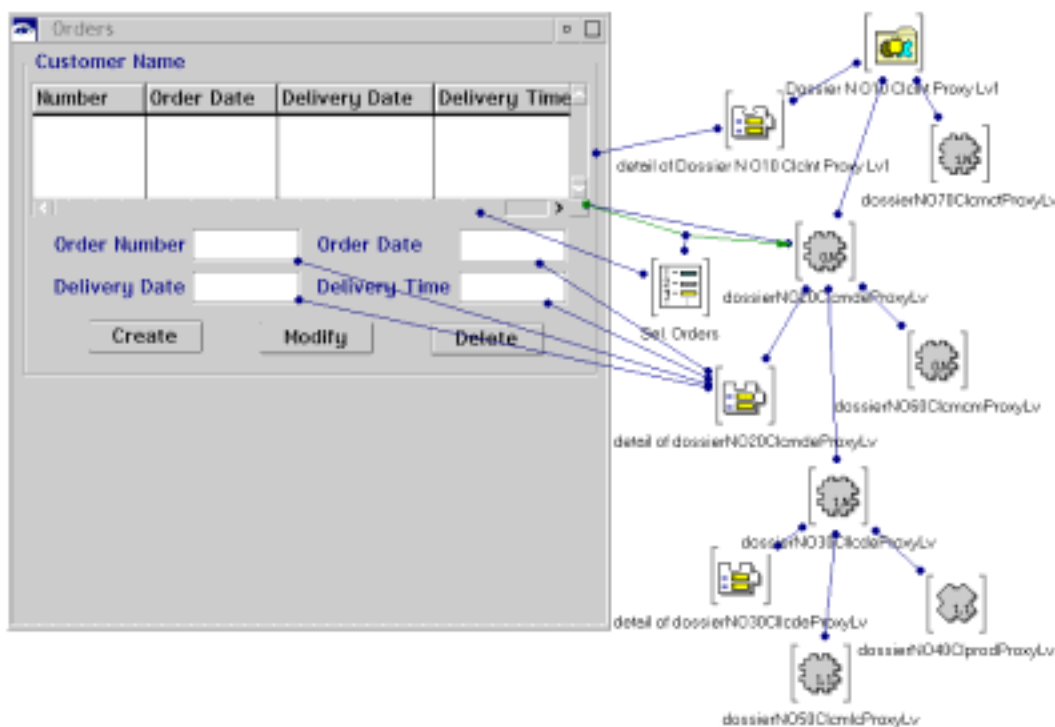
- Make a Quick Form on the **rows** attribute attribute of the **dossierNO20ClcmdeProxyLv** Depending Proxy. Insert the container obtained in the group box.
- You must then program the transfer of the instance selected in the list of orders in the area corresponding to an order detail. The operations required for the programming of the transfer are identical to those required for the Customers node. Refer to point **6.2.3.1.2**.

- **Procedures linked to the detail**

- Make a Quick Form on the **detail** attribute (if necessary after tearing off this attribute to modify the default graphic representation for one or more Data Elements via the **Styles Settings** window).
- To program the update services, you must connect the **clicked** event of the **Create, Modify** and **Delete** buttons to the **createInstance, modifyInstance** and **deleteInstance** actions, respectively, of the Depending Proxy.

- **Result in the Composition Editor**

At the end of these stages, the Composition Editor should look like this:



For better readability, the links between the **Create**, **Modify** and **Delete** buttons and the `dossierNO20ClcdeProxyLv` Depending Proxy are hidden.

### 6.2.3.2.3 Programming Functions Associated with the Order Lines Node

The programming of functions associated with the Order Lines node includes the following stages:

- **Creation of a group box**

In our example, we have gathered everything associated with the management of order lines in a group box: the table displaying the list of order lines, the area displaying an order line detail and push-buttons used to create, modify or delete an order line. The label of this group box is **Order Items**.

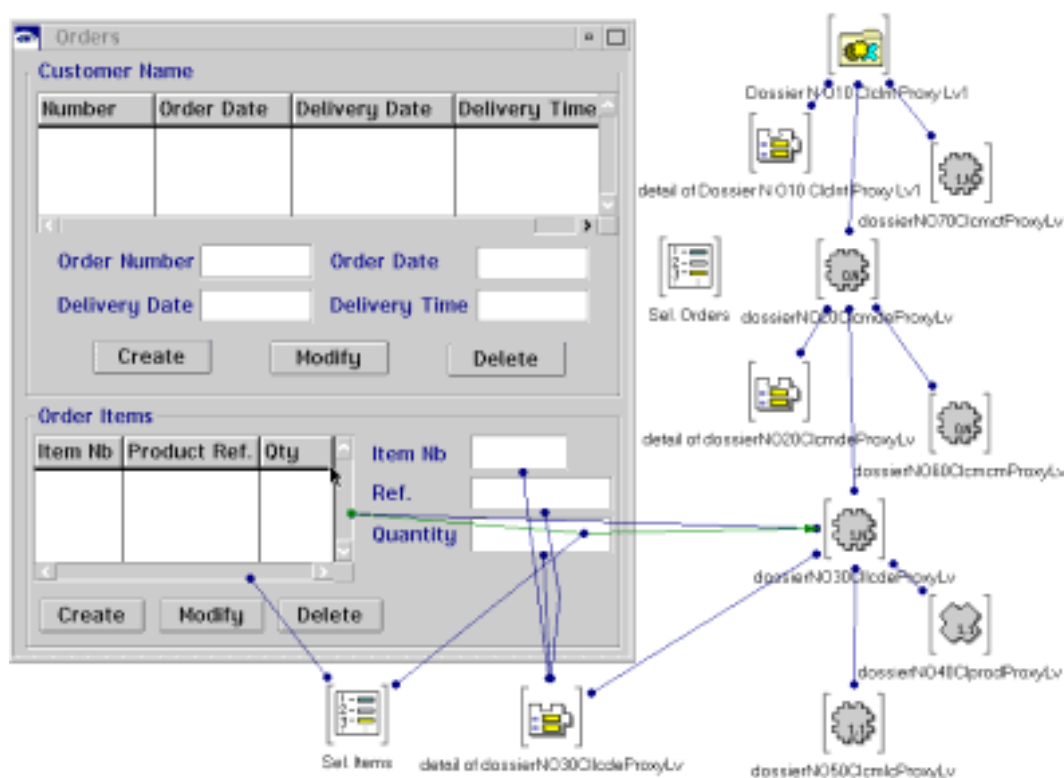
- **Procedures linked to the list**

The implementation of procedures related to the list consists of several stages.

- Make a Quick Form on the `rows` attribute of the `dossierNO30ClcdeProxyLv` Depending Proxy. Insert the container obtained in the group box.
- You must then program the transfer of the instance selected in the list of order lines in the area corresponding to an order line detail. As with the Orders node, the operations required for programming the transfer are identical to those required for the Customers node. Refer to section [6.2.3.1.2](#).

- **Procedures linked to the detail**
  - Generate a Quick Form for the **detail** attribute of the **dossierNO30C11cdeProxyLv** Proxy (if necessary after tearing off this same attribute to modify the default graphic representation of one or more Data Elements via the **Styles Settings** window).
  - To program the update services, you must connect the **clicked** event of the **Create**, **Modify** and **Delete** buttons to the **createInstance**, **modifyInstance** and **deleteInstance** actions, respectively, of the Depending Proxy.
- **Result in the Composition Editor**

At the end of these stages, the Composition Editor should look like this:



For better readability, the links between the **Create**, **Modify** and **Delete** buttons and the **dossierNO30C11cdeProxyLv** Depending Proxy are hidden.

### 6.2.3.3 Development of the Product References Window

The **Product References** window uses the **dossierNO40C1prodProxyLv** Reference Proxy corresponding to the Products node. This window is developed in the same VisualAge View as the **Orders** window.

- **Development of the window**

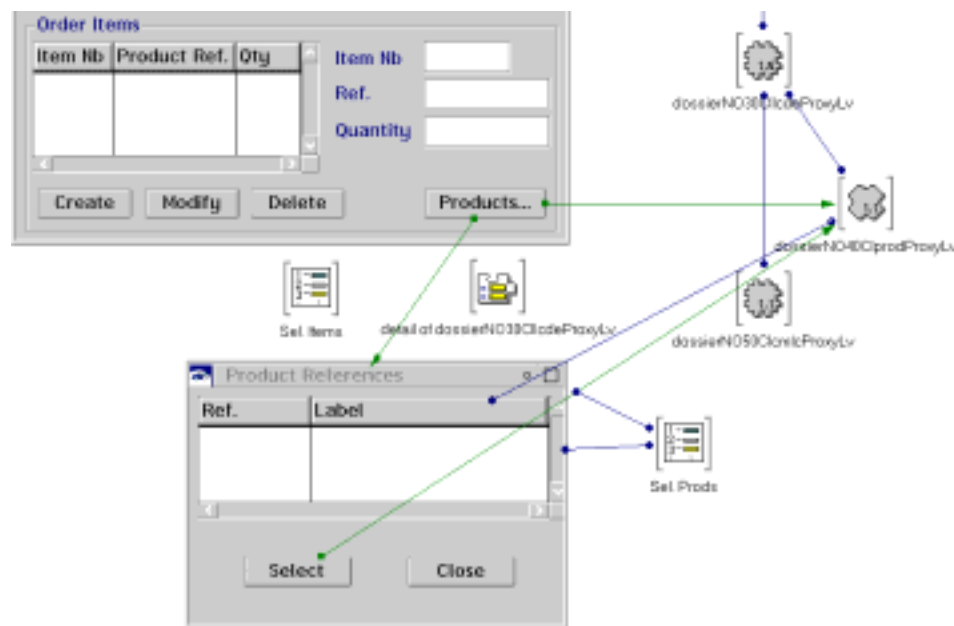
To program this window, you must:

- Initialize the **Product References** window.
- Connect the **clicked** event of the **Products** button to the **openWidget** action of the **Product References** window.
- Connect the **clicked** event of the **Products** button to the **selectInstances** action of the **dossierNO40C1prodProxyLv** Proxy Reference.

- Make a Quick Form on the **rows** attribute of the Reference Proxy and place the obtained container in the window.
- Tear off the container's **selectedItems** attribute and place the obtained Ordered Collection on the Free Form Surface.
- You must now program the transfer of a single line in the products table to the order line detail. Proceed by connecting the **clicked** event of the **Select** button to the **transferReferenceFromSelectedRow::** action of the Reference Proxy. Then connect the **first** attribute of the Ordered Collection to the **aDataDescription** parameter of the link.

• **Result in the Composition Editor**

The Composition Editor should look like this:



## 6.3 Example of a Web Application

### 6.3.1 Introduction

A dynamic Web application developed with Pacbench C/S uses the same application logic as a standard GUI application. With Pacbench C/S, the use of the Web Connection feature presents some specifics which are described in this sub-chapter.

The necessary prerequisites for creating a Web application are described in chapter *Pacbench C/S, Vol. I : Concepts – Architectures – Environments User's Guide*.

Pacbench C/S also supplies two files on the installation CD-ROM which you can install before developing your Web application. These extensions, usable for developing in folder or simple mode, are there to make programming easier.


#### 6.3.1.1 Automatic Conversion of HtmlFormData

The data entered by an end-user in a browser is always in text format. The **cvfdata.dat** extension allows VisualAge to receive date or integer type data, which has already been converted by using the converters associated with the input fields when programming the HTML page. To load this file, carry out the following operations in the Organizer:

- Go to the **AbtRunHtmlPageApp** application. If it is not displayed, select the **View..., Show All Applications** option in the **Applications** menu.
- Then, in the **Parts** menu, select **Import/Export, Import**. In the dialog box that appears, select the **cvfdata.dat** file and then click on **OK**.
- Load the last version of the **AbtRunHtmlPageApp** and **AbtEditHtmlPageApp** applications.
- If they have been imported correctly, the **Web Connection** category from the Composition Editor parts palette contains the following extra part: **HTML Converted Form Data**.

#### 6.3.1.2 HTML Quick-Form Extension

The **qkhtml20.st** extension provides an HTML Quick-Form which behaves identically to a C/S Facility standard Quick-Form function. Also, it allows you to create an HTML table automatically (set of tags allowing the presentation of data in the form of identically sized cells) containing the labels and input fields of the attributes of a proxy component. This extension presents the information elegantly without having to generate HTML tags.

 The **qkhtml20.st** file includes the **R2VA.ALL** file, which is the extension of the VisualAge HTML Quick-Form function.

The **qkhtml20.st** extension generates a graphical representation of the Data Elements of a Logical View as a detail.

To file in this extension, carry out the following operations in the System Transcript:

- In the **File** menu select **Open....** Select the **qkhtml20.st** file. A workspace opens with the file contents.
- Select all of the file contents, then the **File In** option in the **Edit** menu. The code is filed in.
- If the code has been successfully filed in, the **self as HTML table** option appears in the pop-up sub-menu - which opens when you select the **Quick HTML** option - of the torn-off **detail** attribute of a Proxy. The **self as HTML table** option generates the Proxy attributes' update fields as well as the tags that make up an HTML table.

### 6.3.2 Managing an Application Context

For Pacbench C/S, if the application that you want to create uses several window parts, you have to handle an application context in which the same Proxy is used during a dialog with the user.

This implies that a **CGI Link Session Data** part must be used in each page which calls this proxy component. In the **Settings** window of the **CGI Link Session Data** part, you have to indicate the name of the proxy component used in the **Data type of the value** field. Tearing off the **value** attribute from the **CGI Link Session Data** part allows you to obtain a variable instance of the proxy component.



For more information on the **CGI Link Session Data part**, refer to the *VisualAge Web Connection User's Guide*.

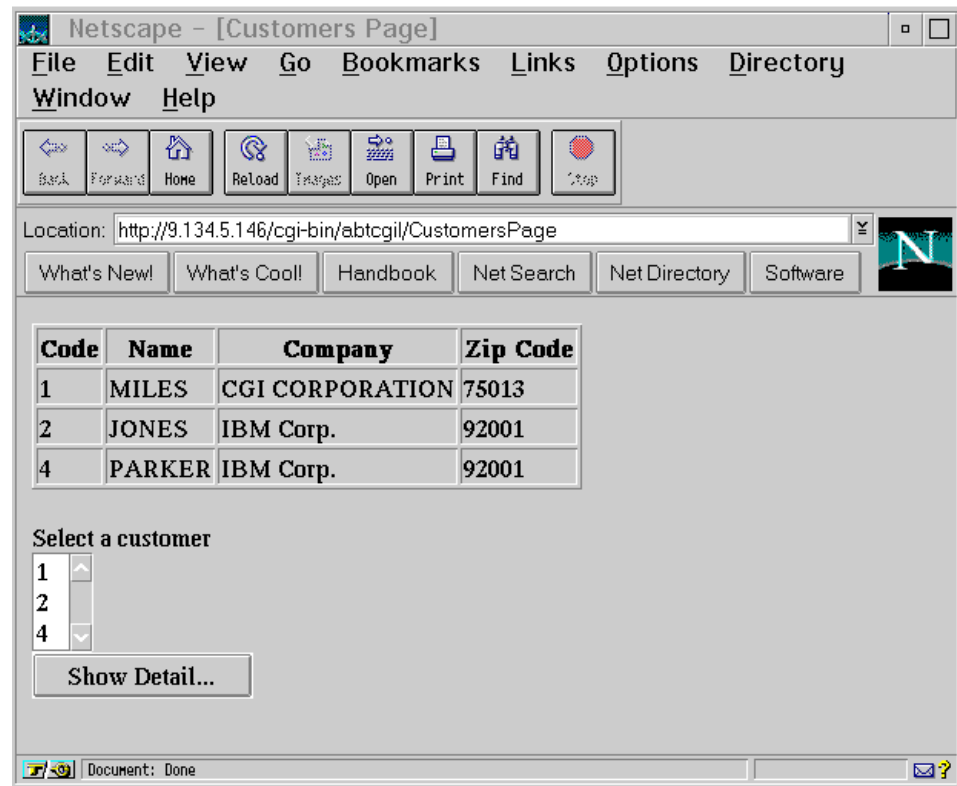
### 6.3.3 Example: Presentation of the User Interface

This application is based on the same Folder View Proxy as that used to develop the standard GUI application.

The example manipulates a single Elementary Proxy, the **dossierNO10clclntProxyLv** Root Proxy, corresponding to the Business Component/Logical View pair which manages the customers in the company's information system.

The application contains two pages:

- **The Customers page**

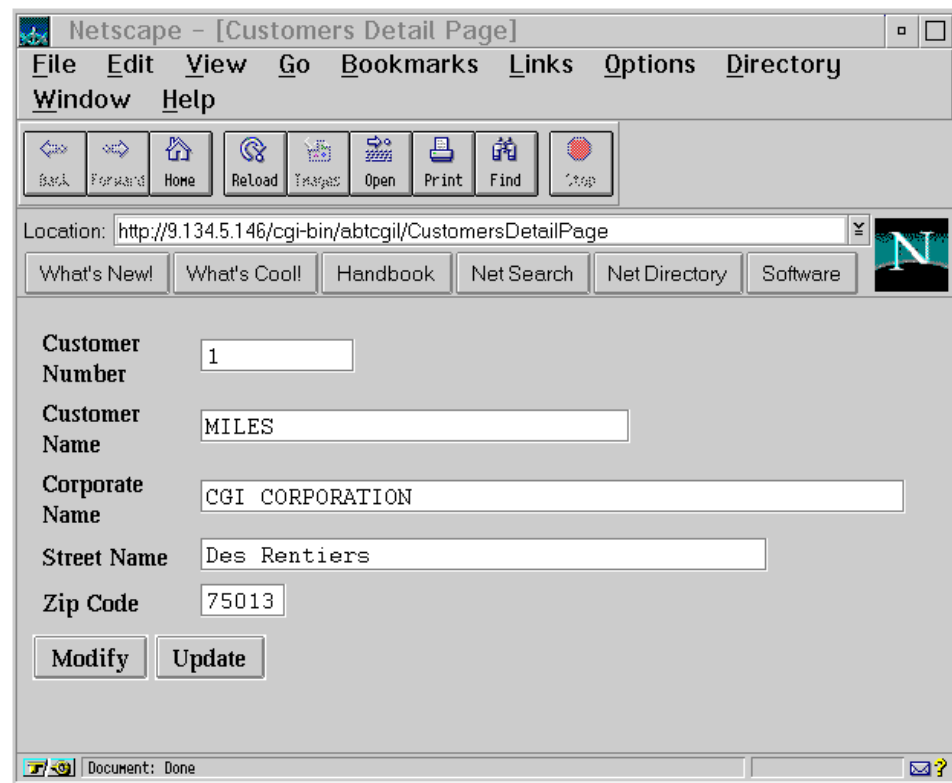


The functions of this page are as follows:

- display, from the time the page is opened, the list of customers in the table containing several columns, each column corresponding to a Logical View Data Element. As the Table is a static component, the user cannot select a customer in the list.
- display an Elementary Proxy attribute, in this case the customer number, in a list which allows the selection.
- navigate to a second page which presents the detail of the selected customer when the user clicks on **Show Detail...**



- **The Customers Detail Page**



The functions of this page are as follows:

- display the information corresponding to the customer previously selected in the **Customers** page in the update fields.
- modify this information: the user must click on the **Modify** button once the modifications are entered in the different fields.
- update the database by taking the modification into account via the **Update** button.

### 6.3.4 Example: Developing the End User Interface

This section details neither the use of the Composition Editor nor the standard use of the Web Connection and the HTML parts. For more information, refer to the various manuals associated with VisualAge which were mentioned at the start of the manual.

It is presumed that you have read the *VisualAge Web Connection User's Guide* before developing this example.

### 6.3.4.1 Developing the Customers Page

Once this page is programmed, the Composition Editor looks like this:



The operations described hereafter are necessary to achieve this result.

- **Application context management and integration of the Folder View Proxy in the Composition Editor**

After initializing the HTML page, you must integrate the FVP into the Composition Editor.

Because two pages are used in the application, a **CGI Link Session Data** part is needed in each of the Pages in order to manage the application context during the same dialog with the end user.

- From the parts palette, select the Web Connection category, then the **CGI Link Session Data** part. Place this part on the Free Form Surface.
- In the **Settings** window for this part, indicate the name of the Root Proxy in the **Data type of the value** field.
- Tear off the **value** attribute from the **CGI Link Session Data** part to obtain the Root Proxy. Place this newly obtained Root Proxy on the Free Form Surface.

- **Programming the display of the customers list in an HTML static table**

This table displays the data in a table resembling a container, but unlike a container is not interactive: the user therefore cannot select a customer for example.

- In the parts palette, Web Connection category, select the **Html Table** part. Place it in the HTML page.
- Then select the **HTML Table Column** part and place it in the **HTML Table** part. Open the **HTML Table Column** part's **Settings** window:
  - ♦ in the **Heading** field, indicate the label that you want displayed for the column in the end-user application.
  - ♦ in the **Data type** field, indicate the type of data for the Data Element (Integer, String, Date...).
  - ♦ In the **Attribute to display** field, indicate the code of the corresponding Data Element.
- Create, in the same way, as many **HTML Table Column** parts as you want columns, i.e. Data Elements, in your table.
- Connect the **rows** attribute of the Root Proxy to the **items** attribute of the **HTML Table** part.
- In order for the table to be filled as soon as the user accesses the page, connect the **selectInstances** action of the Root Proxy to the **aboutToGenerateHtml** event of the **HTML Table** part.

- **Using a form in the page**

The other functions of the page require the use of a form or **HTML Form** part, i.e. the display of the identifier type Data Elements in a list which allows selection and navigation to the **CustomersDetail** page.

**Note** The forms make the behaviour of a Web application and that of a standard GUI application more similar. They allow the integration of similar graphic controls in an HTML page. The forms therefore allow dynamic Web applications to be developed.

To use a form, carry out the following operations:

- Insert an **HTML Form** part in an HTML page.
- In this part's **Settings** window, you must:
  - ♦ select **Part name** in the **Server location** field
  - ♦ enter the name of the HTML page (in our example the name is **CustomersPage**) in the **Name** field. You specify this way the page in which the **HTML Form** part is used and consequently that the processes carried out from the parts contained in the **HTML Form** part are carried out in the indicated page.

- **Programming the display of the identifiers in a dynamic list**

- The **Select a customer** label is an **HTML Text** part.
- To obtain the list, perform a Quick HTML on the Root Proxy's **rows** attribute. Place the part in the **HTML Form part**. The following elements must be indicated in the part's **Settings** window:
  - ♦ For **Data type**, select **Integer**
  - ♦ For **Attribute to display**, enter the code of the identifier type Data Element.

By default, the name of the part is **rowsList**.

- The programming of the display of identifier type Data Elements in this list includes the following steps:
  - ♦ Connect the **rows** attribute of the Root Proxy to the **items** attribute of the list.
  - ♦ Tear off the Root Proxy's **selectionCriteria** attribute and place the part obtained on the Free Form Surface.

- **Programming the loading of the CustomersDetail page**

This step consists in programming the loading of the **CustomersDetail** page with the information relative to the customer selected in the list when the user clicks on **Show Detail....**

- Insert an **HTML Converted Form Data** part on the Free Form Surface. In this part's **Settings** window, for **Name of the part containing the page**, indicate the name of the page in which the processes will be carried out (in our example the name is **CustomersDetailPage**).
- Insert a button, change its label (here **Show Detail....**), then in the pop-up menu of this part select the **Change name...** choice. In the window that opens, enter the name for this button (for example, **ShowDetailButton**).

The **ShowDetailButton clicked** event is then added to the public interface of the **HTML Converted Form Data** part.

- Connect the **ShowDetailButton clicked** event of the **HTML Converted Form Data** part to the torn-off **selectionCriteria** part via the attribute corresponding to the code of the identifier type Data Element. The link requires a parameter: connect the **rowsList** attribute of the **HTML Converted Form Data** part to the **value** parameter of the link.
- Connect the **ShowDetailButton clicked** event of the **HTML Converted Form Data** part to the Root Proxy's **readInstance** action.

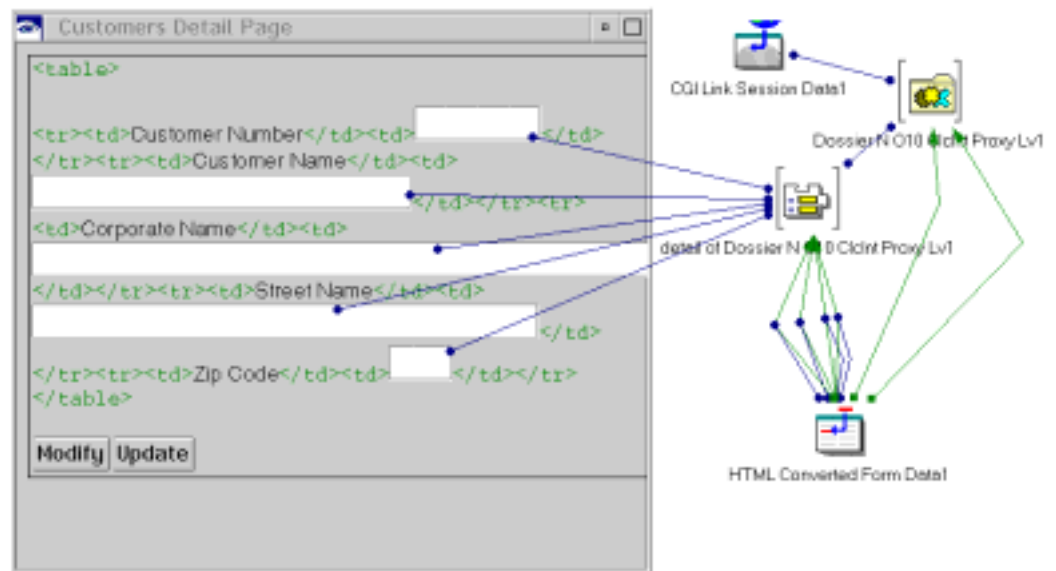
- **Programming the navigation to the CustomersDetail page**

- Insert an **HTML Page Wrapper** part.
- Connect the **ShowDetailButton clicked** event of the **HTML Converted Form Data** part to the **transferRequest** event of the **HTML Page Wrapper** part.
- This step supposes that the secondary **CustomersDetail** page has been built. When this page was built, the Root Proxy's **self** attribute was promoted and this promoted attribute's code is **dossierNO10ClclntProxyLv**.

Back to the main page, connect the **self** attribute of the Root Proxy to the **dossierNO10ClclntProxyLv** attribute of the **HTML Page Wrapper** part.

### 6.3.4.2 Programming the CustomersDetail Page

Once this page is programmed, the Composition Editor should look like this:



- **Programming the application context management and integrating the Folder View Proxy in the Composition Editor**

As the operations to carry out in this step are identical to those described for the **Customers** page, refer to the corresponding section.

- **Programming the display of a customer's detail**

To fill up the update fields with the information corresponding to the previously selected customer:

- Tear off the Root Proxy's `detail` attribute and place the obtained part on the Free Form Surface. By default, the name of each generated update field is coded: `DataElementCodeField`, the `Field` suffix being invariable.
- open the pop-up menu for this part and select the Quick HTML option. Select `self as HTML Table` from the list that appears.
- Place the result in the `HTML Form` part.

- **Programming the modification of a customer**

For this step and the next one, you must use the `HTML Converted Form Data` part. Refer to the corresponding step in the section dedicated to developing the `Customers` page.

It is a local update service ensured by the `Modify` button. To program this button:

- Insert a button, change its label and then change its name to `ModifyButton` for example.
- you must then:
  - ♦ connect the `ModifyButton clicked` event of the `HTML Converted Form Data` part to the part torn-off from the Root Proxy's `detail` attribute using the attribute corresponding to the Data Element code associated with an update field.
  - ♦ then, the link created requiring a parameter, connect the corresponding `DataElementCodeField` attribute of the `HTML Converted Form Data` part to the `value` attribute of the link.

Renew these two successive operations for each update field.

- Connect the `ModifyButton clicked` event of the `HTML Converted Form Data` part to the Root Proxy's `modifyInstance` action.

- **Programming the server update**

The server update is ensured by the Update button. To program it, you must:

- insert a button, change its label, then change its name to `UpdateButton` for example.
- connect the `UpdateButton clicked` event of the `HTML Converted Form Data` part to the Root Proxy's `updateFolder` action.

## 6.4 Error Management

There is no error message displayed during the extraction and generation phases.

To be extracted, a Business Component must have been compiled correctly, because:

- the GVC command in the GPRT procedure extracts the required data without displaying any error messages, even if the Business Component does not contain any Logical View,
- The generator does not control the input file.

However, some error messages can be displayed during the development, test and execution phases of the application. These messages result from local, server or communication errors.

### 6.4.1 Principles

#### 6.4.1.1 Introduction

In VisualAge, error management is ensured by the error manager via a generic class handling all the error types.

The four error types are:

- the local errors, sent by the Client component. They correspond to manipulation or input errors in the Client Component,
- the server errors, sent by the Business Component which are data access errors and user errors set in the server,
- the system errors, sent as well by the Business Component. They correspond to a discrepancy between the Proxy and the Business Logic.
- the communication errors.

The public interface of the error manager includes all the attributes and events required to program errors in the graphic application.

You can customize all error labels in a dedicated file to this use.

#### 6.4.1.2 Event-driven Error Handling

##### 6.4.1.2.1 Local Update Services

After execution, each local update service sends an event which characterizes the result of the executed procedure:

- `noError` when the procedure is correctly executed
- `localError` when the procedure is not correctly executed. In this case, the list of error messages and list of their keys are retrieved in the Error Manager.

### 6.4.1.2.2 Server Update Services

After execution, each server update service sends an event which characterizes the result of the executed procedure. The following different events are likely to be sent:

- **noError** when the procedure is correctly executed
- **serverError** which characterizes two types of functional errors: database access error or user error
- **systemError** which characterizes a synchronization error between the descriptions contained in the different application components
- **fatalError** which characterizes a communication error.

### 6.4.1.3 Visual Programming

The **errorManager** attribute of the root Proxy ensures the error access. You must tear-off this attribute.

The **errorManager** attribute contains an instance of the **VapErrorManager** class which manipulates instances of the **VapError** class. An instance of the **VapError** class represents a local, a server, a system or a communication error.

The instance of the **VapErrorManager** class contains, in its **label** attribute, the error's associated label. This label is read in the **VAPERROR.txt** errors file set up in the VisualAge directory. This file contains all the error label types: local, server, system or communication errors. You can customize these error labels in this file.



The server errors are usually returned by the server. If the server cannot return a particular error or if the label value is blank in the error labels file of the Repository, the error manager fetches the label which corresponds to the key returned in the **vaperror.txt** file.



For more information on the error customizing in the server side, refer to the *Pacbench C/S, Vol. II – Business Logic User's Guide*.

In the application, the management of the error returned by the error manager is based on the creation of connections between different events related to the presence or absence of errors. It is also based on the creation of graphic part's actions.



It is absolutely necessary to connect at least the **localError**, **serverError**, **systemError** and **fatalError** events on the **openWidget** action in a window displaying the **errorList** attribute.

### 6.4.1.4 Customization of the Standard Labels

All error messages are stored in the **vaperror.txt**, installed in VisualAge's directory. When the system is initialized, this file is used to generate an error instance dictionary, which is handled by the **VapErrorManager** class. Developers can customize or add their own messages. These messages can be:

- Errors detected in the client processing.
- User errors detected on the server (**ERU** operator).





If you reinstall, make sure you save the `vaperror.txt` file which you have customized. Possible new elements contained in the file loaded at installation are specified in the `readme` file. You must copy these modifications and paste them in your customized file. Then replace the new file with your own file.

Each error is described in the `vaperror.txt` file on five lines. The French message is on the line beginning with `FLAB` and the English message on the line beginning with `ELAB`. To customize an error message, you just need to overwrite the corresponding line.

Here is an example of an error description as it is presented in this file:

```
ERR:UnknownInstance
FLAB:Instance inconnue
ELAB: Unknown instance
KEY:,,, 'UNKNOW' ,,'L'
TYP:LocalError
```

For example, you can change "Unknown instance " for "selection of an unknown instance ", or change the English label for a message in another language.

#### 6.4.1.5 Error Key Structure

Knowing the error key structure will enable you to manage local error messages in a file different from the `VAPERROR.TXT` file provided. You will have to link labels and the errors' specific keys.

This table contains the errors which are more likely to occur.

| Col 1-3 | Col 4-9 | Col 10-13 | Col 14-19 | Col 20 | Col 21 | Col 22-25  |  |
|---------|---------|-----------|-----------|--------|--------|------------|--|
| lib     | ser     | seg       |           |        | S      | DUPL       | Invalid creation [S]                           |
| lib     | ser     | seg       |           |        | S      | NFND       | Invalid deletion/modification [S]              |
| lib     | ser     |           |           |        | S      | error code | User error [S]                                 |
| lib     | ser     | vie       | dte       | 2      | S      |            | Required Data Element [S]                      |
| lib     | ser     | vie       | dte       | 5      | S      |            | Value error [S]                                |
| lib     | ser     | vie       | ACCESS    |        | S      |            | Data access error [S]                          |
| lib     | ser     | STRU      |           |        | S      |            | View structure error [S]                       |
| lib     | ser     | VERS      |           |        | S      |            | Version error [S]                              |
| lib     | ser     | VIEW      |           |        | S      |            | Unknown View [S]                               |
| lib     | ser     | SERV      |           |        | S      |            | Unknown Service [S]                            |
| lib     | ser     | LTH       |           |        | S      |            | View length error [S]                          |
|         |         |           | MISPCV    |        | S      |            | Components out of phase [S]                    |
|         |         | OPEN      |           |        | C      |            | Open server error [C]                          |
|         |         | CALL      |           |        | C      |            | Call server error [C]                          |
|         |         | CLOS      |           |        | C      |            | Close server error [C]                         |
|         |         |           | CRDVIO    |        | L      |            | Cardinality violation [L]                      |
|         |         |           | CREATE    |        | L      |            | Invalid creation [L]                           |
|         |         |           | DELETE    |        | L      |            | Invalid deletion [L]                           |
|         |         |           | EXTR      |        | L      |            | Invalid extraction method [L]                  |
|         |         |           | FOLDER    |        | L      |            | Folder not updatable [L]                       |
|         |         |           | HIERAR    |        | L      |            | Hierarchy violation [L]                        |
|         |         |           | INVINS    |        | L      |            | Invalid instance [L]                           |
|         |         |           | LENGTH    |        | L      |            | Length error [L]                               |
|         |         |           | LOCATE    |        | L      |            | Unavailable server environment [L]             |
|         |         |           | LOCKED    |        | L      |            | Already locked instance[L,S]                   |
|         |         |           | MISINS    |        | L      |            | Current Instance missing [L]                   |
|         |         |           | MISPAR    |        | L      |            | Parent instance missing [L]                    |
|         |         |           | MISREF    |        | L      |            | Referencing instance missing [L]               |
|         |         |           | MISUSR    |        | L      |            | Current user service instance missing [L]      |
|         |         |           | MODIFY    |        | L      |            | Invalid modification [L]                       |
|         |         |           | NLOCIN    |        | L      |            | Not a lockable instance [L]                    |
|         |         |           | NTLOCK    |        | L      |            | Instance not locked [L,S]                      |
|         |         |           | REQUIR    |        | L      |            | Required Data element [L]                      |
|         |         |           | SUBSCH    |        | L      |            | Data Element does not belong to sub-schema [L] |
|         |         |           | SUSER     |        | L      |            | Invalid user service [L]                       |
|         |         |           | UNKNOW    |        | L      |            | Unknown Instance [L]                           |
|         |         |           | VALUE     |        | L      |            | Value error [L]                                |
|         |         |           | UPDTRQ    |        | L      |            | Server update required [L]                     |

*Legend:*

lib = library code

vie = Logical View code

ser = server code

dte = Data Element code

seg = physical access segment code

[S] = server error

[L] = local error

[C] = communication error

## 6.4.2 Errors list

### 6.4.2.1 Local Errors

The local error messages are listed below:

- **Unknown instance**

This message is displayed when an instance unknown in the local cache is selected.

- **Parent instance missing**

This message is displayed when a depending node instance is selected whereas the parent instance is not found.

- **Instance already locked**

This message is displayed when a lock is requested for an instance which is already locked.

- **Instance not locked**

This message is displayed when an unlock is requested for an instance which is already unlocked.

- **Not a lockable instance**

This message is displayed when the lock action is used on an instance whereas the lock has not been implemented on the server.

- **Current instance missing**

This message is displayed when an action is applied to a **detail** attribute whereas this attribute does not contain an instance.

- **Current user service instance missing**

This message is displayed when a user action is applied to the **userDetail** attribute whereas this attribute does not contain any instance.

- **Referencing instance missing**

This message is displayed when the **transferReferenceFrom-SelectedRow:** action is applied to a reference node instance whereas no instance has been selected in the **detail** attribute of the referencing node.

- **Invalid instance**

This message is displayed when a check is made on values which have not been respected.

- **Folder not updatable**

This message is displayed when an update action is applied to an unlocked Folder whereas the logical lock is set on the Server. To update it, the user must first lock the Folder instance.

- **Invalid creation**

This message is displayed when an instance is created which already exists in the local cache.

- **Invalid modification**

This message is displayed when the instance to be modified does not exist in the local cache.

- **Invalid deletion**

This message is displayed when the instance to be deleted does not exist in the local cache.

- **Server update required**

This message is displayed when an action is applied to an instance whereas the superior instance which has been locally created does not yet exist in the database. A server update of the superior instance is required beforehand.

- **Cardinality violation**

This message is displayed when cardinal values are not respected during the activation of an update action.

- **Hierarchy violation**

This message is displayed when a depending node instance is selected whereas it does not depend on the instance contained in the **detail** attribute of the parent node.

- **Required Data Element**

When creating or modifying an instance, one of the Logical View Data Elements other than a key Data Element has not been filled in.

- **Length error**

This message is displayed when the length of the value entered in a field of the **detail** attribute exceeds the maximum length of the value for this attribute.

- **Value error**

This message is displayed when the value entered for a Data Element is not present in the value table for this Data Element. This error is not displayed if the user selects one of the values provided in the table.

- **Unavailable server environment**

This message is displayed when the middleware is initialized if the parameters required for its implementation are incorrect.

- **Unknown context**

This message is displayed when the response identifier transmitted is not recognized by the Proxy for the current location at the execution of the `getReplyOf:aReplyId` and `resetPendingReplyOf:aReplyId` actions.

- **Field is out of subschema**

This message is displayed when a Data Element belonging to the current instance is updated whereas it was not filled in for this instance on a server access parameterized with a subschema.

- **Action and Protocol**

This message is displayed in the case of an unauthorized request.

- **Reply overflow**

This message is displayed when the `Reply overflow` message is displayed for an asynchronous request. In such a case, the request is not sent.

#### 6.4.2.2 Server errors

These messages are generated in standard by the Business Components. There are three types of messages:

- Irretrievable access errors: **Data access error**

These are irretrievable file or relational database access errors.

- Logical access errors: **Invalid creation** and **Invalid deletion or modification**.

- Control errors on Logical View fields: **Value error** and **Data Element required**.

#### 6.4.2.3 System Errors

The system errors are:

- **Unknown service**

This message is displayed when the service requested by the Proxy is not recognized by the Business Component.

- **View length error**

This message is displayed when the format of a Logical View associated with the Proxy changes and when this Proxy cannot be regenerated.

To solve this problem, you must regenerate the Proxy.

- **Components out of phase**

This error occurs when the Client and Business Components are out of phase.

To solve this problem, you must regenerate the Proxy.

#### 6.4.2.4 Communication Errors

If a communication error message is displayed, inform the person in charge of the communication because the line may be blocked or defective, or a Server may be busy, etc.

Three communication error messages are likely to be displayed:

- Open server error
- Call server error
- Close server error

### 6.4.3 System Errors Generated by the Communication Monitor and the Services Manager

Most of these errors are internal errors that you can solve only by contacting the VisualAge Pacbase support.

#### 6.4.3.1 Errors Generated by the Communication Monitor

| Col 1-3 | Col 4-9 | Col 10-13 | Col 21 |  |
|---------|---------|-----------|--------|--|
| lib     | mon     | LSRV      | S      | Erroneous length of the message received                 |
| lib     | mon     | PNUM      | S      | Erroneous structure of the "service number" parameter    |
| lib     | mon     | PCOD      | S      | Erroneous structure of the "service code" parameter      |
| lib     | mon     | PNOD      | S      | Unknown Folder code                                      |
| lib     | mon     | PCVS      | S      | Erroneous structure of a service request from the client |
| lib     | mon     | PCVF      | S      | Erroneous structure of the message from the client       |
| lib     | mon     | WF00      | S      | Erroneous access to the work file                        |

*Legend*    lib = library code                      mon = Communication Monitor code                      S = System error

#### 6.4.3.2 Errors Generated by the Services Manager

| Col 1-3 | Col 4-9 | Col 10-13 | Col 21 |  |
|---------|---------|-----------|--------|--|
| lib     | serv    | SRV1      | S      | Service not found in the work file                                 |
| lib     | serv    | LNG1      | S      | Erroneous conversion of service length on a multi-messages request |
| lib     | serv    | LNG2      | S      | Erroneous conversion of service length on a single-message request |
| lib     | serv    | NOS1      | S      | Erroneous structure of the "service number" parameter              |
| lib     | serv    | NOS2      | S      | Erroneous conversion of the "service number" parameter             |
| lib     | serv    | SRV1      | S      | Erroneous structure of the "service code" parameter                |
| lib     | serv    | SRV2      | S      | Unknown service code in the Folder                                 |
| lib     | serv    | DOS1      | S      | Missing "Folder name" parameter                                    |
| lib     | serv    | DOS2      | S      | Erroneous length of the "Folder name" parameter                    |
| lib     | serv    | VER2      | S      | Erroneous length of the "version number" parameter                 |
| lib     | serv    | NOD1      | S      | Missing "node name" parameter                                      |
| lib     | serv    | NOD2      | S      | Erroneous length of the "node name" parameter                      |
| lib     | serv    | NOD3      | S      | Unknown node name in the Folder                                    |
| lib     | serv    | TYNO      | S      | Unauthorized service on the node                                   |

| Col 1-3 | Col 4-9 | Col 10-13 | Col 21 |   |
|---------|---------|-----------|--------|---|
| lib     | serv    | SCH1      | S      | Erroneous structure of the "sub-schema code" parameter                                |
| lib     | serv    | SCH2      | S      | Unknown sub-schema code in the Folder   |
| lib     | serv    | NOCP      | S      | Erroneous structure of the "number of occurrences" parameter                          |
| lib     | serv    | NOC1      | S      | Erroneous length of the " number of occurrences" parameter                            |
| lib     | serv    | NOC2      | S      | Erroneous conversion of the "number of occurrences" parameter "                       |
| lib     | serv    | EXT1      | S      | Erroneous structure of the "extraction method" parameter                              |
| lib     | serv    | EXT2      | S      | Unknown extraction method in the Folder   |
| lib     | serv    | USR1      | S      | Missing "User service" parameter  |
| lib     | serv    | USR2      | S      | Erroneous length of the "user service" parameter                                      |
| lib     | serv    | USR3      | S      | Unknown user service in the Folder  |
| lib     | serv    | CHK1      | S      | Missing "Check Option" parameter  |
| lib     | serv    | CHK2      | S      | Erroneous length of the "Check Option" parameter                                      |
| lib     | serv    | RFH1      | S      | Missing "Refresh Option" parameter  |
| lib     | serv    | RFH2      | S      | Erroneous length of the "Refresh Option" parameter                                    |
| lib     | serv    | LCK1      | S      | Missing "Lock Timestamp" parameter  |
| lib     | serv    | LCK2      | S      | Erroneous length of the "Lock Timestamp" parameter                                    |
| lib     | serv    | PCV1      | S      | Erroneous structure of the "Selection Criteria" parameter                             |
| lib     | serv    | PC01      | S      | Erroneous conversion of the "Selection Criteria" parameter                            |
| lib     | serv    | PILO      | S      | Erroneous access to the main record of the work file                                  |
| lib     | serv    | BUF1      | S      | Erroneous structure of user buffer  |
| lib     | serv    | PC02      | S      | Erroneous conversion of a field of the user buffer                                    |
| lib     | serv    | FRWR      | S      | Erroneous write access to the work file   |
| lib     | serv    | FRW2      | S      | Erroneous writing of the last record of the work file                                 |
| lib     | serv    | FRRE      | S      | Erroneous read access to the work file before update                                  |
| lib     | serv    | FRRD      | S      | Erroneous read access to the work file  |
| lib     | serv    | FRRW      | S      | Erroneous update access to the work file  |
| lib     | serv    | CP01      | S      | Erroneous structure of a "Selection Criteria" field from the Business Component       |
| lib     | serv    | CP02      | S      | Erroneous structure of a field of a Logical View instance from the Business Component |
| lib     | serv    | ERKY      | S      | Erroneous structure of the "Selt Message" key from the Business Component             |
| lib     | serv    | ERLA      | S      | Erroneous structure of the "Selt Message" label from the Business Component           |
| lib     | serv    | PCV!      | S      | Erroneous structure of a field of the user buffer from the Business Component         |
| lib     | serv    | PCV3      | S      | Erroneous structure of a field of a Logical View instance from the client             |
| lib     | serv    | PC03      | S      | Erroneous conversion of a field of a Logical View instance from the client            |
| lib     | serv    | PCV4      | S      | Erroneous structure of a "Selection Criteria" field from the client                   |
| lib     | serv    | PC05      | S      | Erroneous conversion of a "Selection Criteria" field from the client                  |
| lib     | serv    | NUVE      | S      | Erroneous "version number" parameter in the elementary Business Component             |
| lib     | serv    | STRU      | S      | Erroneous "structure" parameter in the elementary Business Component                  |
| lib     | serv    | VIEW      | S      | Erroneous "Logical View code" parameter in the elementary Business Component          |
| lib     | serv    | SERV      | S      | Erroneous "operation code" parameter in the elementary Business Component             |
| lib     | serv    | LTH       | S      | Erroneous "length" parameter in the elementary Business Component                     |
| lib     | serv    | PCV5      | S      | Erroneous structure of the action code of a Logical View instance to be updated       |

| Col 1-3 | Col 4-9 | Col 10-13 | Col 21 |   |
|---------|---------|-----------|--------|---|
| lib     | serv    | PCV6      | S      | Erroneous structure of a field of a Logical View instance to be updated from the client           |
| lib     | serv    | PCV7      | S      | Erroneous structure of the presence indicator of a field of a Logical View instance to be updated |
| lib     | serv    | PC06      | S      | Erroneous conversion of a field of a Logical View instance to be updated                          |
| lib     | serv    | ERK1      | S      | Erroneous structure of the user error message key   |
| lib     | serv    | ERL1      | S      | Erroneous structure of the user error message label   |
| lib     | serv    | OCNB      | S      | Erroneous structure, in the user error message, of the field code which bears the error           |
| lib     | serv    | DANA      | S      | Erroneous structure of the erroneous field code in the user error message                         |
| lib     | serv    | PCVS      | S      | Erroneous structure of a service request from the client  |
| lib     | serv    | PCVF      | S      | Erroneous structure of the message from the client  |
| lib     | serv    | WF00      | S      | Erroneous access to the work file   |

*Legend*    lib = library code                      serv = Services Manager code                      S = system error

### 6.4.4 Example

To illustrate error handling in a client application, we will reuse the application from subchapter 6.2 *Example of the Development of a Standard GUI Application*. Let us imagine you want to program the opening of an error message window entitled **Error Manager** when the Error Manager detects an error. This window displays the list and the number of error messages returned for a server update action. It is also used to restore the context of each error.

- **Programming the display of the list and of the number of error messages**

This window is developed in the same VisualAge View as the **Orders** window. The programming includes the following stages:

- To access the Error Manager public interface, you must tear off the Root Proxy's **errorManager** attribute. Place the part obtained on the Free Form Surface.
- To display the number of errors, tear off the Error Manager's **errorCount** attribute.
- To display the list of error messages, create a Quick Form of the Error Manager's **errorList** attribute and insert the obtained container in the **Error Manager** window part. In our example, we only keep the **label** attribute of the **errorList** part.
- To request the Error Manager to return the errors labels in the list, you must connect the **localError**, **serverError**, **systemError** and **fatalError** events of the **errorManager** part to the **openWidget** action of the **Error Manager** window part.

- **Programming the Restoration of the Error Context**

The restoration of the error context is applied to one instance at a time.

To begin with, you must retrieve the items selected in the table in an Ordered Collection, each item corresponding to an instance of the node that causes the error. You must then load the detail of each node with the first item of the Ordered Collection.

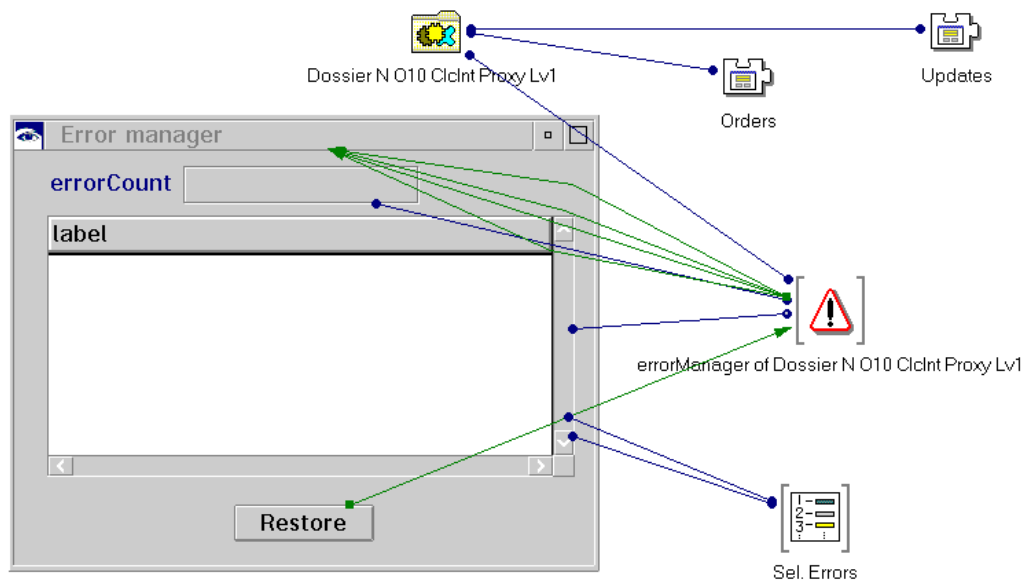


Proceed as follows:

- Tear off the `selectedItems` attribute of the list and place the Ordered Collection on the Free Form Surface.
- You must then connect the `clicked` event of the `Restore` button to the `restoreContextOfSelectedError:` action of the Error Manager. The link appears in dotted lines because the action needs a parameter. To set the necessary parameter, connect the `first` attribute of the Ordered Collection to the `anError` attribute of the link.

• **Result in the Composition Editor**

Once the programming of the `Error Manager` window is finished, here 's how it should look in the Composition Editor:



## 6.5 Communication Management

### 6.5.1 Middleware Components

Pacbench C/S middleware corresponds to a C and C++ application whose functions are distributed in three types of DLLs:

- The first type groups together the interfacing functions with C and C++ languages and the interpreting functions of the middleware and switching services to libraries of specialized functions for a specific Communication protocol.
- The second type groups together the functions for the execution of Pacbench C/S middleware services for a specific Communication protocol.
- The third type groups together functions for managing resources such as the translation in natural language of the encountered exceptions.

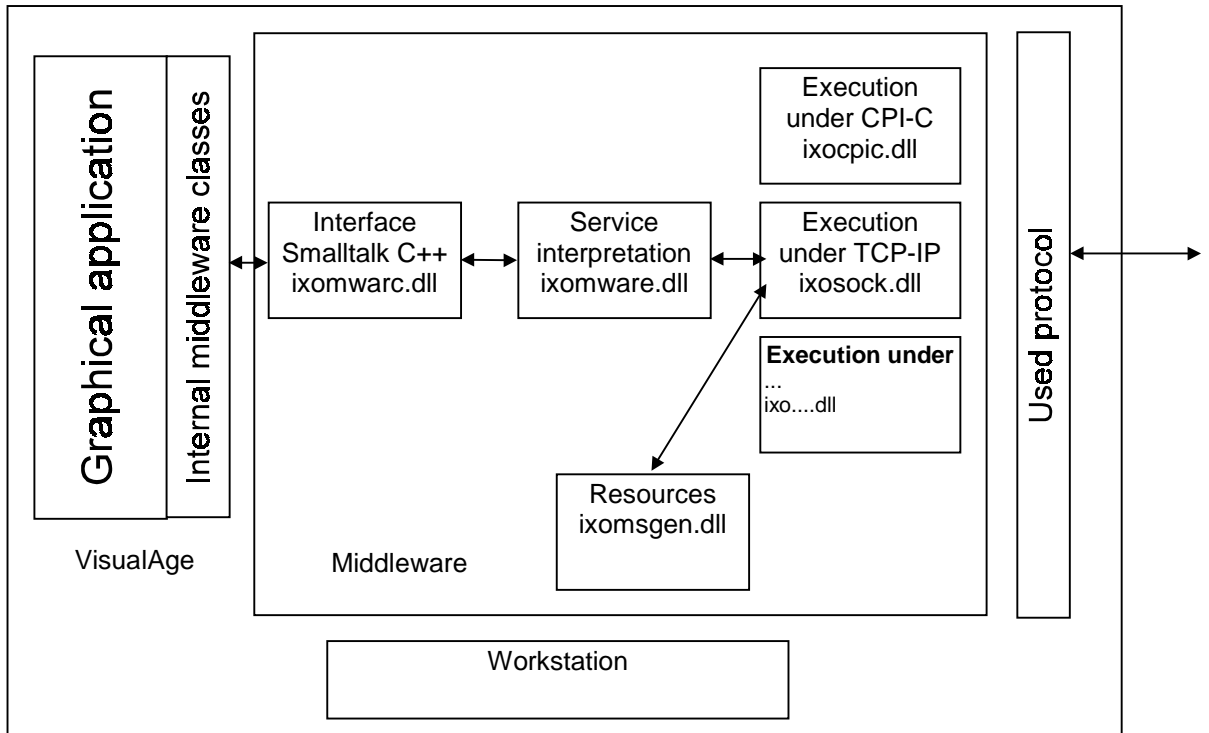
The OS/2 version of Pacbench C/S middleware also uses a run-time materialized by a DLL coded **CSOOM30**.

| <b>DLL name</b> | <b>Function</b> | <b>Platform</b>             | <b>Protocol</b>             |
|-----------------|-----------------|-----------------------------|-----------------------------|
| ixomwarp.dll    | Type-1          | Windows 95/NT<br>OS2        | All                         |
| ixomware.dll    | Type-1          | Windows 95/NT<br>OS2, UNIX  | All                         |
| ixocics.dll     | Type-2          | Windows 95/NT<br>OS2        | CICS-ECI                    |
| ixocpic.dll     | Type-2          | Windows 95/NT<br>OS2        | CPI-C                       |
| ixotux.dll      | Type-2          | Windows 95/NT<br>UNIX       | TUXEDO                      |
| ixosock.dll     | Type-2          | Windows 95/NT<br>OS2, UNIX  | TCP-IP Socket<br>VA Pacbase |
| ixoloc.dll      | Type-2          | Windows 95/NT<br>OS2        | Call of COBOL<br>DLL        |
| ixomqs.dll      | Type-2          | Windows 95/NT<br>OS2, UNIX  | MQSERIES                    |
| ixotmvs.dll     | Type 2          | Windows 95/NT,<br>OS2, UNIX | TCP-IP Socket<br>CICS       |
| ixotcis.dll     | Type 2          | Windows 95/NT,<br>OS2, UNIX | TCP-IP Socket<br>TCIS       |
| ixomsgen.dll    | Type-3          | Windows 95/NT<br>OS2        | All                         |
| csoom30.dll     | runtime C++     | OS2                         | All                         |

Windows(\*) = all Windows platforms

### 6.5.2 Diagram of the Processing of a Request

In VisualAge, middleware services are executed from a set of specific communication classes provided at the installation of the product.



### 6.5.3 Definition of Use Context

The communication management requires the definition of the middleware use context. This context is defined in a specific file where one or more locations are specified for each generated Folder.

A location is used to specify the elements which enable data exchanges between the client applications and the services managers, during the development phase of the client application or during the operation phase. For example, a client application can address a server without any tracepoint for an operating version.

For a folder, a location consists of:

- The location identifier. This identifier is case-sensitive
- The external name of the communications monitor
- The maximum physical length of an exchange message
- A variable number of parameters which make the *middleware* work (IP address, trace, etc.)

The locations are specified in the **VAPLOCAT.INI** file (this file name in upper case is reserved).

☞ The file used by default is the file which is in the startup directory when you start the VisualAge Smalltalk image.

### 6.5.3.1 Structure of the VAPLOCAT.INI File

The locations file is structured in sections and subsections.

Each section corresponds to a Folder and is identified by the Folder code, delimited by [...].

Each subsection corresponds to a location and is identified by the environment name specified in the **LOCATION** option on the Folder **General Documentation** screen. The subsection identifier is delimited by <...>.

Within a subsection, the parameters are identified by keywords. Recognized keywords are as follows:

| <b>keyword</b>  | <b>Meaning</b>   |
|-----------------|--|
| COMMENT         | Comments   |
| LENGTH          | Physical message length (generated)  |
| MONITOR         | Communications monitor external name (generated)   |
| MWADDRESS       | Server's address (for SOCKET, TCPMVS, TCIS, TCPIMSI and TCPIMSE).<br>Required for these protocols<br><br>The following formats are possible:<br>- Decimal coding:<br>* IP address<br>* Space<br>* Port number, in decimal coding<br>- Hexadecimal coding:<br>* [1, 2]: '0x'<br>* [3, 6]: AF_INET domain<br>* [7,10]: port number<br>* [11, 18]: IP address         |
| MWARE           | Communication protocol<br><b>Values :</b><br>TUXEDO<br>CICS<br>LOCAL<br>CPIC<br>SOCKET<br>TCIS<br>MQSERIES<br>TCPMVS<br><b>Protocols</b><br>TUXEDO XA or non XA<br>CICS ECI extend or nonextend<br>Server on the Client workstation<br>CICS CPI-C/APPC or IMS CPI-C/APPC<br>TCP-IP SOCKET for UNIX or Windows/NT<br>TCIS<br>MQSERIES<br>TCP-IP SOCKET for MVS/CICS |
| MWBUFFERTYPE    | Buffer type ( for TUXEDO)<br><b>Values:</b><br>- CARRAY (default)<br>- FML   |
| MWCODEPAGE      | Server page code<br>- Optional   |
| MWFMLNAME       | FML file name ( for TUXEDO)<br>Caution: you need also to state the path in the FLDTBLDIR environment variable and the file name in the FIELDTBLS environment variable.   |
| MWMAXREPLY      | Maximum number of asynchronous answer-waiting requests<br>- Optional<br>- Numeric  |
| MWQUEUEMANAGER  | Queue manager name for asynchronous protocol (MQSERIES)<br>- Required for this protocol<br>- Alphanumeric<br>- 48 characters max.  |
| MWREPLYQUEUE    | Name of the reception-message queue manager for asynchronous protocol (MQSERIES)<br>- Required for this protocol<br>- Alphanumeric<br>- 48 characters max.   |
| MWREQUESTEXPIRY | Duration of an asynchronous request (in seconds)<br>- Optional (default: unlimited)<br>- Numeric   |

| keyword            | Meaning  |
|--------------------|--|
| MWREQUESTQUEUE     | Name of the emission-message queue manager for asynchronous protocol (MQSERIES)<br>- Required for this protocol<br>- Alphanumeric<br>- 48 characters max.  |
| MWTIMEOUT          | Timeout managed by the middleware (in seconds )<br>- Optional<br>- Numeric<br>- Value: [0, 32767]  |
| MWTRANSID          | Identification of the CICS transaction code on 4 characters:<br>- required for the protocol CICS Sockets TCP/IP<br>- optional for the CICS/ECI protocol <ul style="list-style-type: none"> <li>• If the MWTRANSID parameter does not exist or is not valued, the CPMI mirror transaction is used by default and is executed on the CICS server.</li> <li>• If the parameter is valued, the transaction code is executed on the CICS server for each request using this location. This transaction must be defined as a CICS mirror transaction on the server and associated with the DFHMIRS program.</li> <li>• In both cases, the MONITOR parameter is required and must correspond to a program that can be executed on the CICS server.</li> </ul>   |
| IXO_TRANSID(*)     | Identification of the CICS transaction code on 4 characters:<br>- optional and used with the CICS/ECI protocol <ul style="list-style-type: none"> <li>• If the IXO_TRANSID parameter does not exist or is not valued, the CPMI transaction code is used by default and is executed on the CICS server.</li> <li>• If the parameter is valued, the program is called by the CPMI mirror transaction, but it is linked under the name of the transaction code specified in the IXO_TRANSID parameter. Can be used for security problems, for example with DB2 plans.</li> <li>• This variable is ignored if MWTRANSID is used because it has priority.</li> <li>• In all the cases, the MONITOR parameter is required and must correspond to a program that can be executed on the CICS server.</li> </ul> |
| IXO_SYSTEM_NAME(*) | Name of the server for which the ECI request is made on 8 characters:<br>- optional and used with the CICS/ECI protocol <ul style="list-style-type: none"> <li>• If the IXO_SYSTEMNAME parameter does not exist or is not valued, the name of the first server specified in the Server section of the CICSCLI.INI configuration file is used by default.</li> <li>• If the parameter is valued, the name of the server for which the ECI request is made must correspond to a server specified in the Server section of the CICSCLI.INI configuration file</li> </ul>  |

(\*) : Available with release 2.5 V08 only.

You only indicate a single parameter per file line.

Example of a location file:

```
[FOCLNT]
<Location1>
COMMENT=operation monitor
MONITOR=CLCOMM
LENGTH=8192
MWCODEPAGE=850
MWARE=SOCKET
MWADDRESS=0x00021770C0060A5D
<Location2>
COMMENT=monitor with trace
MONITOR=CLCOM2
LENGTH=8192
```

```

MWARE=SOCKET
MWADDRESS=0x00021770C0060A5D
<Location3>
COMMENT=local monitor
MONITOR=CLCOMM
LENGTH=8192
MWARE=LOCAL

```

### 6.5.3.2 Smalltalk Versions from 2.5 V08 onwards

#### 6.5.3.2.1 New structure of the VAPL OCAT.INI file

The locations file contains a list of environments, each of them being identified by its name delimited by <...> (for example <Localhost-Fic-Local>).

It is possible to define for each environment a list of parameters composed of a set of lines. The syntax for each line is: `PARAMETER=VALUE`. (one parameter definition per line).

There are two types of parameter for an environment: parameters defining the general communication functions ( buffer maximum length, monitor used, etc.) and parameters to each communication type (for example, the name of the queue manager for MQSERIES or the name of the FML conversion file for TUXEDO).

The second type parameters are prefixed with `IXO_` and can be added as you wish with a subsection. The pair (parameter name, value) being sent as it is to the Middleware DLL, the Middleware is responsible for taking it into account.

In the following table you find the parameters new names and their corresponding names in versions lower than 2.5 V08 (their meaning are unchanged, of course):

| Versions up to 2.5 V08 | Versions 2.5 V08 and higher    |
|------------------------|--------------------------------|
| COMMENT                | COMMENT                        |
| LENGTH                 | MESSAGE_LENGTH                 |
| MONITOR                | MONITOR                        |
| ADDRESS                | LISTENER_ADDRESS               |
| MWARE                  | COMM_TYPE                      |
| MWTIMEOUT              | MIDDLEWARE_TIME_OUT            |
| MWCODEPAGE             | HOST_CODE_PAGE                 |
| MWMAXREPLY             | MAX_PENDING_REPLY              |
| MWFMLNAME              | IXO_FMLCONVERSIONFILE          |
| MWBUFFERTYPE           | No longer applies <sup>1</sup> |
| MWQUEUEMANAGER         | IXO_QUEUEMANAGER               |
| MWREPLYQUEUE           | IXO_REPLYQUEUE                 |
| MWREPORTQUEUE          | Unused                         |
| MWREQUESTEXPIRY        | IXO_REQUESTEXPIRY              |
| MWREQUESTQUEUE         | IXO_REQUESTQUEUE               |
| MWTRANSID              | IXO_TRANSID                    |
| Non-existing           | FOLDER <sup>2</sup>            |

<sup>1</sup> For TUXEDO, the buffer type depends on the `IXO_FMLCONVERSIONFILE` parameter. If this parameter is not valued, the buffer type will be CARRAY. If it is valued, the buffer type will be FML.

The `FOLDER` parameter allows you to specify that a default location environment will be used for the Folder specified as the parameter value. This value corresponds to the Pacbase name of the Folder.



The **VAPLOCAT.INI** files defined to be used with a version lower than 2.5 V08 remain compatible and can be used as they are with the 2.5 V08.



If the **FOLDER** parameter is not specified for any environment defined in the **VAPLOCAT.INI** file (**that is the case when you use a VAPLOCAT.INI file from a version lower than 2.5 V08**), the first environment in the list will be used to define the default location used by the Proxy. It is highly probable that this default environment is different from the one selected using a version lower than 2.5 V08 **∪ In this case, it is highly recommended to use the FOLDER parameter to avoid any problem.**

### 6.5.3.2 Management of Environments

The environments specified in the **VAPLOCAT.INI** file are no longer created or updated by the generator of the Folder View Proxies. The communication environments being specific to the Client workstation, they are now managed locally on this workstation.

Before being provided with an external tool that is independent of the implementation language, yo can update the structure of an old **VAPLOCAT.INI** file and create, modify or delete environments from this file using the environment editor **VpcsLocationsEditor** stored in the **VpcsToolsApp** application.

### 6.5.3.2.3 Example of Localtions File

```
<Location1>
  COMMENT= Socket Oracle Monitor
  MESSAGE_LENGTH=08192
  MONITOR=MCSKOR
  COMM_TYPE=SOCKET
  MIDDLEWARE_TIME_OUT=120
  LISTENER_ADDRESS=0x0002CACBC0060A5D
  FOLDER=FOCLNA
<Location2>
  COMMENT=Environment Oracle TUXEDO CARRAY
  MESSAGE_LENGTH=08192
  MONITOR=MCTXOR
  COMM_TYPE=TUXEDO
  FOLDER=FOCLNB
  FOLDER=FOCLNC
<Location3>
  COMMENT=Environment Oracle TUXEDO FML
  MESSAGE_LENGTH=08192
  MONITOR=MCTXOR
  COMM_TYPE=TUXEDO
  IXO_FMLCONVERSIONFILE=d:\apps\tuxedo\fieldtbl.vap
```

### 6.5.3.3 Implementation

The **VAPLOCAT.INI** file is required in both development and operation phases.

If the file is missing or if the syntax used is incorrect, the Exchange Manager cannot be instantiated. The incorrect parameter is indicated in the trace file.

In the operation phase, the Folder View Proxy generator is responsible for creating or updating the **VAPLOCAT.INI** file.

- If the **VAPLOCAT.INI** file does not exist, the generator creates it in the current VisualAge directory. For each location, the values of the **MONITOR** and **LENGTH** parameters (generated) are automatically set. The developer must specify the other parameters.
- If it already exists (in the VisualAge directory), it is automatically loaded. For each Folder, the generator compares the locations contained in the extraction file with the existing locations. The existing **VAPLOCAT.INI** file is updated accordingly:
  - ♦ the new locations are added to the existing file
  - ♦ the locations that are no longer in the new extraction file are automatically removed from the existing file
  - ♦ the existing locations are modified according to the modifications contained in the new extraction file.

In case of an update, the parameters set by the developer remain unchanged.



From the version 2.5 V08 onwards, the **VAPLOCAT.INI** is no longer updated.

## 6.6 Test of the Generated Application– Packaging

### 6.6.1 Test of the Generated Application

#### 6.6.1.1 Starting the Application

- complete path and name of the trace file,

To start the VisualAge application, code:

```
abt      -i<image> -v<tracePath>
```

where <image> = image name,

<tracePath> = complete path and name of the trace file (optional).

#### 6.6.1.2 Version Control

If the version control option detects any discrepancy, it means that the Business Component and the Proxy object were not generated with the same version number. So you must:

- regenerate the Proxy object if you have regenerated only a new version of the Business Component,
- or implement, in VisualAge, the generated graphic application including the new proxy component if it has not already been done,
- or implement the generated Business Component if it has not already been done.



As for the version control, see the 1.2 Subchapter, *Compatibility of Business Components* / Proxy.



### 6.6.1.3 Debugging Aid or Trace Mode

Pacbench Client/Server middleware is available in optimized version or in trace version. The extensions of the files associated with the middleware's DLLs allow you to distinguish the two versions.

The files with the '001' extension corresponds to the DLLs used in TRACE mode. With the TRACE mode, you obtain a file containing all events, actions and objects handled by the middleware. This mode is activated from the two following environment variables:

- **IXOTRACE :**

This variable enables to activate (**IXOTRACE=1**) or deactivate (**IXOTRACE=0**) the middleware API trace.

- **IXOTRACE\_FILE :**

This variable enables to specify, when the trace is active (**IXOTRACE=1**), the API middleware trace.

*exemple :* `IXOTRACE_FILE=c:\tmp\ixo_err.txt`


The file containing the events, actions and objects is never reinitialized by the middleware functions. We advise you to frequently destroy this file, as the middleware immediately creates a new one.

The files with the '002' extension corresponds to the optimized version of Pacbench Client/Server middleware DLLs. These DLLs do not contain the code allowing to trace the communication process and do not interpret the environment variables dedicated to this use.

At the installation of Pacbench Client/Server on the workstation, the DLLs ready to be interpreted correspond to those of the middleware in optimized version.


## 6.6.2 Packaging

There are several packaging methods, but some of them are not described in this manual, because they are not specific to VisualAge Pacbase.

 For further information, refer to the *VisualAge for Smalltalk - User's Guide, Enhancing your applications* part, *Packaging your VisualAge application* chapter.

### 6.6.2.1 Packaging an IC

If you are using at least a VisualAge Smalltalk 4.02b version, you can package VisualAge Pacbase runtime as an IC (Image Component).

 You will find information on the "Image Components" in the *Image Component Developer's Guide and Reference* documentation.

To package the runtime represented by the **VpcsRuntimeApp** application as an IC, you must execute the following operations:

- Select the **Tools** menu in the Organizer,
- Select **Packaged Images**,
- Select the **Instructions In Database** page,
- double-click on the **VpcsToolsApp** application to select it,
- Select the **VpcsRuntimeICPackagingInstructions** class,
- Reduce by clicking on the **Reduce** button ,
- Click on **Output Image** to build the component file.

The image is created under the **vpcs.ic** name. It is located in the start directory of the Smalltalk image and can be re-used for the final application packaging.

## 6.7 Application Deployment



To deploy the application, follow the operations indicated in the Visualage Smalltalk documentation.

But you must also install, on the end-user workstation, some files specific to the use of Pacbench C/S :

- ♦ the middleware DLLs,
- ♦ **VAPLOCAT.INI**,
- ♦ **ABTRULES.NLS** (conversion of page code),
- ♦ **VAPERROR.TXT**.

## 7 Developing a COM Standard Client

### 7.1 General Principles

#### 7.1.1 Use of Attributes

An attribute corresponds to a piece of information handled by a Proxy object. This piece of information defines an elementary data, a list of elementary data or a list of composite data instances. An attribute corresponds either to a constant, a parameter or an action result. According to the context, it is initialized by the GUI application or the Proxy.

Two kinds of attributes are found:

- those standing for technological variables. They enable to adjust the behavior of the Proxy objects in VisualAge
- Those corresponding to the Logical View data



The availability of an attribute depends on the Proxy type. All the public interface attributes are documented in the *Graphic Clients Reference Manual: Public Interface of Generated Components*.

##### 7.1.1.1 Local Checks

The Elementary Proxy automatically calls the local checks during the creation and modification of an instance via the `createInstance` and `modifyInstance` actions. Each Data Element belonging to the Logical View is checked.

The following checks are made:

- Checks on value lists defined in the Data Element description
- Checks on value ranges defined in the Data Element description
- Checks on compulsory presence defined when calling Data Elements in a Logical View. The presence of identifier-type Data Elements or foreign key-type Data Elements is automatically checked for a reference relation with a minimum cardinal value of 1.

If local checks detect an error, an error message is set via the Error Manager.

These checks can be selectively triggered for each root or depending-type node in the Folder concerned. The attribute used to activate or deactivate the check of Data Elements on the server is `serverCheckOption`.

However, the Elementary Proxy does not perform numeric or date checks; these are performed by the graphic application.

##### 7.1.1.2 Sub-schema Management

The sub-schemas specified in the Logical View's description can be taken into account when selection/read actions are performed, provided Business Components manage the presence of Data Elements (`VECTPRES=YES` or `CHECKSER=YES` options).

All the nodes with at least one sub-schema have the following attributes:

- **subSchema**, via which you can assign the desired sub-schema when a selection/read action is performed by the Business Component of the node. The value of this attribute can be assigned via the following attributes. It is ignored for update actions.
- **getSubSchemasCount**, and **getSubSchemasElementAt(index)** via which all the sub-schemas available on a node are listed. Since a sub-schema cannot be given a name in the VisualAge Pacbase Repository, each sub-schema is designated by **SubSchema<n>** (with **n** = 01 to 10).

### 7.1.1.3 Check of the Length of the **detail** Attribute Fields

For each field of the **detail** attribute, the checks performed upon the local creation or modification of an instance ensure that the length of the value contained in the attribute does not exceed the maximum length of the value for this attribute.

The length is checked systematically (except if the attribute does not belong to the current sub-schema), even if the attribute has been defined as 'not to be checked in the client'.

A local error is sent if the length is excessive.

☞ The length of the fields included in user buffers is not checked. If it is excessive, the length is truncated to its maximum value.

### 7.1.1.4 Selecting the Local or Server Sort Criterion on a List of Instances

The **localSort** attribute enables you to specify whether the Proxy sorts the instances of the **rows** attribute according to the local sort (**true**) or keeps the instances in the order they were sent by the server (**false**).

You can change the sort type at any moment.

☞ This attribute is not effective in user services.

#### 7.1.1.4.1 Local Sort

In standard, the instances of the **rows** attribute are sorted according to the local criterion if the parameter has not been changed after the generation. In this context, two sort types exist:

- If no sort criterion has been locally defined, the Proxy implicitly sorts the instances in the increasing order of the identifiers defined on the Logical View.
- If a local criterion has been locally defined, the Proxy sorts the instances in the order defined by this criterion.

In all cases, when an instance is created locally, it is inserted according to the current sort criterion applied in the **rows** attribute.

If you change dynamically or cancel the local sort criterion, the instances contained in the **rows** attribute are immediately sorted according to the new criterion.

#### 7.1.1.4.2 Server Sort

The instances of the `rows` attribute are sorted according to the server criterion if the `localSort` attribute is set to `false` or if the standard parameter of the node has been modified. In this context, the instances contained in the `rows` attribute are displayed in the order sent by the server, without taking the local sort criterion into account.

If collections are managed manually or in the case of a paging in extend mode, the instances sent by the server are added at the end of the existing collection in the `rows` attribute.

All the locally-created instances are added at the end of the existing collection in the `rows` attribute. In this context, an instance which is not positioned at the end of a collection but which is deleted and created again locally is transferred to the end of the collection contained in the `rows` attribute.

### 7.1.2 Use of Actions

#### 7.1.2.1 Implementation

An action corresponds to a process which can be executed by a Proxy objet. It is triggered using a connection between an event in the GUI application and the method code of a Proxy.



The availability of an action depends on the Proxy type. All the actions of the public interface are documented in the *Graphic Clients: Public Interface of Generated Components Reference Manual*.

#### 7.1.2.2 The Different Types of Server Actions

The server actions execute procedures implemented in one or more Business Components associated with the Folder. These actions send a query to the Business Components which send back a result on the workstation. The queries and responses generally contain technical parameters, Logical View instances associated with one or more nodes and contextual data defined in a user buffer.

You must distinguish two types of server actions:

- Those which systematically get to the server:
  - ♦ `selectInstances`
  - ♦ `readInstance`
  - ♦ `readInstanceAndLock`
  - ♦ `readWithFirstChildren`
  - ♦ `readWithAllChildren`
  - ♦ `readWithFirstChildrenAndLock`
  - ♦ `readWithAllChildrenAndLock`
  - ♦ `readAllChildrenFromDetail.`
  - ♦ `readWithAllChildrenFrom:`
- Those which do not systematically get to the server:
  - ♦ `readNextPage`: The server is accessed except if the `NO_PAGE_AFTER` event was sent back during the previous selection
  - ♦ `readPreviousPage`: The server is accessed except if the `NO_PAGE_BEFORE` event was sent back during the previous selection.
  - ♦ `readFirstChildrenFromDetail :`

The server is accessed except if the `maxNumberOfRequestedInstances` attributes of Depending Proxies are set to 0 and if their `globalSelection` attributes are set to false.

- ◆ `readWithFirstChildrenFrom`: The server is accessed except if the `maxNumberOfRequestedInstances` attributes of Depending Proxies are set to 0 and if their `globalSelection` attributes are set to false.
- ◆ `checkExistenceOfDependencies`: The server is accessed except if the existence of depending instances can be checked locally.
- ◆ `updateFolder`: The server is only accessed if there is at least one instance of the node concerned, modified in its `updatedFolders` attribute.

### 7.1.2.3 Managing Folder Reading

#### 7.1.2.3.1 Large reading of the Root node

Large reading of a Folder root enables to read from a client component all the instances of the Folder's root node existing in the database. The actions concerned are `selectInstances` and `readNextPage`.

#### 7.1.2.3.2 Provisional Large Reading of Depending Nodes

In the context of client/server architectures, a GUI application handling a Folder strives to anticipate data reading to minimize exchanges with the servers.

In a hierarchical network, various provisional reading actions are possible:

- The `'allChildren'` type action reads *all* the depending instances of the selected instance in the `detail` attribute of the parent Elementary Proxy.
- The `'firstChildren'` type action only reads the instances which are *immediately* dependent of the selected instance in the `detail` attribute of the parent Elementary Proxy.

The first provisional large reading action is available on the Root Proxy only.

The second provisional large reading action is available on the Root Proxy or on the Depending Proxies which themselves hold Depending Proxy objects.

#### 7.1.2.3.3 Transferring an Instance between the Rows and Detail Attributes



The `rows` attribute cannot be accessed directly but only through the methods `getRowCount()` and `getRowElementAt(int i)`.

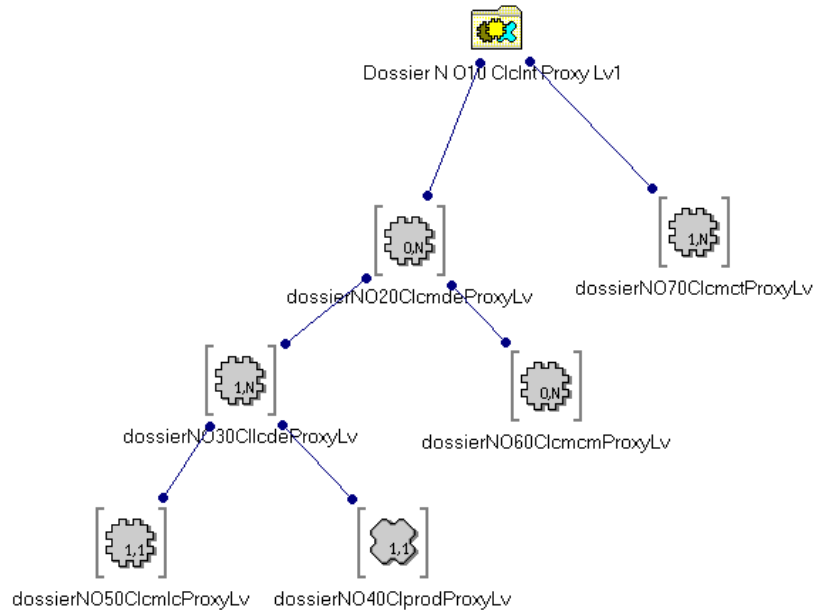
The transfer of an instance between the `rows` and the `detail` attributes enables the `detail` attribute to be loaded with an instance initially retrieved by a large reading action.

This transfer is only available for Root and Depending Proxies. It corresponds to a local reading action which also loads all the local instances of Depending Proxies recognized by the Folder View Proxy. The transfer is made using the `getDetailFromData` action.

### 7.1.2.3.4 Large Reading and Transferring an Instance between the Rows and Detail attributes: Working Mechanism

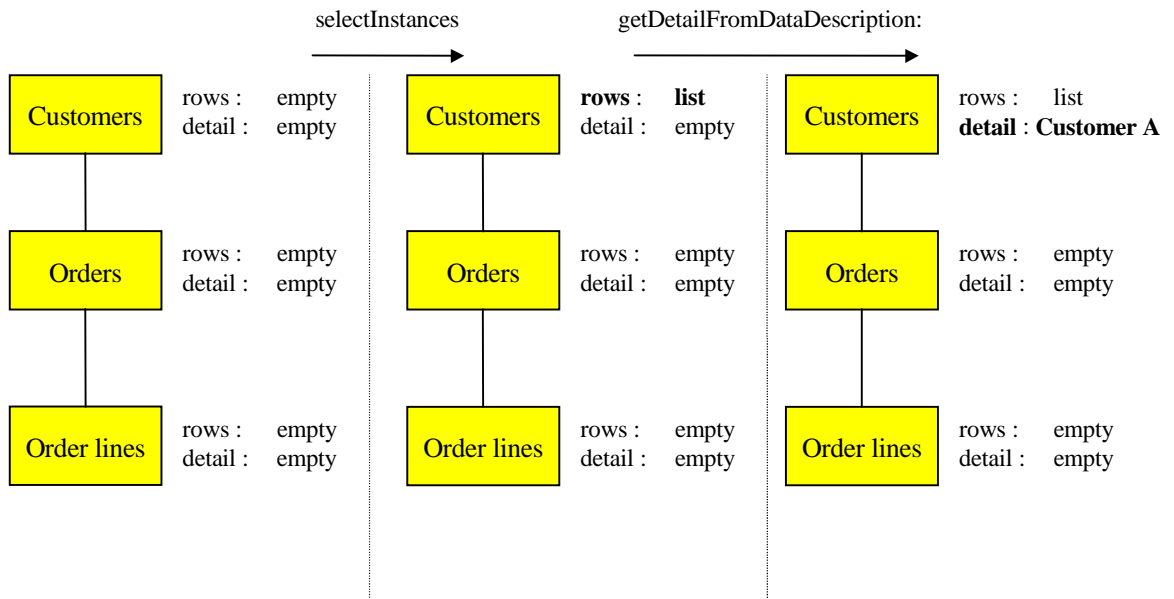
This example illustrates the loading of the **detail** attribute with an instance previously retrieved in the **rows** attribute by large reading a root or depending node, and the principle of the provisional large reading of depending nodes.

It is based on the Folder View Proxy generated in the previous chapter. The Elementary Proxies it contains are recalled in the directory structure below:



For the needs of our example, we will use three Elementary Proxies:

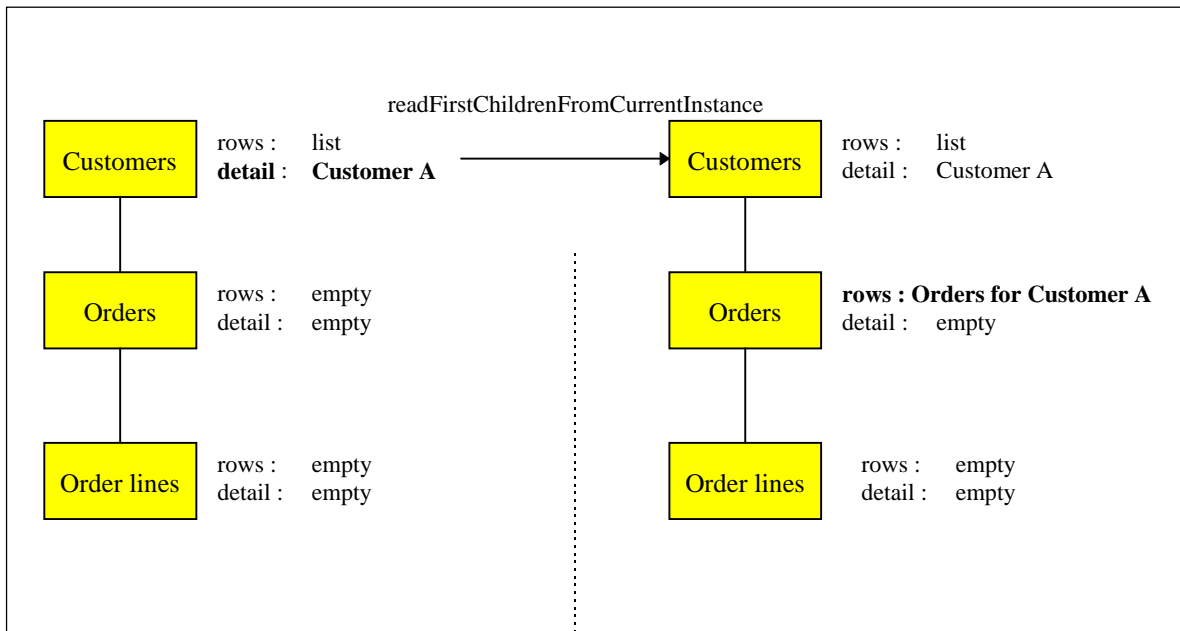
- The **dossierNO10ClclntProxyLv** Root Proxy corresponding to the **Customers** node which manages the customers in the information system described by the Folder
- The **dossierNO20ClcmdeProxyLv** Depending Proxy corresponding to the **Orders** node which manages the orders in the information system described by the Folder
- The **dossierNO30ClcdeProxyLv** Depending Proxy corresponding to the **Order Lines** node which manages the order lines in the information system described by the Folder.



The **detail** attribute of the customer node now contains Customer A. There are three solutions to read Customer A's depending instances (i.e. his associated orders).

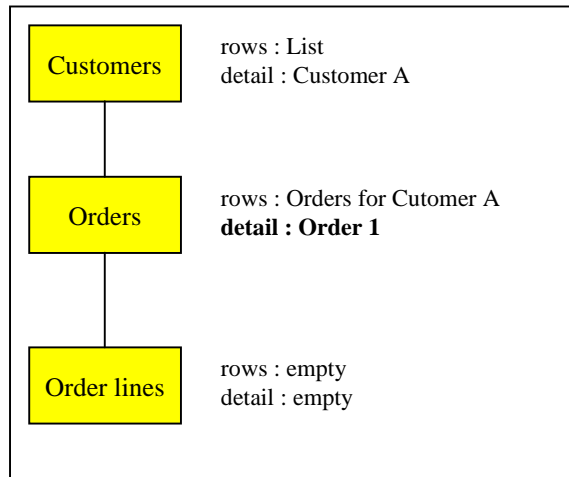


### SOLUTION 1



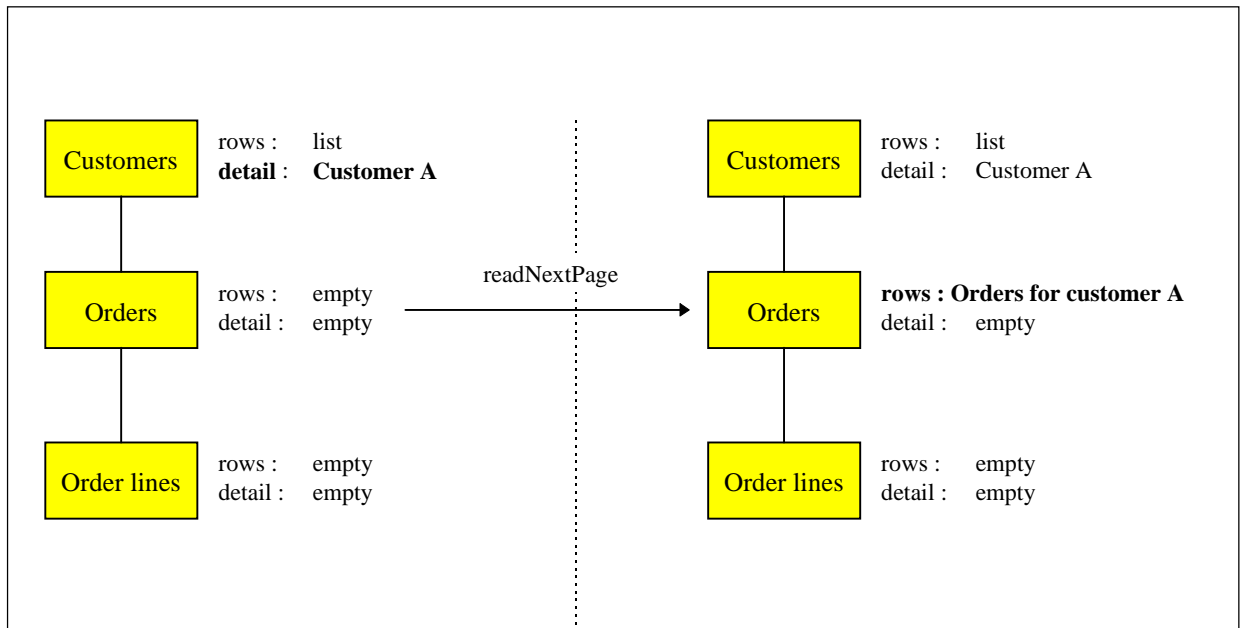
The **readFirstChildrenFromDetail** action on the Customers Root Proxy not only loads the **rows** attribute of the Orders Depending Proxy for Customer A, but also the **rows** attribute of other possible Elementary Proxies directly dependent of the Root Proxy.

In this context, the **getDetailFromData** action on the Orders Depending Proxy initiates:



Then to read the order lines of the Order 1 for Customer A, use the **readFirstChildrenFromDetail** action on Orders or the **readNextPage** action on Order lines.

## SOLUTION 2

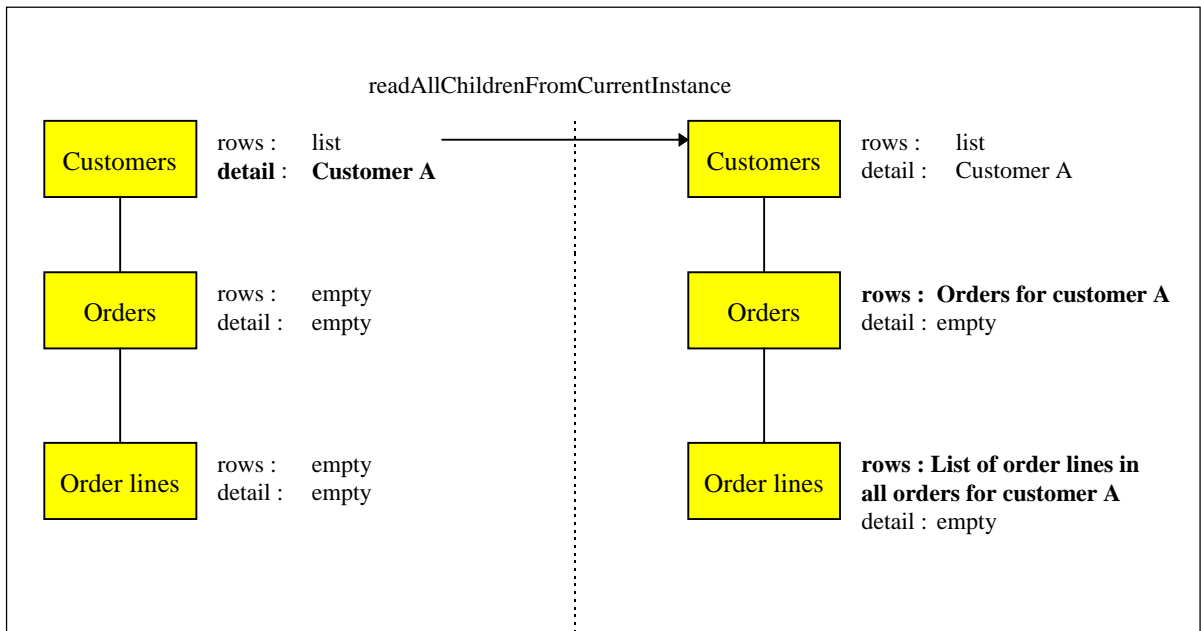


The **readNextPage** action on the Orders Depending Proxy loads its **rows** attribute. The instances of other possible Elementary Proxies directly dependent of the Root Proxy are not read.

In this context, the **getDetailFromData** action on the Orders Depending Proxy initiates the same result as in solution 1.

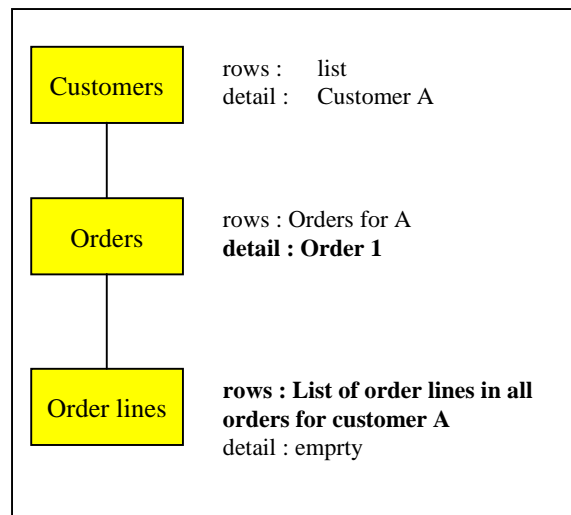
To subsequently read the order lines in Order 1 for Customer A, proceed as in solution 1.

### SOLUTION 3



The `readAllChildrenFromDetail` action on the Customers Root Proxy reads not only all the orders for A but also all the other possible instances dependent of A whatever their hierarchical level can be. In our example, all the order lines for all the orders of A are therefore read.

In this context, the `getDetailFromData` action on the Orders Depending Proxy initiates:



The `getDetailFromData` action automatically loads the `rows` attribute of the Order Lines Depending Proxy with the order lines contained in Order 1 for Customer A. These order lines have previously been transferred to the workstation using the `readAllChildrenFromDetail` action.

### 7.1.2.3.5 Large Reading of Reference Nodes

The reading of a reference node is considered as aid on criteria. It shows the end user a list of information which can be referred to for a depending node.

The information presented to the end user is both necessary and sufficient to assist him in making a choice.

To optimize the volume of characters sent for this type of service, the Logical Views have a « aid on criteria »-type subschema used to select the concerned Data Elements at the server level.

The large reading of reference nodes is executed on request and cannot be involved in provisional large reading. The actions concerned are `selectInstances` and `readNextPage`.

### 7.1.2.3.6 Principle of Paging in a Folder's Nodes

Two types of paging are offered on a Folder's nodes:

- The first, called *non-extend* paging, is used to paginate forwards and backwards on a predefined collection through specific actions. Each action executes a read request to the server and its result overwrites that of the previous read. This type of paging is available only on root or reference nodes.
- The second type of paging, called *extend* paging, is used to gradually retrieve the instances of a defined collection as read requests for following pages are made. In this context, the backwards paging function disappears and is performed locally by the scroll box of the graphic control which presents the list of instances. This type of paging is available to all the nodes of a Folder.

### 7.1.2.3.7 Selection Criteria Associated with Large Reading Actions

The selection criteria associated with large reading actions are elementary or composite attributes associated with each node of a Folder. They are split into two types:

- Functional selection criteria corresponding to the identifier and to elements required for defining extraction methods for the Logical View associated with the node. These criteria are the following ones:
  - ♦ the `selectionCriteria` attribute which defines identifier Data Elements and parameters by value.
  - ♦ the `extractMethodCode` attribute which defines the code of the desired extraction method.
  - ♦ the `getExtractMethodCodesCount()` action and `getExtractMethodCodesElementAt(Int i)` action which enable you to access the list of the available extraction methods.
- Organic selection criteria corresponding to information used to control the volume of instances selected for each node
  - ♦ `globalSelection` is a Boolean attribute which, when set to true, is used to retrieve all instances of the node via a selection query.

- ♦ **maxNumberOfRequestedInstances** is a numerical attribute which specifies, when the **globalSelectionIndicator** attribute is set to false, the number of instances to be retrieved for a node via a selection query. This attribute can hold the value 0. In this case, the concerned part of the tree structure is not read during a provisional large reading action.

#### 7.1.2.3.8 Limitation of the Scope of Large Reading

The result of a large reading action is limited by the selection criteria associated with the node.

However, in the case of a large reading ordered by an **allChildren** type provisional reading, the **globalSelectionIndicator** attribute is considered to be set to true on each node.

#### 7.1.2.3.9 Reading of a Depending or Root Node Instance

The reading of a depending or root node instance enables you to retrieve an instance of the node without previously making a large selection of a collection of instances for this node. It directly loads the node's **detail** attribute.

This type of reading is considered as a collection selection and therefore cancels the previous selection even if it was the result of a large reading action. The loading of the **detail** attribute therefore initializes the **rows** attribute.

The actions which implement this selection function are used to:

- Retrieve the instance without its dependencies (**readInstance**).
- Retrieve the instance with its first level dependencies (**readWithFirstChildren**).
- Retrieve the instance with all its dependencies (**readWithAllChildren**).

#### 7.1.2.3.10 Reading of a Reference Node Instance

The reading of a reference node instance cannot activate the provisional large reading process. It is used to retrieve the entire description of the instance of the node called in its **detail** attribute.

Only the **readInstance** action is therefore available on a reference node.

Update actions are not available on reference nodes. Their **rows** and **detail** attributes. The result of the **readInstance** action does not therefore initialize the **rows** attribute.

The **rows** attribute is used to display sufficient information to assign one of the reference node instances to the referencing node instance.



The **rows** attribute cannot be accessed directly but only through the methods **getRowCount()** and **getRowElementAt(int i)**.

The **detail** attribute is used to view the entire description of a referenced instance.

The structure of these two attributes can thus be different.

#### **7.1.2.3.11 Selection Criteria Associated with Instance Reading**

The identifier of the node instance to be read is specified in the functional selection criteria associated with the node.

For depending nodes, the identifier of the node corresponds to the identifier of the Logical View associated with the node, discarded from the Data Elements which are the identifiers of Logical Views higher in the hierarchy. These hierarchical identifiers are automatically initialized by the Folder View Proxy according to the navigation in the Folder.

### **7.1.2.4 Folder Update Management**

#### **7.1.2.4.1 Local Updates**

Local update services are available on each root or depending type node in the Folder.

- Create a node instance
- Modify a node instance
- Cancel a node instance

However, there are certain rules specific to Folder management:

- The creation of a depending node instance is only authorized if the hierarchy of instances contained in the **detail** attributes of higher nodes exists.
- Cancelling a node instance initiates the recursive cancelling of local instances of depending nodes.

To allow the developer to manage messages which can warn users of the impact of a cascade cancellation, an action for verifying the existence of depending instances is available on root or depending type nodes.

This action (**checkExistenceOfDependencies**) sends a Boolean result which is either true or false. If no dependency is found in the Folder's local cache and the instance concerned is not created locally, this action sends a verification request to the server.

So that a user can undo local manipulations of a Folder instance, an **undoAllLocalFolderUpdates** action can be used to eliminate all local updates on all Folder nodes applied since the last server update action. This action is only available on the Folder's root node. Another action **undoLocalFolderUpdates** can be used to eliminate all updates associated with the instance you transferred as parameter.

#### **7.1.2.4.2 Server Updates**

Only the Root Proxy provides server update actions.

Server updates correspond to actions which enable a client component to send all local updates made since the last server update action.

These updates concern all the modified depending instances. They can concern several Folder instances.

When a server update action sends back errors, the Folder remains with the « Modified Locally » status; new collection selections can be made only by correcting the errors and sending back the updates, or by using the `undoAllLocalFolderUpdates` action or the `undoLocalFolderUpdates` action.

Before the request is sent to the server, the server update actions check the Folder integrity for locally created instances. For each locally created instance of the node, this action checks the minimum cardinal values of each link and sends an error if the number of depending instances does not respect the properties of the associated links.

A server update action can be accompanied by a request to refresh the updated instances if some of their Data Elements, such as the identifiers, are calculated by the server. This refreshment request is made using the `refreshOption` attribute.

#### **7.1.2.4.3 Management of Effective Transactions**

The management of effective transactions is automatically carried out by the local cache.

It consists in calculating the resulting update of various local updates made on the same instance of the Folder's node. It controls the creation of duplicate instances. If several local updates have been made on the same instance of the node, only the last one will be sent to the server.

#### **7.1.2.4.4 Changing the Collection Selection**

The `ABOUT_TO_CHANGE_SELECTION` event is returned to the Client by the Root Proxy when a server selection action on a Folder node overwrites its current local collection and when this collection or the depending collections contain local modifications which are candidates for a server update. If the event is not intercepted during programming, the updates will be lost.

#### **7.1.2.4.5 Re-initializing instances in the local cache**

The `resetCollection` action is used to remove all the instances from the cache of a Folder View Proxy before initializing a new collection of instances. This action can be executed by all types of nodes in a Folder View Proxy containing a `rows` attribute.



The `rows` attribute cannot be accessed directly but only through the methods `getRowCount()` and `getRowElementAt(int i)`.

#### **7.1.2.4.6 Managing collections of instances**

The management of collections of instances can be carried out automatically, or manually by positioning the `manualCollectionReset` boolean attribute which is available on all types of nodes in a Folder View Proxy. The manual management mode is used to create collections of heterogeneous collections through a series of selection and paging actions.

### **7.1.2.5 User Services**

Each root or depending type node contains the following elements to implement a user service:

- Two actions enable you to obtain the list of available user services on the node. These actions are `getUserServiceCodesCount()` and `getUserServiceCodesElementAt(Int i)`.
- Two actions enable you to store locally Logical View instances to be processed for the next user service. These actions are `getUserInputRowsCount()` and `getUserInputRowsElementAt(Int i)`.
- Two actions enable you to present various Logical View instances sent by a user service. These actions are `getUserOutputRowsCount()` and `getUserOutputRowsElementAt(Int i)`.
- An attribute which presents the Logical View instances which are candidates for the execution of the next user service. This attribute is `userDetail`.
- Local actions used to memorize each Logical View instance to be sent to the server for the execution of a user service. These actions are `createUserInstance`, `modifyUserInstance` and `deleteUserInstance`.

The root node of the Folder also has the following elements:

- An action used to execute all the user services on each Folder node. This action is `executeUserServices`.
- An action used to delete all the local instances stored for all the Folder nodes. This action is `resetUserRows`.
- An action used to delete the current instance stored locally. This action is `resetUserServiceCodes`.

This principle means that 1 to n user services can be executed, in the same query, distributed on the different nodes. The execution sequence of these services corresponds to the hierarchical order of nodes, browsing the tree from top to bottom and from left to right.

#### 7.1.2.6 Database Logical Lock

The upload-download mechanisms associated with a Folder increase the elapsed time between reading the initial Folder image and displaying the result of an update.

In this context, with no lock mechanism, two users can modify the same Folder instance. The result of accumulated updates are therefore difficult to manage.

To enable the user to use a Folder in an exclusive appropriation mode, two types of locks for a node instance are available:

- The optimistic lock which works on the principle of verifying the change of a `TimeStamp` before executing the update procedure.
- The pessimistic lock which uses an entity for exclusive update by recording a specific resource. In this case, the Folder update procedure is carried out before freeing up the exclusive resource.

The server lock procedure is triggered by the explicit execution of a specific action available on the Root Proxy. This action is `lock`.

The server unlock procedure is triggered automatically with the execution of a server update action or explicitly by the execution of a specific action available on the Root Proxy. This specific action is `unLock`.



The developer is responsible for writing the lock processing in the root Business Component.

This processing receives the identifier of the Logical View instance to be locked as well as the request type (lock or unlock) to be executed.

In return, it must set a status used to grant or refuse the lock and the `TimeStamp` or the name of the resource used.

If the lock is refused, the Root Proxy sends a `LOCK_FAILED` event whereby all subsequently executed local or server update actions are disabled for the specified instance of the Folder. In this case, the Folder changes to the « Read only » status.

The concept of the logical lock is defined in the Folder entity or in the Business Component for a single-view development.

When the logical lock option is active on a Folder, all read requests for the `detail` attribute of the Root Proxy can be accompanied by a logical lock request on the server.

#### 7.1.2.7 Management of Data Element Presence

The two following actions enable you to manage the presence of the Logical View's Data Elements at the Proxy level.

The `is<delco>Present` action enables you to test the presence or absence of the `delco` Data element. It is generated for all `DataDescription` and `UserDataDescription` classes.

The `set<delco>Present(aBoolean)` action enables you to specify the presence or absence of the `delco` Data element. It is generated for all `DataDescription` and `UserDataDescription` classes.

By default, all the Data Elements are considered to be absent, except if a default value and/or a list of values have been indicated in the VisualAge Package description..

#### 7.1.2.8 Management of Data Element Check

The following two actions enable you to manage the check of the Logical View's Data Elements at the Proxy level.

The `setCheck<fieldIndex,aBoolean>` action enables you to activate or inhibit the server checks on a Data Element before any local update action.

The `getCheck<fieldIndex>` action enables you to test whether the server checks are activated on a Data Element.

Both these actions are generated for all the `DataDescription` classes of the root or depending nodes whose Business Components include the `NULLMNGT=YES` and `CHECKSER=YES` options and an update service.

By default, all the Data Elements are to be checked (if the `serverCheckOption` attribute is set to `true`).

#### 7.1.2.9 Sub-schema Management

The server selection or read actions take into account the sub-schema present in the `subSchema` attribute and return the values of the Data Elements belonging to the sub-schema. If a selection action is followed by a paging action, the sub-schema taken into account is that associated with the selection action.

The local creation actions do not refer to any sub-schema.

The local modification/deletion actions refer to the sub-schema associated with the instance, that is:

- if the modification/deletion is performed on an instance which was created locally, the sub-schema is empty.
- if the modification/deletion is performed on a read instance, the sub-schema is that associated with the selection of this instance.

Moreover the following two actions are specific to the sub-schema management.

The **resetSubSchema** action enables you to reset the **subSchema** attribute, that is to select no sub-schema.

The **completeInstance** action enables you to retrieve the values of the Data Elements which do not belong to the sub-schema, by calling the Business Component associated with the Logical View.

The **belongsToSubSchema** action enables you to know whether the Data Element passed as a parameter belongs to the sub-schema associated with the **detail** attribute.

### 7.1.3 Use of Events

The events sent by an Elementary Proxy are used to trigger application actions belonging to the GUI application. This processing is performed by connecting a Proxy event to one or more actions in the GUI application. The conditional execution of actions is facilitated by the fact that an event is always accompanied by its opposite event; both events cannot be sent at the same time. After being sent, the event is stored in a stack. Therefore, the graphic application must access the stack regularly using the **getServerEventsCount** and **popServerEvent** methods.



The availability of an event depends on the type of Proxy. All the Public Interface events are documented in the *Graphic Clients: Public Interface of Generated Components Reference Manual*.

#### 7.1.3.1 Event-driven Management of Large Reading

Event-driven management of large reading provides the developer with information on the state of the collection of instances contained in a node. Each available paging action offers its own event-driven paging system.

The paging action in non-extend mode can send the following four events:

- **NO\_PAGE\_BEFORE**: This event is sent by a root or reference node at the end of the execution of a collection selection or paging action when it does not return any error and when the read page is the first in the current collection.
- **PAGE\_BEFORE**: This event is sent by a root or reference node at the end of the execution of a collection selection or paging action when it does not return any error and when the read page is not the first in the current collection.
- **NO\_PAGE\_AFTER**: This event is sent by a root or reference node at the end of the execution of a collection selection or paging action when it does not return any error and when the read page is the last in the current collection.

- **PAGE\_AFTER**: This event is sent by a root or reference node at the end of the execution of a collection selection or paging action when it does not return any error and when the read page is not the last in the current collection.

The paging action in extend mode can send the following two events:

- **PAGE\_AFTER**: This event is sent by any type of node at the end of the execution of a collection selection or forwards paging action when it does not return any error and when the number of instances contained in the node is not the total number of instances contained in the database.
- **NO\_PAGE\_AFTER**: This event is sent by any type of node at the end of the execution of a collection selection or forwards paging action when it does not return any error and when the number of instances contained in the node is the total number of instances contained in the database when the query is made.

### 7.1.3.2 Event-driven Management of Instance Reading

A Logical View can be mapped on one or more physical storage entities. In this context, the event-driven management of a Logical View instance reading can send two events:

- **NOT\_FOUND** when the instance searched for is not found in the database. This event can be sent when the Logical View is mapped on one or more tables.

## 7.2 Error Management

There are four types of errors:

- Local errors, sent by the Client Component, which correspond to manipulation or input errors in the Client Component.
- Server errors, sent by the Business Component, which are data access errors and user errors set in the server.
- System errors, sent by the Business Component, which corresponds to a discrepancy between the Proxy and the Business Components,
- Communication errors.

### 7.2.1 Local Errors

- **ASYNCHRONOUS\_VIOLATION**  
This error is produced when a lock, unlock or existence check of depending instances in asynchronous mode is requested.
- **CARDINALITY\_VIOLATION**  
This error is produced if the cardinalities are not respected when an update action is activated.
- **CURRENT\_INSTANCE\_MISSING**  
This error is produced when a action is applied to the **detail** attribute whereas the latter does not contain any instance.
- **CURRENT\_USER\_INSTANCE\_MISSING**  
This error is produced when a user action is applied to the **userDetail** attribute whereas this attribute does not contain any instance.
- **FOLDER\_USER\_CONTEXT\_LENGTH\_ERROR**  
This error is produced when the length of the value of a Data Element belonging to the user buffer exceeds the authorized length for this Data Element.
- **INSTANCE\_ALREADY\_LOCKED**  
This error is produced when a lock action is requested on an instance, which has already been locked on the server.
- **INSTANCE\_NOT\_LOCKED**  
This error is produced when a lock action is requested on an unlocked instance on the server.
- **INVALID\_CHANGE**  
This error is produced when the instance to be modified does not exist in the local cache.
- **INVALID\_CREATION**  
This error is produced when an instance is created though it already exists in the local cache.
- **INVALID\_DELETION**  
This error is produced when the instance to be deleted does not exist in the local cache.
- **INVALID\_INSTANCE**  
This error is produced when a primary key of the current instance is not valid.
- **LENGTH\_ERROR**  
This error is produced when the length of the value of a Data Element belonging to the current instance exceeds the authorized length for this Data Element.
- **PARENT\_INSTANCE\_MISSING**  
This error is produced when a depending node instance is selected though the parent instance does not exist.
- **REFERING\_INSTANCE\_MISSING**

This error is produced when the **transferReference** action does not find any instance in the **detail** of the referring node.

- **SERVER\_UPDATE\_REQUIRED**

This error is produced when a action is applied to an instance whereas the parent instance created locally does not exist in the database yet. A server update is previously required for the parent instance.

- **SUBSCHEMA\_ERROR**

This error is produced when a Data Element belonging to the current instance is updated whereas it was not filled in for this instance on a server access parameterized with a sub-schema.

- **UNKNOWN\_ASYNCHRONOUS\_REQUEST\_ID**

This error occurs when a response retrieval request is sent and when the response identifier is not recognized or has expired.

- **UNKNOWN\_INSTANCE**

This error is produced when the selected instance is not known by the local cache.

- **VALUE\_ERROR**

This error is produced when the contents of a Data Element belonging to the current instance are not valid.

- **VALUE\_REQUIRED**

This error is produced when the contents of a Data Element belonging to the current instance are considered to be absent whereas they are required.

## 7.2.2 Server Errors

These messages are generated in standard by the Business Components. There are three types of messages:

- Irretrievable access errors: **Data access error**

These are irretrievable file or relational database access errors.

- Logical access errors: **Invalid creation** and **Invalid deletion or modification**.
- Control errors on Logical View fields: **Value error** and **Data Element required**.

## 7.2.3 System Errors

The system errors are:

- **Unknown service**

This message is displayed when the service requested by the Proxy is not recognized by the Business Component.

- **View length error**

This message is displayed when the format of a Logical View associated with the Proxy changes and when this Proxy cannot be regenerated.

To solve this problem, you must regenerate the Proxy.

- **Components out of phase**

This error occurs when the Client and Business Components are out of phase.

To solve this problem, you must regenerate the Proxy.

## 7.2.4 Communication Error Messages

If a communication error message is displayed, inform the person in charge of the communication because the line may be blocked or defective, or a Server may be busy, etc.

Three communication error messages are likely to be displayed:

- **Open server error**
- **Call server error**
- **Close server error**

## 7.2.5 System Errors Generated by the Communication Monitor and the Services Manager

Most of these errors are internal errors that you can solve only by contacting the VisualAge Pacbase support.

### 7.2.5.1 Errors Generated by the Communication Monitor

| Col 1-3 | Col 4-9 | Col 10-13 | Col 21 |  |
|---------|---------|-----------|--------|--|
| lib     | mon     | LSRV      | S      | Erroneous length of the message received                 |
| lib     | mon     | PNUM      | S      | Erroneous structure of the "service number" parameter    |
| lib     | mon     | PCOD      | S      | Erroneous structure of the "service code" parameter      |
| lib     | mon     | PNOD      | S      | Unknown Folder code                                      |
| lib     | mon     | PCVS      | S      | Erroneous structure of a service request from the client |
| lib     | mon     | PCVF      | S      | Erroneous structure of the message from the client       |
| lib     | mon     | WF00      | S      | Erroneous access to the work file                        |

*Legend*    lib = library code                      mon = Communication Monitor code                      S = System error

### 7.2.5.2 Errors Generated by the Services Manager

| Col 1-3 | Col 4-9 | Col 10-13 | Col 21 |  |
|---------|---------|-----------|--------|--|
| lib     | serv    | SRV1      | S      | Service not found in the work file                                 |
| lib     | serv    | LNG1      | S      | Erroneous conversion of service length on a multi-messages request |
| lib     | serv    | LNG2      | S      | Erroneous conversion of service length on a single-message request |
| lib     | serv    | NOS1      | S      | Erroneous structure of the "service number" parameter              |
| lib     | serv    | NOS2      | S      | Erroneous conversion of the "service number" parameter             |
| lib     | serv    | SRV1      | S      | Erroneous structure of the "service code" parameter                |

| Col 1-3 | Col 4-9 | Col 10-13 | Col 21 |   |
|---------|---------|-----------|--------|---|
| lib     | serv    | SRV2      | S      | Unknown service code in the Folder  |
| lib     | serv    | DOS1      | S      | Missing "Folder name" parameter   |
| lib     | serv    | DOS2      | S      | Erroneous length of the "Folder name" parameter                                       |
| lib     | serv    | VER2      | S      | Erroneous length of the "version number" parameter                                    |
| lib     | serv    | NOD1      | S      | Missing "node name" parameter   |
| lib     | serv    | NOD2      | S      | Erroneous length of the "node name" parameter   |
| lib     | serv    | NOD3      | S      | Unknown node name in the Folder   |
| lib     | serv    | TYNO      | S      | Unauthorized service on the node  |
| lib     | serv    | SCH1      | S      | Erroneous structure of the "sub-schema code" parameter                                |
| lib     | serv    | SCH2      | S      | Unknown sub-schema code in the Folder   |
| lib     | serv    | NOCF      | S      | Erroneous structure of the "number of occurrences" parameter                          |
| lib     | serv    | NOC1      | S      | Erroneous length of the " number of occurrences" parameter                            |
| lib     | serv    | NOC2      | S      | Erroneous conversion of the "number of occurrences" parameter "                       |
| lib     | serv    | EXT1      | S      | Erroneous structure of the "extraction method" parameter                              |
| lib     | serv    | EXT2      | S      | Unknown extraction method in the Folder   |
| lib     | serv    | USR1      | S      | Missing "User service" parameter  |
| lib     | serv    | USR2      | S      | Erroneous length of the "user service" parameter                                      |
| lib     | serv    | USR3      | S      | Unknown user service in the Folder  |
| lib     | serv    | CHK1      | S      | Missing "Check Option" parameter  |
| lib     | serv    | CHK2      | S      | Erroneous length of the "Check Option" parameter                                      |
| lib     | serv    | RFH1      | S      | Missing "Refresh Option" parameter  |
| lib     | serv    | RFH2      | S      | Erroneous length of the "Refresh Option" parameter                                    |
| lib     | serv    | LCK1      | S      | Missing "Lock Timestamp" parameter  |
| lib     | serv    | LCK2      | S      | Erroneous length of the "Lock Timestamp" parameter                                    |
| lib     | serv    | PCV1      | S      | Erroneous structure of the "Selection Criteria" parameter                             |
| lib     | serv    | PC01      | S      | Erroneous conversion of the "Selection Criteria" parameter                            |
| lib     | serv    | PILO      | S      | Erroneous access to the main record of the work file                                  |
| lib     | serv    | BUF1      | S      | Erroneous structure of user buffer  |
| lib     | serv    | PC02      | S      | Erroneous conversion of a field of the user buffer                                    |
| lib     | serv    | FRWR      | S      | Erroneous write access to the work file   |
| lib     | serv    | FRW2      | S      | Erroneous writing of the last record of the work file                                 |
| lib     | serv    | FRRE      | S      | Erroneous read access to the work file before update                                  |
| lib     | serv    | FRRD      | S      | Erroneous read access to the work file  |
| lib     | serv    | FRRW      | S      | Erroneous update access to the work file  |
| lib     | serv    | CP01      | S      | Erroneous structure of a "Selection Criteria" field from the Business Component       |
| lib     | serv    | CP02      | S      | Erroneous structure of a field of a Logical View instance from the Business Component |
| lib     | serv    | ERKY      | S      | Erroneous structure of the "Selt Message" key from the Business Component             |
| lib     | serv    | ERLA      | S      | Erroneous structure of the "Selt Message" label from the Business Component           |
| lib     | serv    | PCV!      | S      | Erroneous structure of a field of the user buffer from the Business Component         |
| lib     | serv    | PCV3      | S      | Erroneous structure of a field of a Logical View instance from the client             |
| lib     | serv    | PC03      | S      | Erroneous conversion of a field of a Logical View instance from the client            |
| lib     | serv    | PCV4      | S      | Erroneous structure of a "Selection Criteria" field from the client                   |

| Col 1-3 | Col 4-9 | Col 10-13 | Col 21 |   |
|---------|---------|-----------|--------|---|
| lib     | serv    | PC05      | S      | Erroneous conversion of a "Selection Criteria" field from the client                              |
| lib     | serv    | NUVE      | S      | Erroneous "version number" parameter in the elementary Business Component                         |
| lib     | serv    | STRU      | S      | Erroneous "structure" parameter in the elementary Business Component                              |
| lib     | serv    | VIEW      | S      | Erroneous "Logical View code" parameter in the elementary Business Component                      |
| lib     | serv    | SERV      | S      | Erroneous "operation code" parameter in the elementary Business Component                         |
| lib     | serv    | LTH       | S      | Erroneous "length" parameter in the elementary Business Component                                 |
| lib     | serv    | PCV5      | S      | Erroneous structure of the action code of a Logical View instance to be updated                   |
| lib     | serv    | PCV6      | S      | Erroneous structure of a field of a Logical View instance to be updated from the client           |
| lib     | serv    | PCV7      | S      | Erroneous structure of the presence indicator of a field of a Logical View instance to be updated |
| lib     | serv    | PC06      | S      | Erroneous conversion of a field of a Logical View instance to be updated                          |
| lib     | serv    | ERK1      | S      | Erroneous structure of the user error message key   |
| lib     | serv    | ERL1      | S      | Erroneous structure of the user error message label   |
| lib     | serv    | OCNB      | S      | Erroneous structure, in the user error message, of the field code which bears the error           |
| lib     | serv    | DANA      | S      | Erroneous structure of the erroneous field code in the user error message                         |
| lib     | serv    | PCVS      | S      | Erroneous structure of a service request from the client  |
| lib     | serv    | PCVF      | S      | Erroneous structure of the message from the client  |
| lib     | serv    | WF00      | S      | Erroneous access to the work file   |

*Legend*    lib = library code                      serv = Services Manager code                      S = system error

## 7.3 Communication Management

### 7.3.1 Middleware Components

Pacbench C/S middleware corresponds to a C and C++ application whose functions are distributed in three types of DLLs:

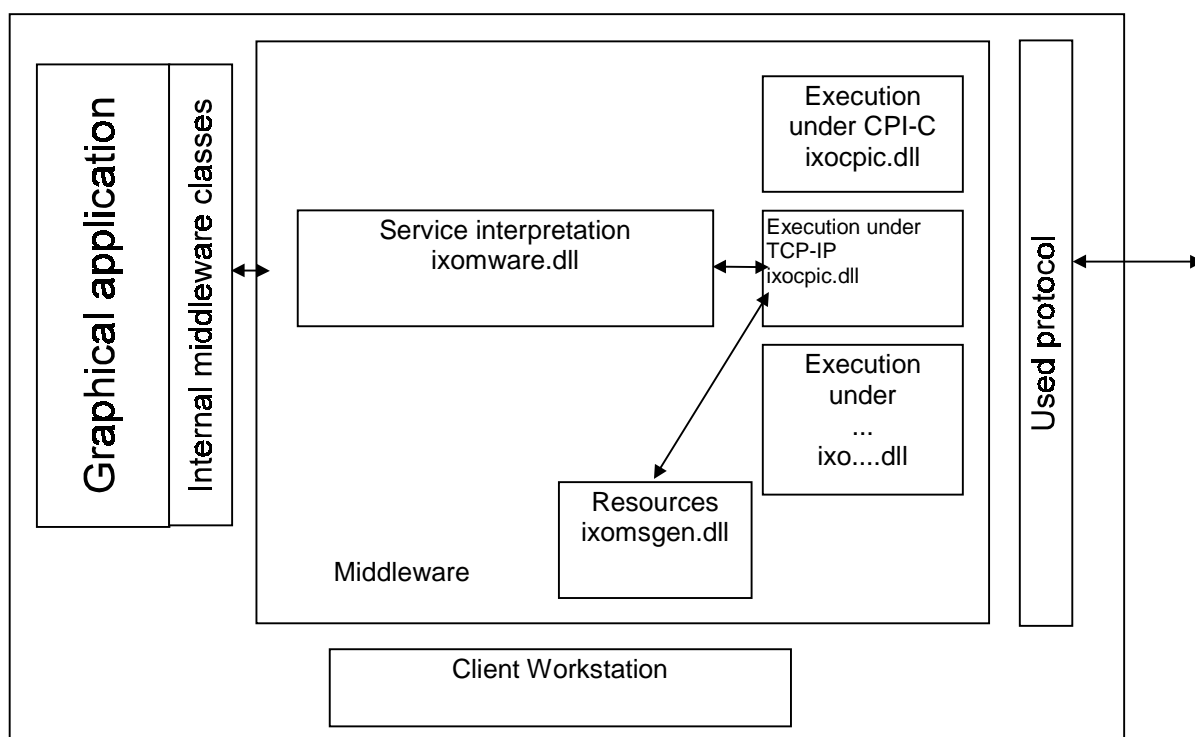
- The first type groups together the interfacing functions with C and C++ languages and the interpreting functions of the middleware and switching services to libraries of specialized functions for a specific Communication protocol.
- The second type groups together the functions for the execution of Pacbench C/S middleware services for a specific Communication protocol.
- The third type groups together functions for managing resources such as the translation in natural language of the encountered exceptions.



| DLL Name     | Function    | Platform     | Protocol             |
|--------------|-------------|--------------|----------------------|
| ixomware.dll | Type-1      | Windows95/NT | All                  |
| ixocics.dll  | Type-2      | Windows95/NT | CICS-ECI             |
| ixocpic.dll  | Type-2      | Windows95/NT | CPI-C                |
| ixotux.dll   | Type-2      | Windows95/NT | TUXEDO               |
| ixosock.dll  | Type-2      | Windows95/NT | SOCKET TCP-IP        |
| ixoloc.dll   | Type-2      | Windows95/NT | Call of DLL<br>COBOL |
| ixomqs.dll   | Type-2      | Windows95/NT | MQSERIES             |
| ixomsgen.dll | Type-3      | Windows95/NT | All                  |
| csoom30.dll  | runtime C++ | Windows95/NT | All                  |
| msvcr20.dll  | runtime C++ | Windows95/NT | All                  |

### 7.3.2 Diagram of the Processing of a Request

In your client, the middleware services are executed from a set of specific communication classes at the installation of the product.



### 7.3.3 Definition of the Use Context

The communication management requires the definition of the middleware use context. This context is defined in a specific file where one or more locations are specified for each generated Folder.

A location is used to specify the elements which enable data exchanges between the client applications and the services managers, during the development phase of the client application or during the operation phase. For example, a client application can address a server without any tracepoint for an operating version.

For a folder, a location consists of:

- The location identifier. This identifier is case-sensitive
- The external name of the communications monitor
- The maximum physical length of an exchange message
- A variable number of parameters which make the *middleware* work (IP address, trace, etc.)

The locations are specified in the **VAPLOCAT.INI** file (this file name in upper case is reserved).



The file used by default is the file which is in the startup directory when you start the VisualAge Smalltalk image.

#### 7.3.3.1 Structure of the VAPLOCAT.INI File

The locations file is structured in sections and subsections.

Each section corresponds to a Folder and is identified by the Folder code, delimited by [...].

Each subsection corresponds to a location and is identified by the environment name specified in the **LOCATION** option on the Folder **General Documentation** screen. The subsection identifier is delimited by <...>.

Within a subsection, the parameters are identified by keywords. Recognized keywords are as follows:

| <b>keyword</b> | <b>Meaning</b>  |
|----------------|---|
| COMMENT        | Comments  |
| LENGTH         | Physical message length (generated)   |
| MONITOR        | Communications monitor external name (generated)  |
| MWADDRESS      | Server's address (for SOCKET, TCPMVS, TCIS, TCPIMSI and TCPIMSE). Required for these protocols<br><br>The following formats are possible:<br>- Decimal coding:<br>* IP address<br>* Space<br>* Port number, in decimal coding<br>- Hexadecimal coding:<br>* [1, 2]: '0x'<br>* [3, 6]: AF_INET domain<br>* [7,10]: port number<br>* [11, 18]: IP address |
| MWARE          | Communication protocol<br><b>Values :</b><br>TUXEDO<br><b>Protocols</b><br>TUXEDO XA or non XA  |

|                 |   |  |
|-----------------|---|--|
|                 | <p>CICS<br/>                 LOCAL<br/>                 CPIC<br/>                 SOCKET<br/>                 TCIS<br/>                 MQSERIES<br/>                 TCPMVS</p>  | <p>CICS ECI extend or nonextend<br/>                 Server on the Client workstation<br/>                 CICS CPI-C/APPC or IMS CPI-C/APPC<br/>                 TCP-IP SOCKET for UNIX or Windows/NT<br/>                 TCIS<br/>                 MQSERIES<br/>                 TCP-IP SOCKET for MVS/CICS</p> |
| MWBUFFERTYPE    | <p>Buffer type ( for TUXEDO)<br/> <b>Values:</b><br/>                 - CARRAY (default)<br/>                 - FML</p>   |  |
| MWCODEPAGE      | <p>Server page code<br/>                 - Optional</p>   |  |
| MWFMLNAME       | <p>FML file name ( for TUXEDO)<br/>                 Caution: you need also to state the path in the FLDTBLDIR environment variable and the file name in the FIELDTBLS environment variable.</p>                         |  |
| MWMAXREPLY      | <p>Maximum number of asynchronous answer-waiting requests<br/>                 - Optional<br/>                 - Numeric</p>  |  |
| MWQUEUEMANAGER  | <p>Queue manager name for asynchronous protocol (MQSERIES)<br/>                 – Required for this protocol<br/>                 - Alphanumeric<br/>                 - 48 characters max.</p>                          |  |
| MWREPLYQUEUE    | <p>Name of the reception-message queue manager for asynchronous protocol (MQSERIES)<br/>                 - Required for this protocol<br/>                 - Alphanumeric<br/>                 - 48 characters max.</p> |  |
| MWREQUESTEXPIRY | <p>Duration of an asynchronous request (in seconds)<br/>                 – Optional (default: unlimited)<br/>                 - Numeric</p>   |  |

| <b>keyword</b> | <b>Meaning</b>   |
|----------------|--|
| MWREQUESTQUEUE | Name of the emission-message queue manager for asynchronous protocol (MQSERIES)<br>- Required for this protocol<br>- Alphanumeric<br>- 48 characters max.  |
| MWTIMEOUT      | Timeout managed by the middleware (in seconds )<br>- Optional<br>- Numeric<br>- Value: [0, 32767]  |
| MWTRANSID      | Code of the transaction CICS on 4 characters:<br>- required for the MVS TCP-IP socket protocol<br>- optional for the CICS/ECI protocol <ul style="list-style-type: none"> <li>• If the MWTRANSID parameter does not exist or is not valued, the CPMI transaction is automatically executed under CICS. In this case, the MONITOR parameter is required and must correspond to a program running under CICS.</li> <li>• If the parameter is valued, the transaction code is executed under CICS for each request using this location. In this case, the value entered for the MONITOR parameter is not taken into account.</li> </ul> |

You only indicate a single parameter per file line.

Example of a location file:

```
[ FOCLNT ]
<Location1>
COMMENT=operation monitor
MONITOR=CLCOMM
LENGTH=8192
MWCODEPAGE=850
MWARE=SOCKET
MWADDRESS=0x00021770C0060A5D
<Location2>
COMMENT=monitor with trace
MONITOR=CLCOM2
LENGTH=8192
MWARE=SOCKET
MWADDRESS=0x00021770C0060A5D
<Location3>
COMMENT=local monitor
MONITOR=CLCOMM
LENGTH=8192
MWARE=LOCAL
```

### 7.3.3.2 Implementation

The **VAPLOCAT.INI** file is required in both development and operation phases.

If the file is missing or if the syntax used is incorrect, the Exchange Manager cannot be instantiated. The incorrect parameter is indicated in the trace file.

In the operation phase, the Folder View Proxy generator is responsible for creating or updating the **VAPLOCAT.INI** file.

- If the **VAPLOCAT.INI** file does not exist, the generator creates it in the current VisualAge directory. For each location, the values of the **MONITOR** and **LENGTH** parameters (generated) are automatically set. The developer must specify the other parameters.
- If it already exists (in the VisualAge directory), it is automatically loaded. For each Folder, the generator compares the locations contained in the extraction file with the existing locations. The existing **VAPLOCAT.INI** file is updated accordingly:
  - ♦ the new locations are added to the existing file
  - ♦ the locations that are no longer in the new extraction file are automatically removed from the existing file
  - ♦ the existing locations are modified according to the modifications contained in the new extraction file.
 In case of an update, the parameters set by the developer remain unchanged.

## 7.4 Application Deployment



To deploy the application, follow the operations described in the documentation of your graphical development tool.

But you must also install some files specific to the use of Pacbench C/S.

- **For a Web application**

The end-user workstation must be equipped with a browser.

- **For a *standalone* application:**

- If no gateway is used, you must install, on the end-user workstation:
  - ♦ the middleware DLLs,
  - ♦ **VAPLOCAT.INI**,
  - ♦ **CHARCONV.TXT** (conversion of page code),
  - ♦ **VAPRUNTIME.DLL**,
  - ♦ **MWADAPTER.DLL**.
- If a gateway is used, you must install, on the station where the gateway is installed:
  - ♦ **GATEWAY.EXE**,
  - ♦ the middleware DLLs,
  - ♦ **VAPLOCAT.INI**,
  - ♦ **CHARCONV.TXT** (conversion of page code),
  - ♦ **GWADAPTER.DLL**.

In this case, you must not install any of these files on the end-user workstation. However, on this workstation, you must install the **VAPRUNTIME.DLL** file.



## 8 Pacbench Client/Server Bridge

### 8.1 Introduction

The Pacbench C/S bridge enables you to:

- Use a unique reference database, the Repository.
- Store VisualAge objects in this database. In doing so these objects benefit from the management, cross-references and security functions offered by the Repository.

The bridge comprises a Backup procedure, a Purge procedure and a Restoration procedure (for VisualAge Smalltalk only).

- The Backup procedure stores the VisualAge objects in the Repository according to the Information Model described in the following subchapter.

For Smalltalk, it also enables you to store these object's sources.

The VisualAge Pacbase WorkStation (Pacbench module) allows you to access the VisualAge storage entities in consultation mode only.

- The Purge procedure searches the VisualAge entities that are not used and deletes them from the database.
- The Smalltalk Restoration procedure allows you to download objects stored in the Repository back in the VisualAge environment.

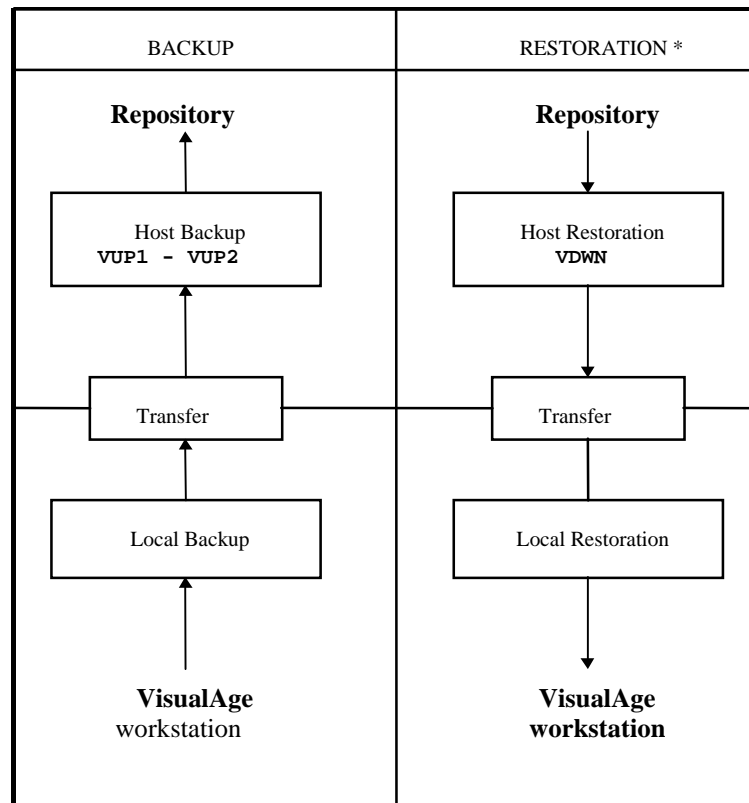
The Backup and Restoration procedures comprise two parts:

- one is executed locally in the VisualAge environment,
- the other is executed on the host site by batch procedures.

The Purge procedure is a batch procedure.

To execute these batch procedures, it is necessary to have a level 4 global authorization. The technical specificities for each platform are described in the corresponding *Operations Manual*.

## *The Backup and Restoration steps*



\* The Restoration procedure is only for VisualAge for Smalltalk workstation.

## 8.2 Pacbench Client/Server Information Model (Graphic Client)

A number of User Entities - dedicated to the storing of VisualAge objects - must be installed in the Repository prior to any bridge operations.



For technical details on this operation, refer to the *VisualAge Pacbase - Batch Procedures - Administrator's Guide Operations Manual*, chapter *Integration of VisualAge Dictionary*.

The Pacbench C/S Information Model with a graphic client component comprises two sub-schemas:

- the Business Logic sub-schema,
- the Graphical User Interface sub-schema.

These two sub-schemas contain Repository entities and entities specific to the client graphic development environment, separated by a dotted arc. They also show the cross-references between entities, and their relationships and cardinalities.



### 8.2.1 Business Logic Sub-schema

This sub-schema can be accessed from the VisualAge Pacbase **Workstation Manager** window (**Pacbench** module, **Metamodel** menu, **Business Logic** option).

On the server side it comprises the Folder, Business Logic, Logical View, Data Element, Segment, Database Block and Program entities.

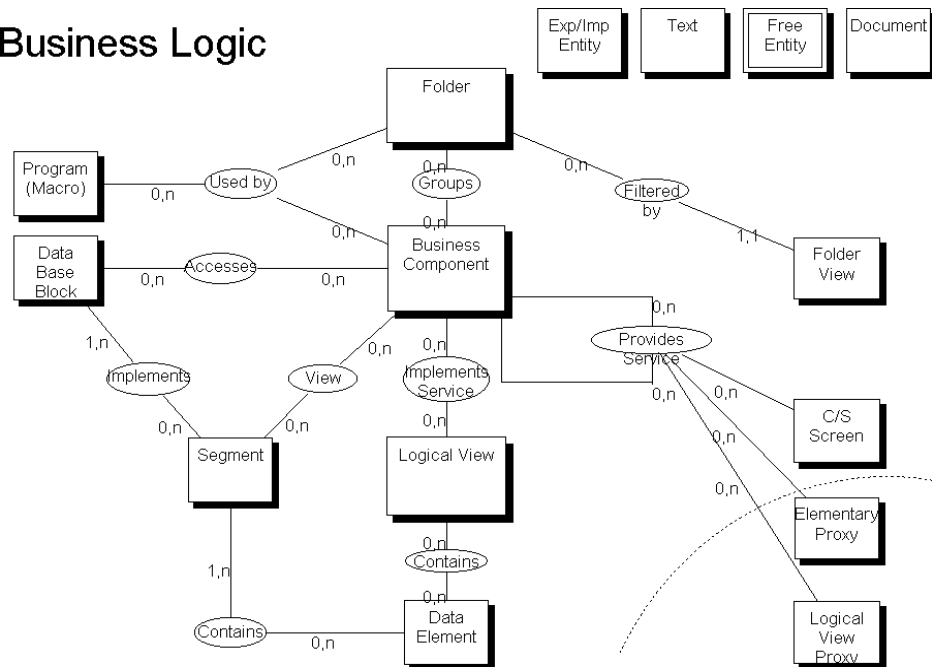
On the client side it comprises:

- the Folder View and C/S Screen entities for TUI clients,
- the Logical View Proxy (simple mode) and Elementary Proxy (folder mode) entities for GUI clients.



The TUI clients are not described in this manual. For more informations, refer to the *Pacbench C/S, Vol. II – Business Logic User's Guide*.

### Business Logic



### 8.2.2 Graphical User Interface Sub-schema

This sub-schema can be accessed from the VisualAge Pacbase **Workstation Manager** window (**Pacbench** module, **Metamodel** menu, **Graphical User Interface** option).

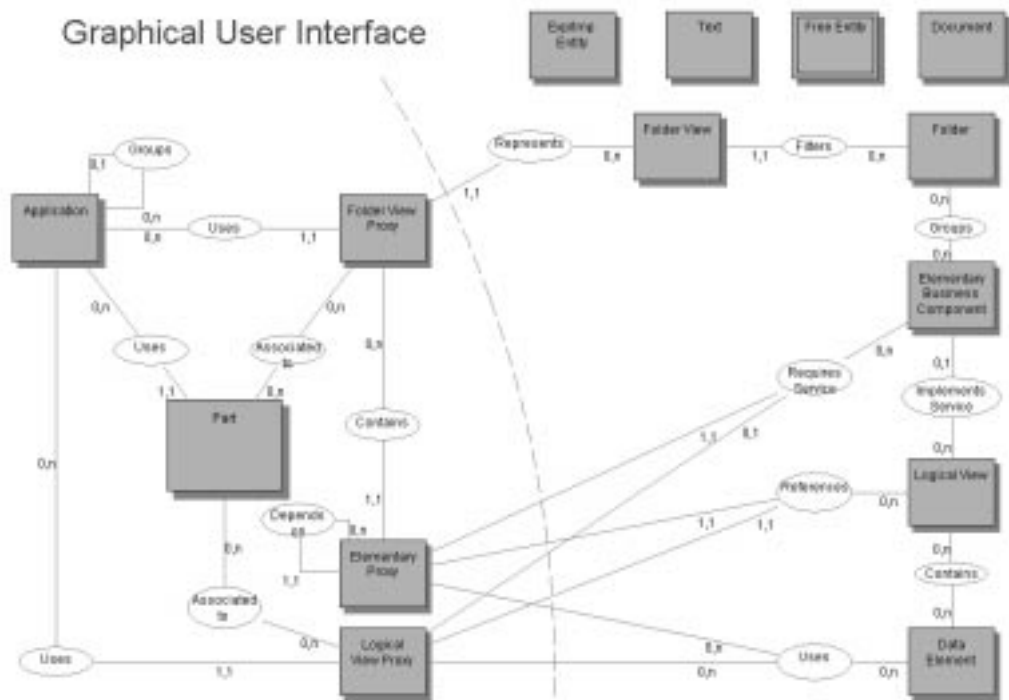
On the server side it comprises the Folder, Folder View, Elementary Business Component (manages a single Logical View), Logical View and Data Element entities.

On the client side it comprises:

- Part and Application entities, common to the folder and simple modes,
- Folder View Proxy and Elementary Proxy entities, specific to the folder mode,
- Logical View Proxy entity, specific to the simple mode.



For more details on the Information Models and their conventions, refer to the *VisualAge Pacbase Workstation Reference Manual*, chapter *The Metamodels*.



### 8.2.3 VisualAge Entities

The Part, Application, Folder View Proxy, Elementary Proxy and Logical View Proxy entities are User Entities dedicated to the storing of the corresponding VisualAge objects.

These entities are described in the following paragraphs subdivided into the following headings:

- Definition,
- Descriptions,
- Cross-references with other VisualAge User Entities,
- and the call code in the Repository.



These entities can be accessed through the **VisualAge Pacbase Workstation** (**Pacbench** module) from the **Entity** box of the **Workstation Manager** window. They can be consulted but they must not be modified.

### 8.2.3.1 Application Entity

The Application entity is a group of Parts describing a VisualAge application.

#### Definition

Occurrence code

Label: VisualAge Identifier

Keywords

Code of the parent Application

Label of the parent Application

#### Descriptions

-D1 to -D6

Source Code

-D9

VisualAge Identifier

#### Cross-references

Child Applications

Applications' parts

Used Folder View Proxy

Used Logical View Proxies

#### Call code

§77

### 8.2.3.2 Part Entity

The Part entity is used to store the source code of a Part.

#### Definition

Occurrence Code

Label: VisualAge Identifier

Keywords

Code of the associated application

#### Descriptions

-D1 à -D6

Source Code

-D8

List of associated Folder View  
Proxy

-D7

List of associated Logical View  
Proxy

-D9

VisualAge Identifier

#### Cross-references

Application

#### Call code

§78

### 8.2.3.3 Folder View Proxy Entity

The Folder View Proxy entity establishes the link with the Folder View and the Elementary Business Component from which it was generated.

#### Definition

Occurrence Code

Label: VisualAge Identifier

Keywords

Code of the associated Folder View

Label of the associated Folder View

Code of the associated Application

Label of the associated Application

#### Descriptions

-D8 Elementary Proxy

-D9 VisualAge Folder View Proxy Identifier

Part

#### Call code

§7X

### 8.2.3.4 Elementary Proxy Entity

The Elementary Proxy entity is not autonomous and is part of a Folder View Proxy. It establishes the link with the Logical View and the Elementary Business Component from which it was generated.

#### Definition

Occurrence Code

Label: VisualAge Identifier

Keywords

Code of the referenced Logical View

Label of the referenced Logical View

Code of the called Elementary Business Component

Label of the called Elementary Business Component

Code of the parent Elementary Proxy

Type of dependence (R/L/D)

Label of the parent Elementary Proxy

#### Descriptions

|     |                                       |
|-----|---------------------------------------|
| -D5 | List of called Data Elements          |
| -D9 | VisualAge Elementary Proxy Identifier |

#### Cross-references

Folder View Proxy

Dependant Proxy

#### Call code

\$7Y

### 8.2.3.5 Logical View Proxy Entity

The Logical View Proxy entity establishes the link with the Logical View and the Business Component from which it was generated.

#### Definition

Occurrence Code

Label: VisualAge Identifier

Keywords

Code of the referenced Logical View

Label of the referenced Logical View

Code of the associated Business Component

Label of the associated Business Component

Code of the associated Application

Label of the associated Application

#### Descriptions

-D5 List of called Data Elements

-D9 VisualAge Logical View Proxy Identifier

#### Cross-references

Part

#### Call code

§79



## 8.3 Operating Principle s

A VisualAge Pacbase database is made up of a network of libraries organized hierarchically and forming a tree structure.

Frozen sessions enable you to manage several images of this network in parallel. These images can evolve ('T'-type frozen session or current session) or remain unchanged ('H'-type frozen session).

Data manipulated in VisualAge, either initially generated from the Repository or directly created in VisualAge, are stored in the Repository as entity occurrences created during the Backup procedure.



You can consult these occurrences but cannot modify them

## 8.4 Backup: Upload to the Repository

The Backup procedure is used to store, in the Repository, generated components after you manipulated them in VisualAge, and the components created locally in VisualAge.

The Backup procedure comprises two steps, one is executed on the host, the other one is executed locally.

### 8.4.1 Java Local Backup

#### 8.4.1.1 Functions

The VisualAge Pacbase for Java Bridge runs on any platform which supports the version 1.1 of the Java Development Kit (JDK).

It is used to create a file containing cross-references between the servers and Java applications. This file can be imported in the Repository.

#### 8.4.1.2 CLASSPATH Principle

The **CLASSPATH** variable is used in VisualAge Pacbase for Java Bridge to analyze classes installed on the client workstation. The value of this variable is read from the machine's environment but you can change this value.

During the first analysis, the program searches, in the access path, for all the files ending with the **.class** extension and those ending with the **.jar** or **.zip** extension. All class files which do not reference any VisualAge Pacbase Proxy objects are discarded immediately. This includes references to the following packages:

- Java Packages,
- All packages starting with **sun**,
- All packages containing the VisualAge Pacbase classes,

All class files which are identified as valid are analyzed. During this process, each class file is decompiled to retrieve the following:

- The class name,

- The parent class,
- Any interfaces implemented by the class,
- All other classes referenced by the class (this includes class dependencies as well as those that are referenced as method arguments, return type, field types and ancestor class).
- Any VisualAge Pacbase constants which are contained in the `cvapFolder`, `cvapServe` and `cvapView` fields.

This information is stored into a table which will be used to save the cross-references.

Once all classes have been analyzed, the program displays a list of the packages which have been found. The program displays the Folder View Proxy objects in a Folders list.

The output file contain a list of all the Folder View Proxy objects you have selected and all the classes that reference them directly or indirectly. If the reference is indirect, the Proxy is not referenced directly in the class but the class references another class which itself references the Proxy.

#### 8.4.1.3 Starting the Program

To start VisualAge Pacbase for Java Bridge, execute the following command in a DOS window:

```
<Java VM executable> com.ibm.vap.bridge.Main [options]
```

Option `-lang <language>` :

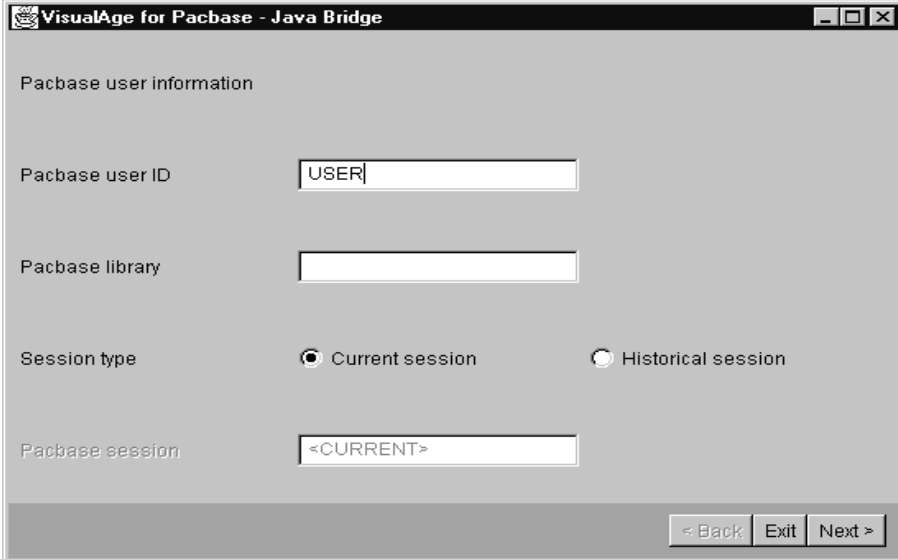
This option enables you to select the language used in the presentation of the application's user interface: `Fr` for French and `Eng` for English. By default, the language of the system is used.

If the bridge has been imported in the VisualAge workstation, you can execute the program as follow:

- In the Workbench, select the Project in which the bridge has been imported.
- Display the pop-up menu, select the `Run` submenu, then choose `Run Main...`
- The `Command Line Argument` window is displayed with the default language option: `-lang fr`.
- Click on `Run`.

#### 8.4.1.4 User Identification

At the program start, the following window opens:



The screenshot shows a window titled "VisualAge for Pacbase - Java Bridge". The window contains the following fields and controls:

- Pacbase user information** (Section Header)
- Pacbase user ID**: A text input field containing the text "USER".
- Pacbase library**: An empty text input field.
- Session type**: Two radio buttons. The first is labeled "Current session" and is selected. The second is labeled "Historical session".
- Pacbase session**: A text input field containing the text "<CURRENT>".
- At the bottom right, there are three buttons: "< Back", "Exit", and "Next >".

Enter the following information in the user identification window:

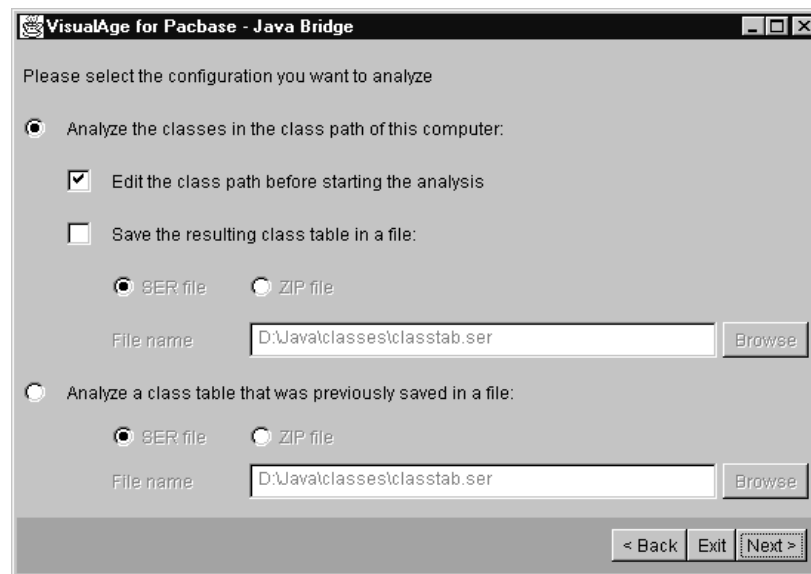
- **user ID**: your VisualAge Pacbase identification (up to 8 characters)
- **VisualAge Pacbase library**: the library allocated to VisualAge Pacbase Repository (up to 3 characters)
- **session Type**: current or frozen
- In the **Pacbase Session** field, enter the four digits that identify the frozen session.

Click on the **Next** button to display the window which enable you to start the classes analysis.

#### 8.4.1.5 Search Options

After you have entered the information on the user context, press the **Next** button.

The following window opens:



This window offers two primary options. You can:

- Search for classes defined in the class path variable.
- Start the classes analysis from a file containing a class table resulting from a previous search. In this case, you must define the format of the file to be analyzed: **.SER** or **.ZIP** files.

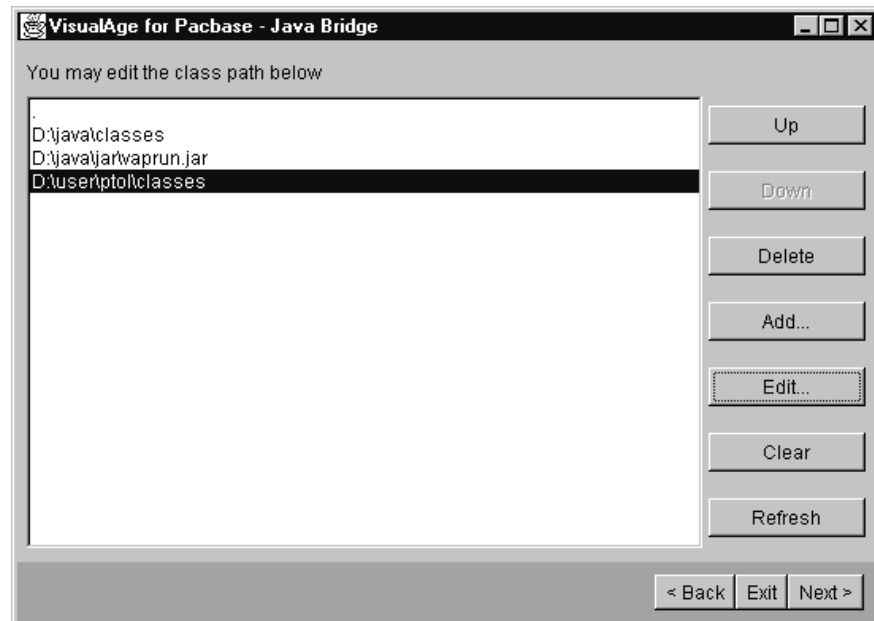
When you choose the **Analyze the classes in the class path** option, and click on the **Next** button, the program executes a global search of all classes available in the client workstation.

You can however limit the scope of the class search. To do so, check the **edit the class path before starting the analysis** option. When you click on the **Next** button, the window which enables you to modify the class path variable is displayed.

Another option enables you to save the result of the search in a class table file for a later use. The program provides two formats for saving this file: compressed or uncompressed, with the **zip** or **SER** extension.

### 8.4.1.6 Modifying the Class Path

If you choose to modify the class path before starting the analysis, the following window opens:



This window is used to edit, add, or delete the class path components. These changes will affect the class analysis only, and will not affect the system environment variable or the behavior of the program.

The buttons of the class path editor window have the following functions:

- **Up**: enables to move up in the list.
- **Down**: enables to move down in the list.
- **Delete**: enables to remove a class path component from the list.
- **Add**: displays a dialog box where you can enter the name of the new component to be added in the list.
- **Edit**: displays a dialog box in which you can modify a file name or a directory name of the list.
- **Clear**: clears all files of the list.
- **Refresh**: resets the class path to its default value.

Press the **Next** button to start the class path analysis and to display the next window.

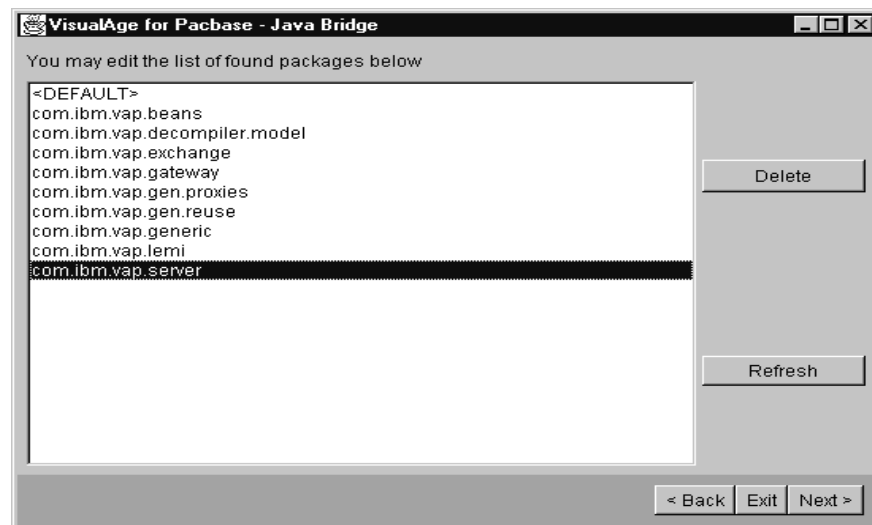
### 8.4.1.7 Modifying the List of Packages

After the class path has been completed or when the class table has been read, the package editing window is displayed.

In this window, you can delete the packages that you do not want to save in the output file.

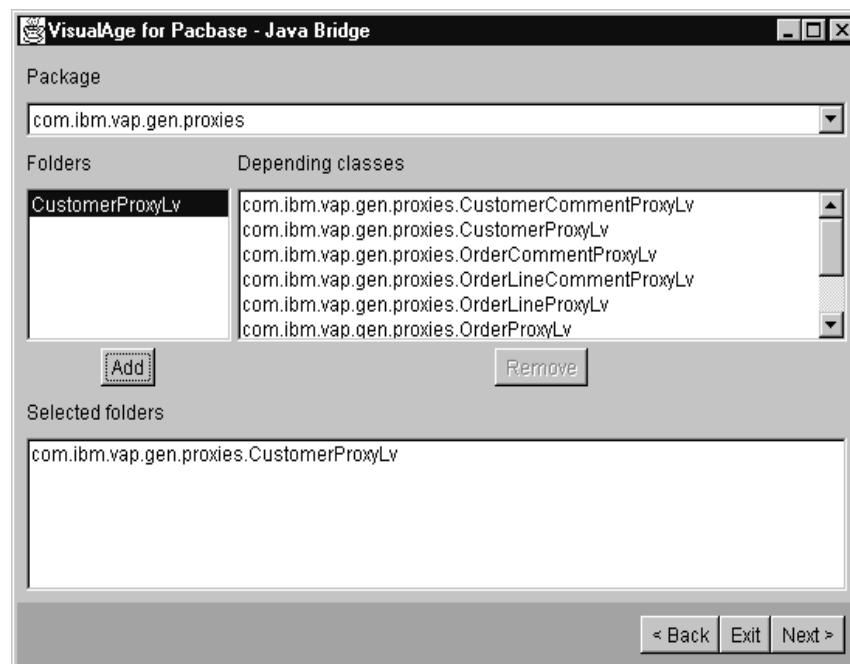
This window's function is the same as that of the class path editor. It enables you to perform the following actions:

- **Delete:** remove a package.
- **Refresh:** reset the package list to its initial value.



### 8.4.1.8 Selecting Folder View Proxy

After the package list has been modified, the program displays a window with the list of folders that have been found.

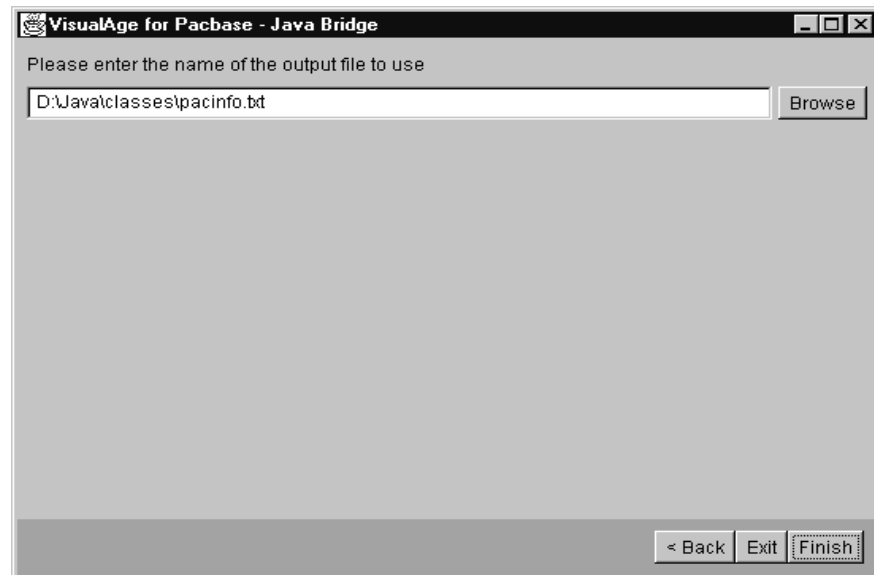


The packages are shown in a combo box at the top of the window. When you select a package, the Folder View Proxy objects of the package are displayed in the **Folders** field. You can then select any Proxy, and use the Add and Remove buttons to add them to the selection list.

Then click on the **Next** button to go to output file selection window.

#### 8.4.1.9 Selecting and Generating the Output File

This window allows you to specify the name of the output file. Once you have selected the file, click on the **Finish** button to start generating the file.



### 8.4.2 Smalltalk Local Backup

#### 8.4.2.1 Functions

The local Backup analyzes the contents of the selected Application source file, formats the backup transactions and prepares the creation of entity cross-references.

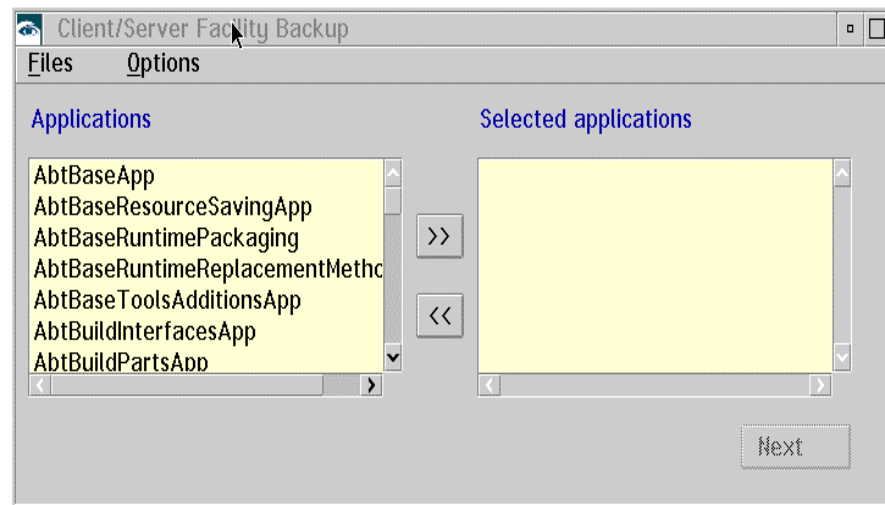
It produces a backup output file containing the update transactions and used as input to the **VUP1** procedure.



Before starting the local backup, in order to prevent certain errors from occurring, check that the advice given in chapter 6 has been respected.

### 8.4.2.2 Local Backup Main Window

To select the Application to backup, open the Pacbench C/S Backup window from VisualAge Organizer, **Tools** menu, **Backup** option.



This window supplies the list of all charged Applications. The angle brackets are used to transfer your applications from one list to the other.

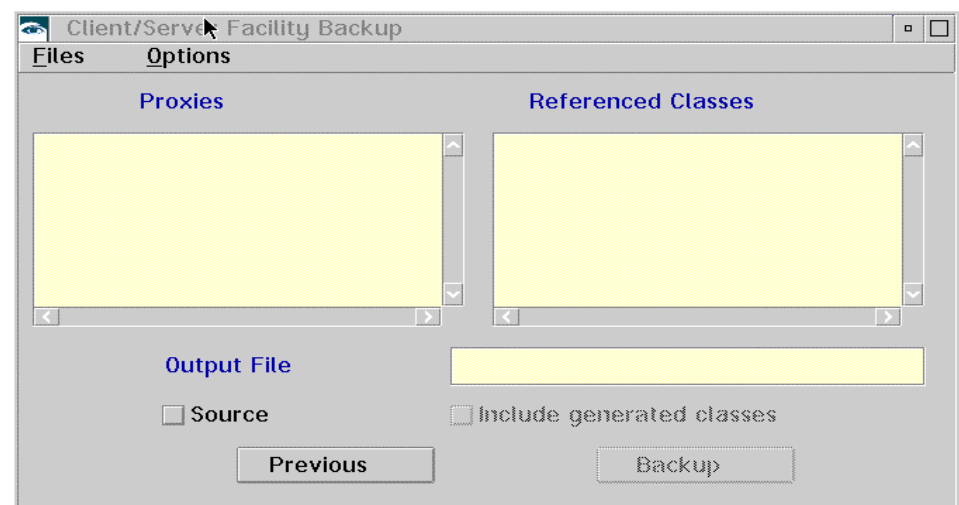
The language option can be accessed from the **Options** menu in the menu bar. The selected language is that used for the presentation of the window.

In the **Options** menu you will find the **User context** option which is used to open the Pacbase contextual information input window.

The **Files** menu contains the **Exit** sub-menu to close the Bridge.

The **Next** button displays the window with the list of proxies resulting from the choice of Applications.

### 8.4.2.3 Window Listing Proxy Objects



The window displays the list of proxies which have been referenced according to the Applications selected in the previous window.



A double click on a Proxy displays the list of proxies referenced by the selected proxy. The references can be direct or indirect. If they are indirect, the proxy is not referenced directly in the class but the class references another class which itself references the proxy.

Enter in the **Output file** area, the name of the extraction file that will receive the Backup result and will be used as input to the VUPI procedure. You can access the list of files through the **Files** menu.

The Backup window contains two options:

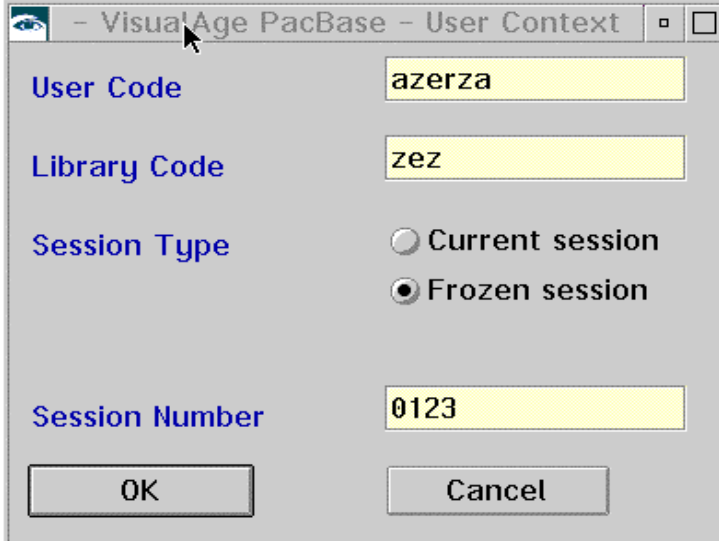
- the **Source** option is used to specify if you want to upload the sources of applications selected in the **Referenced classes** list.
- the **Include generated classes** option is used to upload the sources of the generated classes of the proxies present in an application for which the source is uploaded too.

From the **Files** menu, you can choose the language option (English or French) which will be used for the presentation of the window and access the VisualAge Pacbase contextual information input window via the **User context** choice.

Press the **Backup** button to backup your application. This operation enables you to initiate the creation of a backup file.

#### 8.4.2.4 User Context

The following window opens automatically after initiating the backup. It enables to define the backup environment.



The screenshot shows a dialog box titled "VisualAge PacBase - User Context". It contains the following fields and options:

- User Code:** a text box containing "azerza".
- Library Code:** a text box containing "zez".
- Session Type:** two radio buttons, "Current session" (unselected) and "Frozen session" (selected).
- Session Number:** a text box containing "0123".

At the bottom of the dialog are two buttons: "OK" and "Cancel".

In this window, enter the following information:

- your user code,
- the library code,
- the session type: current session or frozen session,
- the session number.

The information entered in this window are saved in the **vapback.ini** file. This file is read upon the window's opening.

Once the local procedure is executed, you get a transaction file ready to be transferred onto the host with a file transfer facility.

Each VisualAge object of the export source file corresponds to a set of transactions in the backup file.

### 8.4.3 Host Backup

The Host backup processing includes two batch procedures: **VUP1** and **VUP2**. Each procedure's functions, user input and results are described below.

#### 8.4.3.1 VUP1 Procedure: Computing Entity Codes

##### 8.4.3.1.1 Functions

The **VUP1** procedure is essentially used to compute entity codes for VisualAge objects. This procedure creates elements that will be used as input to the **VUP2** procedure. It uses, as input, the backup file produced by the backup procedure local processing and transferred to the host.

This procedure includes the following steps:

- Extraction of entity codes for VisualAge objects with the creation of a correspondence file relating to the entity codes and the VisualAge identifiers.
- Analysis of the backup file and computation of the entity codes of the new entities created in VisualAge to build a file of new codes.

According to the backup file resulting from the backup local processing, the **VUP1** procedure determines the entity codes following these steps:

- Conversion of lowercase characters into uppercase,
- Replacement/deletion of special characters,
- If the VisualAge identifier is made up of 6 characters, search to see if the corresponding entity code exists. Otherwise the first 6 characters of the VisualAge identifier are kept to make up the entity code,
- The existence of the new code is checked first in the VisualAge Pacbase correspondence table, then in the list of already computed codes,
- In the case of a collision, the last character is incremented according to the alphanumeric sequence. At the end of the sequence, if the problem remains, an iterative computation is performed on the second last character, and so on.

*Example for a code such as NOMCLI, the computed sequence is as follows: NOMCLI, NOMCLA, NOMCLB, ..., NOMCLZ, NOMCA, NOMCB ... , NOMCZ, NOMCAA, NOMCAB, NOMCAZ ....., NOMCBA ... etc...*

You can confirm or modify each computed code.



For more information on possible modifications, refer to paragraph **8.4.3.2**.

##### 8.4.3.1.2 User Input

The file created by the backup local processing includes a user assignment line (\*) indicating the VisualAge Pacbase backup environment, followed by description lines, one for each VisualAge object.

You must complete this line with:

- the password
- the Product code and Change Number if the database is under DSMS control.

| POSITION  | LENGTH    | VALUE | MEANING  |
|-----------|-----------|-------|--|
| 02        | 02        | *     | Line code  |
| 04        | 08        |       | User code  |
| <b>12</b> | <b>08</b> |       | <b>Password</b>  |
| 20        | 03        |       | VisualAge Pacbase library code                                       |
| 23        | 05        |       | Session number and status  |
|           |           | ' '   | Current session  |
| <b>58</b> | <b>09</b> |       | <b>Product + Change number if the database is under DSMS control</b> |

#### 8.4.3.1.3 Results

This procedure produces three files:

- Correspondence file between the VisualAge entity codes and identifiers (VP file),
- File with the new entity codes for the new entities created in VisualAge (VV file),
- File with the transactions corresponding to the file resulting from the local backup procedure once duplicates have been removed (VG file).

#### 8.4.3.2 VUP2 Procedure: Generation of the Update

##### 8.4.3.2.1 Functions

The **VUP2** procedure analyses the transaction file resulting from **VUP1** and updates the database by considering the differences between the transaction file and the database. The procedure creates an update transaction file that will be used as input of the UPDT batch procedure.

##### 8.4.3.2.2 User Input

The **VUP2** procedure uses the three input files resulting from the **VUP1** procedure, and takes into account the modifications you may have made in the new codes file (VN). This file is made up of a user assignment line (\*) and a transaction line to update the database.

The **VUP2** includes two different types of user input:

- **Transaction File (VG)**

This file includes a '\*' line and lines used to generate update transactions for the database. The '\*' line is required.

The '\*' line is as follows:

| POSITION | LENGTH | VALUE | MEANING   |
|----------|--------|-------|---|
| 03       | 01     | *     | Line code   |
| 12       | 08     |       | Password  |
| 58       | 09     |       | Product + Change Number if the database is under DSMS control |

- **New Codes File (VN)**

This file contains a transaction line you can modify if you want to assign to a VisualAge entity, an entity code different to that automatically computed by **VUP1**.

Use an editor to perform the changes.

The transaction line is as follows:

| POSITION | LENGTH | VALUE | MEANING         |
|----------|--------|-------|-----------------|
| 46       | 06     |       | New entity code |

#### 8.4.3.2.3 Results

The **VUP2** procedure produces an update transaction file which will be the input of the **UPDT** procedure.

## 8.5 Restoration of Data to the VisualAge Smalltalk Workstation

The Restoration procedure recuperates, in your VisualAge Workstation, objects whose code, produced by the Export function, was saved initially in the VisualAge Pacbase Repository.

The Restoration work unit is a VisualAge Application with associated Parts and child Applications.

The Restoration procedure includes two parts: one is executed on the host, the other is executed locally.

### 8.5.1 Host Restoration

#### 8.5.1.1 VDWN Procedure

Before executing this procedure, it is necessary to allocate the following files:

- File containing the result of the Host Restoration,
- File containing the Logical View Proxy generation commands.

##### 8.5.1.1.1 Functions

The **VDWN** procedure creates a transaction file which will be imported into your VisualAge Workstation.

### 8.5.1.1.2 User Input

The **VDWN** procedure includes two types of user input.

- **Line defining the library-session to be processed**
- **Ligne de commande d'extraction**

| POSITION | LENGTH | VALUE | MEANING                                      |
|----------|--------|-------|--|
| 02       | 01     | *     | Line code                                    |
| 03       | 08     |       | User code                                    |
| 11       | 08     |       | Password                                     |
| 19       | 03     |       | Library code                                 |
| 22       | 05     |       | Number and session status<br>Current session |

- **Extract command line**

| POSITION | LENGTH | VALUE | MEANING                      |
|----------|--------|-------|------------------------------|
| 02       | 01     | Y3    | Line code                    |
| 04       | 02     |       | Object class                 |
|          |        | 77    | VisualAge Application        |
| 06       | 06     |       | VisualAge object entity code |

### 8.5.1.1.3 Results

The **VDWN** procedurees two files:

- Restoration Host file to store the objects extracted from the Repository. This file must be transferred to your VisualAge Workstation to be processed by the local Restoration. The source file produced will be imported into VisualAge.
- Command file used to generate Logical View Proxies if they are used in the extracted objects. This file allows you to generate the Logical View Proxies if necessary. These commands are used as input to the **GPRT** procedure.

## 8.5.2 Local Restoration into VisualAge Pacbase for Smalltalk

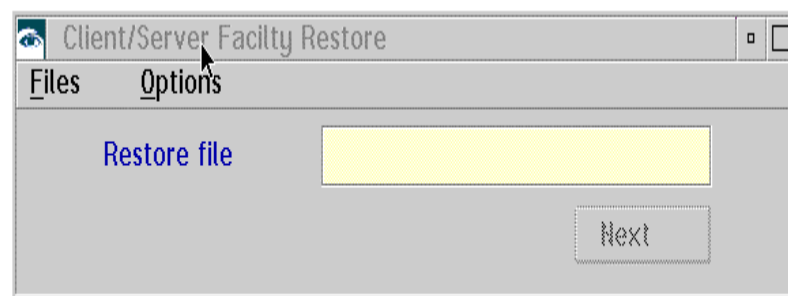
### 8.5.2.1 Functionalities

The local restoration procedure builds an entire description file under the form of a source file from the Restoration Host file resulting from the **VDWN** procedure. This procedure allows you to get a source file ready to be imported into VisualAge using the **File in** function.

### 8.5.2.2 Access to the Local Restoration Window

This window can be accessed from the VisualAge Organizer, **Tools** menu, **Restore** option.

The following window is displayed:



### 8.5.2.3 Pacbench C/S Restore window

In this window, you must choose an input file. The input file is the transactions file produced by the **VDWN** Host Restoration procedure and transferred to your workstation using a file transfer utility.

To specify the name of the file containing the applications to restore must be entered in the **Restore file** box. The other way is to select the input file in the list of files from the **Files** menu, **Open** sub-menu.



You are strongly advised to use the **.APP** extension, which will make file management easier.

In the **Options** menu, choose the language required for the window presentation.

The **Restore** button is used to display the next window.

Choose **File** menu, **Exit** sub-menu to quit.

An error box displays either error messages in case of problem with the restoration file (the **OK** button of the error window is shaded) or the list of applications contained in the restoration.

The **Restore** button is used to pursue the importation.

If you want to stop the importation process, click on the Restoration button, the previous window is displayed.

## 8.6 Purging VisualAge Entities in the Repository

### 8.6.1 Functions

The **VPUR** Purge batch procedure deletes the unused VisualAge entities from the Repository. It searches for these entities in the libraries network. The following are considered as unused:

- the Parts which do not belong to any Application,
- the Applications which have neither archive, parent nor child Applications.

It is possible to limit the search field by specifying a given list of library codes and session numbers.

### 8.6.2 User Input

- **User identification line**

| POSITION | LENGTH | VALUE | MEANING   |
|----------|--------|-------|-----------|
| 02       | 01     | *     | Line code |
| 03       | 08     |       | User code |
| 11       | 08     |       | Password  |

- **Library selection lines**

If no line is specified all libraries are searched.

| POSITION | LENGTH | VALUE | MEANING               |
|----------|--------|-------|-----------------------|
| 02       | 02     | 'SL'  | Line code             |
| 04       | 03     |       | Selected library code |

- **Session selection lines**

If no line is specified all sessions are searched including the current session.

| POSITION | LENGTH | VALUE | MEANING   |
|----------|--------|-------|---|
| 02       | 02     | 'SS'  | Line code   |
| 04       | 03     |       | Session code and status<br>(current session: 9999Z) |

### 8.6.3 Results

A transaction file is produced by **VPUR**, and used as input of the **UPDT** update procedure.





## 9 Appendix 1: Parameterizing Prerequisite Software

To implement the communication of a VisualAge Pacbase application, it is necessary to use specific parameters in the prerequisite software.

The following configuration examples correspond to configurations which have been tested. Therefore, they depend on a particular technical environment. For example, when communicating with the MVS host, the workstations located on the Token-Ring invoke a Communications Manager Gateway, which in turn invokes a 3172 controller to keep the MVS host.

This configuration parameterizing solution only suits a particular context, so an adapted parameterizing solution must be implemented on each site.

### 9.1 IMS CPI-C

#### 9.1.1 MVS and IMS Configuration

##### 9.1.1.1 VTAM Definitions

Prerequisite (minimum): VTAM 3.3 Version

##### 9.1.1.1.1 Definition of ATCSTR of the VTAM

```
*****
NOPROMPT,CONFIG=00,SSCPID=01,
MAXSUBA=31,SUPP=NOSUP,
SSCPNAME=A01M,
SSCPORD=DEFINED,
NETID=NETCGI,
HOSTSA=1,
CRPLBUF=(550,,20,,40,40),
IOBUF=(420,182,25,,40,40),
LFBUF=(300,,0,,20,10),
LPBUF=(50,,0,,5,5),
SFBUF=(50,,0,,5,5),
SPBUF=(90,,0,,5,5),
NOTRACE,TYPE=VTAM
*****
```

##### 9.1.1.1.2 MAC TIC Address (LAN attachment) 3745 controler in NCP :

```
*****
*      TIC BNN                                05742090
*****
A01L1TK LINE ADDRESS=(1088,FULL),
      PORTADD=01,
      LOCADD=400003172000,
      ISTATUS=ACTIVE,
      UACB=(X$P1AX,X$P1AR)
*
A01TK1PU PU ISTATUS=ACTIVE,
      ADDR=01
*****
```

### 9.1.1.1.3 APPC/IMS Definition

A specific APPC/IMS application for LU6.2 must be defined in VTAM (different from the APPL IMS used for the 3270 terminals)

```

*/ * LIB: SYS1.VTAMLST(A0100)
*/ *
A01IMS62 APPL      EAS=50,                ESTIMATED CONCURRENT SESSIONS      *
                   MODETAB=MTLU62,        ⇨      name of the modes table
                   DLOGMOD=LU62,          ⇨      name of the mode in the table
                   APPC=YES,              ⇨      compulsory parameter
                   ACBNAME=A01IMS62,      APPLID FOR ACB
                   AUTH=(ACQ,BLOCK,PASS)  IMS CAN ACQUIRE & PASS TMLS

```

### 9.1.1.1.4 Mode Definition

- Definition of characteristics for the LU6.2 session.
- The SNASVCMG mode is used with the "Parallels Sessions" session.

```

TITLE '--- "MODTABLE" CONCERNANT LES LU 6.2 ---'
*
MTLU62  MODETAB
        SPACE 4
SNASVCMG MODEENT LOGMODE=SNASVCMG, FMPROF=X'13',
          TSPROF=X'07', PRIPROT=X'B0', SECPROT=X'B0',
          COMPROT=X'D0B1', RUSIZES=X'8585', ENCR=B'0000',
          PSERVIC=X'0602000000000000000000000300'
LU62    MODEENT LOGMODE=LU62, TYPE=X'00', FMPROF=X'13',
          TSPROF=X'07', PRIPROT=X'B0', SECPROT=X'B0',
          COMPROT=X'50B1', RUSIZES=X'8787', SRCVPAC=X'00',
          PSNDPAC=X'00', SSNDPAC=X'00',
PSERVIC=X'06020000000000000000000002C00'

```

### 9.1.1.1.5 SNA Definition

```

** LIB: SYS1.VTAMLST(SW1TKR)
**
**
** SWITCHED MAJOR NODE TOKEN-RING ST-MARC :           - 06/07/95.
** ----> LIEN XCA MAJOR NODE ==> XCA1TKR (IBM3172-3)
** ----> LIEN GROUPE XCA      ==> GRP02
**
** -----
** - MODEL FOR IDBLK X'05D' - OS/2 COMMUNICATIONS MANAGER
** -----
**
SW1TKR  VBUILD TYPE=SWNET,MAXNO=99,MAXGRP=10
**
** -----
** ----->  DEFINING A GATEWAY TOKEN-RING --> GTWTK1  <-----
** -----
W1TK00  PU      ADDR=50,
          CPNAME=GTWTK1,
          IDBLK=05D,
          IDNUM=00002,
          DYNLU=YES,
          MAXPATH=1,
          DISCNT=NO,
          IRETRY=YES,
          VPACING=7,
          PACING=7,
          SSCPFM=USSSCS,
          MAXDATA=4096,
          PUTYPE=2

```

With the option DYNLU=YES it is not necessary to give an additional definition of the LU 6.2 in VTAM (for the workstations of the TR network).

### 9.1.1.1.6 Independent LU Definition

Despite what has been stated above, it may be worthwhile defining an independent LU for the first tests:

```

** LIB: SYS1.VTAMLST(SW1TKR)
**
**
** SWITCHED MAJOR NODE TOKEN-RING ST-MARC :           - 06/07/95.
**
**
** INDEPENDENT LUS
**
IMS4349      LU      LOCADDR=0,
              ISTATUS=ACTIVE,
              DLOGMOD=LU62,
              MODETAB=MTLU62

```

### 9.1.1.2 APPC/MVS Definitions

Prerequisite (minimum): MVS/ESA Version 4.2

Two SYS1.PARMLIB members are required to define the characteristics of the Local LU APPC/MVS and ASCH (APPC scheduler).

### 9.1.1.2.1 Local LU APPC/MVS Definition

- Parameterization

```
BROWSE -- SYS1.PARMLIB(APPCPM00) - 01.12 ----- LINE 0000
COMMAND ==>
***** TOP OF DATA *****
/*****/
/* THE FOLLOWING PARAMETERS ARE FOR THE APPC ADDRESS SPACE. */
/* THE APPC ADDRESS SPACE HANDLES THE ACTUAL COMMUNICATIONS. */
/*
/* THESE MEMBERS PROVIDE THE LINKAGE BETWEEN LU NAMES AND */
/* TRANSACTION SCHEDULERS. */
/*****/
LUADD
    ACBNAME(A01IMS62)           => corresponds to APPL APPC/IMS of the VTAM
    SCHED(CGIB)
    BASE
    TPDATA(UTI.APPCTP)
    TPLEVEL(SYSTEM)
SIDEINFO DATASET(UTI.APPCSI)
```

- Startup

```
BROWSE -- SYS1.PROCLIB(APPC) -----
COMMAND ==>
***** TOP OF DATA *****
//APPC PROC APPC=00
//APPC EXEC PGM=ATBINITM,PARM='APPC=&APPC',REGION=0K
***** BOTTOM OF DATA *****
```

### 9.1.1.2.2 scheduler APPC/MVS Definition

- Parameterization

```
BROWSE -- SYS1.PARMLIB(ASCHPM00) - 01.00 -----
COMMAND ==>
***** TOP OF DATA *****
/*****/
/* THE FOLLOWING IS ADDED TO ENABLE APPC/MVS.
/*****/
CLASSADD CLASSNAME(SVSAMP)
    MAX(10)
    MIN(2)
    RESPGOAL(0.02)
    MSGLIMIT(700)
OPTIONS DEFAULT(SVSAMP)
    SUBSYS(JES2)
TPDEFAULT REGION(4M)
    TIME(10,30)
    MSGLEVEL(1,1)
    OUTCLASS(R)
***** BOTTOM OF DATA *****
```

- Startup

```
BROWSE -- SYS1.PROCLIB(ASCH) -----
COMMAND ==>
***** TOP OF DATA *****
//ASCH PROC ASCH=00
//ASCH EXEC PGM=ASBSCHIN,PARM='ASCH=&ASCH',REGION=0K
***** BOTTOM OF DATA *****
```

### 9.1.1.3 IMS Definitions

Prerequisite (minimum): IMS 4.1 Version

#### 9.1.1.3.1 IMSCTRL Macro

To generate IMS, it is necessary to use the version of the MVS/ESA libraries corresponding to the third parameter of the SYSTEM keyword. The minimum MVS/ESA version required is 4.2.

```
BROWSE -- EX.IMS410.SOURCE(STAGE1) - 01.28 -----
COMMAND ==>
*
* IMSCTRL MACRO --
*
  IMSCTRL SYSTEM=(VS/2,CTLBLKS,4.2),
          DBRC=(YES,NO),
          DBRCNM=DBRC41,
          DLINM=DLISAS41,
          DCLWA=YES,
          IMSID=CGIB,           => corresponds to parameter SCHED of the
                                APPCPM00
          NAMECHK=(YES,S1),
          MAXIO=(,015),
          MAXREGN=(008,512K,A,A),
          MCS=(8),
          DESC=7,
          MAXCLAS=020
```

#### 9.1.1.3.2 Startup

For IMS to be able to make a connection with APPC/MVS, it is necessary to specify APPC=Y in the IMS Startup Job (DFSPBxxx where xxx is the region suffix).

#### 9.1.1.3.3 Transaction Definition

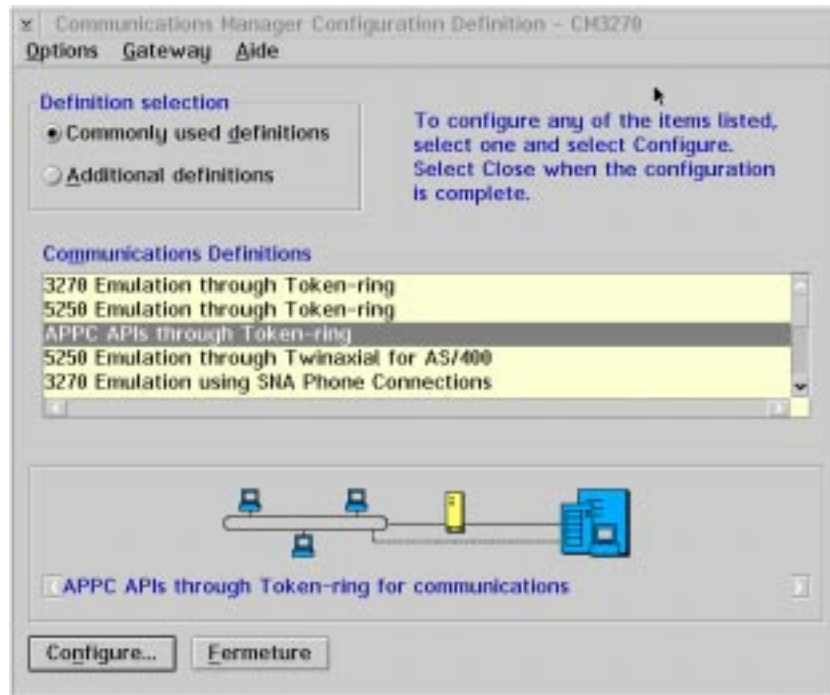
```
BROWSE -- EX.IMS410.SOURCE(STAGE1) - 01.28 -----
COMMAND ==>
***** TRANSACTION DB2 POUR BABY CLIENT (PTAB) *****
APPLCTN PSB=BABIVG
TRANSACT CODE=BABI,SEGSIZE=00000,MODE=SNGL,SEGNO=00000,
          PRTY=(07,10,00002),PROCLIM=(00005,00015),EDIT=ULC,
          MSGTYPE=(SNGLSEG,RESPONSE,4)
```

## 9.1.2 OS/2 Configuration

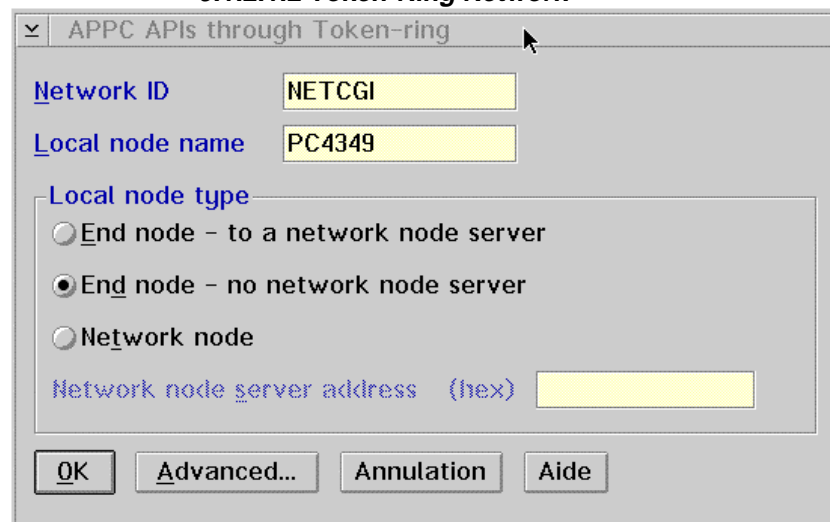
### 9.1.2.1 Communication Manager/2: APPC Configuration

Prerequisite (minimum): Communication Manager/2 Version 1.11

#### 9.1.2.1.1 Communication Choice



#### 9.1.2.1.2 Token-Ring Network



Network ID: SNA network identifier to which the workstation is connected (NETID of VTAM)

Local node name: workstation identification in the network

**9.1.2.1.3 Token-Ring DLC Adapter**

Token Ring or Other LAN Types DLC Adapter Parameters

Adapter

Free unused links

Send alert for beaoning

Maximum activation attempts  (1 - 99)

Maximum link stations  (1 - 255)

Maximum I-field size  (265 - 16393)

Percent of incoming calls (%)  (0 - 100)

Link establishment retransmission count  (1 - 127)

Retransmission threshold  (1 - 127)

Local sap (hex)  (04 - 9C)

C&SM LAN ID

Connection network name (optional)  .

Maximum I-field size: This value depends on the CM/2 gateway.

#### 9.1.2.1.4 Local Node

Local Node Characteristics

Network ID

Local node name

Node type

End node to network node server

End node - no network node server

Network node

Your network node server address (hex)

Local node ID (hex)

#### 9.1.2.1.5 Connections List

Connections List

Choose the type of node to change or create connections to nodes of that type.

Selecting a partner type will display connections to nodes of that type in the list.

Partner type

To network node  To peer node  To host

| Link Name | Adapter                          | Adapter Number |
|-----------|----------------------------------|----------------|
| HOST0001  | Réseau en anneau à jeton ou tout | 0              |

Comment



Connection to a Host

Link name: HOST0001  Activate at startup

Local PU name: PC4349  APPN support

Node ID (hex): 05D 00000

LAN destination address (hex): 400003172000 Address format: Token Ring Remote SAP (hex): 04

Adjacent node ID (hex):

Partner network ID: NETCGI

Partner node name: A01M (Required for partner LU definition)

Use this host connection as your focal point support

Optional comment:

Buttons: OK, Define Partner LUs..., Annulation, Aide

Node ID: Not used in our environment because we invoke the CM/2 Gateway via its MAC Address.  
 LAN destination address: MAC Address of the CM/2 Gateway  
 Partner Node Name: VTAM node name

**9.1.2.1.6 Partner LU**

To create the Partner LU, select the Define Partner LUs... push button from the previous screen.

Partner LU

Fully qualified LU name: NETCGI . A01IMS62

Alias: A01IMS62

Conversation security verification

Dependent partner LU

Partner LU is dependent

Uninterpreted name:

Optional comment:

Buttons: OK, Annulation, Aide

LU Name: NETID parameter of the VTAM (NETCGI) definition + APPL parameter for IMS definition in the VTAM List (A01IMS62).

### 9.1.2.1.7 Local LU

Local LU

LU name

Alias

NAU address

Independent LU

Dependent LU NAU ...  (1 - 254)

Host link

Use this local LU as your default local LU alias

Optional comment

LU Name: no equivalent on the Host if the Dynamic LU option DYNLU=YES

### 9.1.2.1.8 Mode

Mode Definition

Mode name

Class of service

Mode session limit  (0 - 32767)

Minimum contention winners  (0 - 32767)

Receive pacing window  (0 - 63)

Compression

Compression need

PLU->SLU compression level

SLU->PLU compression level

RU size

Default RU size

Maximum RU size  (256 - 16384)

Optional comment

Mode Name: mode defined in the VTAM mode table, LOGMODE parameter of the MODEENT macro-instruction.

### 9.1.2.1.9 Side Info

Symbolic destination name

**Partner LU**  
 Fully qualified name

Alias

**Partner TP**  
 Service TP  
 TP name

**Security type**  Same  None  Program

Mode name

Optional comment

This screen is used in the CPI-C communications to connect the Client and the Server programs using APPC.

The value of Symbolic destination name must correspond to the external name of the communications monitor in the Repository.

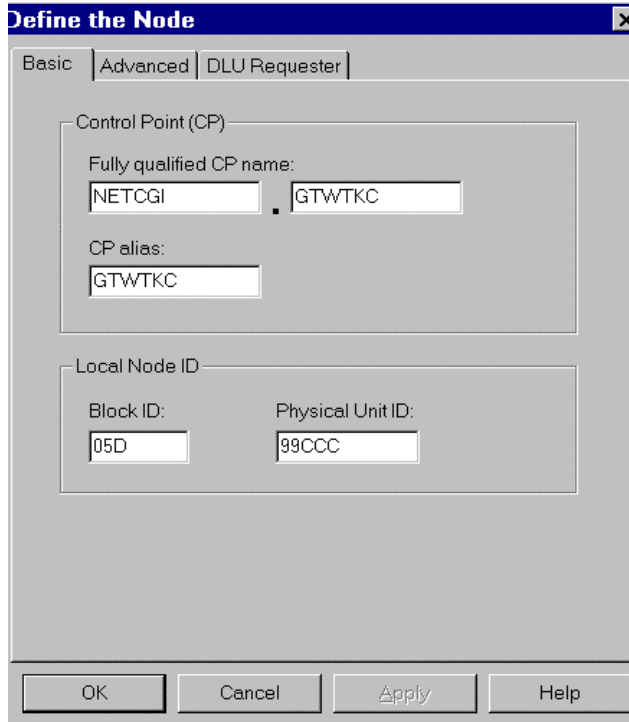
TP name: transaction code activating the communications monitor on the Host.

## 9.1.3 WINDOWS 95/NT Configuration

### 9.1.3.1 Personal Communications : APPC Configuration

Minima requis : Personal Communications Version 4.2

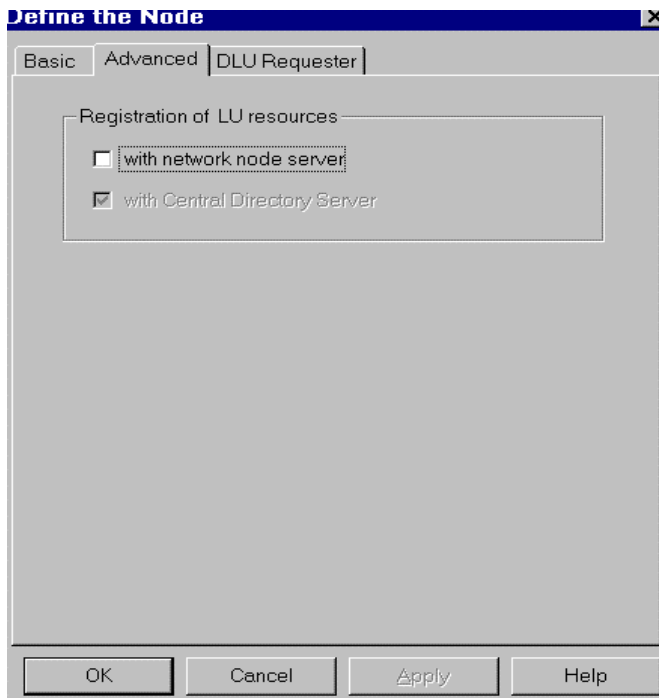
### 9.1.3.1.1 Define the Node



The screenshot shows the 'Define the Node' dialog box with the 'Basic' tab selected. The 'Control Point (CP)' section contains a 'Fully qualified CP name' field with the value 'NETCGI.GTWTKC' and a 'CP alias' field with the value 'GTWTKC'. The 'Local Node ID' section contains a 'Block ID' field with the value '05D' and a 'Physical Unit ID' field with the value '99CCC'. At the bottom, there are buttons for 'OK', 'Cancel', 'Apply', and 'Help'.

Network ID: SNA network identifier to which the workstation is connected (NETID of VTAM).

Local node name: workstation identification in the network.



The screenshot shows the 'Define the Node' dialog box with the 'DLU Requester' tab selected. The 'Registration of LU resources' section contains two checkboxes: 'with network node server' (unchecked) and 'with Central Directory Server' (checked). At the bottom, there are buttons for 'OK', 'Cancel', 'Apply', and 'Help'.

### 9.1.3.1.2 Define the Lan Device

The screenshot shows the 'Define a LAN Device' dialog box with the 'Basic' tab selected. The 'Port name' field contains 'LAN0\_04'. Below it, the 'Adapter number' is set to '0' and the 'Local SAP' is set to '04'. The dialog has 'OK', 'Cancel', 'Apply', and 'Help' buttons at the bottom.

|                 |         |
|-----------------|---------|
| Port name:      | LAN0_04 |
| Adapter number: | 0       |
| Local SAP:      | 04      |

The screenshot shows the 'Define a LAN Device' dialog box with the 'Performance' tab selected. It contains several timeout and count fields:

|                                    |      |         |
|------------------------------------|------|---------|
| Idle timeout:                      | 30   | seconds |
| Busy state timeout:                | 15   | seconds |
| REJ response timeout:              | 10   | seconds |
| Acknowledgement delay:             | 100  | ms      |
| POLL response timeout:             | 3000 | ms      |
| Acknowledgement timeout:           | 1000 | ms      |
| Anticipated outstanding transmits: | 16   |         |
| Receive buffer count:              | 16   |         |

### 9.1.3.1.3 Define the Lan Connection

The screenshot shows the 'Define a LAN Connection' dialog box with the 'Basic' tab selected. The fields are as follows:

- Link station name: LINK0000
- Device name: LAN0\_04
- Discover network addresses... button
- Destination address: 4000000A3172
- Remote SAP: 04

Buttons at the bottom: OK, Cancel, Apply, Help.

Destination address: MAC Address of the TIC IBM 3745

The screenshot shows the 'Define a LAN Connection' dialog box with the 'Security' tab selected. The fields and options are as follows:

- Activate link at start
- HPR support
- APPN support
- Auto-activate support
- Link to preferred NN server
- Solicit SSCP sessions
- PU name: GTWTKC
- Local Node ID:
  - Block ID: 05D
  - Physical Unit ID: 99CCC

Buttons at the bottom: OK, Cancel, Apply, Help.

**Define a LAN Connection**

Basic | Advanced | Security

Adjacent CP name: [ ] . [ ]

Adjacent CP type: APPN Node TG number: 0

Adjacent node ID:

Block ID: 000 Physical Unit ID: 00000

OK Cancel Apply Help

**9.1.3.1.4 Define a Partner LU 6.2**

**Define a Partner LU 6.2**

Basic | Advanced

Partner LU name: NETCGI . A01IMS62

Partner LU alias: A01IMS62

Fully qualified CP name: NETCGI . A01M

OK Cancel Apply Help

LU Name: NETID parameter of the VTAM (NETCGI) definition + APPL parameter for IMS definition in the VTAM List (A01IMS62).

**Define a Partner LU 6.2**

Basic | Advanced

Maximum LL record size:  
32767

Conversation security support

Parallel session support

OK Cancel Apply Help

### 9.1.3.1.5 Define a Local LU 6.2

**Define a Local LU 6.2**

Basic

Local LU name:  
CICSFBFB  Dependent LU

Local LU alias: CICSFBFB

PU name: [dropdown]

NAU address: [dropdown]

LU session limit: 0

OK Cancel Apply Help

LU Name: no equivalent on the Host if the Dynamic LU option DYNLU=YES  
 In Windows 95 environment variables define the local LU6.2 used by default  
 => SET APPCLU=CICSFBFB



**9.1.3.1.6 Define a Mode**

The screenshot shows the 'Define a Mode' dialog box with the 'Basic' tab selected. The 'Mode name' field contains 'LU62'. The 'PLU mode session limit' field contains '4'. The 'Minimum contention winner sessions' field contains '2'. At the bottom, there are four buttons: 'OK', 'Cancel', 'Apply', and 'Help'.

Mode Name: mode defined in the VTAM mode table, LOGMODE parameter of the MODEENT macro-instruction.

The screenshot shows the 'Define a Mode' dialog box with the 'Advanced' tab selected. The 'Maximum negotiable session limit' field contains '4'. The 'Receive pacing window size' field contains '1'. The 'Class of Service name' dropdown menu is set to '#CONNECT'. There is an unchecked checkbox for 'Use cryptography'. A sub-section contains a checked checkbox for 'Use default RU size' and a 'Maximum RU size' field containing '4096'. At the bottom, there are four buttons: 'OK', 'Cancel', 'Apply', and 'Help'.

### 9.1.3.1.7 Define CPI-C Side Information

The screenshot shows the 'Define CPI-C Side Information' dialog box with the 'Basic' tab selected. The fields are as follows:

- Symbolic destination name:
- Mode name:
- Partner LU name:  .
- TP name:
- Service TP

Buttons at the bottom: OK, Cancel, Apply, Help.

This screen is used in the CPI-C communications to connect the Client and the Server programs using APPC.

The value of *Symbolic destination name* must correspond to the external name of the communications monitor in the Repository.

TP name: transaction code activating the communications monitor on the Host.

The screenshot shows the 'Define CPI-C Side Information' dialog box with the 'Security' tab selected. The fields are as follows:

- Conversation security:
- Security user ID:
- Security password:

Buttons at the bottom: OK, Cancel, Apply, Help.

## 9.2 CICS CPI-C

### 9.2.1 MVS and CICS Configuration

#### 9.2.1.1 VTAM Definitions

Prerequisite (minimum): VTAM Version 3.3

##### 9.2.1.1.1 ATCSTR of the VTAM Definition

```
*****
NOPROMPT,CONFIG=00,SSCPID=01,
MAXSUBA=31,SUPP=NOSUP,
SSCPNAME=A01M,
SSCPORD=DEFINED,
NETID=NETCGI,
HOSTSA=1,
CRPLBUF=(550,,20,,40,40),
IOBUF=(420,182,25,,40,40),
LFBUF=(300,,0,,20,10),
LPBUF=(50,,0,,5,5),
SFBUF=(50,,0,,5,5),
SPBUF=(90,,0,,5,5),
NOTRACE,TYPE=VTAM
*****
```

##### 9.2.1.1.2 MAC Address of TIC (LAN attachment) 3745 controller in NCP :

```
*****
*      TIC BNN                05742090
*****
A01L1TK LINE ADDRESS=(1088,FULL),
      PORTADD=01,
      LOCADD=40003172000,
      ISTATUS=ACTIVE,
      UACB=(X$P1AX,X$P1AR)
*
A01TK1PU PU ISTATUS=ACTIVE,
      ADDR=01
*****
```

##### 9.2.1.1.3 CICS Definition

```
A01CICS1 APPL  EAS=160 ,                ESTIMATED CONCURRENT SESSIONS  *
              ACBNAME=CICST ,          APPLID FOR ACB                *
              AUTH=( ACQ , VPACE , PASS ) , CICS CAN ACQUIRE & PASS TMLS  *
*                                                    CICS CAN REQUEST BLOCKED INPUT
              PARSESS=YES                ⇒ Parallel Sessions Supports
              SONSCIP=YES ,              *
              MODETAB=MTLU62            ⇒ name of the modes table
```

#### 9.2.1.1.4 Mode Definition

- Definition of the characteristics for the LU6.2 Sessions.
- The SNASVCMG mode is used with the "Parallels Sessions" support.

```
TITLE '--- "MODTABLE" CONCERNANT LES LU 6.2 ---'
*
MTLU62  MODETAB
        SPACE 4
SNASVCMG MODEENT LOGMODE=SNASVCMG, FMPROF=X'13',
          TSPROF=X'07', PRIPROT=X'B0', SECPROT=X'B0',
          COMPROT=X'D0B1', RUSIZES=X'8585', ENCR=B'0000',
          PSERVIC=X'06020000000000000000000300'
LU62    MODEENT LOGMODE=LU62, TYPE=X'00', FMPROF=X'13',
          TSPROF=X'07', PRIPROT=X'B0', SECPROT=X'B0',
          COMPROT=X'50B1', RUSIZES=X'8787', SRCVPAC=X'00',
          PSNDPAC=X'00', SSNDPAC=X'00',
PSERVIC=X'060200000000000000000002C00'
```

#### 9.2.1.1.5 SNA Definition

```
*/ * LIB: SYS1.VTAMLST(SW1TKR)
*/ *
*/ *
*/ * SWITCHED MAJOR NODE TOKEN-RING ST-MARC :           - 06/07/95.
*/ * ---> LIEN XCA MAJOR NODE ==> XCA1TKR (IBM3172-3)
*/ * ---> LIEN GROUPE XCA      ==> GRP02
*
*   - MODEL FOR IDBLK X'05D' - OS/2 COMMUNICATIONS MANAGER
*
*
SW1TKR  VBUILD TYPE=SWNET,MAXNO=99,MAXGRP=10
*
*
* ----->  DEFINING A GATEWAY TOKEN-RING -->  GTWK1  <-----
*
*
W1TK00  PU      ADDR=50,
          CPNAME=GTWTK1,
          IDBLK=05D,
          IDNUM=00002,
          DYNLU=YES,
          MAXPATH=1,
          DISCNT=NO,
          IRETRY=YES,
          VPACING=7,
          PACING=7,
          SSCPFM=USSSCS,
          MAXDATA=4096,
          PUTYPE=2
```

With the option DYNLU=YES it is not necessary to give an additional definition of the LU 6.2 in VTAM (for the workstations of the TR network).

### 9.2.1.1.6 Independent LU Definition

Despite what has been stated above, it may be worthwhile defining an independent LU for the first tests:

```

** LIB: SYS1.VTAMLST(SW1TKR)
**
**
** SWITCHED MAJOR NODE TOKEN-RING ST-MARC :           - 06/07/95.
**
**
* INDEPENDENT LUS
*
CICSFBFB LU LOCADDR=0,
           ISTATUS=ACTIVE,
           DLOGMOD=LU62,
           MODETAB=MTLU62

```

### 9.2.1.2 APPC/MVS Definitions

Prerequisite (minimum): MVS/ESA Version 4.2

There is no specific definition as we use the APPC layer delivered with the CICS version.

### 9.2.1.3 CICS Definitions

#### 9.2.1.3.1 InterSystem Communication Parameter in the SIT Table

ISC=YES

#### 9.2.1.3.2 Connection

```

Connection      : SGFB
Group           : GRPISC5
DEscription     :
CONNECTION IDENTIFIERS
Netname        : CICSFBFB  ⇨ same declaration as Local LU in CM/2 (free code)
INDsys         :
REMOTE ATTRIBUTES
REMOTESystem   :
REMOTENAME     :
CONNECTION PROPERTIES
ACcessmethod   : Vtam
Protocol       : Appc
SInglesess     : No        ⇨ If independent LU
DAstream       : User
RECORDformat   : U
OPERATIONAL PROPERTIES
AUtoconnect    : Yes
INService      : Yes
SECURITY
SEcurityname   : PTPD
ATTachsec      : Verify    ⇨ userid and password check at the conversation
BINDPassword   :
BINDSecurity   : No

```

**9.2.1.3.3 Session**

```

Sessions      : SESSIOFB
Group        : GRPISC5
DEscription  :
SESSION IDENTIFIERS
Connection   : SGFB           ⇒ code of the connection defined above
SESSName     :
NETnameq    :
MOdename    : LU62           ⇒ mode defined in the VTAM MODTABLE
SESSION PROPERTIES
Protocol     : Appc
MAXimum     : 004 , 002
RECEIVEPfx  :
RECEIVECount :
SENDPfx     :
SENDCount   :
SENDSize    : 08192
RECEIVESize : 08192
SESSPriority : 000
Transaction :
OPERATOR DEFAULTS
OPERId      :
OPERPriority : 000
OPERRsl     : 0
OPERSecurity : 1
PRESET SECURITY
USERId     :
OPERATIONAL PROPERTIES
Autoconnect : Yes
INservice   :
Buildchain  : Yes
USERArealen : 000
IOarealen   : 00000 , 00000
RELreq     : Yes
DIScreq    : No
NEPclass   : 000
RECOVERY
RECOVOption : Sysdefault
RECOVNotify : None

```

**9.2.1.3.4 Transaction Definition**

OBJECT CHARACTERISTICS

CICS RELEASE =

0330

CEDB View

|             |        |   |  |
|-------------|--------|---|--|
| TRansaction | : VIC0 | ⇒ | code to report in the TP name of the CPIC Side Information of CM/2 |
|-------------|--------|---|--|

|       |          |  |  |
|-------|----------|--|--|
| Group | : VISUAL |  |  |
|-------|----------|--|--|

|             |   |  |  |
|-------------|---|--|--|
| DEscription | : |  |  |
|-------------|---|--|--|

|         |          |   |                                    |
|---------|----------|---|------------------------------------|
| PROgram | : AVVMSV | ⇒ | code of the communications monitor |
|---------|----------|---|------------------------------------|

|         |         |  |         |
|---------|---------|--|---------|
| TWAsize | : 00000 |  | 0-32767 |
|---------|---------|--|---------|

|         |            |  |  |
|---------|------------|--|--|
| PROfile | : DFHCICST |  |  |
|---------|------------|--|--|

|              |   |  |  |
|--------------|---|--|--|
| PARTitionset | : |  |  |
|--------------|---|--|--|

|        |           |  |                    |
|--------|-----------|--|--------------------|
| STatus | : Enabled |  | Enabled ! Disabled |
|--------|-----------|--|--------------------|

|            |         |  |         |
|------------|---------|--|---------|
| PRIMedsize | : 00000 |  | 0-65520 |
|------------|---------|--|---------|

|             |         |  |             |
|-------------|---------|--|-------------|
| TASKDATAloc | : Below |  | Below ! Any |
|-------------|---------|--|-------------|

|             |        |  |             |
|-------------|--------|--|-------------|
| TASKDATAKey | : User |  | User ! Cics |
|-------------|--------|--|-------------|

REMOTE ATTRIBUTES

|         |      |  |          |
|---------|------|--|----------|
| DYnamic | : No |  | No ! Yes |
|---------|------|--|----------|

|              |   |  |  |
|--------------|---|--|--|
| REMOTESystem | : |  |  |
|--------------|---|--|--|

|            |   |  |  |
|------------|---|--|--|
| REMOTENAME | : |  |  |
|------------|---|--|--|

|        |   |  |  |
|--------|---|--|--|
| TRProf | : |  |  |
|--------|---|--|--|

|        |   |  |          |
|--------|---|--|----------|
| Localq | : |  | No ! Yes |
|--------|---|--|----------|

SCHEDULING

|          |       |  |       |
|----------|-------|--|-------|
| PRIOrity | : 001 |  | 0-255 |
|----------|-------|--|-------|

|        |      |  |           |
|--------|------|--|-----------|
| TClass | : No |  | No ! 1-10 |
|--------|------|--|-----------|

ALIASES

|       |   |  |  |
|-------|---|--|--|
| Alias | : |  |  |
|-------|---|--|--|

|         |   |  |  |
|---------|---|--|--|
| TASKReq | : |  |  |
|---------|---|--|--|

|         |   |  |  |
|---------|---|--|--|
| XTRanid | : |  |  |
|---------|---|--|--|

|        |   |  |  |
|--------|---|--|--|
| TPName | : |  |  |
|--------|---|--|--|

|  |   |  |  |
|--|---|--|--|
|  | : |  |  |
|--|---|--|--|

|         |   |  |  |
|---------|---|--|--|
| XTPname | : |  |  |
|---------|---|--|--|

|  |   |  |  |
|--|---|--|--|
|  | : |  |  |
|--|---|--|--|

|  |   |  |  |
|--|---|--|--|
|  | : |  |  |
|--|---|--|--|

RECOVERY

|         |      |  |             |
|---------|------|--|-------------|
| DTImout | : No |  | No ! 1-6800 |
|---------|------|--|-------------|

|         |           |  |                         |
|---------|-----------|--|-------------------------|
| Indoubt | : Backout |  | Backout ! Commit ! Wait |
|---------|-----------|--|-------------------------|

|         |      |  |          |
|---------|------|--|----------|
| REStart | : No |  | No ! Yes |
|---------|------|--|----------|

|        |       |  |          |
|--------|-------|--|----------|
| SPurge | : Yes |  | No ! Yes |
|--------|-------|--|----------|

|        |       |  |          |
|--------|-------|--|----------|
| TPUrge | : Yes |  | No ! Yes |
|--------|-------|--|----------|

|      |       |  |          |
|------|-------|--|----------|
| DUmp | : Yes |  | Yes ! No |
|------|-------|--|----------|

|       |       |  |          |
|-------|-------|--|----------|
| TRACe | : Yes |  | Yes ! No |
|-------|-------|--|----------|

SECURITY

|       |      |  |          |
|-------|------|--|----------|
| RESec | : No |  | No ! Yes |
|-------|------|--|----------|

|        |      |  |          |
|--------|------|--|----------|
| Cmdsec | : No |  | No ! Yes |
|--------|------|--|----------|

|        |      |  |          |
|--------|------|--|----------|
| Extsec | : No |  | No ! Yes |
|--------|------|--|----------|

|         |      |  |      |
|---------|------|--|------|
| TRANsec | : 01 |  | 1-64 |
|---------|------|--|------|

|     |      |  |               |
|-----|------|--|---------------|
| RS1 | : 00 |  | 0-24 ! Public |
|-----|------|--|---------------|

Additionally, if a transaction accesses a DB2 database, this database must be authorized and linked to the DB2 plan. The transaction code and the DB2 plan must therefore be declared in the RCT table. In this example, the transaction VIC0 and the DB2 plan ATDF are used by the client application:

```
DSNRCT TYPE = ENTRY, TXID=(VIC0),
          THRDM=6, THRDA=6, PLAN=ATDF, AUTH=(USERID, *, *)
```

## 9.2.2 OS/2 Configuration

### 9.2.2.1 Communication Manager/2: APPC configuration

Prerequisite (minimum): Communication Manager/2 Version 1.11

#### 9.2.2.1.1 Communication Choice

Refer to the screen contained in the equivalent paragraph in the 9.1 Subchapter, *IMS CPI-C*.

#### 9.2.2.1.2 Token Ring Network

Refer to the screen contained in the equivalent paragraph in the 9.1 Subchapter, *IMS CPI-C*.

#### 9.2.2.1.3 Token Ring DLC Adapter

Refer to the screen contained in the equivalent paragraph in the 9.1 Subchapter, *IMS CPI-C*.

#### 9.2.2.1.4 Local Node

Refer to the screen contained in the equivalent paragraph in the 9.1 Subchapter, *IMS CPI-C*.

#### 9.2.2.1.5 Connections List

Refer to the screen contained in the equivalent paragraph in the 9.1 Subchapter, *IMS CPI-C*.

#### 9.2.2.1.6 Partner LU

The Partner LU must be created from the Define Partner LUs...push button in the previous screen.

LU Name: NETID parameter of the VTAM (NETCGI) general definition  
+ CICS definition APPL parameter in the VTAM List  
A01CICS1).



### 9.2.2.1.7 Local LU

Local LU

LU name: CICSFBFB

Alias: CICSFBFB

NAU address:

- Independent LU
- Dependent LU NAU: [ ] (1 - 254)

Host link: HOST0001

Use this local LU as your default local LU alias

Optional comment: [ ]

Buttons: OK, Annulation, Aide

LU Name: This Local LU must be declared in the CICS Host in the connection Netname.

### 9.2.2.1.8 Mode

Mode Definition

Mode name: LU62

Class of service: #CONNECT

Mode session limit: 4 (0 - 32767)

Minimum contention winners: 2 (0 - 32767)

Receive pacing window: 4 (0 - 63)

Compression:

- Compression need: PROHIBITED
- PLU->SLU compression level: NONE
- SLU->PLU compression level: NONE

RU size:

- Default RU size
- Maximum RU size: [ ] (256 - 16384)

Optional comment: [ ]

Buttons: OK, Annulation, Aide

Mode Name: mode defined in the VTAM mode table, LOGMODE parameter of the MODEENT macro-instruction.

### 9.2.2.1.9 Side Info

Symbolic destination name

**Partner LU**  
 Fully qualified name    
 Alias

**Partner TP**  
 Service TP  
 TP name

**Security type**  Same  None  Program

**Mode name**

**Optional comment**

This screen is used in the CPI-C communications to connect the Client and the Server programs using APPC.

The value of Symbolic destination name must correspond to the external name of the communications monitor in the Repository.

TP name: transaction code activating the communications monitor on the Host.

## 9.2.3 WINDOWS 95/NT Configuration

### 9.2.3.1 Personal Communications : APPC configuration

Prerequisite (minimum): Personal Communications 4.2 Version

### 9.2.3.1.1 Define the Node

Define the Node

Basic | Advanced | DLU Requester

Control Point (CP)

Fully qualified CP name:

NETCGI . GTWTKC

CP alias:

GTWTKC

Local Node ID

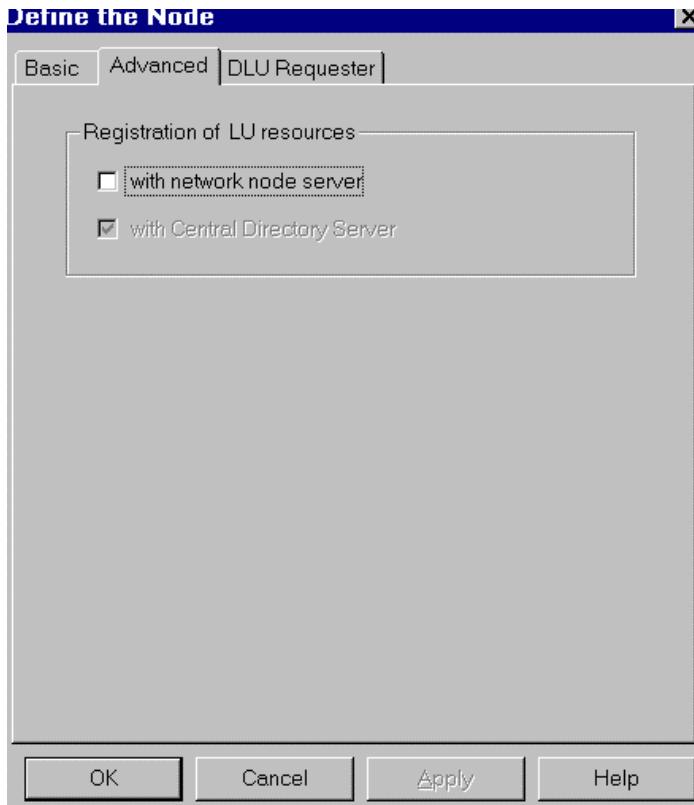
Block ID: Physical Unit ID:

05D 99CCC

OK Cancel Apply Help

Network ID: SNA network identifier to which the workstation is connected (NETID of VTAM)

Local node name: workstation identification in the network.



### 9.2.3.1.2 Define the Lan Device



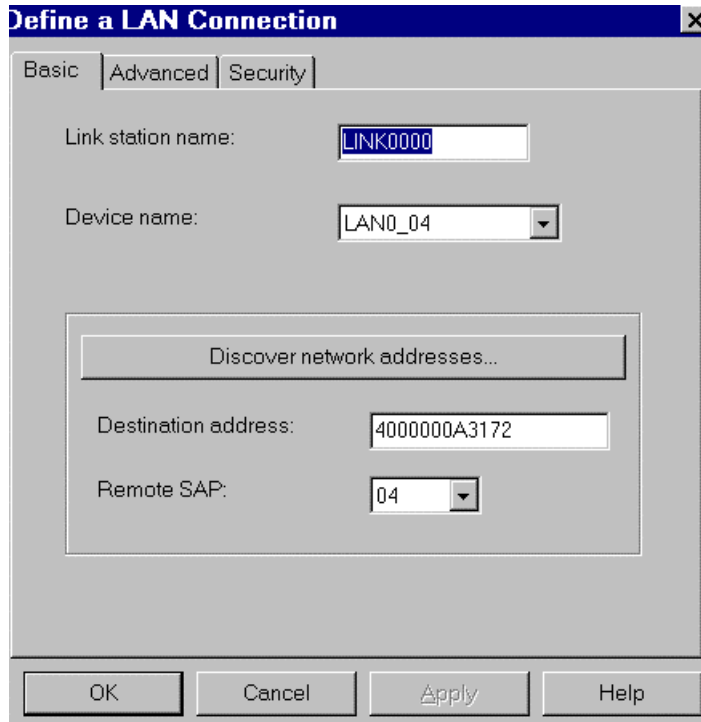
**Define a LAN Device** [X]

Basic | Activation | Performance

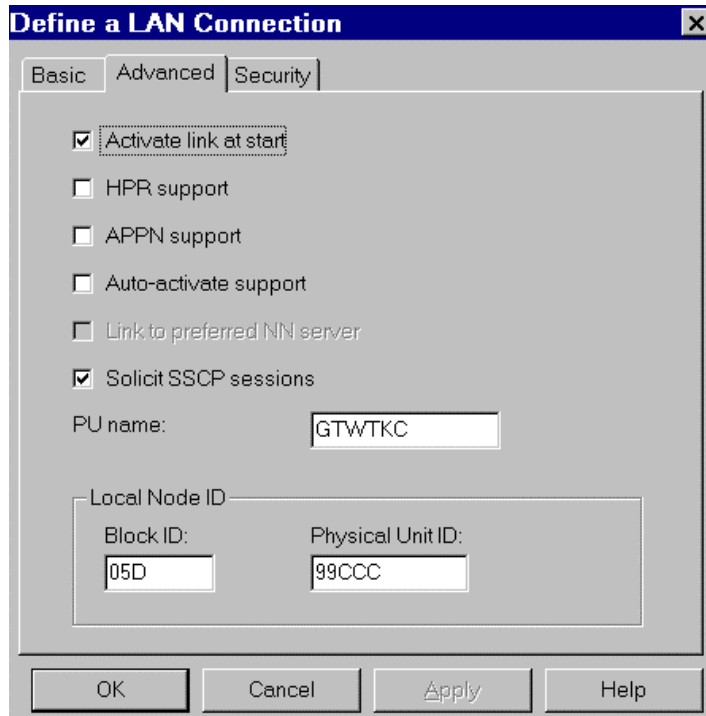
|                                    |                                   |         |
|------------------------------------|-----------------------------------|---------|
| Idle timeout:                      | <input type="text" value="30"/>   | seconds |
| Busy state timeout:                | <input type="text" value="15"/>   | seconds |
| REJ response timeout:              | <input type="text" value="10"/>   | seconds |
| Acknowledgement delay:             | <input type="text" value="100"/>  | ms      |
| POLL response timeout:             | <input type="text" value="3000"/> | ms      |
| Acknowledgement timeout:           | <input type="text" value="1000"/> | ms      |
| Anticipated outstanding transmits: | <input type="text" value="16"/>   |         |
| Receive buffer count:              | <input type="text" value="16"/>   |         |

OK Cancel Apply Help

### 9.2.3.1.3 Define the Lan Connection



Destination address: MAC Address of TIC IBM 3745



The screenshot shows a dialog box titled "Define a LAN Connection" with a close button (X) in the top right corner. It has three tabs: "Basic", "Advanced", and "Security", with "Basic" selected. The dialog contains the following fields:

- Adjacent CP name:** Two text input fields separated by a period.
- Adjacent CP type:** A dropdown menu with "APPN Node" selected.
- TG number:** A dropdown menu with "0" selected.
- Adjacent node ID:** A container box containing two sub-fields:
  - Block ID:** A text input field with "000" entered.
  - Physical Unit ID:** A text input field with "00000" entered.

At the bottom of the dialog are four buttons: "OK", "Cancel", "Apply", and "Help".

#### 9.2.3.1.4 Define a Partner LU 6.2

The screenshot shows a dialog box titled "Define a Partner LU 6.2" with a close button (X) in the top right corner. It has two tabs: "Basic" and "Advanced", with "Basic" selected. The dialog contains the following fields:

- Partner LU name:** Two text input fields separated by a period, containing "NETCGI" and "A01CICS1".
- Partner LU alias:** A text input field containing "A01CICS1".
- Fully qualified CP name:** Two text input fields separated by a period, containing "NETCGI" and "A01M".

At the bottom of the dialog are four buttons: "OK", "Cancel", "Apply", and "Help".

LU Name : NETID parameter of the VTAM (NETCGI) definition + parameter APPL for the CICS definition in the VTAMLST (A01CICS1).

**Define a Partner LU 6.2**

Basic | Advanced

Maximum LL record size:  
32767

Conversation security support

Parallel session support

OK Cancel Apply Help

### 9.2.3.1.5 Define a Local LU 6.2

**Define a Local LU 6.2**

Basic

Local LU name: CICSFBFB  Dependent LU

Local LU alias: CICSFBFB

PU name: [dropdown]

NAU address: [dropdown]

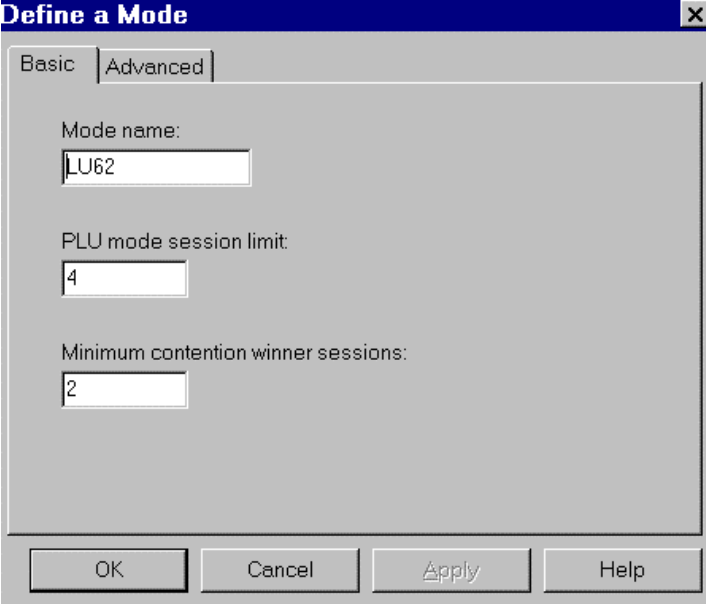
LU session limit: 0

OK Cancel Apply Help

LU Name: no equivalent on the Host if the Dynamic LU option (DYNLU=YES)  
 In the Windows 95 environment, define the local LU6.2 used by  
 default => SET APPCLLU=CICSFBFB

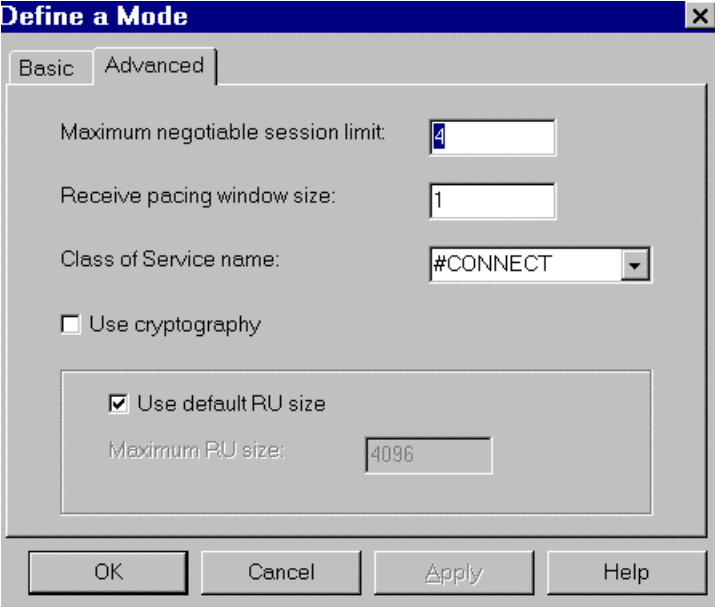


### 9.2.3.1.6 Define a Mode



The screenshot shows the 'Define a Mode' dialog box with the 'Basic' tab selected. The dialog has a title bar with a close button (X). Below the title bar are two tabs: 'Basic' and 'Advanced'. The 'Basic' tab contains three input fields: 'Mode name:' with the value 'LU62', 'PLU mode session limit:' with the value '4', and 'Minimum contention winner sessions:' with the value '2'. At the bottom of the dialog are four buttons: 'OK', 'Cancel', 'Apply', and 'Help'.

Mode Name: mode defined in the VTAM mode table, LOGMODE parameter of the MODEENT macro-instruction.



The screenshot shows the 'Define a Mode' dialog box with the 'Advanced' tab selected. The dialog has a title bar with a close button (X). Below the title bar are two tabs: 'Basic' and 'Advanced'. The 'Advanced' tab contains several settings: 'Maximum negotiable session limit:' with a value of '4', 'Receive pacing window size:' with a value of '1', and 'Class of Service name:' with a dropdown menu showing '#CONNECT'. There is an unchecked checkbox for 'Use cryptography'. Below this is a section with a checked checkbox for 'Use default RU size' and a text box for 'Maximum RU size:' with a value of '4096'. At the bottom of the dialog are four buttons: 'OK', 'Cancel', 'Apply', and 'Help'.

### 9.2.3.1.7 Define CPI-C Side Information

The screenshot shows the 'Define CPI-C Side Information' dialog box with the 'Basic' tab selected. The fields are as follows:

- Symbolic destination name:
- Mode name:
- Partner LU name:  .
- TP name:
- Service TP

Buttons at the bottom: OK, Cancel, Apply, Help.

This screen is used in the CPI-C communications to connect the Client and the Server programs using APPC.

The value of Symbolic destination name must correspond to the external name of the communications monitor in the Repository.

TP name: transaction code activating the communications monitor on the Host.

The screenshot shows the 'Define CPI-C Side Information' dialog box with the 'Security' tab selected. The fields are as follows:

- Conversation security:
- Security user ID:
- Security password:

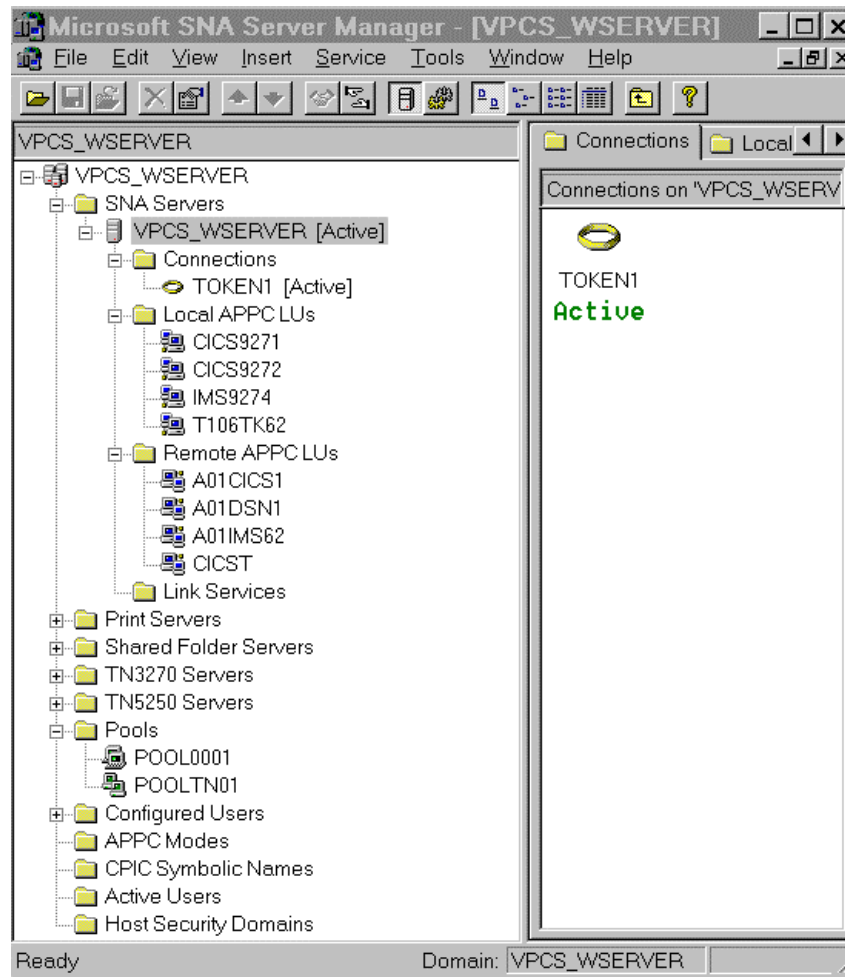
Buttons at the bottom: OK, Cancel, Apply, Help.

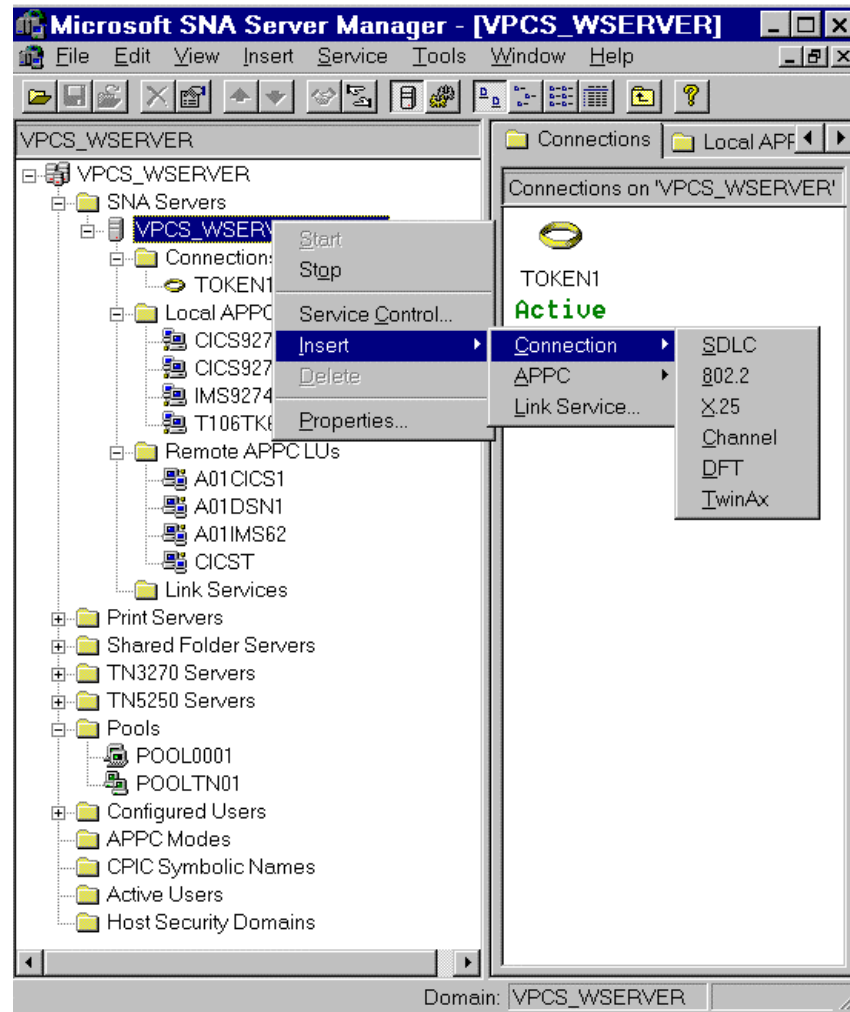
## 9.2.4 Windows NT Configuration

### 9.2.4.1 SNA Server 3.0A Configuration

(The Pack 2 Service must be applied for APPC connections).

The SNA Server parameters must correspond to the VTAM, NCP and CICS definitions of the MVS central site.





#### 9.2.4.1.1 Server Properties

**Network Name** = identifier of the SNA network (NETID in ATCSTRxx of VTAM)

**Control Point Name** = corresponding to the CPNAME in the PU definition of VTAM

**VPCS\_WSERVER Properties**

General | Server Configuration

Server Name: VPCS\_WSERVER

Comment:

SNA Network Control Point Name

Network Name: NETCGI

Control Point Name: GTWTKK

- Type of transport used between the client and the SNA server (example: TCP/IP protocol)

**VPCS\_WSERVER Properties**

General | Server Configuration

Server: VPCS\_WSERVER

Subdomain: VPCS\_WSERVER

Server Role: Primary

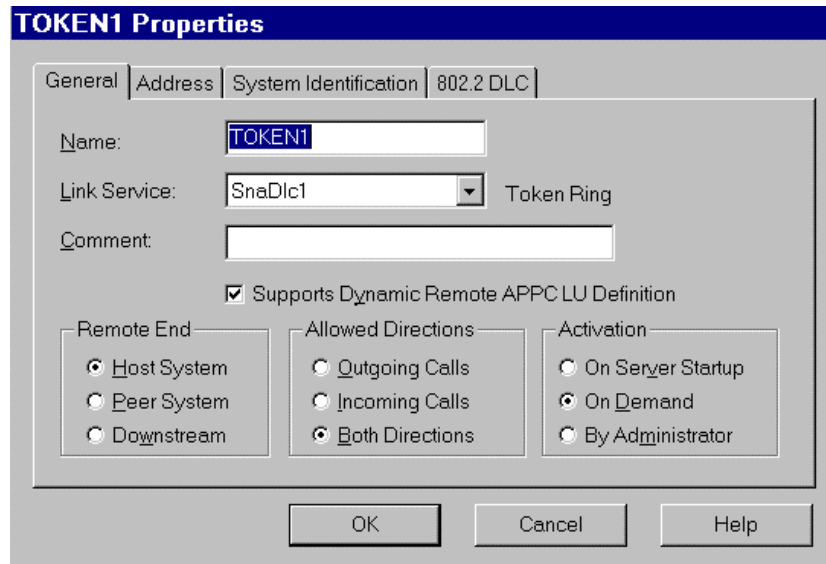
Network Transports: TCP/IP

Warning: Server Registry Not Accessible.

#### 9.2.4.1.2 Link Properties

**Link Service :** The link type must be chosen at installation or installed later on using the Setup program. This is the component of SNA Server which communicates with the network card driver. The SNA DLC 802.2 Link service is assigned for the communication with the central site in a Token Ring or Ethernet.LAN

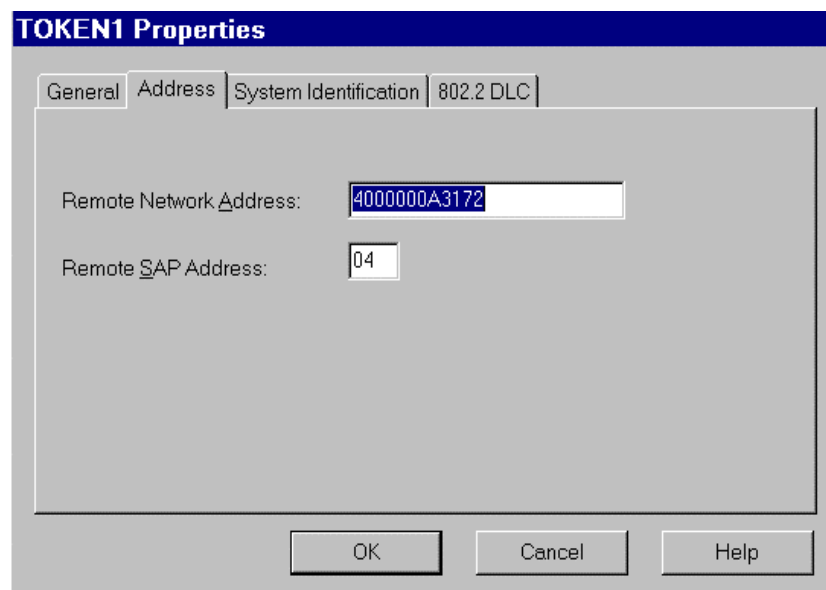
Connection type = 802.2



- Definition of the controller TIC 3745 in NCP (LAN attachment).

```
*****
*      TIC BNN                               05742090
*****
A01L1TK LINE ADDRESS=(1088,FULL),
      PORTADD=01,
      LOCADD=4000000A3172,
      ISTATUS=ACTIVE,
      UACB=(X$P1AX,X$P1AR)
*
A01TK1PU PU ISTATUS=ACTIVE,
      ADDR=01
*****
```

Remote Network Address = LOCADD (MAC Address) in the LINE macro of NCP.



- SSCPNAME and NETID parameters of VTAM ATCSTRxx (required for the configuration of Remote Node Name in SNA Server) :

```
*****
NOPROMPT,CONFIG=00,SSCPID=01,
MAXSUBA=31,SUPP=NOSUP,
SSCPNAME=A01M,
SSCPORD=DEFINED,
NETID=NETCGI,
HOSTSA=1,
CRPLBUF=(550,,20,,40,40),
IOBUF=(420,182,25,,40,40),
LFBUF=(300,,0,,20,10),
LPBUF=(50,,0,,5,5),
SFBUF=(50,,0,,5,5),
SPBUF=(90,,0,,5,5),
NOTRACE,TYPE=VTAM
*****
```

- Definition of the PU in VTAM corresponding to the SNA Server gateway (used in the configuration of the Local Node Name for SNA Server) :

```
*****
*/ * SWITCHED MAJOR NODE *
*****
*
SW6TKR VBUILD TYPE=SWNET,MAXNO=12,MAXGRP=06
*
W6TK00 PU ADDR=55,
        CPNAME=GTWTKK,
        IDBLK=05D,
        IDNUM=0FF44,
        DYNLU=YES,
        MAXPATH=1,
        DISCNT=YES,
        IRETRY=YES,
        VPACING=7,
        PACING=7,
        SSCPFM=USSSCS,
        USSTAB=USSTAB2,
        MAXDATA=4096,
        PUTYPE=2,
        MAXOUT=7,
        DATMODE=FULL
*
* ==> INDEPENDENT LU
*
CICS9271 LU LOCADDR=0,
           ISTATUS=ACTIVE,
           MODETAB=MTLU62,
           DLOGMOD=LU62
*
*****
```

- Local Node Name :
- Local Node ID = IDBLK & IDNUM
- Control Point Name = CPNAME
- Network Name = NETID (ATCSTRxx)
- Remote Node Name :
- Control Point Name = SSCPNAME
- Network Name = NETID (ATCSTRxx)

**TOKEN1 Properties**

General | Address | System Identification | 802.2 DLC

Local Node Name

Network Name: NETCGI

Control Point Name: GTWTKK

Local Node ID: 05D 0FF44

Remote Node Name

Network Name: NETCGI

Control Point Name: A01M

Remote Node ID:

XID Type

Format 1

Format 3

Peer DLC Role

Primary

Secondary

Negotiable

OK Cancel Help

Max BTU Length (frame size) corresponds to MAXDATA of the PU in VTAM.

- for Token ring adapter of 4 Mbps, must be less than or equal to 4195
- for Ethernet adapter, must be less than or equal 1493.

**TOKEN1 Properties**

General | Address | System Identification | 802.2 DLC

Max BTU Length: 4096

Receive ACK Threshold (frames): 2

Unacknowledged Send Limit (frames): 8

802.2 Timeouts

Response (t1): Default

Receive Ack (t2): Default

Inactivity (ti): Default

Retry Limit: 10

XID Retries: 3

Connection Retry Limits

Maximum Retries: No Limit

Delay After Failure: Default

OK Cancel Help

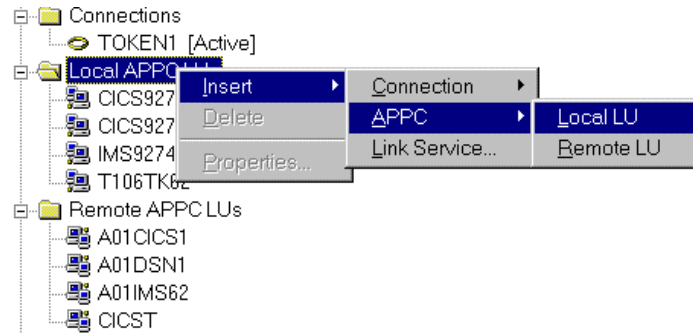
#### 9.2.4.1.3 Local APPC LU

APPC uses a local LU (independent or dependant) and one or more remote LUs. The APPC sessions communicate between two LUs (Local and Remote).



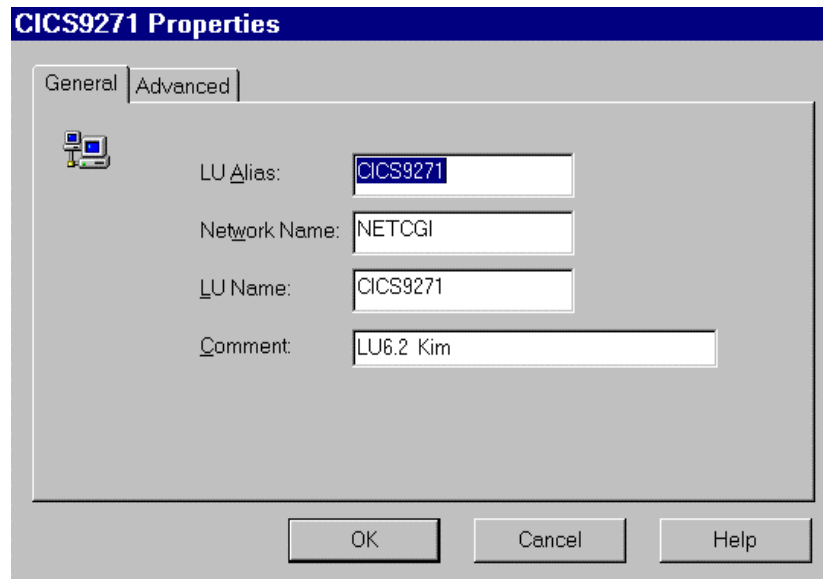
The minimum prerequisites for MVS to communicate with a TP (Transaction Program) :

- VTAM version 3.2
- NCP version 5.2 (3745)
- NCP version 4.3 (3725)



Local LU is defined with the PU in VTAM as an independent LU with LOCADDR = 0:

- LU Name = CICS9271
- Network Name = NETID dans ATCSTRxx
- LU Alias = Identifier for the local LU for TPs (Transaction Programs)



#### 9.2.4.1.4 Remote APPC LU

- Definition of the CICS APPL in VTAM :

```

*****
*/ * THIS MEMBER CONTAINS VTAM APPLICATION DEFINITION
*/ * . NAME ACBNAME
*/ * -----
*/ * . A01CICS1 CICST
*****
A01CICS1 APPL EAS=160,
ACBNAME=CICST,          APPLID FOR ACB
AUTH=(ACQ,VPACE,PASS),
PARSESS=YES,
SONSCIP=YES,
MODETAB=MTLU62
*****

```

- In the CICS SIT table, InterSystem Communication must be activated: ISC=YES
- Definition of the CONNECTION in CICS :

Netname (CICS9271) correspond au Local APPC LU

```

-----
OVERTYPE TO MODIFY          CICS RELEASE = 0330
CEDA ALter
  Connection   : SG71
  Group       : GRPISC9
  Description  ==> CONNEXION LU6.2 SNA SERVER
CONNECTION IDENTIFIERS
  Netname     ==> CICS9271
INDsys       ==>
REMOTE ATTRIBUTES
  REMOTESystem ==>
  REMOTENAME  ==>
CONNECTION PROPERTIES

```

*ACcessmethod* ==> *Vtam*  
*Protocol* ==> *Appc*  
*Singlesess* ==> *No*  
*DATAstream* ==> *User*  
*RECORDformat* ==> *U*  
*OPERATIONAL PROPERTIES*  
*AUTOconnect* ==> *Yes*  
*INService* ==> *Yes*  
*SECURITY*  
*SECURITYname* ==> *SYTD*  
*ATTachsec* ==> *Verify*  
*BINDPassword* ==>  
*BINDSecurity* ==> *No*

APPLID=CICST

- 
- Definition of the SESSION in CICS :
    - Connection (SG71) corresponds to the connexion code specified above
    - MOdename (LU62) corresponds to the Mode name defined in SNA Server
    - MAximum (004 , 002) corresponds to the parameters Parallel Session Limit and to the Partner Min Contention Winner defined in the SNA Server Mode (LU62)
    - SENDSize and RECEIVESize correspond to the parameters Max Receive RU Size et Max Send RU Size SNA Server Mode (LU62).

---

*OVERTYPE TO MODIFY* *CICS RELEASE = 0330*  
*CEDA ALter*  
*Sessions* : *SESSIO71*  
*Group* : *GRPISC9*  
*DEscription* ==> *SESSION LU6.2 SNA Server*  
*SESSION IDENTIFIERS*  
*Connection* ==> ***SG71***  
*SESSName* ==>  
*NETnameq* ==>  
*MOdename* ==> ***LU62***  
*SESSION PROPERTIES*  
*Protocol* ==> *Appc*  
*MAximum* ==> ***004 , 002***  
*RECEIVEPfx* ==>  
*RECEIVECount* ==>  
*SENDPfx* ==>  
*SENDCount* ==>  
*SENDSize* ==> ***08192***  
*RECEIVESize* ==> ***08192***  
*SESSPriority* ==> *000*  
*Transaction* :  
*OPERATOR DEFAULTS*  
*OPERId* :  
*OPERPriority* : *000*  
*OPERRsl* : *0*  
*OPERSecurity* : *1*  
*PRESET SECURITY*  
*USERId* ==>  
*OPERATIONAL PROPERTIES*  
*Autoconnect* ==> *Yes*  
*INservice* :

*Buildchain* ==> *Yes*  
*USERArealen* ==> *000*  
*IOarealen* ==> *00000, 00000*  
*RELreq* ==> *Yes*  
*DIScreq* ==> *No*  
*NEPclass* ==> *000*  
**RECOVERY**  
*RECOVOption* ==> *Sysdefault*  
*RECOVNotify* ==> *None*

APPLID=CICST

---

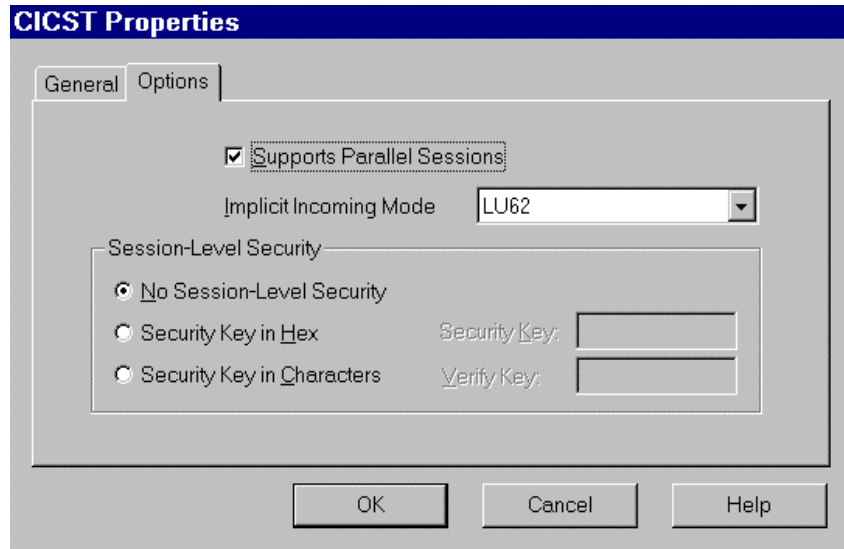
- Definition of the Remote APPC LU in SNA Server:
  - Network Name = NETID in ATCSTRxx
  - LU Alias = Identifier of the LU in local TPs (Transaction Programs)
  - LU Name = APPL ID CICS in VTAM (A01CICS1)

The screenshot shows a dialog box titled "CICST Properties" with two tabs: "General" and "Options". The "General" tab is active. It contains a small icon of a computer monitor and keyboard. Below the icon are several labeled input fields:

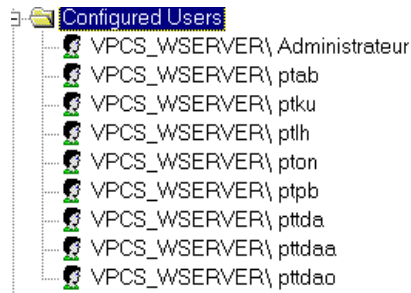
- Connection: A dropdown menu showing "TOKEN1".
- LU Alias: A text box containing "CICST".
- Network Name: A text box containing "NETCGI".
- LU Name: A text box containing "A01CICS1".
- Uninterpreted Name: An empty text box.
- Comment: An empty text box.

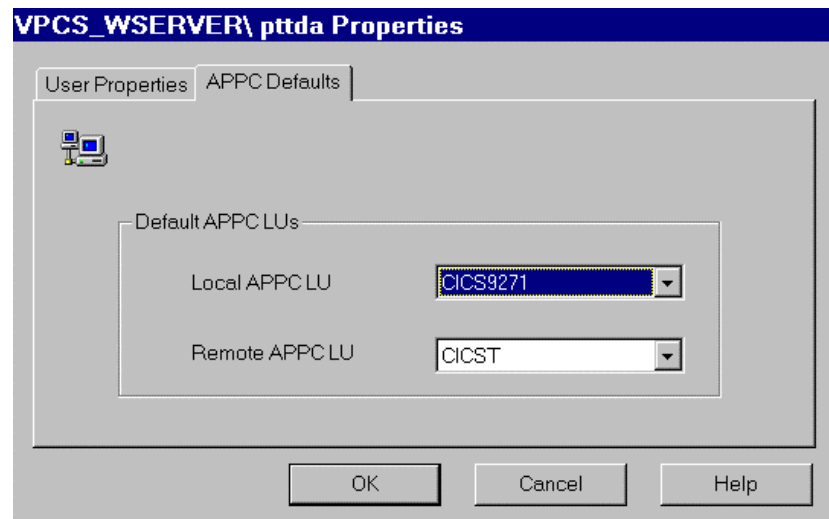
At the bottom of the dialog box are three buttons: "OK", "Cancel", and "Help".

- Implicit Incoming Mode = Mode Name used for Supports Parallel Sessions and defined in the APPC Modes in SNA Server

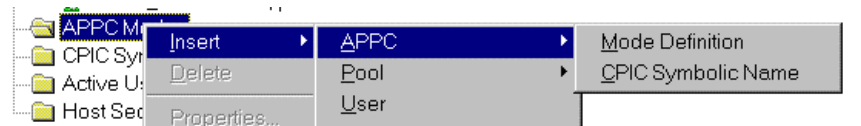


### 9.2.4.1.5 Configuration of Users





#### 9.2.4.1.6 Definition of the MODE



- Definition of the LU6.2 MODES in the tables of Modes in VTAM :

The LU62 Mode is an example of the configured and used mode with APPC (LU6.2). The SNASVCMG mode est included in SNA Server and is used by the "Supports Parallels Sessions".

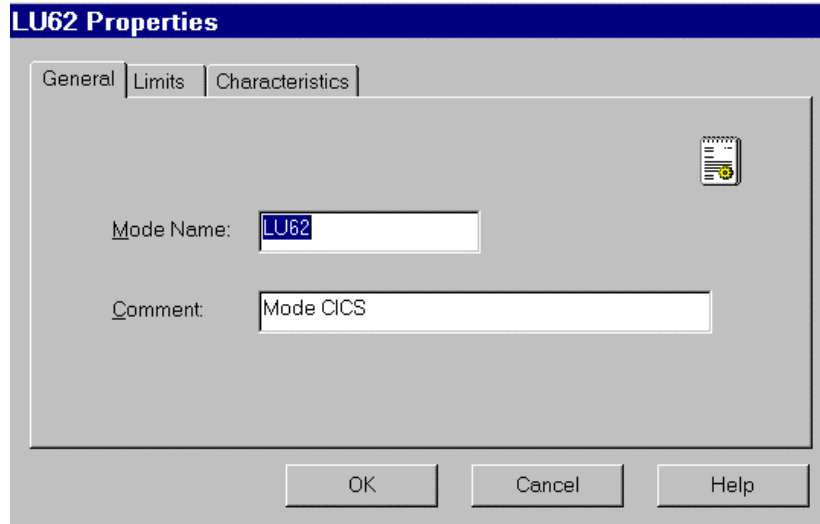
```

-----
*
***  TITLE '--- "MODTABLE" RELATED TO THE LU 6.2 ---'  *****
*
SNASVCMG MODEENT LOGMODE=SNASVCMG,
          FMPROF=X'13',
          TSPROF=X'07',
          PRIPROT=X'B0',
          SECPROT=X'B0',
          COMPROT=X'D0B1',
          RUSIZES=X'8585',
          ENCR=B'0000',
          PSERVIC=X'06020000000000000000300'
*
LU62  MODEENT LOGMODE=LU62,
          TYPE=X'00',
          FMPROF=X'13',
          TSPROF=X'07',
          PRIPROT=X'B0',
          SECPROT=X'B0',
          COMPROT=X'50B1',
          RUSIZES=X'8989',
          SRCVPAC=X'00',
          PSNDPAC=X'00',
          SSNDPAC=X'00',
          PSERVIC=X'060200000000000000002C00'

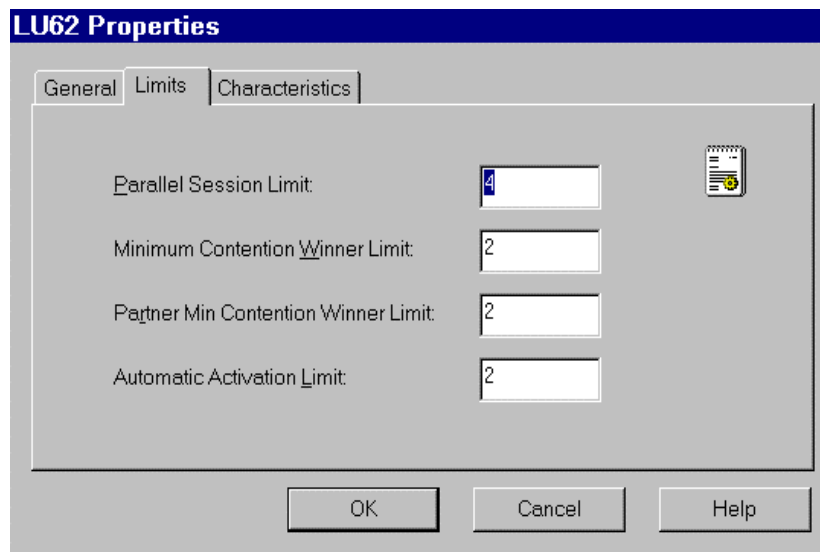
```

- Definition of the Mode in SNA Server :

Mode Name = Mode defined in the table of Modes in VTAM



The image shows the 'LU62 Properties' dialog box with the 'General' tab selected. The 'Mode Name' field contains 'LU62' and the 'Comment' field contains 'Mode CICS'. There are 'OK', 'Cancel', and 'Help' buttons at the bottom.



The image shows the 'LU62 Properties' dialog box with the 'Limits' tab selected. The 'Parallel Session Limit' is set to 4, and the 'Minimum Contention Winner Limit', 'Partner Min Contention Winner Limit', and 'Automatic Activation Limit' are all set to 2. There are 'OK', 'Cancel', and 'Help' buttons at the bottom.

The image shows the 'LU62 Properties' dialog box with the 'Limits' tab selected. The dialog has three tabs: 'General', 'Limits', and 'Characteristics'. The 'Limits' tab contains the following fields and controls:

- Pacing Send Count: 4
- Pacing Receive Count: 4
- Max Send RU Size: 8192
- Max Receive RU Size: 8192
- High Priority Mode

At the bottom of the dialog are three buttons: 'OK', 'Cancel', and 'Help'.

#### 9.2.4.1.7 Definition of the CPI-C Symbolic Destinations Names (Side Information)

CPI-C is the interface of the communication API. It provides a standard group of function calls for all the APPC platforms that handle CPI-C.

Name corresponds to the external name of the Pacbase communications monitor (makes the connection between the Clients and the server programs using APPC).

The image shows the 'CLCFOL Properties' dialog box with the 'Partner Information' tab selected. The dialog has two tabs: 'General' and 'Partner Information'. The 'Partner Information' tab contains the following fields and controls:

- Name: CLCFOL
- Comment: Tests VAPB Lionel
- Conversation Security:
  - None
  - Same
  - Program
- Mode Name: LU62

At the bottom of the dialog are three buttons: 'OK', 'Cancel', and 'Help'.

- Definition of the CICS transaction:

```

-----
OVERTYPE TO MODIFY                CICS RELEASE = 0330
CEDA ALter
Transaction : VP20
Group      : VISUAL
Description ==> VISUAL/CLIENT WIN/NT CPIC
PROGram    ==> CLCFOL
TWAsize    ==> 00000
PROFile    ==> DFHCICST
  
```



```

PArtitionset ==>
STatus ==> Enabled
PRIMedsize : 00000
TASKDATALoc ==> Below
TASKDATAKey ==> User
REMOTE ATTRIBUTES
DYnamic ==> No
REMOTESystem ==>
REMOTENAME ==>
TRProf ==>
Localq ==>
SCHEDULING
PRIOrity ==> 001
TClass ==> No
ALIASES
Alias ==>
TASKReq ==>
XTRanid ==>
TPName ==>
==>
XTPname ==>
==>
==>
RECOVERY
DTimout ==> No
Indoubt ==> Backout
REStart ==> No
SPurge ==> No
TPUrge ==> No
DUmp ==> Yes
TRACe ==> Yes
SECURITY
RESSec ==> No
Cmdsec ==> No
Extsec : No
TRANsec : 01
RSI : 00

```

*APPLID=CICST*

-----

If a program uses DB2 during the transaction, the DB2 plan must be linked to the transaction of the server monitor. The transaction must be declared in the RCT table of CICS/ESA :

-----

**DSNCRCT TYPE=ENTRY, TXID=(VP20), THRDM=2,  
THRDA=2, PLAN=VP20, AUTH=(USERID, \*, \*)**

If the application accesses DB2, the user transaction must be authorized and linked to the DB2 plan in the RCT table of the CICS area:

-----

**DSNCRCT TYPE=ENTRY, TXID=(VP20), THRDM=6,  
THRDA=6, PLAN=VP20, AUTH=(USERID, \*, \*)**

-----

- Definition of Partner TP Name :

Application TP = Transaction code activating the communications monitor on the Host and defined above

- Definition of Partner LU Name :

Alias = Alias defined in Remote APPC LU

## 9.3 CICS ECI

### 9.3.1 MVS and CICS Configuration

#### 9.3.1.1 VTAM Definitions

Prerequisite (minimum): VTAM Version 3.3

##### 9.3.1.1.1 Definition of the ATCSTR of the VTAM

```
*****
NOPROMPT,CONFIG=00,SSCPID=01,
MAXSUBA=31,SUPP=NOSUP,
SSCPNAME=A01M,
SSCPORD=DEFINED,
NETID=NETCGI,
HOSTSA=1,
CRPLBUF=(550,,20,,40,40),
IOBUF=(420,182,25,,40,40),
LFBUF=(300,,0,,20,10),
LPBUF=(50,,0,,5,5),
SFBUF=(50,,0,,5,5),
SPBUF=(90,,0,,5,5),
NOTRACE,TYPE=VTAM
*****
```

##### 9.3.1.1.2 MAC Address of the TIC (LAN attachment) 3745 controller in NCP:

```
*****
*      TIC BNN                05742090
*****
A01L1TK LINE ADDRESS=(1088,FULL),
      PORTADD=01,
      LOCADD=400003172000,
      ISTATUS=ACTIVE,
      UACB=(X$P1AX,X$P1AR)
*
A01TK1PU PU ISTATUS=ACTIVE,
      ADDR=01
*****
```

### 9.3.1.1.3 CICS Definition

```
A01CICS1 APPL  EAS=160,                ESTIMATED CONCURRENT SESSIONS  *
                ACBNAME=CICST,         APPLID FOR ACB                 *
                AUTH=(ACQ,VPACE,PASS),  CICS CAN ACQUIRE & PASS TMLS  *
                PARSESS=YES,            ⇨parallel Sessions Supports *
                SONSCIP=YES,
                MODETAB=MTLU62         ⇨    name of the modes table
```

### 9.3.1.1.4 Mode Definition

- Definition of the characteristics for the LU6.2 session.
- The SNASVCMG mode is used with the "Parallels Sessions" support.

```
TITLE '--- "MODTABLE" RELATED TO THE LU 6.2 ---'
*
MTLU62  MODETAB
        SPACE 4
SNASVCMG MODEENT LOGMODE=SNASVCMG, FMPROF=X'13',
           TSPROF=X'07', PRIPROT=X'B0', SECPROT=X'B0',
           COMPROT=X'D0B1', RUSIZES=X'8585', ENCR=B'0000',
           PSERVIC=X'060200000000000000000000300'
LU62    MODEENT LOGMODE=LU62, TYPE=X'00', FMPROF=X'13',
           TSPROF=X'07', PRIPROT=X'B0', SECPROT=X'B0',
           COMPROT=X'50B1', RUSIZES=X'8787', SRCVPAC=X'00',
           PSNDPAC=X'00', SSNDPAC=X'00',
PSERVIC=X'0602000000000000000000002C00'
```

### 9.3.1.1.5 SNA Definition

```
*/ * LIB: SYS1.VTAMLST(SW1TKR)
*/ *
*/ *
*/ * SWITCHED MAJOR NODE TOKEN-RING ST-MARC :           - 06/07/95.
*/ *   --->   LIEN XCA MAJOR NODE ==> XCA1TKR  (IBM3172-3)
*/ *   --->   LIEN GROUPE XCA      ==> GRP02
*
*   - MODEL FOR IDBLK X'05D' - OS/2 COMMUNICATIONS MANAGER
*
*
SW1TKR  VBUILD TYPE=SWNET,MAXNO=99,MAXGRP=10
*
*   ----->  DEFINING A GATEWAY TOKEN-RING --> GTWTK1  <-----
*
W1TK00  PU      ADDR=50,
           CPNAME=GTWTK1,
           IDBLK=05D,
           IDNUM=00002,
           DYNLU=YES,
           MAXPATH=1,
           DISCNT=NO,
           IRETRY=YES,
           VPACING=7,
           PACING=7,
           SSCPFM=USSSCS,
           MAXDATA=4096,
```

PUTYPE=2

The DYNLU=YES option allows to avoid Lu 6.2 definitions at the VTAM level (for workstations on the TR network).

### 9.3.1.1.6 Definition of an Independent LU

Despite what has been stated above, it may be worthwhile defining an independant LU for the first communication tests:

```

** LIB: SYS1.VTAMLST(SW1TKR)
**
**
** SWITCHED MAJOR NODE TOKEN-RING ST-MARC :           - 06/07/95.
**
**
* INDEPENDENT LUS
*
CGI5075          LU      LOCADDR=0,
                  ISTATUS=ACTIVE,
                  DLOGMOD=LU62,
                  MODETAB=MTLU62

```

### 9.3.1.2 APPC/MVS Definitions

Prerequisite (minimum): MVS/ESA Version 4.1

There is no specific definition as we use the APPC layer delivered with the CICS version.

### 9.3.1.3 CICS Definitions

#### 9.3.1.3.1 InterSystem Communication Parameter in the SIT Table

ISC=YES

#### 9.3.1.3.2 Connection

```

Connection      : SGFB
Group           : GRPISC5
DEscription     :
CONNECTION IDENTIFIERS
Netname        : CGI5075   ⇨   same declaration as Local LU in the CM/2 (free code)
INdsys         :
REMOTE ATTRIBUTES
REMOTESystem   :
REMOTENAME     :
CONNECTION PROPERTIES
ACcessmethod   : Vtam
Protocol       : Appc
SIngleless     : No       ⇨   If independent LU
DATAstream     : User
RECORDformat   : U
OPERATIONAL PROPERTIES
AUtoconnect    : Yes
INService      : Yes
SECURITY
SEcurityname   : PTPD
ATTachsec      : Verify   ⇨   Ckeck the userid and password at the conversation
BINDPassword   :
BINDSecurity   : No

```

**9.3.1.3.3 Session**

```

Sessions      : SESSIOFB
Group         : GRPISC5
DEscription   :
SESSION IDENTIFIERS
Connection    : SGFB      ⇒   code of the connection defined above
SESSName     :
NETnameq     :
MOdename     : LU62      ⇒   mode defined in the VTAM MODTABLE
SESSION PROPERTIES
Protocol      : Appc
MAXimum      : 004 , 002
RECEIVEPfx   :
RECEIVECount :
SENDPfx      :
SENDCount    :
SENDSize     : 08192
RECEIVESize  : 08192
SESSPriority  : 000
Transaction  :
OPERATOR DEFAULTS
OPERId       :
OPERPriority  : 000
OPERRsl      : 0
OPERSecurity  : 1
PRESET SECURITY
USERId       :
OPERATIONAL PROPERTIES
Autoconnect  : Yes
INservice    :
Buildchain   : Yes
USERArealen  : 000
IOarealen    : 00000 , 00000
RELreq       : Yes
DIScreq      : No
NEPclass     : 000
RECOVERY
RECOVOption  : Sysdefault
RECOVNotify  : None

```

### 9.3.1.3.4 Transaction Definition

Definition of a user mirror transaction:

```

-----
CICS RELEASE = 0410

TRANSaction      : VEC1
Group            : VPCS250
DEscription      :
PROGram          : DFHMIRS
TWAsize          : 00000          0-32767
PROfile         : DFHCICSA
PARTitionset     :
STATus           : Enabled          Enabled ! Disabled
PRIMedsize      : 00000          0-65520
TASKDATAloc     : Below           Below ! Any
TASKDATAkey     : User            User ! Cics
STOrageclear    : No              No ! Yes
RUNaway         : System          System ! 0-2700000
SHutdown        : Disabled        Disabled ! Enabled
ISolate         : Yes              Yes ! No
REMOTE ATTRIBUTES
+ DYNAMIC        : No              No ! Yes
REMOTESystem    :
  REMOTENAME     :
  TRProf        :
  Localq        :                  No ! Yes
SCHEDULING
PRIOrity        : 001             0-255
TClass         : No              No ! 1-10
TRANClass      : DFHTCL00
ALIASES
Alias           :
TASKReq        :
XTRanid        :
TPName         :
               :
XTPname        :
               :
RECOVERY
DTimeout       : No              No ! 1-6800
INDoubt        : Backout         Backout ! Commit ! Wait
REStart        : No              No ! Yes
SPurge         : No              No ! Yes
TPUrge         : No              No ! Yes
DUmp           : Yes             Yes ! No
TRACe          : Yes             Yes ! No
CONfdata       : No              No ! Yes
SECURITY
RESec          : Yes             No ! Yes
CMdsec         : Yes             No ! Yes
Extsec         : No
TRANSec        : 01              1-64
RS1            : 00              0-24 ! Public

```

If a program uses DBS during the transaction, this transaction must be linked to the DB2 plan. The Server monitor transaction code must then be declared in the RCT with the CSPM transaction which makes the link with the CICS OS/2

Server and/or with the CPMI transaction which makes the link with the CICS OS/2 Client; these transactions should be linked to the DB2 plan (ATDF):

```
DSNRCT TYPE = ENTRY, TXID=(CPMI, CSPM, VICL),
        THRDM=2, THRDA=2, PLAN=ATDF, AUTH=(USERID, *, *)
```

If the CPMI mirror transaction causes an Abend ACN1, it means that the DFHCNV conversion table is not defined in the CICS area. It is required for using this transaction. Define this table, then assemble and LinkEdit using the following parameters:

```
*****
*****
```

```
//EXBCCNV JOB (009), 'BC', CLASS=X, MSGCLASS=X, NOTIFY=SYTD
//CICSCNV EXEC DFHAUPLE,
// PARM.LNKEDT='RENT,REUS,LIST,XREF,LET,NCAL,AMODE=31,RMODE=ANY'
//ASM.SYSLIB DD DSN=CICS330.SDFHMAC, DISP=SHR
//ASSEM.SYSUTI DD DSN=PT$EXP.CICST330.SOURCE(DFHCNV), DISP=SHR
//LNKEDT.SYSLMOD DD DSN=PT$PDV.PB80204.MTR8, DISP=SHR
```

```
DFHCNV TYPE=INITIAL
DFHCNV TYPE=ENTRY, RTYPE=PC, RNAME=TESTVP, CLINTCP=(850,437), *
        SRVERCP=297
DFHCNV TYPE=SELECT, OPTION=DEFAULT
DFHCNV TYPE=FIELD, OFFSET=0, DATATYP=CHARACTER, DATALEN=8051, *ES
        LAST=YES
DFHCNV TYPE=FINAL
END
```


```
*****
*****
```

## 9.3.2 OS/2 Configuration


### 9.3.2.1 Communication Manager/2: APPC configuration

Prerequisite (minimum): Communication Manager/2 Version 1.11


#### 9.3.2.1.1 Communication Choice

 Refer to the screen contained in the equivalent paragraph in Subchapter 9.1, *IMS CPI-C*.


#### 9.3.2.1.2 Token Ring Network

 Refer to the screen contained in the equivalent paragraph in Subchapter 9.1, *IMS CPI-C*.


#### 9.3.2.1.3 Token Ring DLC Adapter

 Refer to the screen contained in the equivalent paragraph in Subchapter 9.1, *IMS CPI-C*.

#### 9.3.2.1.4 Local Node

 Refer to the screen contained in the equivalent paragraph in Subchapter 9.1, *IMS CPI-C*.

#### 9.3.2.1.5 Connections List

 Refer to the screen contained in the equivalent paragraph in Subchapter 9.1, *IMS CPI-C*.

### 9.3.2.1.6 Partner LU

The Partner LU must be created from the Define Partner LUs...push button in the previous screen.

LU Name: NETID parameter of the VTAM (NETCGI) general definition + CICS definition APPL parameter in the VTAM List A01CICS1).

### 9.3.2.1.7 Local LU

LU Name: This Local LU must be declared in the CICS Host in the connection Netname.



### 9.3.2.1.8 Mode

Mode Name: mode defined in the VTAM mode table, LOGMODE parameter of the MODEENT macro-instruction.

### 9.3.2.2 CICS OS/2

#### 9.3.2.2.1 CICS Universal Client for OS/2 Version 3.xx

The CICS OS/2 Client configuration is made directly in the CICSCLI.INI file in the \CICSCLI\BIN directory (standard installation).

```

;*****
;* IBM CICS Client - Initialization File
;*****

;-----
; Client section - This section defines the local CICS client. There
; should only be one Client section.

Client = CGI5075           ; Auto-install client on the server
  MaxServers = 1           ; Only allow one server connection
  MaxRequests = 20         ; Limit the maximum server interaction
  MaxBufferSize = 32      ; Allow for a 32K maximum COMMAREA
  LogFile = CICSCLI.LOG    ; Set the error log file name
  TraceFile = CICSCLI.TRC  ; Set the trace log file name
  DosMemory = 48           ; The DOS client's memory pool size

;-----
; Server section - This section defines a server to which the client may
; connect. There may be several Server sections.

Server = CICSNA           ; Arbitrary name for the server

```

```

Description = CICS Server ; Arbitrary description for the server
Protocol = SNA ; Matches with a Driver section below
NetName = NETCGI.A01CICS1 ; The server's NetBIOS name
Adapter = 0 ; Use NetBIOS on LAN adapter 0
UpperCaseSecurity = N ; Fold Userid and Password to uppercase
LocalLUName=CGI5075
Modename=LU62

```

```

Driver = CM2APPC ; Matches the Server's Protocol value
DriverName = CCLIBMSN ; Use the IBM CM/2 APPC

```

## 9.4 TCP-IP Socket via UNIX

### 9.4.1 Configuration

The SOCKET Middleware API allows a VisualAge application and COBOL Microfocus Servers in a UNIX environment to communicate via a listener running on a distant target UNIX machine.

Therefore, the only parameters to set are the Compile and Link options and the SQL and Microfocus libraries needed. You can find them in the following makefile example.

#### 9.4.1.1 Makefile Example

```

CC=cc
CFLGS= -c -D_PAC_AIX
CFLGSX= -c -D_PAC_AIX
COB=cob
COBFLGS=-x -B static
LIB=/cgi/oracle/procob/lib/cobsqlintf.o \
    /cgi/oracle/lib/libsql.a \
    /cgi/oracle/lib/osntab.o \
    -lora \
    -lsqlnet \
    -lnlsrtl \
    -lcv6 \
    -lcore \
    -lnlsrtl \
    -lcv6 \
    -lcore -lm -lld

LDLFLAGS=-L/cgi/oracle/lib

all: serveurur

serveur.o: serveur.c serveur.h
    $(CC) $(CFLGS) serveur.c

serveur: serveur.o
    $(COB) $(COBFLGS) -e "main" $(LDLFLAGS) $(LIB) serveur.o -o serveur

```

This example of makefile is used to compile a Server accessing an Oracle database.

#### 9.4.1.2 Execution of the Listener on UNIX

To execute the listener process of the port on UNIX, enter the following command:

```
server path port &
```

with path = path to access the Server DLLs  
port = unused port in the services file

## 9.5 Windows/NT TCP-IP Socket

### 9.5.1 Configuration

The SOCKET Middleware API allows a VisualAge application and COBOL Microfocus Servers on a Windows/NT platform to communicate via a listener running on a Windows/NT machine.

The only parameters to consider are those of the Microfocus compiler.

#### 9.5.1.1 Microfocus Compiling Parameterization on Windows/NT

This parameterization is defined in the COBOL.DIR file in the \COBOL32\LBR directory (standard installation):

```
VERBOSE
TRACE
VSC2
PERFORM-TYPE "OSVS"
NOBOUND
IBMCOMP
NATIVE "EBCDIC"
ASSIGN "EXTERNAL"
SEQUENTIAL "LINE"
```

To compile, enter the following command: `CBLINK -v -d -s serveur.cbl`

The parameterization in the example compiles a Logical View server accessing indexed files.

The compilation command produces the COBOL Servers' DLLs.

#### 9.5.1.2 Execution of the Listener on Windows/NT

To execute the listener process of the port on Windows/NT, enter the following command:

```
serveur path port
```

with path = path to access the Server DLL  
port = unused port in the services file

## 9.6 CICS TCP/IP Sockets Interface

### 9.6.1 Configuration CICS TCP/IP

#### 9.6.1.1 Prerequisites

MVS/ESA :

- TCP/IP Version 3, Release 1
- CICS/ESA Version 3, Release 3
- CICS TCP/IP Socket Interface Version 3.1

#### 9.6.1.2 CICS Startup

Modification of the startup JCL for the CICS area.

```
-----
//PMTICST JOB (008),'CICS TEST PAC',MSGLEVEL=(2,0),CLASS=O,
// MSGCLASS=X
//CICST PROC INDEX=CICS330,
// UTINDX='PT$EXP.CICST330',
// REGSZE=6M,
// START='COLD',
// SIP=T,
//DFHRPL DD DSN=&UTINDX..LNK,DISP=SHR
// DD DSN=SYS1.TCPIP310.SEZALINK,DISP=SHR
// DD DSN=TCPIP310.SEZATCP,DISP=SHR
//TCPDATA DD SYSOUT=&OUTC,DCB=(DSORG=PS,RECFM=V,BLKSIZE=136)
-----
```

#### 9.6.1.3 Definition of CICS TCP/IP transactions

- Listener Task

```
-----
TRansaction : CSKL
Group       : TCPIPI
DEscription : Listener Task
PROGram     : EZACIC02
TWAsize     : 00000
PROFile     : DFHCICST
PArTitionset :
STatus      : Enabled
PRIMedsize  : 00000
TASKDATAloc : Below
TASKDATAKey : Cics
REMOTE ATTRIBUTES
DYnamic     : No
REMOTESystem :
REMOTEName  :
TRProf      :
Localq      :
-----
```

\*\*\*\*\*

[\\*TCP/IP Version 3.1.0](#)

- Enable the Socket Interface

```

-----
TTransaction : CSKE
Group       : TCPIPI
DEscription : Enable Sockets Interface
PROGram    : EZACIC00
TWAsize    : 00000
PROFile    : DFHCICST
PArTitionset :
STatus     : Enabled
PRIMedsize : 00000
TASKDATAloc : Below
TASKDATAKey : Cics
REMOTE ATTRIBUTES
DYnamic    : No
REMOTESystem :
REMOTENAME :
TRProf     :
Localq     :
-----

```

- Terminate the socket interface

```

-----
TTransaction : CSKD
Group       : TCPIPI
DEscription : Disable Sockets Interface
PROGram    : EZACIC00
TWAsize    : 00000
PROFile    : DFHCICST
PArTitionset :
STatus     : Enabled
PRIMedsize : 00000
TASKDATAloc : Below
TASKDATAKey : Cics
REMOTE ATTRIBUTES
DYnamic    : No
REMOTESystem :
REMOTENAME :
TRProf     :
+ Localq   :
-----

```

\*\*\*\*\*

[\\*TCP/IP Version 3.2.0](#)

- Configure the socket interface

```

-----
TTransaction : EZAC
Group       : TCPIPI
DEscription : CONFIGURE SOCKETS INTERFACE
PROGram    : EZACIC23
TWAsize    : 00000
PROFile    : DFHCICST
PArTitionset :
STatus     : Enabled
PRIMedsize : 00000
TASKDATAloc : Below
TASKDATAKey : Cics
REMOTE ATTRIBUTES

```

DYnamic : No  
 REMOTESystem :  
 REMOTENAME :  
 TRProf :  
 Localq :

---

- Enable the socket interface
- 

TRansaction : **EZAO**  
 Group : TCPIPI  
 DEscription : *ENABLE SOCKETS INTERFACE*  
 PROGram : **EZACIC00**  
 TWAsize : 00000  
 PROFile : DFHCICST  
 PArtitionset :  
 SStatus : Enabled  
 PRIMedsize : 00000  
 TASKDATAloc : Below  
 TASKDATAKey : Cics  
 REMOTE ATTRIBUTES  
 DYnamic : No  
 REMOTESystem :  
 REMOTENAME :  
 TRProf :  
 Localq :

---

- Terminate the socket interface
- 

TRansaction : **EZAP**  
 Group : TCPIPI  
 DEscription : *DISABLE SOCKETS INTERFACE*  
 PROGram : **EZACIC22**  
 TWAsize : 00000  
 PROFile : DFHCICST  
 PArtitionset :  
 SStatus : Enabled  
 PRIMedsize : 00000  
 TASKDATAloc : Below  
 TASKDATAKey : Cics  
 REMOTE ATTRIBUTES  
 DYnamic : No  
 REMOTESystem :  
 REMOTENAME :  
 TRProf :

---

#### 9.6.1.4 Definition of CICS TCP/IP p programs

---

PROGram : **EZACIC00**  
 Group : TCPIPI  
 DEscription : Connection Manager  
 Language : Assembler  
 RELoad : No  
 RESident : No  
 USAge : Transient

USEIpcopy : No  
 Status : Enabled  
 RSI : 00  
 Cedf : Yes  
 DAtalocation : Any  
 EXECKey : CICS  
 REMOTE ATTRIBUTES  
 REMOTESystem :  
 + REMOTENAME :  
   Transid :  
 EXECUTIONset : Fullapi

---

**PROGRAM : EZACIC02**  
 Group : TCPIPI  
 Description : Listener  
 Language : Assembler  
 REload : No  
 RESident : Yes  
 USAge : Normal  
 USEIpcopy : No  
 Status : Enabled  
 RSI : 00  
 Cedf : Yes  
 DAtalocation : Any  
 EXECKey : CICS  
 REMOTE ATTRIBUTES  
 REMOTESystem :  
 + REMOTENAME :  
 Transid :  
 EXECUTIONset : Fullapi

---

**Mapset : EZACICM**  
 Group : TCPIPI  
 Description : Mapset for Connection Manager  
 RESident : No  
 USAge : Transient  
 USEIpcopy : No  
 Status : Enabled  
 RSI : 00

---

**PROGRAM : EZACIC01**  
 Group : TCPIPI  
 Description : Task Related User Exit  
 Language : Assembler  
 REload : No  
 RESident : Yes  
 USAge : Normal  
 USEIpcopy : No  
 Status : Enabled  
 RSI : 00  
 Cedf : Yes  
 DAtalocation : Any  
 EXECKey : CICS  
 REMOTE ATTRIBUTES  
 REMOTESystem :  
 + REMOTENAME :  
 Transid :

EXECUtionset : Fullapi

---

PROGram : **EZACIC20**  
Group : TCPIPI  
DEscription : Initialization/termination for CICS Sockets  
Language : Assembler  
RELoad : No  
RESident : No  
USAge : Transient  
USElpacopy : No  
Status : Enabled  
RSI : 00  
Cedf : Yes  
DAtalocation : Any  
EXECKey : CICS  
REMOTE ATTRIBUTES  
REMOTESystem :  
+ REMOTENAME :  
Transid :  
EXECUtionset : Fullapi

---

PROGram : **EZACIC21**  
Group : TCPIPI  
DEscription : Initialization Module for CICS Sockets  
Language : Assembler  
RELoad : No  
RESident : No  
USAge : Transient  
USElpacopy : No  
Status : Enabled  
RSI : 00  
Cedf : Yes  
DAtalocation : Any  
EXECKey : CICS  
REMOTE ATTRIBUTES  
REMOTESystem :  
+ REMOTENAME :  
Transid :  
EXECUtionset : Fullapi

---

PROGram : **EZACIC22**  
Group : TCPIPI  
DEscription : Termination Module for CICS Sockets  
Language : Assembler  
RELoad : No  
RESident : No  
USAge : Transient  
USElpacopy : No  
Status : Enabled  
RSI : 00  
Cedf : Yes  
DAtalocation : Any  
EXECKey : CICS  
REMOTE ATTRIBUTES  
REMOTESystem :  
+ REMOTENAME :  
Transid :



EXECUtionset : Fullapi

---

PROGram : **EZACIC23**  
 Group : TCPIPI  
 DEscription : Primary Module for Transaction EZAC  
 Language : Assembler  
 REload : No  
 RESident : No  
 USAge : Transient  
 USElpacopy : No  
 Status : Enabled  
 RSI : 00  
 Cedf : Yes  
 DAtalocation : Any  
 EXECKey : User  
 REMOTE ATTRIBUTES  
 REMOTESystem :  
 + REMOTENAME :  
 Transid :  
 EXECUtionset : Fullapi

---

PROGram : **EZACIC24**  
 Group : TCPIPI  
 DEscription : Message Delivery Module for CICS Sockets  
 Language : Assembler  
 REload : No  
 RESident : No  
 USAge : Transient  
 USElpacopy : No  
 Status : Enabled  
 RSI : 00  
 Cedf : Yes  
 DAtalocation : Any  
 EXECKey : CICS  
 REMOTE ATTRIBUTES  
 REMOTESystem :  
 + REMOTENAME :  
 Transid :  
 EXECUtionset : Fullapi

---

PROGram : **EZACIC25**  
 Group : TCPIPI  
 DEscription : Cache Module for the Domain Name Server  
 Language : Assembler  
 REload : No  
 RESident : No  
 USAge : Normal  
 USElpacopy : No  
 Status : Enabled  
 RSI : 00  
 Cedf : Yes  
 DAtalocation : Any  
 EXECKey : User  
 REMOTE ATTRIBUTES  
 REMOTESystem :  
 + REMOTENAME :  
 Transid :

EXECUtionset : Fullapi

---

PROGram : **EZACICME**  
 Group : TCPIPI  
 DEscription : US English Text Delivery Module  
 Language : Assembler  
 RELoad : No  
 RESident : No  
 USAge : Normal  
 USElpacopy : No  
 Status : Enabled  
 RSI : 00  
 Cedf : Yes  
 DAtalocation : Any  
 EXECKey : CICS  
 REMOTE ATTRIBUTES  
 REMOTESystem :  
 + REMOTENAME :  
 Transid :  
 EXECUtionset : Fullapi

---

#### 9.6.1.5 Definition of the DCT Table

Definition of a TCPM transitional data queue for the LISTENER, in the DCT table.

---

TCPDATA DFHDCT TYPE=SDSCI, TCP/IP OUTPUT  
 BLKSIZE=136,  
 BUFNO=1,  
 DSCNAME=TCPDATA,  
 RECSIZE=132,  
 RECFORM=VARUNBA,  
 TYPEFLE=OUTPUT  
 \*  
 TCPM DFHDCT TYPE=EXTRA, USED FOR MESSAGES - SEE  
 DESTID=TCPM, INDDDEST=TCPM BELOW  
 DSCNAME=TCPDATA  
 \*  
 TCPIN DFHDCT TYPE=INTRA, TCP/IP  
 DESTID=TRAA,  
 DESTFAC=FILE,  
 TRIGLEV=1,  
 TRANSID=TRAA

#### 9.6.1.6 Definition and initialization of Configuration files (\*TCP/IP Version 3.2.0)

**EZACONFG** : CICS Sockets configuration file

---

File : **EZACONFG**  
 Group : SOCKETS  
 DEscription : CICS SOCKETS CONFIGURATION FILE  
 VSAM PARAMETERS  
 DSName : **CICS.STM9.SOCKETS.CFG**  
 Password : PASSWORD NOT SPECIFIED  
 Lsrpoolid : 1 1-8 ! None  
 DSNSharing : Allreqs Allreqs ! Modifyreqs

STRings : 001 1-255  
 Nsrgroup :  
 REMOTE ATTRIBUTES  
 REMOTESystem :  
 REMOTENAME :  
 RECORDSsize : 1-32767  
 Keylength : 1-255  
 INITIAL STATUS  
 SStatus : Enabled Enabled ! Disabled ! Unenabled  
 Opentime : Startup Firstref ! Startup  
 DIsposition : Share Share ! Old  
 BUFFERS  
 Databuffers : 00002 2-32767  
 Indexbuffers : 00001 1-32767  
 DATATABLE PARAMETERS  
 Table : No No ! Cics ! User  
 Maxnumrecs : 16-16777215  
 DATA FORMAT  
 RECORDFormat : V V ! F  
 OPERATIONS  
 Add : No No ! Yes  
 BRowse : Yes No ! Yes  
 DELete : No No ! Yes  
 REAd : Yes Yes ! No  
 Update : No No ! Yes  
 AUTO JOURNALLING  
 JOurnal : No No ! 1-99  
 JNLRead : None None ! Updateonly ! Readonly ! All  
 JNLSYNCRRead : No No ! Yes  
 JNLUpdate : No No ! Yes  
 JNLAdd : None None ! Before ! After ! All  
 JNLSYNCRWrite : No Yes ! No  
 RECOVERY PARAMETERS  
 RECOVery : None None ! Backoutonly ! All  
 Fwdrecovlog : No No ! 1-99  
 BAckuptype : Static Static ! Dynamic  
 SECURITY  
 RESsecnum : 00 0-24 ! Public

-----

**EZACACHE** : File required for the Domain Name Server Cache function

-----

File : **EZACACHE**  
 Group : SOCKETS  
 DEScription : DOMAIN NAME SERVER CACHE CONFIGURATION FILE  
 VSAM PARAMETERS  
 DSName : **CICS.STM9.SOCKETS.EZACACHE**  
 Password : PASSWORD NOT SPECIFIED  
 Lsrpoolid : 1 1-8 ! None  
 DSNSharing : Allreqs Allreqs ! Modifyreqs  
 STRings : 020 1-255  
 Nsrgroup :  
 REMOTE ATTRIBUTES  
 REMOTESystem :  
 REMOTENAME :  
 RECORDSsize : 1-32767  
 Keylength : 1-255  
 INITIAL STATUS  
 SStatus : Enabled Enabled ! Disabled ! Unenabled  
 Opentime : Startup Firstref ! Startup  
 DIsposition : Old Share ! Old

BUFFERS  
 Databuffers : 00060            2-32767  
 Indexbuffers : 02000           1-32767  
 DATATABLE PARAMETERS  
 Table : User            No ! Cics ! User  
 Maxnumrecs : 00004000        16-16777215  
 DATA FORMAT  
 RECORDFormat : V            V ! F  
 OPERATIONS  
 Add : Yes            No ! Yes  
 BRowse : Yes        No ! Yes  
 DElete : Yes        No ! Yes  
 REAd : Yes           Yes ! No  
 Update : Yes        No ! Yes  
 AUTO JOURNALLING  
 JOurnal : No        No ! 1-99  
 JNLRead : None        None ! Updateonly ! Readonly ! All  
 JNLSYNRead : No        No ! Yes  
 JNLUpdate : No        No ! Yes  
 JNLAdd : None        None ! Before ! AFter ! ALI  
 JNLSYNWrite : No        Yes ! No  
 RECOVERY PARAMETERS  
 RECOVery : None        None ! Backoutonly ! All  
 Fwdrecovlog : No        No ! 1-99  
 BAckupType : Static      Static ! Dynamic  
 SECURITY  
 RESsecnum : 00        0-24 ! Public

---

**JCL for the definition of the VSAM EZACONFG file and configuration of the  
 EZACICD macro for the CICS Sockets**

```

/*****
/* THE FOLLOWING JOB DEFINES AND THEN LOADS THE VSAM */
/* FILE USED FOR CICS/TCP CONFIGURATION. THE JOBSTREAM */
/* CONSISTS OF THE FOLLOWING STEPS. */
/* 1). DELETE A CONFIGURATION FILE IF ONE EXISTS */
/* 2). DEFINE THE CONFIGURATION FILE TO VSAM */
/* 3). ASSEMBLE THE INITIALIZATION PROGRAM */
/* 4). LINK THE INITIALIZATION PROGRAM */
/* 5). EXECUTE THE INITIALIZATION PROGRAM TO LOAD THE */
/* FILE */
/*****
//PTCONFIG JOB MSGLEVEL=(1,1)
/*
/* THIS STEP DELETES AN OLD COPY OF THE FILE
/* IF ONE IS THERE.
/*
//DEL EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
DELETE -
CICS.STM9.SOCKETS.CFG -
PURGE -
ERASE
/*
/* THIS STEP DEFINES THE NEW FILE
/*
//DEFILE EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
  
```

```

DEFINE CLUSTER (NAME(CICS.STM9.SOCKETS.CFG) VOLUMES(CICSVOL) -
  CYL(1 1) -
  IMBED -
  RECORDSIZE(150 150) FREESPACE(0 15) -
  INDEXED ) -
  DATA ( -
    NAME(CICS.STM9.SOCKETS.CFG.DATA) -
    KEYS (16 0) ) -
  INDEX ( -
    NAME(CICS.STM9.SOCKETS.CFG.INDEX) )
/*
/**
/** THIS STEP ASSEMBLES THE INITIALIZATION PROGRAM
/**
/**PRGDEF EXEC PGM=IEV90,PARM='OBJECT,TERM',REGION=1024K
/**SYSLIB DD DISP=SHR,DSNAME=SYS1.MACLIB
/** DD DISP=SHR,DSNAME=TCPV32.SEZACMAC
/**SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(5,1))
/**SYSUT2 DD UNIT=SYSDA,SPACE=(CYL,(2,1))
/**SYSUT3 DD UNIT=SYSDA,SPACE=(CYL,(2,1))
/**SYSPUNCH DD DISP=SHR,DSNAME=NULLFILE
/**SYSLIN DD DSNAME=&&OBJSET,DISP=(MOD,PASS),UNIT=SYSDA,
/** SPACE=(400,(500,50)),
/** DCB=(RECFM=FB,BLKSIZE=400,LRECL=80)
/**SYSTEM DD SYSOUT=*
/**SYSPRINT DD SYSOUT=*
/**SYSIN DD *
EZACICD TYPE=INITIAL, X
  PRGNAME=EZACICDF, X
  FILNAME=EZAONCFG
EZACICD TYPE=CICS, X
  TCPADDR=TCPIP, X
  NTASKS=20, X
  DPRTY=10, X
  CACHMIN=10, X
  CACHMAX=20, X
  CACHRES=5, X
  ERRORTD=CSKN, X
  APPLID=A6ECCSM9
EZACICD TYPE=LISTENER, X
  TRANID=CSKL, X
  PORT=9953, X
  BACKLOG=40, X
  ACCTIME=30, X
  GIVTIME=10, X
  REATIME=300, X
  NUMSOCK=100, X
  WLMGN1=CICSSTM9, X
  MINMSGL=4, X
  APPLID=A6ECCSM9
EZACICD TYPE=FINAL
/*
/**
/** THIS STEP LINKS THE INITIALIZATION PROGRAM
/**
/**LINK EXEC PGM=IEWL,PARM='LIST,MAP,XREF',
/** REGION=512K,COND=(4,LT)
/**SYSPRINT DD SYSOUT=*
/**SYSUT1 DD SPACE=(CYL,(5,1)),DISP=(NEW,PASS),UNIT=SYSDA
/**SYSLMOD DD DSNAME=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSDA,

```

```
//      SPACE=(TRK,(1,1,1)),
//      DCB=(DSORG=PO,RECFM=U,BLKSIZE=32760)
//SYSLIN DD DSNAME=&&OBJSET,DISP=(OLD,DELETE)
//*
/* THIS STEP EXECUTES THE INITIALIZATION PROGRAM
/*
//FILELOAD EXEC PGM=*.LINK.SYSLMOD,COND=(4,LT)
//EZACONFG DD DSNAME= CICS.STM9.SOCKETS.CFG,DISP=OLD
```

☞ You can also configure the CICS Sockets TCP/IP interface using the **EZAC** transaction.

JCL for the definition of the VSAM **EZACACHE** file and configuration of the **EZACICR** macro to use the DNS Cache

```
-----
//*****//
/* THE FOLLOWING JOB DEFINES AND THEN LOADS THE VSAM  *//
/* FILE USED FOR THE CACHE. THE DEFINITION CONSISTS OF *//
/* TWO IDCAMS STEPS TO PERFORM THE VSAM DEFINITION  *//
/* AND A STEP USING EZACICR TO BUILD THE FILE LOAD  *//
/* PROGRAM. THE FINAL STEP EXECUTES THE FILE LOAD  *//
/* PROGRAM TO CREATE THE FILE.  *//
//*****//
//PTCACHE JOB MSGLEVEL=(1,1)
/*
/* THIS STEP DELETES AN OLD COPY OF THE FILE
/* IF ONE IS THERE.
/*
//DEL EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
DELETE -
CICS.STM9.SOCKETS.EZACACHE -
PURGE -
ERASE
/*
/* THIS STEP DEFINES THE NEW FILE
/*
//DEFILE EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
DEFINE CLUSTER (NAME(CICS.STM9.SOCKETS.EZACACHE) VOLUMES(CICSVOL) -
CYL(1 1) -
IMBED -
RECORDSIZE(500 1000) FREESPACE(0 15) -
INDEXED ) -
DATA ( -
NAME(CICS.STM9.SOCKETS.EZACACHE.DATA) -
KEYS (255 0) ) -
INDEX ( -
NAME(CICS.STM9.SOCKETS.EZACACHE.INDEX) )
/*
/*
/* THIS STEP DEFINES THE FILE LOAD PROGRAM
/*
//PRGDEF EXEC PGM=IEV90,PARM='OBJECT,TERM',REGION=1024K
//SYSLIB DD DISP=SHR,DSNAME=SYS1.MACLIB
// DD DISP=SHR,DSNAME=TCPV32.SEZACMAC
//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(5,1))
//SYSUT2 DD UNIT=SYSDA,SPACE=(CYL,(2,1))
```

```

//SYSUT3 DD UNIT=SYSDA,SPACE=(CYL,(2,1))
//SYSPUNCH DD DISP=SHR,DSNAME=NULLFILE
//SYSLIN DD DSNAME=&&OBJSET,DISP=(MOD,PASS),UNIT=SYSDA,
//      SPACE=(400,(500,50)),
//      DCB=(RECFM=FB,BLKSIZE=400,LRECL=80)
//SYSTEM DD SYSOUT=*
//SYSPRINT DD SYSOUT=*

//SYSIN DD *
EZACICR TYPE=INITIAL
EZACICR TYPE=RECORD,NAME=ESSONVS1
EZACICR TYPE=FINAL
/*
//LINK EXEC PGM=IEWL,PARM='LIST,MAP,XREF',
//      REGION=512K,COND=(4,LT)
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD SPACE=(CYL,(5,1)),DISP=(NEW,PASS),UNIT=SYSDA
//SYSLMOD DD DSNAME=&&LOADSET(GO),DISP=(MOD,PASS),UNIT=SYSDA,
//      SPACE=(TRK,(1,1,1)),
//      DCB=(DSORG=PO,RECFM=U,BLKSIZE=32760)
//SYSLIN DD DSNAME=&&OBJSET,DISP=(OLD,DELETE)
/*
/* THIS STEP EXECUTES THE FILE LOAD PROGRAM
/*
//LOAD EXEC PGM=*.LINK.SYSLMOD,COND=((4,LT,ASM),(4,LT,LINK))
//EZACICRF DD DSN= CICS.STM9.SOCKETS.EZACACHE,DISP=OLD

```

### 9.6.1.7 Definition of the PLT table (\*TCP/IP V320)

For the automatic startup/stop of the CICS Sockets interface (\*TCP/IP V320) the EZACIC20 module must be added into the PLT table.

For the automatic startup, it must be added in PLTPI after the DFHDELIM entry.

```

-----
DFHPLT TYPE=ENTRY,PROGRAM=EZACIC20
-----

```

\*\*\*\*\*

For the automatic stop, it must be added in PLTSD before the DFHDELIM entry.

```

-----
DFHPLT TYPE=ENTRY,PROGRAM=EZACIC20
-----

```

## 9.6.2 Configuration TCP/IP MVS/ESA

### 9.6.2.1 Modification of the TCP/IP Configuration

For the use of the CICS TCP/IP Sockets, a PORT must be defined for the CICS area in the TCP/IP "hlq.PROFILE.TCPIP" configuration file.

```

,*****
;
;PROFILE.TCPIP
;=====
;
;
;-----
;

```

; NOTES:

;
  
; A port that is not reserved in this list can be used by any user.
  
; If you have TCP/IP hosts in your network that reserve ports
  
; in the range 1-1023 for privileged applications, you should
  
; reserve them here to prevent users from using them.
  
;
  
; The port values below are from RFC 1060, "Assigned Numbers."
  
;

PORT

1415 TCP CSQ9CHIN ; MQSeries CSQ9
  
9011 TCP PTMBRUNT ; TeamConnection
  
9950 TCP EXCICS9 ; CICS Socket for CICS9
  
**9953 TCP CICSSTM9 ; CICS Socket for A6ECCSM9**

### 9.6.2.2 TCPJOBNAME Parameter in the *hlq.TCPIP.DATA* File

For the initialization of CICS TCP/IP, you must know the name of the MVS TCP/IP procedure specified in the *hlq.TCPIP.DATA* file, *TCPIPJOBNAME* parameter.

```

;
;*****
; Name of Data Set:  TCPIP.DATA          *
;
;
; COPYRIGHT = NONE.                    *
;
;
; This data, TCPIP.DATA, is used to specify configuration *
; information required by TCP/IP client programs.         *
;
;
; Syntax Rules for the TCPIP.DATA configuration data set: *
; treated as a comment.                               *
;
;*****
; TCPIPJOBNAME specifies the name of the started procedure that was
; used to start the TCPIP address space.  TCPIP is the default.
;
TCPIPJOBNAME TCPIP

```

## 9.6.3 Manual Start and Stop of CICS TCP/IP

### 9.6.3.1 Start of CICS TCP/IP

Execute the CICS -> **CSKE** transaction to start CICS TCP/IP manually (\*TCP/IP V310).

```

-----
CSKE                                EZACIC00
                                CICS TASK-RELATED USER EXIT
                                CONNECTION MANAGER
                                ENABLE CICS-TCP/IP API

```



TCPIPJOBNAME **tcpip**--- PORT **9953**-  
 PF1=HELP PF3=QUIT EZAME00

\*\*\*\*\*

Execute the CICS -> EZA0 transaction to start CICS TCP/IP manually (\*TCP/IP V320).

### EZA0,START,CICS

APPLID= ==> **A6ECCSM9** APPLID of CICS

\*\*\*\*\*

\*Next, check with **CEMT I TAS** that the CSKL transaction is active :

### CEMT I TAS

STATUS: RESULTS - OVERTYPE TO MODIFY  
 Tas(0000023) Tra(CKAM) Sus Tas Pri( 255 )  
 Tas(0000024) Tra(CKTI) Sus Tas Pri( 001 )  
 Tas(0000027) Tra(DSNC) Sus Tas Pri( 255 )  
 Tas(0000034) Tra(ISER) Sus Tas Pri( 254 )  
**Tas(0000046) Tra(CSKL) Sus Tas Pri( 255 )**

### 9.6.3.2 Stop of CICS TCP/IP

Execute the CICS => **CSKD** transaction stop CICS TCP/IP manually (\*TCP/IP V310).

CSKD EZACIC00  
 CICS TASK-RELATED USER EXIT  
 CONNECTION MANAGER

DISABLE CICS-TCP/IP API

==> 1 QUIESCENT DISABLE-API  
 ==> 2 IMMEDIATE DISABLE-API

ENTER ONE OF THE ABOVE DISABLE-API OPTION NUMBERS: **2**

PF1=HELP PF3=QUIT EZAMD00

\*\*\*\*\*

Execute the CICS => **EZA0** transaction to stop CICS TCP/IP manually (\*TCP/IP V320).

### EZA0,STOP,CICS

APPLID ==> **A6ECCSM9** APPLID of CICS  
 IMMEDIATE ==> **Y** Enter Yes!No

## 9.6.4 Cobol Compilation

### 9.6.4.1 Cobol Compilation of the VisualAge Pacbase Communications Monitor Program

Modification of the JCL for the compilation and link-edit to integrate the CICS Sockets TCP/IP interface.

```
-----
//PTTDASOC JOB (661),LH,CLASS=X,MSGCLASS=X,MSGLEVEL=(0,0)
//*JCLLIB ORDER=DSNY220.JCLLIB
/*****
/* Jcl of compil/link for VAP communications monitor *
/* with the CICS SOCKET TCP/IP interface *
/***** **
//CBL EXEC DB2SOCK,MEMBER=CLTMVS,LOAD=PT$VIC.VIC.MTR8,
// DBRMLIB=PT$VIC.VIC.DBRMLIB,SOURCE=PT$VIC.CICS.SOURCE
//LKED.SYSLIB DD DSN=PT$VIC.TCPIP310,SEZATCP,LIB=SHR
//LKED.SYSIN DD *
    INCLUDE SYSLIB(EZACICAL)
    INCLUDE SYSLIB(EZACIC04)
    INCLUDE SYSLIB(EZACIC05)
    INCLUDE DB2LOAD(DSNCLI)
    NAME CLTMVS(R)
//
```

## 9.6.5 CICS definitions for the VisualAge Pacbase application

### 9.6.5.1 Definition of Transaction Code

```
-----
TRansaction : VSO1
  Group      : VISUAL
  DDescription : VISUAL/CLIENT Transaction SOCKET CICS TCP/IP
  PROGRAM    : CLTMVS
  TWasize    : 00000
  PROFile    : DFHCICST
  PArTitionset :
  SStatus    : Enabled
  PRIMedsize : 00000
  TASKDATA Loc : Below
  TASKDATA Key : User
  REMOTE ATTRIBUTES
  DYnamic     : No
  REMOTESystem :
  REMOTENAME  :
  TRProf      :
+ Localq     :
```

### 9.6.5.2 Definition of the Communications Monitor Program

```
-----
PROGram     : CLTMVS
  Group      : VISUAL
```

```

DEscription   : Communications monitor SOCKET CICS TCP/IP
Language      : CObol
RELoad       : No
RESident     : No
USAge        : Normal
USEIpacopy   : No
Status       : Enabled
RSI          : 00
Cedf         : Yes
DAtalocation : Below
EXECKey      : User
REMOTE ATTRIBUTES
REMOTESystem :
+ REMOTENAME :
+ Transid    :
EXECUTIONset : Fullapi

```

---

### 9.6.5.3 Definition of the VSAM Workfile

```

File          : FRVABI
Group         : VISUAL
DEscription   :
VSAM PARAMETERS
DSName       : PT$VIC.CICS.FRBIS
Password     :
Lsrpoolid    : 1
DSNSharing   : Allreqs
STRings      : 001
Nsrgroup     :
REMOTE ATTRIBUTES
REMOTESystem :
REMOTENAME   :
RECORDSize   :
Keylength    :
INITIAL STATUS
+ STATUS     : Enabled

```

---

## 9.6.6 Client Configuration

### 9.6.6.1 VAPLOCAT.INI File Parameters

The environment variables used for the communication with the CICS Sockets must be defined in the VAPLOCAT.INI.

```

<EnvSocketMVS>
LENGTH=08192
MONITOR=CLTMVS          <- Name of the CICS communications monitor program
MWARE=TCPMVS           <- Type of the Middleware Communication protocol
MWTRANSID=VSO1        <- CICS Transaction Code used for the application
MWTIMEOUT=220
MWCODEPAGE=297         <- Host Page Code for ASCII <-> EBCDIC transcoding
MWADDRESS=9.134.16.10 9953 <- TCP/IP MVS Address and CICS/ESA area port

```

---

**9.134.16.10** corresponds to the TCP/IP address of the MVS/ESA host

9951 corresponds to the port defined for the CICS area => PMTCICST

### 9.6.6.2 VP Middleware

The VisualAge Pacbase Middleware DLL used for the la communication with CICS TCP/IP Sockets is named => IXOTMVS.DLL.

### 9.6.6.3 Traces

When a communication problem occurs between the Client application and the Server, refer to the messages displayed in the LOG of the concerned CICS area.

To check that the transaction is actually activated by the client application, it is necessary to pass under CICS the display command of first called program (Communications Monitor), before and after the client application execution. Then check that the USE is up +1. This means that the program has been executed.

#### CEMT I PROG(CLTMVS)

```
STATUS: RESULTS - OVERTYPE TO MODIFY
Prog(TESTMQ ) Len(0084584) Cob Pro Ena Pri   Ced
Res(000) Use(000001946) Bel Uex Ful
```

## 9.7 Local Middleware

### 9.7.1 Configuration

The LOCAL Middleware API is used for the communication between a VisualAge application and COBOL Microfocus Servers which are on the same workstation. The only parameters to consider are the Microfocus compiler ones.

#### 9.7.1.1 Microfocus OS/2 Compiling Parameterization

This parameterization is defined in the COBOL.DIR file in the \COBOL32\LBR directory (standard installation):

```
VERBOSE
TRACE
VSC2
PERFORM-TYPE "OSVS"
NOBOUND
IBMCOMP
NATIVE "EBCDIC"
ASSIGN "EXTERNAL"
SEQUENTIAL "LINE"
SQL "DB2"
SQLINIT
SQLDB "PTATDF"           =>   Name of the DB2 plan
SQLBIND " "
```

To compile, run the following command: CBLINK -v -d -s  
serveur.cbl

This example of parameterization is used to compile a Logical View Server accessing a DB2/2 database.

The compile command produces the COBOL Server DLLs.

Once the Microfocus compiler is installed, it is necessary to build the following DLLs: `SQLINIT.DLL`, `_SQLE3X.DLL` and `_SQLPRLD.DLL`.

## 9.8 TUXEDO

### 9.8.1 Client

Tuxedo /WS version 6.x

The only configuration to consider consists in indicating the server address and the port associated with the application via the `WSNADDR` environment variable.

`WSNADDR` must respect the following syntax:

```
WSNADDR=0X0002ppppaaaaaaaa
      | | ↪ 8-char. long IP address in hexa
      | ↪ 4-char. Lon port no. in hexa
      ↪ AF_INET domain.
```

```
example : WSNADDR=0X00020BB8C0060A5D
              | ↪ 192.6.10.93
              ↪ 3000
```

If the Tuxedo /WS version allows it (version 6.4 or higher), this address must be specified as follows:

```
WSNADDR=//server:port
          | ↪ Port No
          ↪ Logical name or server IP address
```

```
example : WSNADDR=//9.143.96.178 :3005
```

### 9.8.2 Server

Refer to the TUXEDO manual.

The services' names in the Tuxedo server must correspond to the programs' external names.

## 9.9 MQSERIES

### 9.9.1 CICS Adapter

Example of a simple configuration for a CICS application triggered by the Trigger Monitor MQSeries.

- REPLY QUEUE Definition

```
Queue name . . . . . VAP.REQUEST.Q
Description . . . . . : REQUEST QUEUE

Put enabled . . . . . : Y Y=Yes,N=No
Get enabled . . . . . : Y Y=Yes,N=No
Usage . . . . . : N N=Normal,X=XmitQ
```

```

Storage class . . . . . : DEFAULT
Creation method . . . . . : PREDEFINED
Output use count . . . . . : 0
Input use count . . . . . : 0
Current queue depth . . . . . : 0
Default persistence . . . . . : Y Y=Yes,N=No
Default priority . . . . . : 5 0 - 9
Message delivery sequence . . . . . : F P=Priority,F=FIFO
Permit shared access . . . . . : Y Y=Yes,N=No
Default share option . . . . . : S E=Exclusive,S=Shared
Index type . . . . . : N N=None,M=MsgId,C=CorrelId
Maximum queue depth . . . . . : 100000 0 - 999999999
Maximum message length . . . . . : 4194304 0 - 4194304
Retention interval . . . . . : 999999999 0 - 999999999 hours
Creation date . . . . . : 1999-06-23
Creation time . . . . . : 12.59.47

```

#### Trigger Definition

```

Trigger type . . . . . : E F=First,E=Every,D=Depth,N=None

Trigger set . . . . . : Y Y=Yes,N=No
Trigger message priority : 0 0 - 9
Trigger depth . . . . . : 1 1 - 999999999
Trigger data . . . . . :

Process name . . . . . : VAP.PROCESS.DEF
Initiation queue . . . . . : CICS.INITQ

```

#### • REPLY QUEUE Definition

```

Queue name . . . . . : VAP.REPLY.Q
Description . . . . . : output QUEUE

```

```

Put enabled . . . . . : Y Y=Yes,N=No
Get enabled . . . . . : Y Y=Yes,N=No
Usage . . . . . : N N=Normal,X=XmitQ
Storage class . . . . . : DEFAULT
Creation method . . . . . : PREDEFINED
Output use count . . . . . : 0
Input use count . . . . . : 0
Current queue depth . . . . . : 0
Default persistence . . . . . : Y Y=Yes,N=No
Default priority . . . . . : 0 0 - 9
Message delivery sequence . . . . . : F P=Priority,F=FIFO
Permit shared access . . . . . : Y Y=Yes,N=No
Default share option . . . . . : S E=Exclusive,S=Shared
Index type . . . . . : N N=None,M=MsgId,C=CorrelId
Maximum queue depth . . . . . : 999999999 0 - 999999999
Maximum message length . . . . . : 4194304 0 - 4194304
Retention interval . . . . . : 999999999 0 - 999999999 hours
Creation date . . . . . : 1999-06-23
Creation time . . . . . : 13.00.10

Trigger type . . . . . : N F=First,E=Every,D=Depth,N=None

Trigger set . . . . . : N Y=Yes,N=No
Trigger message priority : 0 0 - 9
Trigger depth . . . . . : 1 1 - 999999999

```

```

Trigger data . . . . . :
Process name . . . . . :
Initiation queue . . . . . :

```

- INITIATION QUEUE Definition

```

Queue name . . . . . : CICS.INITQ
Description . . . . . : CKTI initiation queue

Put enabled . . . . . : Y Y=Yes,N=No
Get enabled . . . . . : Y Y=Yes,N=No
Usage . . . . . : N N=Normal,X=XmitQ
Storage class . . . . . : SYSTEM
Creation method . . . . . : PREDEFINED
Output use count . . . . . : 0
Input use count . . . . . : 1
Current queue depth . . . . . : 0
Default persistence . . . . . : Y Y=Yes,N=No
Default priority . . . . . : 5 0 - 9
Message delivery sequence . . . . . : F P=Priority,F=FIFO
Permit shared access . . . . . : Y Y=Yes,N=No
Default share option . . . . . : E E=Exclusive,S=Shared
Index type . . . . . : N N=None,M=MsgId,C=CorrelId
Maximum queue depth . . . . . : 100 0 - 999999999
Maximum message length . . . . . : 4194304 0 - 4194304
Retention interval . . . . . : 999999999 0 - 999999999 hours
Creation date . . . . . : 1999-05-07
Creation time . . . . . : 18.30.18
Trigger Definition

Trigger type . . . . . : N F=First,E=Every,D=Depth,N=None

Trigger set . . . . . : N Y=Yes,N=No
Trigger message priority . . . . . : 0 0 - 9
Trigger depth . . . . . : 1 1 - 999999999
Trigger data . . . . . :

Process name . . . . . :
Initiation queue . . . . . :
Event Control

```

- PROCESS Definition

```

Process name . . . . . : VAP.PROCESS.DEF
Description . . . . . : VAP Process

Application type . . . . . : CICS
Application ID . . . . . : AMQM ← Transaction application Serveur
User data . . . . . : VAP.REQUEST.Q QMGR

```

## 9.10 CPIC-C/XCP2

### 9.10.1 CPI-C/XCP2 – TDS Configuration

#### 9.10.1.1 Prerequisites

- GCOS7 ==> **V7 - TS7458**  
 - CNP7 or DATANET ==> **PID ISO/DSA**  
 - GCOS7 or GCOS8 ==> **CPI-C/XCP2**  
 - DPX20 or ESCALA ==> **Stack OSI et CPI-C/OSI**

#### 9.10.1.2 Frontal CNP7 Configuration

- Definition of Ethernet Controller linked to the LAN and of ISO workstation

CNP7 must have the `ISOPLG` option for the ISO/DSA connections(non-standard option requiring a separate license).

```
&*****
&* LOCAL MICROFEP GENERATION *
&*****
&
SC CNP7 LOC -ADDR 001:003 -BRK AT -ISOPLG
TS CNP7 LOC -ADDR 001:003
EX CNP7 -MEM 70 -NBCREQ 50
&
&*****
&* LOCAL NETWORK CONTROLLERS *
&*****
SC STMB RMT -NAT DSA -ADDR 001:001 -SR SR00
SR SR00 ISO -TS TS00
TS TS00 DIWS -NR NR00 -TPDU 512
NR NR00 LAN1 -PL PL00
PL PL00 CSM1 -ETHAD 0800385F0100 -CB CB01
PL PL01 CSM2 -CB CB01
CB CB01 LAN1 -PL RLN1
PL RLN1 CSMA -CT RLN1 -ETHAD 0800385F0200
CT RLN1 RLNA -PHAD 1
&*****
&* Definition of the attachment to the Ethernet LAN *
&*****
CB CB02 LAN1 -PL RLN2
PL RLN2 CSMA -ETHAD 0800385F0092 -CT RLN2
CT RLN2 RLNA -PHAD 2
&*****
&* Definition of STID STM4 (Escala) on LAN *
&*****
SC STM4 RMT -NAT ISO -ADDR 001:086 -SR SR10
SR SR10 ISO -TS TS10
TS TS10 DIWS -NR NR10 -TPDU 512
NR NR10 LAN1 -PL PL10
PL PL10 CSM1 -ETHAD 080038210FF8 -CB CB02
&-----
```

#### 9.10.1.3 GCOS7 Network Configuration (NETGEN command)

- Definition of the Remote Node (NETWORK command)

```
NET STMB QM=1 QFBLKSZ=512 QMBLKSZ=512 SIMU=2;
```



```

COMM '-----';
COMM '-- SYSTEME LOCAL --';
COMM '-----';
SYS STMB PF=LSYS SCID=001:001 ISL=(5F-01-00,EA01,CBL_MAIN) OBJLIST;
COMM '-----';
COMM '-- SYSTEMES - RELAIS / VOISINS -';
COMM '-----';
SYS CNP7 PF='CNP7/CNS7/A2' SCID=001:003 ISL=(5F-02-00) OBJLIST;
DEF CT SET WATCH=3000 ;
COMM '-----';
COMM '-- SYSTEMES A DISTANCE --';
COMM '-----';
SYS STM4 PF='STID/ISO/SID4' SCID=001:086 PT=CNP7 OBJLIST ;
COMM '-----';

```

- Definition TDS Workstation and XCP2 correspondents (DIRECTORY command)

```

DIR ;
COMM '*****'
COMM '** APPLICATION TDS1 CPI-C/XCP2 VISUALAGE PACBASE **'
COMM '*****'
TDSWKS NAME=TDS1 TMSSESS=28 XCP2WKS=WKS1 ;
XCP2WKS NAME=WKS1 MBX=XKS1 MAXSC=16
MAXTX=10, MAXC=10, MAXPOOL=10, MAXPCS=10, MAXCONV=8
SYNCPT=0, SCBUFSZ=(32512,32512), CONVBUFSZ=64512
CONV_ACCEPT=1, CONVCK=1, CONV_USERID=MANDATORY ;
XCP2COR NAME=LUTXCP2 XCP2WKS=WKS1 ;
XCP2COR NAME=LUJAS2 SC=STM4 MBX=SESJAS2 PARALLEL=1 ;
XCP2POOL NAME=POOLTDS1 XCP2COR=LUTXCP2 XCP2WKS=WKS1 MAXSC=4 ;
XCP2POOL NAME=MODEXCP2 XCP2COR=LUJAS2 XCP2WKS=WKS1 MAXSC=4
WINSRCE=2 WINTRGT=2 WINAUTO=2
DRSRCE=0 DRTRGT=1 ;
EDIR;

```

#### 9.10.1.4 GCOS7 Configuration

- Site Catalog Configuration

Modification of the project associated with the TDS application using the CPI-C/XCP2 services.

Under MNCAT (Maintain\_Catalog) :

- add into the application list the *mailbox of the XCP2 WorkStation* (MBX=XKS1) defined in the XCP2WKS command of the Gcos7 Networkconfiguration.
- add into the workstation list, the ISO workstation corresponding to the Escala (STM4)
- add into the user list, with no password:
  - the XCP2 Administrator with the generic name tdsname\_ADM (tdsname being the name of the TDS application).
  - the XCP2 remote correspondent (XCP2COR NAME=LUJAS2 SC=STM4)

```

-----
PROJECT          -MODIFDATE- -----JOBCLASS-----
->PT             12.09.97  OB:BDFLT QC:IOFCL KB KC AB AC
                  AD AE AF JB JD A  B  C  D  E  F

```

G H I J K L M N O

DFLTOUTC :

ATTRIBUTES

```

STD NMAIN NSTATION NRMS
mstupb=SITE ostupb=PROJECT mstupi=SITE ostupi=PROJECT
mass storage volumes
  STM000  STM010  STM090  STM100
magnetic tape volumes
*
application          -tdscode-
->IOF      DFLT
->TDS1          0FFFFFFF
->XKS1          0FFFFFFF
station
->MAIN
->STMB      DFLT
->STM4
user          -modifdate-
->LUJAS2     DFLT 12.09.97
->PTTDA      DFLT 12.29.97
->TDS1_ADM   DFLT 12.09.97
billing      -modifdate- credit - charge - balance -
->613       DFLT 09.26.88 99999999 0 99999999

```

### 9.10.1.5 TDS Configuration

- Preparation of the TDS environment

To use the XCP2 protocol with TDS, the PPCLOG must be assigned using the TP7PREP utility:

- XCP2=YES
- XCP2SZ=size
- XCP2MD=media  
(DEAL=N to keep the existing files)

```

-----
15 OVL HOLD;
16 VL PRY='SYSFILE=CAT,FILESTAT=CAT,CATNAME=PT,IMPORT=NO',
17   PREV5=MTPREP,PREV6=TP7PREP,
18   PRN='SYSFILE=RSD,FILESTAT=UNCAT',
19   FF='TDS1,MS/D500,STM130,MS/D500,STM130,DEAL=N',
20   GG='DBGSZ=1,MAXDBG=3,CBSZ=1,SMSZ=15,MAXSM=20,XCP2=YES,NBSW=2,
    SW2SZ=4',
21   VLVL='VL=(&FF','&PRY';
22 IV &PREV6 SYS.HSLLIB &VLVL,&GG);
23 SEND '====> PREPARATION OF 'TDS1' SUCCESSFUL <====';
-----

```

- TDS Generation

To use CPI-C/XCP2 with TDS, the STDS of the tdsname.SLLIB must be modified, then a new TDS generation (TP7GEN) must be carried out.

- TDS Section

This clause defines the waiting period for the CPI-C calls (this command must be inserted before the USE PROCEDURE clause):

```

MAXIMUM XCP2-WAITTIME
-----
TDS SECTION.
PROGRAM-ID. TDS1.
BTNS          IS BTNS.
NUMBER OF DUMMY CORRESPONDENT IS 4 MAXIMUM IS 8.
SIMULTANEITY  5.
RESERVE       16 AREAS.
NUMBER MODULES 10.
MESSAGE-LENGTH 6000.
TPR-TIME-LIMIT 45000.
MAXIMUM OF XCP2-WAITTIME IS 300.
USE "MENU" TRANSACTION-MENU.
USE ZAR100.
USE ZAR200.
-----
    
```

- Transaction Section

The transaction definition with CPI-C/XCP2 in the MESSAGE-ID clause must contain the sub-clause:

```
XCP2 SERVICE USED
```

To open the CPI-C/XCP2 sessions with TDS, a transaction must have been previously defined using this clause.

```

-----
MESSAGE "VIC2" ASSIGN CLCFOL
IMPLICIT COMMITMENT
XCP2 SERVICE USED
PAGES          50
WITH TPR ACCOUNTING
AUTHORITY-CODES 31
TRANSACTION-STORAGE SIZE 500.
-----
    
```

- management of POOLs and CORRESPONDANTs CPI-C/XCP2

The commands will be executed on the TDS Master.

- Open of XCP2 session Pool

The OPEN\_COR\_POOL command (abbrev. OCPOOL) allows to open one or more session pools linking a local TDS application and a partner application.

```
OCPOOL LUJAS2 MODEXCP2 TDS=TDS1
```

```

-----
1/1          OPEN_COR_POOL          -->:

          open correspondent pool

COR          + correspondent name          LUJAS2
POOL         pool name (or *) (xcp2-cor only) MODEXCP2
ATTRIBUTE    address extension (xcp1-cor only)
ACTIVE_SE    active session number (xcp1-cor only)
TDS          tds name                    TDS1
-----
    
```

- List of XCP2 session Pool:

```
LSCPOOL LUJAS2 MODEXCP2 TDS=TDS1
```

```
-----
1/1          LIST_COR_POOL          -->:

          list correspondent pool

COR          + correspondent name          LUJAS2
POOL         pool name (or *) (xcp2-cor only)  MODEXCP2
ATTRIBUTE    address extension (or *) (xcp1-cor only)
NETGEN       known to netgen?             0
PRINT_MEMBER print member name
TDS          ds name                      TDS1
-----
```

- Close of XCP2 session Pool:

The CLOSE\_COR\_POOL command (abbrev. CLCPOOL) allows to close one or more session pools linking a local TDS application and a partner application.  
CLCPOOL LUJAS2 MODEXCP2 TDS=TDS1

```
-----
1/1          CLOSE_COR_POOL          -->:

          close correspondent pool

COR          + correspondent name          LUJAS2
POOL         pool name (or *) (xcp2-cor only)  MODEXCP2
ATTRIBUTE    address extension (or *) (xcp1-cor only)
STRONG       abnormal termination?         0
DRAIN_SOURCE drain-source (xcp2-cor only)
DRAIN_TARGET drain-target (xcp2-cor only)
TDS          tds name                      TDS1
-----
```

- Tests and Debugging of transactions using CPI-C/XCP2

For debugging, start a trace for the transaction. The data is sent to a member (NameTX\_\_\_\_Userid\_\_\_\_\_01) of the Tdsname.DEBUG.

Example of a trace for the VIC2 transaction:

```
SEND_TDS 'TRACE PRINT TX=VIC2 USER=*' TDS=TDS1
```

## 9.10.2 CPI-C/OSI Configuration on AIX Server

Ensure that the stack OSI communication services are installed and loaded onto the AIX server. They are required for the CPI-C/OSI services.

In the "System Management" menu

- Select -> Communication service applications
  - > OSI Networking
  - > OSI Configuration
  - > Load the last selected configuration

### 9.10.2.1 Definition of the XCP2 user profile on AIX (required configuring CPI-C/XCP2)

```
# User profile XCP2
```

```
DACUAB=/usr/xcp2/up
```

```

DACBIN=/usr/xcp2/ti
DACRSC=/usr/xcp2/conf
DACMEN=/usr/xcp2/men
DACCFG=/usr/xcp2/conf/conf_confcgi
DACSID=/usr/xcp2/side/s2side_cgi
DACSES=/usr/xcp2/ses/s2session_confcgi
DACTRC=1
DACLOG=/usr/xcp2/log
DACTRD=/usr/xcp2/log
export DACBIN DACUAB DACRSC DACCFG DACSES
export DACSID DACMEN DACLOG DACTRD DACTRC
PATH=/usr/bin:/etc:/usr/sbin:/usr/ucb:$HOME/bin:/usr/bin/X11:/sbin:$DACUAB:.
export PATH

```

### 9.10.2.2 CPI-C/OSI Configuration

The configuration CPI-C/OSI must correspond to the definitions of the TDS WorkStation (XCP2WKS) in the Gcos7 Network configuration.

Local LU = Correspondent XCP2 (XCP2COR NAME=LUJAS2)

Remote LU = Correspondent XCP2 (XCP2COR NAME=LUTXCP2)

Mode = Pool XCP2 (XCP2POOL NAME=MODEXCP2)

- smit xcp2
- configuration menu
- define resource menu

- **NODE Definition**

- define Node Server
- add

|  |                |
|--|----------------|
| * add Local Node fields in                         | <b>NODEJAS</b> |
| * Memory Size ( between 50 and 10 000 Kilo bytes ) | <b>[256]</b>   |
| * Node Operator Messages                           | <b>1</b>       |
| * Logging Messages                                 | <b>1</b>       |
| * Verb Message Buffer Size                         | <b>512</b>     |
| * Node Message Buffer Size                         | <b>512</b>     |

ENTER to validate, then F3 to quit

\*\*\*\*\*

- **LOCAL LU Definition**

**\_\_\_ - define Local LU**  
- add

|  |               |
|--|---------------|
| * Add Local LU fields in                         | <b>LUJAS2</b> |
| * Logical Unit ID ( between 1 to 254 )           | <b>[2]</b>    |
| * Network Qualifier                              | <b>[NONE]</b> |
| * Network Name                                   | <b>[NONE]</b> |
| * Default Timeout ( -1 to 3600 seconds or NONE ) | <b>[-1]</b>   |
| * Maximum Transaction Programs ( 1 to 255 )      | <b>[25]</b>   |

ENTER to validate, then F3 to quit

\*\*\*\*\*

- REMOTE LU Definition

- **define Remote LU**

- *add*

|  |                |
|--|----------------|
| * Add Remote LU fields in                      | <b>LUTXCP2</b> |
| * Network Qualifier                            | [NONE]         |
| * Network Name                                 | [NONE]         |
| * Parallel Sessions                            | 1              |
| * LU-LU Password in hexadecimal                | [NONE]         |
| * Security Acceptance                          | <b>NONE</b>    |
| * Logical Unit ID ( between 1 to 254, or NONE) | [5]            |

ENTER to validate, then F3 to quit

\*\*\*\*\*

- MODE Definition

- **define Mode**

- *add*

|   |                 |
|---|-----------------|
| * Add Mode fields in                          | <b>MODEXCP2</b> |
| * Minimum RU Length ( 8 to 256 )              | [256]           |
| * Maximum RU Length ( 256 to 4096 )           | [4092]          |
| * Session Reinit                              | <b>OPER</b>     |
| * Maximum Sessions ( 1 to 255 )               | [4]             |
| * Minimum First Speaker Sessions ( 0 to 255 ) | [2]             |
| * Auto-initiated Sessions ( 0 to 255 )        | [2]             |
| * Allow Queued Binds                          | 1               |

ENTER to validate, then F3 to quit

\*\*\*\*\*

- PARTNER LU definition

- \_\_\_ - **define Partner**

- *add*

|                         |                 |
|-------------------------|-----------------|
| * Add Partner fields in | <b>PARTTDS1</b> |
| * Remote LU Name        | <b>LUTXCP2</b>  |
| * Mode Name             | <b>MODEXCP2</b> |

ENTER to validate, then F3 to quit

\*\*\*\*\*

- TP Definition

- **define Transaction Program**

- *add*

|                                     |              |
|-------------------------------------|--------------|
| * Add Transaction Program fields in | <b>TPALL</b> |
| * Program Name                      | [*]          |

|                     |        |
|---------------------|--------|
| * Network Name      | [NONE] |
| * Status            | ENABLE |
| * Conversation Type | EITHER |
| * Sync Level        | CNFRM  |
| * Assign LUW        | 0      |

ENTER to validate

|                                     |                         |
|-------------------------------------|-------------------------|
| * Add Transaction Program fields in | TPCNOS                  |
| * Program Name                      | [/usr/xcp2/ti/dac_cnos] |
| * Network Name                      | [06F1]                  |
| * Status                            | ENABLE                  |
| * Conversation Type                 | EITHER                  |
| * Sync Level                        | CNFRM                   |
| * Assign LUW                        | 0                       |

ENTER to validate, then F3 to quit

\*\*\*\*\*

- Configuration Definition

- **define Configuration**
- *add a Configuration*

\* Enter Configuration Name [CONFCGI]

- ENTER to validate, then F3 to quit

- *Set / Change the definitions of an existing Configuration*

- *Select Node Server, ENTER, then position the cursor on CONFCGI*

\* Editing configuration CONFCGI

\* Select the Configuration's Node Server Name

- *Then F4 to list then position the cursor on NODEJAS*

\* Editing configuration CONFCGI

\* Select the Configuration's Node Server Name NODEJAS

- ENTER to validate, then F3 to quit

- *Select Local LU(s), ENTER*

- *Add Resources Definitions, ENTER, then position the cursor on CONFCGI*

\* Editing configuration CONFCGI

\* Select Local LU Name

- *Then F4 to list, then position the cursor on LUJAS2*

\* Editing configuration CONFCGI

\* Select Local LU Name LUJAS2

- ENTER to validate, then F3 to quit

- *Select Partner(s), ENTER*

- *Add Resources Definitions, ENTER, then position the cursor on CONFCGI*

\* **Enter Local LU to edit** +

- *Then F4 to list, then position the cursor on LUJAS2*

\* Editing configuration CONFCGI

\* Local LU to edit LUJAS2

\* Select Partner(s) to Add in the Configuration

- F4 to list, then position the cursor on PARTTDS1

\* Editing configuration **CONFCGI**  
 \* Local LU to edit **LUJAS2**  
 \* Select Partner(s) to Add in the Configuration **PARTTDS1**

- ENTER to validate, then F3 to quit

- Select Transaction Program(s), ENTER

- Add Resources Definitions, ENTER, then position the cursor on CONFCGI

\* Enter Local LU to edit

- F4 to list, then position the cursor on LUJAS2

\* Editing configuration **CONFCGI**  
 \* Local LU to edit **LUJAS2**  
 \* Select TP(s) to Add in the Configuration

- Then F4 to list, then position the cursor on TPALL

\* Editing configuration **CONFCGI**  
 \* Local LU to edit **LUJAS2**  
 \* Select TP(s) to Add in the Configuration **TPALL**

- ENTER to validate

F4 to list, then position the cursor on TPCNOS

\* Editing configuration **CONFCGI**  
 \* Local LU to edit **LUJAS2**  
 \* Select TP(s) to Add in the Configuration **TPCNOS**

- ENTER to validate, the F3 to quit

\*\*\*\*\*

- Display Configuration

- Come back to the "Configuration Menu"

- Select -> Display Configuration

\* Choose Configuration

- F4 to list, then position the cursor on CONFCGI

- ENTER to list the configuration CONFCGI

LU Name = LUJAS2 NetName = NONE Wait = -1

| Partner  | Rmt LU  | Mode     |
|----------|---------|----------|
| -----    | -----   | -----    |
| PARTTDS1 | LUTXCP2 | MODEXCP2 |

Program = TPALL Network Name = NONE  
 Program = TPCNOS Network Name = 06F1

Press the keys <s> and <Enter> to STOP the display of the Configuration  
 or just <Enter> to CONTINUE...

- ENTER, then F3 to quit



\*\*\*\*\*

**\* Configuration Generation**

- Come back to the "Configuration Menu"
- Select -> Generate Configuration

**\* Choose Configuration**

- F4 to list, then position the cursor on CONFCGI then ENTER

**Configuration 'CONFCGI' Generated.**

- F3 to quit

\*\*\*\*\*

- Update of Configuration file

- Come back to the "Configuration Menu"
- Select -> Set Current Configuration

**\* Choose Configuration**

- F4 to list, then position the cursor on CONFCGI then ENTER

**Current Configuration File is CONFCGI.Z.**

- F3 to quit

\*\*\*\*\*

- ISO Configuration

\_\_\_\_\_ \* The ISO configuration refers to the definitions of the TDS WorkStation (XCP2WKS) in the Gcos7 Network configuration and the CNP7 configuration.

**\* Definition of Local Site LU**

- \_\_\_\_\_ \* Session Selector = Mailbox (MBX) du XCP2COR LUJAS2
- \_\_\_\_\_ \* Transport Selector = DSA Name of the AIX Node (STM4)
- \_\_\_\_\_ \* Network Selector = AIX Ethernet Address
- \_\_\_\_\_ \* Network = Network type (ETH for Ethernet)

- Come back to the "Configuration Menu"
- Select -> ISO Session Management
  - > Define Configuration
  - > Define Local Site LUs
  - > Add

**\* Enter the local site LU name**

□

- F4 to list, then position the cursor on LUJAS2, then ENTER

|                      |                       |
|----------------------|-----------------------|
| * Add Local Site Lu  | <b>LUJAS2</b>         |
| * Session Selector   | <b>[SESJAS2]</b>      |
| * Transport Selector | <b>[STM4]</b>         |
| * Network Selector   | <b>[080038210FF8]</b> |
| TRANSPORT Parameters |                       |
| * Preferred class    | <b>4</b>              |

```

* Alternative class          4
* Flow control              1
* Credit (1 to 15 )       [3]
* TPDU size                [512]
* Checksum                 0
* Network                  ETH

```

BE CAREFUL :

The environment variable DACSES (in the profile file) must contain the PATH of a FILE in which is written the PATH of the SESSION FILE.

- ENTER, then F3 to quit

\*\*\*\*\*

- Definition of Remote Site LU

```

_____ * Session Selector = Mailbox of the TDS WorkStation (MBX=XKS1)
* Transport Selector = DSA Name of the GCOS7 Node (STMB)
* Network Selector = Ethernet Address of the CNP7 Controller
* Network = Network type (ETH for Ethernet)

```

-> Define Remote Site LUs  
-> Add

```

* Enter the Remote site LU name      []

```

- F4 to list, then position the cursor on LUTXCP2, then ENTER

```

* Add Remote Site Lu          LUTXCP2
* Session Selector           [XKS1]
* Transport Selector         [STMB]
* Network Selector           [0800385F0092]
TRANSPORT Parameters
* Preferred class            4
* Alternative class          4
* Flow control               1
* Credit (1 to 15 )         [3]
* TPDU size                  [512]
* Checksum                   0
Network                      ETH

```

BE CAREFUL :

The environment variable DACSES (in the profile file) must contain the PATH of a FILE in which is written the PATH of the SESSION FILE.

- ENTER, then F3 to quit

\*\*\*\*\*

- Activation Trace ISO

```

_____ -> ISO Session Trace Options

```

Enter ISO Session trace options:

```

* Trace Level for ISO Session Interface Calls      2
* Trace Filename ("stdout" for screen)           [trace_xcp2]

```

(Trace file will be created in \$DACLOG directory.)

BE CAREFUL :

The environment variable *DACSES* (in the profile file) must contain the *PATH* of a *FILE* in which is written the *PATH* of the *SESSION FILE*.

- ENTER, then F3 to quit

\*\*\*\*\*

- Display ISO Configuration

- Come back to the "ISO Session Management" Menu  
- Select -> Display Configuration

**ISO SESSION FILE : /usr/xcp2/ses/session\_confcgi.**

| LU_name       | S_Selector | T_Selector | N_Selector   | N_Type |
|---------------|------------|------------|--------------|--------|
| LUJAS2 (loc)  | SESJAS2    | STM4       | 080038210FF8 | ETH    |
| LUTXCP2 (rmt) | XKS1       | STMB       | 0800385F0092 | ETH    |

- ENTER, then F3 to quit

\*\*\*\*\*

- Current Configuration File

- Come back to the "**CPI-C OSI Services**" Menu  
- Select -> Network Operations Menu  
-> Set Current Configuration

\* Choose Configuration []

- F4 to list, then position the cursor on *CONFCGI*, then ENTER

**Current Configuration File is CONFCGI.Z.**

- F3 to quit

\*\*\*\*\*

- Activation of the NODE

- Come back to the "**CPI-C OSI Services**" Menu  
- Select -> Network Operations Menu  
-> Activate Node Server

\* Really do this 1

- ENTER,

**Node Server Activated.**

- F3 to quit

- Select -> Display XCP2 Daemons

PID 46960 Node Server: dac\_schd  
PID 39538 Node Server: dac\_tpi

- F3 to quit

\*\*\*\*\*

- Initialization of CPI-C/XCP2 Sessions

- Come back to the "**CPI-C OSI Services**" Menu
  - Select -> *CPI-C OSI Session Operations Menu*
  - > *Initialize Service Manager Mode*

\* Local LU Name  
\* Remote LU Name

\_\_\_\_\_ - Then F4 to list, then position the cursor on LUJAS2

\* Local LU Name                   **LUJAS2**  
\* Remote LU Name

- Then F4 to list, then position the cursor on LUTXCP2

\* Local LU Name                   **LUJAS2**  
\* Remote LU Name                 **LUTXCP2**

- ENTER to validate

**Mode Initialization Complete.**

- F3 to quit

- Select -> *Initialize Session Limits*

\* Local LU Name  
\* Remote LU Name  
\* Mode Name  
SESSION Limits  
\* Maximum Sessions                 []  
\* Minimum First Speaker Sessions   []  
\* Minimum Bidder Sessions           []

- Then F4 to list "Local LU Name", position the cursor on LUJAS2 then ENTER
- Select "Remote LU Name", F4, position the cursor on LUTXCP2, ENTER
- Select "Mode Name", F4, position the cursor on MODEXCP2 then ENTER
- Specify the Maximum and Minimum Sessions

\* Local LU Name                   **LUJAS2**  
\* Remote LU Name                 **LUTXCP2**  
\* Mode Name                       **MODEXCP2**  
SESSION Limits  
\* Maximum Sessions               **[4]**  
\* Minimum First Speaker Sessions **[2]**  
\* Minimum Bidder Sessions       **[2]**

- ENTER to validate

**Mode Initialization Complete.**  
**Limits were negotiated.**

- F3 to quit

- Display Status LUs and CPI-C/XCP2 Sessions

- Select -> LU Status Display

**Configuration: /usr/xcp2/conf/CONF CGI.Z**  
**LU Status Display**

| LU     | REMOTE LU | CONFIGURED MODE | MAXIMUM | CURRENT MAXIMUM | ACTIVE FSPK | ACTIVE BIDR |
|--------|-----------|-----------------|---------|-----------------|-------------|-------------|
| LUJAS2 | LUTXCP2   | SNASVCMG        | 2       | 2               | 1           | 0           |
|        |           | MODEXCP2        | 4       | 4               | 2           | 2           |

- F3 to quit

- Select -> Display Session Status

**Configuration: /usr/xcp2/conf/CONF CGI.Z**  
**Session Status Display**

| LU     | REMOTE  | MODE     | ID      | FSPK | CNVID | TPID | PPID |
|--------|---------|----------|---------|------|-------|------|------|
| LUJAS2 | LUTXCP2 | SNASVCMG | 0126404 | F    |       |      |      |
| LUJAS2 | LUTXCP2 | MODEXCP2 | 0127780 | B    |       |      |      |
| LUJAS2 | LUTXCP2 | MODEXCP2 | 0128428 | B    |       |      |      |
| LUJAS2 | LUTXCP2 | MODEXCP2 | 0127052 | F    |       |      |      |
| LUJAS2 | LUTXCP2 | MODEXCP2 | 0125732 | F    |       |      |      |

- F3 to quit

\*\*\*\*\*

- Definition SIDE INFORMATION

- Come back to the "Configuration Menu" Menu
  - Select -> Side Information Management
    - > Update the current SIDE\_INFORMATION
    - > Add

Symbolic Destination Name **[CLCFOL]**

- Partner TP name, indicate the name of the TDS server transaction that calls the Communications Monitor program

- Then F4 to list "Partner LU Name", position the cursor on LUTXCP2 then ENTER
- Select "Mode Name", F4, position the cursor on MODEXCP2 then ENTER
- Indicate the Path and thename of the active configuration
- Select "Local LU Name", F4, position the cursor on LUJAS2 then ENTER

```

* Adding Symbolic Destination          CLCFOL
* Partner TP name                      [VIC2]
* Partner LU name (= Remote LU )      [LUTXCP2]
* Mode_name                            [MODEXCP2]
Partner User name                      []
Partner Password                       []
* Security Type                        0
* Node file path                       [/usr/xcp2/conf/CONF CGI.Z]
* Local LU name                        [LUJAS2]
BE CAREFUL :
```

The environment variable DACSID (in the profile file) must contain the PATH of a FILE in which is written the PATH of the SIDE INFO FILE.

*- ENTER to validate, then F3 to quit*

## 10 Appendix 2: Use of a Customized Middleware

### 10.1 VisualAge for Java

To implement your own middleware, proceed as follows:

- overwrite the `ExchMgrImpl` class with the `ExchMgrImpl.java` file. This file contains the source code of this class which implements the Exchange Manager. It is delivered with the setup CD-ROM in the same directory as the `Vaprun.jar` file corresponding to the Pacbench C/S Runtime.
- inside this class, you must redefine the `getNewServer` method.
- and create a class which contains all the methods implementing the `com.ibm.vap.server.Server` interface.

### 10.2 VisualAge for Smalltalk

The use of customized middleware consists in replacing Pacbench C/S middleware by specific middleware.

This operation is only possible when using Pacbench C/S with VisualAge.

Usually, this operation consists in using inheritance mechanisms to modify the basic actions associated with the calls of remote Servers.

The classes that manage Servers calls are generic. They are loaded once when installing the product and are independent of the variable classes associated with each Folder View Proxy.

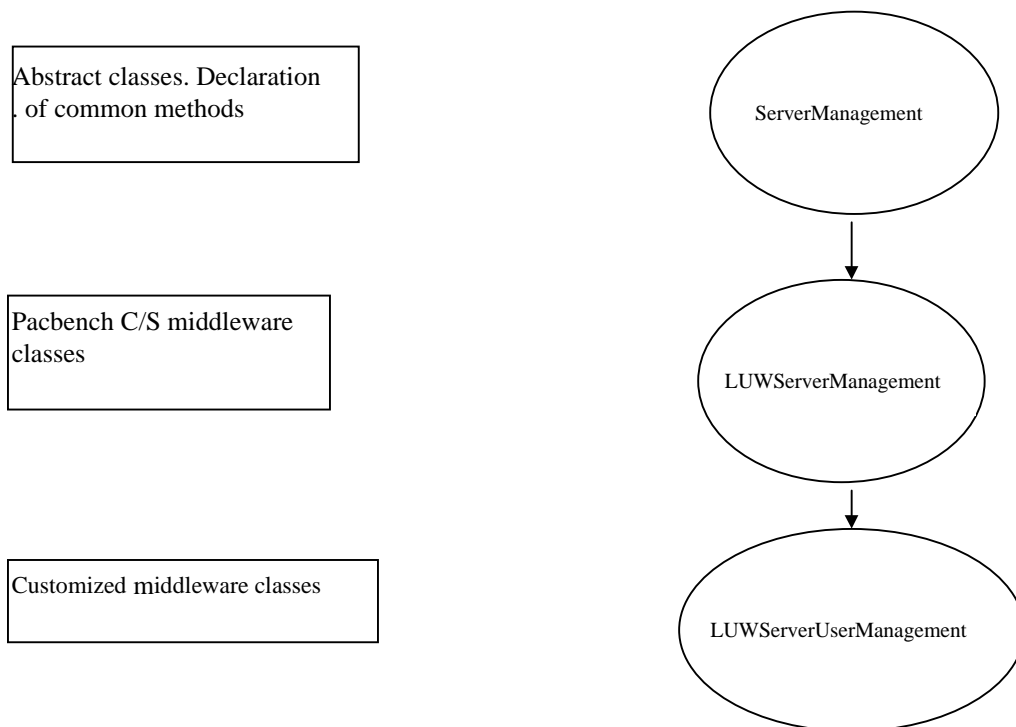
These generics enable server call actions to be modified once only so that all the Folder View Proxy can use the new middleware.

These modifications are permanent; they will not be overwritten by importing any new Folder View Proxy or by re-loading generic classes.

#### 10.2.1 Schema of Inheritance of Server Call Classes

When you use the VisualAge for Client middleware, executed actions are inherited from the `LUWServerManagement` class.

When you use customized middleware, executed actions correspond to actions that have been redefined in the `LUWServerUserManagement` class.



### 10.2.2 Instantiating Objects from the Pacbench Client/Server middleware

The `LUWServerUserManagement` class can only have one instance. The Proxy manages the instantiation and the verification of its object uniqueness.

The type of object used to execute an action is chosen by the Proxy according to the communication protocol defined in the Communications Monitor used by the Proxy.

### 10.2.3 Actions to Load again for Integrating Specific Middleware

These actions correspond to the entry points used to implement the specific middleware services.

#### 10.2.3.1 Action callServer: with: with: with

This instance action is common to the `LUWServerManagement` class.

It is executed by Proxy instance for each request sent to a Server.

It sends a message to a Server and waits for the reply to then send it to the Proxy component.

It receives three `string`-type parameters corresponding respectively to the code of the communication server to be called, the message to be sent to this server and the local buffer code, if it exists.



It returns an **array** object with two fields:

- The first field holds a **#OK** symbol when the exchange is successful or a **#Error** code when a communication error interrupts the exchange.
- The second field holds the Server response when the exchange is successful or an Ordered Collection of error messages when a communication error interrupted the exchange. This list of error messages will be automatically copied into the **aErrorLabels** attribute of the Logical View Proxy concerned.

### 10.2.3.2 Creation of a Local User Buffer

This buffer is used to send information, other than the code of the service called and the communication area, required by the customized communication layer. This information is sent to the communication layer via the Logical View Proxy.



To generate this buffer, you must code the **LOCALBUF** option on the server. For further details, refer to the *Pacbench C/S, Vol. II –Business Logic User's Guide*.

#### 10.2.3.2.1 Generated Part Corresponding to the Description of the Buffer

During generation, a local buffer creates a non-visual part which can be called on the Free Form Surface.

This buffer consists of fields which you fill in from other graphic objects or specific Smalltalk methods. These fields cannot be modified by the communication layer which only retrieves the data from this buffer.

#### 10.2.3.2.2 LocalBuffer Attribute

This attribute represents the local buffer. It is generated in the **ProxyLV** class. It enables you to connect the local buffer to the Proxy.



**11 Index**

## A

## Actions [COM]

|                                       |                    |
|---------------------------------------|--------------------|
| belongsToSubSchema                    | 210                |
| checkExistenceOfDependencies          | 198, 206           |
| completeInstance                      | 210                |
| createInstance                        | 195                |
| createUserServiceInstance             | 208                |
| deleteUserInstance                    | 208                |
| executeUserServices                   | 208                |
| getCheck<fieldIndex>                  | 209                |
| getDetailFromData                     | 198, 201, 202, 203 |
| getDetailFromDataDescription          | 203                |
| getExtractMethodCodesCount()          | 204                |
| getExtractMethodCodesElementAt(Int i) | 204                |
| getRowCount                           | 198, 205, 207      |
| getRowsElementAt(Int i)               | 198, 205, 207      |
| getUserInputRowCount                  | 208                |
| getUserInputRowsElementAt(Int i)      | 208                |
| getUserOutputRowCount                 | 208                |
| getUserOutputRowsElementAt(Int i)     | 208                |
| getUserServiceCodesCount              | 208                |
| getUserServiceCodesElementAt(Int i)   | 208                |
| is<delco>Present                      | 209                |
| lock                                  | 208                |
| modifyInstance                        | 195                |
| modifyUserInstance                    | 208                |
| readAllChildrenFromDetail             | 197, 203           |
| readFirstChildrenFromDetail           | 197, 201           |
| readInstance                          | 197, 205           |
| readInstanceAndLock                   | 197                |
| readNextPage                          | 197, 198, 202, 204 |
| readPreviousPage                      | 197                |
| readWithAllChildren                   | 197, 205           |
| readWithAllChildrenAndLock            | 197                |
| readWithAllChildrenFrom               | 197                |
| readWithFirstChildren                 | 197, 205           |
| readWithFirstChildrenAndLock          | 197                |
| readWithFirstChildrenFrom             | 198                |
| ResetCollection                       | 207                |
| resetSubSchema                        | 210                |
| resetUserRows                         | 208                |
| resetUserServiceCodes                 | 208                |
| selectInstances                       | 197, 198, 204      |
| set<delco>Present(a Boolean)          | 209                |
| setCheck<fieldIndex,a Boolean>        | 209                |
| undoAllLocalFolderUpdates             | 206, 207           |
| undoLocalFolderUpdates                | 206, 207           |
| updateFolder                          | 198                |

## Actions [Smalltalk]

|                                      |                         |
|--------------------------------------|-------------------------|
| aReplyId                             | 144                     |
| belongsToSubSchema                   | 147                     |
| checkExistenceOfDependentInstances   | 134, 142, 145           |
| completeInstance                     | 147                     |
| createInstance                       | 133                     |
| createUserServiceInstance            | 145                     |
| deleteUserServiceInstance            | 145                     |
| executeUserService                   | 144                     |
| executeUserServices                  | 146                     |
| getDetailFromDataDescription         | 135, 139, 157           |
| getReplyOf                           | 181                     |
| isNull<delco>                        | 147                     |
| lock                                 | 144, 146                |
| modifyInstance                       | 133                     |
| modifyUserServiceInstance            | 145                     |
| readAllChildren(data)                | 145                     |
| readAllChildrenFrom                  | 134, 145                |
| readAllChildrenFromCurrentInstance   | 134, 139, 145, 160      |
| readFirstChildren(data)              | 145                     |
| readFirstChildrenFrom                | 134, 145                |
| readFirstChildrenFromCurrentInstance | 134, 145                |
| readInstance                         | 134, 141, 144, 145, 172 |

|                                      |                    |
|--------------------------------------|--------------------|
| readInstanceAndLock                  | 134, 144, 145      |
| readInstanceWithAllChildren          | 134, 141, 144, 145 |
| readInstanceWithAllChildrenAndLock   | 134, 144, 145      |
| readInstanceWithFirstChildren        | 134, 141, 144, 145 |
| readInstanceWithFirstChildrenAndLock | 134, 144, 145      |
| readNextPage                         | 134, 140, 144, 157 |
| readPreviousPage                     | 134, 145, 157      |
| ResetCollection                      | 143                |
| resetPendingReplyOf                  | 181                |
| resetSubSchema                       | 147                |
| resetUserServiceCodes                | 146                |
| resetUserServiceInputInstances       | 146                |
| selectInstances                      | 134, 140, 144, 164 |
| setCheck:a Boolean on:<delco>        | 147                |
| setNull:a Boolean on:<delco>         | 147                |
| transferReferenceFromSelectedRow     | 165                |
| undoAllLocalFolderUpdates            | 142, 143           |
| undoLocalFolderUpdates(DataUpdate)   | 142                |
| undoLocalFolderUpdatesFor            | 143                |
| unlock                               | 144                |
| unLock                               | 146                |
| updateFolder                         | 134, 159, 174      |

## Attributes [COM]

|                               |                                   |
|-------------------------------|-----------------------------------|
| action                        | 49                                |
| detail                        | 198, 199, 205, 206, 209, 212, 213 |
| extractMethodCode             | 204                               |
| getsubSchemasCount            | 196                               |
| getSubSchemasElementAt(index) | 196                               |
| globalSelection               | 198, 204                          |
| globalSelectionIndicator      | 205                               |
| localSort                     | 196                               |
| ManualCollectionReset         | 207                               |
| maxNumberOfRequestedInstances | 198, 205                          |
| refreshOption                 | 207                               |
| rows                          | 196, 198, 199, 205                |
| selectionCriteria             | 204                               |
| serverCheckOption             | 209                               |
| ServerCheckOption             | 195                               |
| subSchema                     | 196                               |
| subSchema                     | 209                               |
| updatedFolders                | 49, 198                           |
| UpdatedInstancesCount         | 49                                |
| userDetail                    | 208                               |

## Attributes [Smalltalk]

|                                   |  |
|-----------------------------------|--|
| detail                            | 135, 141, 142, 146, 157, 158, 162, 164, 174, 180 |
| errorManager                      | 176  |
| extractMethodCode                 | 140  |
| extractMethodList                 | 140  |
| globalSelectionIndicator          | 134, 140, 141                                    |
| lastMessageId                     | 144  |
| localSort                         | 132  |
| ManualCollectionReset             | 143  |
| maximumNumberOfRequestedInstances | 134, 141   |
| refreshOption                     | 143  |
| rows                              | 132, 135, 141, 156, 157, 162, 163, 165           |
| selectedItems                     | 165  |
| selectionCriteria                 | 140, 157, 172                                    |
| serverCheckOption                 | 147  |
| ServerCheckOption                 | 133  |
| subSchema                         | 133  |
| subSchema                         | 147  |
| subSchemaList                     | 133  |
| updatedFolders                    | 134  |
| userDetail                        | 145  |
| userServiceCode                   | 145  |
| userServiceInputRows              | 145  |
| userServiceList                   | 145  |
| userServiceOutputRows             | 145  |

## E

## Events [COM]

|                           |     |
|---------------------------|-----|
| ABOUT_TO_CHANGE_SELECTION | 207 |
|---------------------------|-----|

|  |                |
|--|----------------|
| LOCKED_FAILED.....                                   | 209            |
| NO_PAGE_AFTER.....                                   | 197, 210, 211  |
| NO_PAGE_BEFORE.....                                  | 197, 210       |
| NOT_COMPLETE.....                                    | 211            |
| NOT_FOUND.....                                       | 211            |
| PAGE_AFTER.....                                      | 211            |
| PAGE_BEFORE.....                                     | 210            |
| Events [Java]  |                |
| aboutToChangeSelection.....                          | 70             |
| lockFailed.....                                      | 74             |
| noPageAfter.....                                     | 61, 77, 78     |
| noPageBefore.....                                    | 61, 77         |
| notComplete.....                                     | 78             |
| notFound.....  | 78             |
| pageAfter.....                                       | 77, 78         |
| PageBefore.....                                      | 77             |
| Events [Smalltalk]                                   |                |
| aboutToChangeSelection.....                          | 143            |
| asyncRequest.....                                    | 144            |
| getLastSelectResponseStatus.....                     | 149            |
| lockFailed.....                                      | 146            |
| noPageAfter.....                                     | 134, 148       |
| noPageBefore.....                                    | 134, 148       |
| notFound.....  | 149            |
| notRead.....   | 149            |
| pageAfter.....                                       | 148            |
| PageBefore.....                                      | 148            |
| replyPending.....                                    | 144            |
| <br>   |                |
| <b>G</b>   |                |
| Generated classes [COM]                              |                |
| [Prefix]Buffer.....                                  | 48             |
| [Prefix]Data.....                                    | 48             |
| [Prefix]DataUpdate.....                              | 48             |
| [Prefix]SelectionCriteria.....                       | 48             |
| [Prefix]UserData.....                                | 49             |
| [Prefix]VapError.....                                | 49             |
| Generated classes [Java]                             |                |
| [Prefix]Buffer.....                                  | 27             |
| [Prefix]Data.....                                    | 27             |
| [Prefix]DataUpdate.....                              | 27             |
| [Prefix]SelectionCriteria.....                       | 27             |
| [Prefix]TableModel.....                              | 27             |
| [Prefix]UpdateTableModel.....                        | 28             |
| [Prefix]UserData.....                                | 27             |
| DataDescription.....                                 | 19             |
| selectionCriteria.....                               | 19             |
| Generated classes [Smalltalk]                        |                |
| [Prefix][ClassCode][ProxyLv].....                    | 42             |
| DataDescription[LogicalViewCode][Suffix].....        | 42             |
| SelectionCriteria[LogicalViewCode][Suffix].....      | 42             |
| UpdatedDataDescription[LogicalViewCode][Suffix]..... | 42             |
| UserContext[UserBufferCode][Suffix].....             | 42             |
| UserDataDescription[LogicalViewCode][Suffix].....    | 42             |
| ValuesOf[DataElementCode][Suffix].....               | 42             |
| ValuesOf[DataElementCode][Suffix]Converter.....      | 42             |
| Generic classes [Java]                               |                |
| CommunicationError.....                              | 19             |
| DependentNode.....                                   | 19             |
| DependentProxyLv.....                                | 19             |
| Folder.....  | 19             |
| HierarchicalNode.....                                | 19             |
| HierarchicalProxyLv.....                             | 19             |
| LocalException.....                                  | 19             |
| Node.....  | 19             |
| Pacbase Date Choice.....                             | 20             |
| Pacbase Date Field.....                              | 20             |
| Pacbase Date Swing Field.....                        | 20             |
| Pacbase Decimal Choice.....                          | 20             |
| Pacbase Decimal Field.....                           | 20             |
| Pacbase Integer Choice.....                          | 20             |
| Pacbase Integer Field.....                           | 20             |
| Pacbase Long Choice.....                             | 20             |
| Pacbase Long Field.....                              | 20             |
| Pacbase Swing Date ComboBox.....                     | 20             |
| Pacbase Swing Date RadioButtonGroup.....             | 20             |
| Pacbase Swing Decimal ComboBox.....                  | 20             |
| Pacbase Swing Decimal Field.....                     | 20             |
| Pacbase Swing Decimal RadioButtonGroup.....          | 20             |
| Pacbase Swing Integer ComboBox.....                  | 20             |
| Pacbase Swing Integer Field.....                     | 20             |
| Pacbase Swing Integer RadioButtonGroup.....          | 20             |
| Pacbase Swing Long ComboBox.....                     | 20             |
| Pacbase Swing Long Field.....                        | 20             |
| Pacbase Swing Text ComboBox.....                     | 20             |
| Pacbase Swing Text Field.....                        | 20             |
| Pacbase Swing Time ComboBox.....                     | 20             |
| Pacbase Swing Time Field.....                        | 20             |
| Pacbase Text Choice.....                             | 20             |
| Pacbase Text Field.....                              | 20             |
| Pacbase Time Choice.....                             | 20             |
| Pacbase Time Field.....                              | 20             |
| ProxyLv.....   | 19             |
| ReferenceNode.....                                   | 19             |
| ReferenceProxyLv.....                                | 19             |
| RootNode.....  | 19             |
| ServerException.....                                 | 19             |
| SystemError.....                                     | 19             |
| VapDependentProxyProperties.....                     | 19             |
| VapException.....                                    | 19             |
| VapFolderProperties.....                             | 19             |
| VapHierarchicalProxyProperties.....                  | 19             |
| VapProxyProperties.....                              | 19             |
| VapReferenceProxyProperties.....                     | 19             |
| Generic classes [Smalltalk]                          |                |
| LUWServerManagement.....                             | 343, 344       |
| LUWServerUserManagement.....                         | 343, 344       |
| VapError.....  | 176            |
| VapErrorManager.....                                 | 176            |
| VapLUWServerManagement.....                          | 34             |
| VapLUWServerUserManagement.....                      | 34             |
| VapServerManagement.....                             | 34             |
| VapUserServiceError.....                             | 34             |
| VpcsValues.....                                      | 42             |
| VpcsValuesConverter.....                             | 42             |
| <br>   |                |
| <b>M</b>   |                |
| Methods [Java]                                       |                |
| belongsToSubSchema.....                              | 77             |
| checkExistenceOfDependentInstances.....              | 61, 69, 72     |
| completeInstance.....                                | 77             |
| createInstance.....                                  | 58             |
| createUserInstance.....                              | 74             |
| deleteUserInstance.....                              | 74             |
| executeUserService.....                              | 71             |
| executeUserServices.....                             | 74             |
| get<delco>Index.....                                 | 76             |
| getCheck(int index).....                             | 76             |
| getDetailFromDataDescription.....                    | 62, 64, 66     |
| initializeInstance.....                              | 70             |
| is<delco>Present.....                                | 76             |
| lock.....  | 72, 74         |
| modifyInstance.....                                  | 58             |
| modifyUserInstance.....                              | 74             |
| readAllChildren(data).....                           | 72             |
| readAllChildrenFrom.....                             | 61, 72         |
| readAllChildrenFromCurrentInstance.....              | 72             |
| readAllChildrenFromDetail.....                       | 61, 66         |
| readFirstChildren(data).....                         | 72             |
| readFirstChildrenFrom.....                           | 72             |
| readFirstChildrenFromCurrentInstance.....            | 72             |
| readFirstChildrenFromDetail.....                     | 61, 64         |
| readInstance.....                                    | 61, 68, 71, 72 |
| readInstanceAndLock.....                             | 61, 71, 72     |
| readInstanceWithAllChildren.....                     | 61, 68, 71, 72 |

|  |                        |
|--|------------------------|
| readInstanceWithAllChildrenAndLock.....      | 61, 71, 72             |
| readInstanceWithFirstChildren.....           | 61, 68, 71, 72         |
| readInstanceWithFirstChildrenAndLock.....    | 61, 71, 72             |
| readNextPage.....                            | 61, 62, 64, 65, 67, 71 |
| readPreviousPage.....                        | 61, 72                 |
| resetCollection.....                         | 70                     |
| resetSubSchema.....                          | 77                     |
| resetUserServiceCodes.....                   | 74                     |
| resetUserServiceInputInstances.....          | 74                     |
| selectInstances.....                         | 61, 62, 67, 71         |
| setNull<delco>Present(boolean aBoolean)..... | 76                     |
| undoAllLocalFolderUpdates.....               | 69                     |
| undoLocalFolderUpdates.....                  | 69                     |
| unlock.....                                  | 72                     |
| updateFolder.....                            | 61                     |

## P

### Properties [Java]

|  |                          |
|--|--------------------------|
| action.....                            | 27                       |
| detail.....                            | 62, 68, 69, 75, 100, 101 |
| extractMethodCode.....                 | 67                       |
| extractMethodCodes.....                | 67                       |
| globalSelection.....                   | 61, 67, 68               |
| localSort.....                         | 59                       |
| manualCollectionReset.....             | 70                       |
| maximumNumberOfRequestedInstances..... | 61, 67                   |

|                            |                    |
|----------------------------|--------------------|
| readInstance.....          | 68                 |
| refreshOption.....         | 69                 |
| rows.....                  | 19, 24, 59, 62, 68 |
| selectionCriteria.....     | 67                 |
| serverCheckOption.....     | 76                 |
| ServerCheckOption.....     | 58                 |
| subSchema.....             | 60                 |
| subSchema.....             | 76                 |
| subSchemaList.....         | 61                 |
| updatedFolders.....        | 61                 |
| UpdatedFolders.....        | 27                 |
| updateInstancesCount.....  | 27                 |
| userDetail.....            | 73                 |
| userServiceCode.....       | 73                 |
| userServiceCodes.....      | 73                 |
| userServiceInputRows.....  | 73                 |
| userServiceOutputRows..... | 73                 |

## V

|                   |                              |
|-------------------|------------------------------|
| vaperror.txt..... | 176, 179                     |
| VAPLOCAT.INI..... | 115, 117, 187, 192, 218, 221 |
| VDWN.....         | 244, 245                     |
| VUP1.....         | 239, 242, 243                |
| VUP2.....         | 242, 243, 244                |

This index is not an exhaustive list of the public interface elements.



To obtain the list of elements, refer to the *Graphic Clients: Public Interface of Generated Components Reference Manual*.