

VisualAge Generator Templates Standard Functions



User's Guide

Version 4.5

VisualAge Generator Templates Standard Functions



User's Guide

Version 4.5

Note

Before using this document, read the general information under “Notices” on page vii.

First Edition (August 2000)

This edition applies to the following licensed programs:

- VisualAge Generator Templates Version 4.5

Order publications by phone or fax. IBM Software Manufacturing Solutions takes publication orders between 8:30 a.m. and 7:00 p.m. eastern standard time (EST). The phone number is (800) 879-2755. The fax number is (800) 284-4721.

You can also order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address below.

A form for comments appears at the back of this publication. If the form has been removed, address your comments to:

IBM SWS - Paris Laboratory
1, place Jean-Baptiste Clément
93881 Noisy-le-Grand CEDEX
France.

You can fax comments to +33 1 49 31 57 91 (provisional)

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 1997, 2000. All rights reserved.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	vii
Trademarks	ix
About this Document	xi
Conventions Used in this Book	xi
Style conventions	xi
Symbols	xi
Other VAGTemplates Documentation	xii
<hr/>	
Part 1. VAGTemplates Overview	1
Chapter 1. Introduction	3
General Presentation	3
Objectives and Use Context	5
<hr/>	
Part 2. The VAGTemplates Workbench	7
Chapter 2. The Workbench	9
General Functions	9
Accessing the Workbench	9
Defining a Workspace	10
Importing a Relational Database	10
Browsing Instances	11
Creating, Modifying or Deleting Instances	11
Accessing the Generation Tools	12
The VAGTemplates Workbench	12
Presentation of the Browser	12
File Menu	15
Workspace Menu	15
Entity Menu	36
Instance Menu	37
View Menu	45
Tools Menu	46
Options Menu	49
Help Menu	49
How to Use VAGTemplates On-Line Help	50
<hr/>	
Chapter 3. Information Model Entities and their Editors	51
Introduction	51

The Information Model Entities	52
The Definition Editor	52
The Default generation Parameters editor and the Generation Parameters Editor	53
Editor General Characteristics	54
Business Object	56
What is a Business Object?	56
Default Generation Parameters	57
How to Define a Business Object	67
How to Specify the Business Object Parameters	74
How to Specify the Business Object Extensions	75
Data Element	76
What is a Data Element?	76
Default Generation Parameters	76
How to Define a Data Element	79
How to Specify the Data Element Parameters	84
How to Specify the Data Element Extensions	85
Interface Unit	85
What is an Interface Unit?	85
Default Generation Parameters	86
How to Define an Interface Unit	89
How to Specify the Interface Unit Parameters	92
How to Specify the Interface Unit Extensions	93
Relational Table	94
What is a Relational Table?	94
Default Generation Parameters	94
How to Define a Relational Table	96
How to Specify the Relational Table Parameters	101
How to Specify the Relational Table Extensions	102
Value Style	102
What is a Value Style?	102
How to Define a Value Style	102
How to Specify the Value Style Extensions	106

Part 3. Standard Use of VAGTemplates 107

Chapter 4. Exploring VAGTemplates Basic Functions 111

Presenting a List and a Detail in the Same Window (GUI) or Map (TUI). 112

 Creating an Interface Unit Presenting a List and a Detail 114

 Defining the Interface Unit 114

 Setting the Interface Unit Generation Parameters 116

 Modifying the Business Object 116

 TUI Only: Modifying the Presentation of the Business Object 117

 TUI Only: Modifying the MainMenu Interface Unit 117

 Generating your Application 117

 Enhancing the GUI Client Application 118

 Testing your Application 119

Presenting a List and Detail in Two Different Windows (GUI) or Maps (TUI) 122

 Creating an Interface Unit Presenting the List 124

 GUI Only: Modifying the Detail Interface Unit 126

 GUI Only: Modifying the Business Object's Behavior 126

 TUI Only: Modifying the MainMenu Interface Unit 126

 Generating your Application 127

 Testing your Application 127

Generating On-Line Help (VAGTemplates on Smalltalk Example) 128

 Generating On-Line Help for the GUI Client Application 128

 Generating On-Line Help for the TUI Application 129

 Testing On-Line Help 129

 Enhancing the GUI On-Line Help (IPF file) 131

 Enhancing the GUI On-Line Help (RTF file) 135

 Enhancing the TUI On-Line Help 138

Using Foreign Keys to Provide a Help List 139

 Help List Principles 139

 Help List Specification 140

 Testing your Help List 142

Chapter 5. Standard Functions and Layouts of Generated Applications 145

Standard Functions 146

 Management of Persistent Data 147

 Error Handling in GUI Client applications 153

 Error Handling in TUI Applications 156

 Management of the Navigation 157

 On-Line Help 160

 Edition Functions (GUI) 165

 Prompt on close 167

 Windows Menu 167

 BiDi Applications 168

Standard Layouts of GUI Client applications 169

 Fields. 169

 Detail Business Objects 181

 List Business Objects 191

 Windows 196

Standard Layouts of TUI Applications 199

 Maps 199

 Fields. 212

 Detail Business Objects 217

 List Business Objects 221

Chapter 6. Application Generation and Enhancement. 225

Standard Generation 225

 List of Available Generators 225

 Instance Only / Instance Generation Option 226

 With associates/Cascaded Generation Option 231

 With Associates and Predefined Beans / Cascaded Generation With Predefined Parts Option 232

 Application Storage 232

Enhancements and Re-generation 233

 Traceability Information 234

 How the Generators Use the Traceability Information 235

 What Generator Do You Use When Re-generating 237

Generated Architecture and Principles 240

 Introduction 240

 Generated Components Naming Policy 241

 Predefined Beans/Parts 244

 Server Architecture 244

 Client Architecture 247

Overview of Generated Code 253

Servers and their Hooks	253	Application Enhancement: Public Interface	
Clients	265	of GUI Generated Components	324
Components Generated by Entities	286	Resource Object Bean/Part Interface	324
Components Generated from a Data		Business Object Bean/Part Interface	325
Element	286	List Manager Bean/Part Interface	327
Components Generated from a Business			
Object	287	<hr/>	
Components Generated from a		Part 4. Appendixes	329
Relational Table	302		
Components Generated from an		Glossary	331
Interface Unit	306		
Components Generated From a		Index	333
Workspace: Predefined Beans/Parts	312		

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 the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk NY 10504-1785, U.S.A.

Licensees 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 the information which has been exchanged, should contact the SWS General Legal Counsel, IBM Corporation, Department TL3 Building 062, P. O. Box 12195, Research Triangle Park, NC 27709-2195. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

IBM has made reasonable efforts to ensure the accuracy of the information contained in this publication. If a softcopy of this publication is provided to you with the product, you should consider the information contained in the softcopy version the most recent and most accurate. However, this publication is presented "as is" and IBM makes no warranties of any kind with respect to the contents hereof, the products listed herein, or the completeness or accuracy of this publication.

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

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

DB2

IBM

OS/2

VisualAge

The following terms are trademarks of other companies:

Microsoft, Windows, VisualBasic, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

About this Document

The abbreviated name of VisualAge Generator Templates is **VAGTemplates**.

This document is divided into three parts:

- Part 1 gives an overview of VAGTemplates.
- Part 2, *The VAGTemplates Workbench* describes the general functions of the Workbench and the entity editors.
- Part 3, *Standard Use of VAGTemplates* shows you how to modify the GUI and TUI end user interface you built following the steps described in the *Getting Started* Part in the *Introducing VisualAge Generator Templates* book or how to implement additional functionalities to the sample application. It also describes the generated application's functions, behavior and components. This part is of interest for all developers using *VAGTemplates*.
- The VisualAge Generator Templates Complete User's Guide includes a fourth part that is of particular interest for developers in charge of adapting the VAGTemplates product. It describes the generation technology and provides examples of generator customization and creation.

In this document, we assume that you are already familiar with either the VisualAge for Java or VisualAge Smalltalk Enterprise environments.

Conventions Used in this Book

We call "*Customizer*" the developer in charge of creating new generators or customizing the standard VAGTemplates Generators.

Style conventions

- References to other parts of the documentation or other VAGTemplates manuals are written in *italic*: 'For more information, refer to *Introducing VAGTemplates*'.
- Names and values of attributes and parameters are written in *italic* type.
- Names of classes are written in **bold** type.
- *Italic* type is also occasionally used for *emphasis* of words.
- New terms are shown in **bold** the first time they are used. Their definitions appear in the glossary. The terms also appear in the documentation index.

Symbols

The symbols used in this documentation are :

Note

note, remark

"" reference to another part of the documentation, or another manual

TIP

tip or helpful hint

!!! proceed with caution (risky or irreversible action, etc.)

VAGT

action to be carried out in VAGTemplates or paragraph specific to VAGTemplates

JAVA

action to be carried out in VisualAge for Java or paragraph specific to VisualAge for Java

VAST

action to be carried out in VisualAge Smalltalk Enterprise or paragraph specific to VisualAge Smalltalk Enterprise

VAG

action to be carried out in VisualAge Generator or paragraph specific to VisualAge Generator

Other VAGTemplates Documentation

- *Introducing VisualAge Generator Templates*: This book provides an overview of VAGTemplates objectives and the functions it offers. If you are new to VAGTemplates, we suggest that you read this book first in order to get an overall idea of the product.
- *VisualAge Generator Templates Customizer on Java — Reference Guide*: This document is a reference to the API of the VAGTemplates on Java components:
 - the Information Model API;
 - the Generation Framework API;
 - the generators API;
 - the APIs used when building VisualAge Generator parts and VisualAge for Java beans.
- *VisualAge Generator Templates Customizer on Smalltalk — Reference Guide*: This document is a reference to the API of the VAGTemplates on Smalltalk components:
 - the Information Model API;
 - the Generation Framework API;
 - the generators API;

- the APIs used when building VisualAge Generator and VisualAge Smalltalk Enterprise parts.

Part 1. VAGTemplates Overview

Chapter 1. Introduction

General Presentation	3	Objectives and Use Context	5
--------------------------------	---	--------------------------------------	---

General Presentation

VAGTemplates is a Rapid Architected Application Development add-on feature of VisualAge Generator.

VAGTemplates automates the development of business applications by generating application components from existing database definitions.

These definitions populate the VAGTemplates **Information Model** which allows you to formalize high-level specifications via its dedicated entities.

The integration of VisualAge Generator and VAGTemplates enables you to easily and rapidly produce a complete operational client / server application:

- VAGTemplates enables you to build GUI client components in either a VisualAge for Java or VisualAge Smalltalk Enterprise environment,
- VAGTemplates enables you to build both server and TUI client components.

Note: Instances of the Information Model entities and the VisualAge Generator application components are stored in the VisualAge Library. This central library includes advanced team development facilities such as version control, multi-user support and release management.

The generated components may be easily enhanced by adding specific business logic using VisualAge Generator. Additions to the generated components are preserved, thus providing the ability to maintain the functional specifications at the VAGTemplates Information Model level.

VAGTemplates consists of the following main components:

- a *Workbench*;
 - an *Information Model*;
 - *generators*.
1. The VAGTemplates **Workbench** is a multi-windowed graphical interface through which you can import a relational database, model your application, edit your application specifications via the editors of the Information Model entities, and activate the generation. If required, you can modify the automatically built layouts or add business logic code using the VisualAge for Java or VisualAge Smalltalk Enterprise integrated development environment.

The VAGTemplates Workbench is an integrated feature of the VisualAge for Java or VisualAge Smalltalk Enterprise workstation. You open the VAGTemplates Workbench *via* a specific menu in the VisualAge for Java Workbench or VisualAge Smalltalk Enterprise Organizer. This menu also enables you to migrate V3.0 or V3.1 specifications into V4.0 specifications.

2. The VAGTemplates **Information Model** provides you with a number of entities for specifying business applications at a logical level, i.e. independently from the physical structure of the database. This set of reusable entities provides a general formalism allowing you to describe your application in a consistent, non-redundant and systematic manner. Using the *import* function, VAGTemplates allows you to automatically integrate an existing relational database from the catalogs. The import creates instances of the Information Model entities representing relational tables and elementary data; the specification task consists of assembling application data into business objects according to your application needs. VAGTemplates supports DB2 and Oracle 7 database management systems.
3. The VAGTemplates *generators* convert the application specifications into fully operational *VisualAge Generator* Graphical User Interface (GUI), Web applications or Textual User Interface (TUI) client/server applications. The VAGTemplates' *automatic layout function* provides your applications with well-designed presentations very quickly. You only have to specify a few presentation parameters and then let the VAGTemplates' automatic layout function create template layouts for the graphic components or maps being generated. The VAGTemplates *reverse engineering* function automatically builds a generator from an application that you have developed manually using the VisualAge workstation.

VAGTemplates ensures that prototyping does not simply produce "throw-away" components but that any code generated during prototyping remains part of the application as it develops.

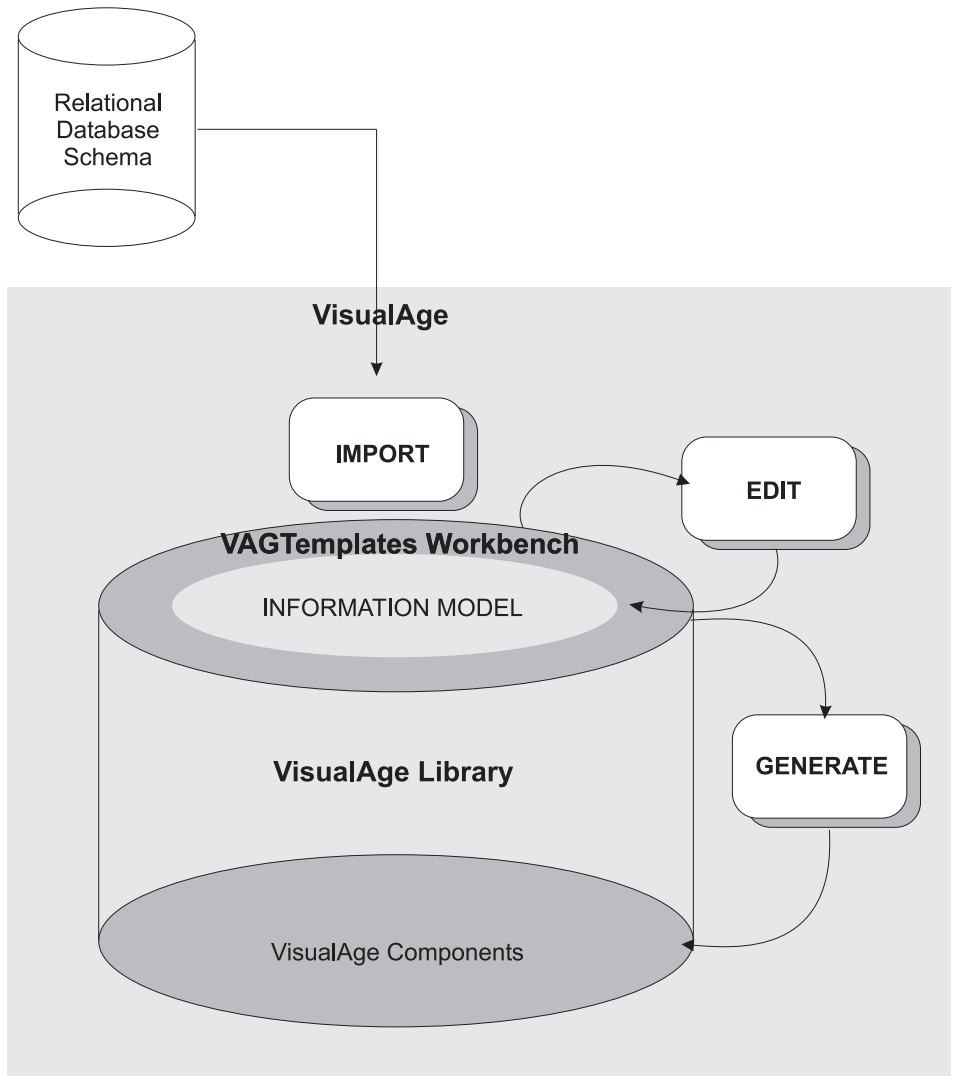


Figure 1. VAGTEMPLATES OVERVIEW

Objectives and Use Context

The objective of this book is to give you a complete understanding of the development concepts used in the VAGTemplates GUI/TUI application specification and generation facility.

1. The first part of this document presents the VAGTemplates Workbench and its tools.

2. The second part describes the standard use of the Workbench.
 - It shows you how you can modify the behavior and look of the sample application you built in the *Introducing VisualAge Generator Templates* book.
 - “Chapter 5. Standard Functions and Layouts of Generated Applications” on page 145 gives you an illustrated documentation on the functions and behavior of the generated applications.
 - “Chapter 6. Application Generation and Enhancement” on page 225 explains the generation principle and provides a description of the generated components and their role within the generated application.

We have chosen not to dissociate the GUI, Web and TUI functions provided by VAGTemplates as the product itself is built on reusability. The functional aspect of the Information Model is common to GUI, Web, and TUI applications. The parameterizing aspect takes into account the specificity of the GUIs and TUIs and that of the Web applications, when needed.

Part 2. The VAGTemplates Workbench

Chapter 2. The Workbench	9	Instance menu / Set Generation	
General Functions	9	Parameters to Default choice	40
Accessing the Workbench	9	Instance menu / Move ... choice	40
Starting the Browser	9	Instance menu / Copy choice	40
Closing the Browser	10	Instance menu / Delete choice.	40
Defining a Workspace	10	Instance menu / Create Business	
Importing a Relational Database	10	Object from RT ... choice	40
Browsing Instances	11	Instance menu / Create Interface Unit	
Creating, Modifying or Deleting		from BO... choice	41
Instances	11	Instance menu / Generate choice	41
Accessing the Generation Tools	12	View Menu	45
The VAGTemplates Workbench	12	View menu / Refresh Now choice	45
Presentation of the Browser	12	View menu / Select All choice.	45
File Menu	15	View menu / Deselect All choice.	45
File menu / Exit VisualAge Templates		View menu / Sort by choice	45
choice	15	View menu / Reorder Columns choice	45
Workspace Menu	15	View menu / Reorder Status Bar Text	
Workspace menu / Open... choice	15	choice	45
Workspace menu / Close choice	15	Tools Menu	46
Workspace menu / Load choice	16	Tools menu / Import from Database	
Workspace menu / Editions choice	16	choice	46
Workspace menu / Move... choice	16	Tools menu / Show Duplicate	
Workspace menu / Copy ... choice	16	Instances choice	49
Workspace menu / Delete ... choice	16	Options Menu	49
Workspace menu / Generate choice	16	Options menu / Save Settings as	
Workspace menu / Definition choice	20	Default choice	49
Entity Menu	36	Help Menu	49
Entity menu / Default Generation		Help menu / Help Index choice	49
Parameters choice	36	Help menu / General Help choice	49
Entity menu / Show Duplicate		Help menu / Using Help choice	49
Generation Parameters choice	37	Help menu / Product Information	
Instance Menu	37	choice	49
Instance menu / New ... choice	37	How to Use VAGTemplates On-Line Help	50
Instance menu / Definition choice	38		
Instance menu / Generation		Chapter 3. Information Model Entities and	
Parameters choice	38	their Editors	51
Instance menu / Load choice	38	Introduction	51
Instance menu / Editions choice	39	The Information Model Entities	52
Instance menu / References ... choice	39	The Definition Editor	52
Instance menu / Associates choice	39	The Default generation Parameters editor	
Instance menu / Referenced		and the Generation Parameters Editor	53
Workspaces choice.	39	Editor General Characteristics.	54
Instance menu / Copy Generation		Navigation within an Editor	54
Parameters from... choice	39	Push Buttons	55
		Pop-up Menus	56

Editor Extensions Panels	56	How to Specify the Value Style	
Business Object	56	Extensions	106
What is a Business Object?.	56		
Default Generation Parameters	57		
Default Generation Parameters Editor	57		
How to Define a Business Object	67		
Definition Editor	68		
How to Specify the Business Object			
Parameters	74		
Generation Parameters Editor	75		
How to Specify the Business Object			
Extensions	75		
Data Element	76		
What is a Data Element?	76		
Default Generation Parameters	76		
Default Generation Parameters Editor	77		
How to Define a Data Element	79		
Definition Editor	80		
How to Specify the Data Element			
Parameters	84		
Generation Parameters Editor	84		
How to Specify the Data Element			
Extensions	85		
Interface Unit	85		
What is an Interface Unit?	85		
Default Generation Parameters	86		
Default Generation Parameters Editor	86		
How to Define an Interface Unit	89		
Definition Editor	89		
How to Specify the Interface Unit			
Parameters	92		
Generation Parameters Editor	93		
How to Specify the Interface Unit			
Extensions	93		
Relational Table	94		
What is a Relational Table?	94		
Default Generation Parameters	94		
Default Generation Parameters Editor	94		
How to Define a Relational Table	96		
Definition Editor	96		
How to Specify the Relational Table			
Parameters	101		
Generation Parameters Editor	101		
How to Specify the Relational Table			
Extensions	102		
Value Style	102		
What is a Value Style?	102		
How to Define a Value Style.	102		
Definition Editor	103		

Chapter 2. The Workbench

General Functions

VAGTemplates is a feature of *VisualAge Generator* 4.0. Once VAGTemplates is loaded into your VisualAge for Java or VisualAge Smalltalk Enterprise image, the VAGTemplates Workbench is integrated to the corresponding VisualAge workstation.

- You can access the VAGTemplates on Java functions *via* the **Workspace** menu from the VisualAge for Java Workbench.
 - The **Open VAGTemplates Browser** choice opens the VAGTemplates Browser.
 - The **VAGT Import/Export** function is documented in the *VisualAge Generator Migration Guide*.
- You can access the VAGTemplates on Smalltalk functions *via* the **VAGT tools** submenu from the VisualAge Smalltalk Enterprise **Tools** menu.
 - The **Open VAGTemplates Browser** choice opens the VAGTemplates Browser.
 - The **VAGT Import/Export** function is documented in the *VisualAge Generator Migration Guide*.

The Workbench allows you:

- to *view existing entity instances*;
- to *create, modify or delete entity instances*;
- to *activate the generation tools*.

Accessing the Workbench

Starting the Browser

For VAGTemplates on Java, you access VAGTemplates by selecting the **Open VAGTemplates Browser** choice from the **Workspace** menu in the VisualAge for Java Workbench.

For VAGTemplates on Smalltalk, you access VAGTemplates *via* the **Open VAGTemplates Browser** choice from the **VAGT tools** submenu in the VisualAge Smalltalk Enterprise Organizer **Tools** menu.

The first time you start VAGTemplates, the *Select workspace* window automatically opens and prompts you to choose between three options by checking the corresponding radio-buttons:

- *Create a new workspace*

You need to specify a name for the Workspace and the package/application where it will be stored, then click *OK* to open the VAGTemplates Browser.

- *Open an existing workspace*

You need to select a Workspace in the list and click *OK* to open the VAGTemplates Browser.

- *Do not open a workspace (edit functional properties only)*

You just need to click *OK* to open the VAGTemplates Workbench. Selecting this option prevents you from accessing the entity Generation Parameters editors. You can only edit the functional description of your application using the entity Definition editors.

Note: Uncheck the *Show this window at startup* option if you do not want the *Select workspace* window to open the next time you start VAGTemplates.

Closing the Browser

To close the Browser, double-click on the system menu. A confirmation message appears. When you confirm the close request, the Browser closes along with all the editor panels that may still be open.

Defining a Workspace

Defining a Workspace is required:

- to import a relational database into VAGTemplates

For more information, see “**Importing a Relational Database**”.

- to define the generation parameters you need to generate your application

A Workspace gathers the following generation parameters:

- *Generation options* shared by *all* entities to be generated from the Workspace, for example the naming policy of the generated components
- *Default generation parameters* specified for each entity
- *Default generation parameters* specified for each instance
- *Parameter extensions* that can be defined as shared by all Workspaces or specific to a given Workspace

For more information on how to define a Workspace, refer to “**Workspace Menu**” on page 15.

Importing a Relational Database

The import function allows you to use the description of one or several relational database to build your application. To import a relational database into VAGTemplates, a Workspace must be defined: a database import is always made in relation to one Workspace. The import of the relational database into VAGTemplates automatically creates instances of both the Relational Table and Data Element entities.

In addition, some information is automatically imported in the current Workspace as generation parameters. For example, the Data Element *Label* input field is automatically filled in with the corresponding information from the database.

The Relational Table and Data Element entities are described in “Chapter 3. Information Model Entities and their Editors” on page 51.

Caution: You can import two databases in the same Workspace, but keep in mind that if two tables have the same name the second imported table will override the first one. No standard function allows access to more than one database in the same generated application.

Browsing Instances

The *Instances* area displays all the existing instances for one or more entities selected in the *Entities* area. For each instance in the list, the following information can be displayed:

- the icon representing the entity
- the instance name

The instance name is a character string that designates one and only one instance among all other instances of the entity.

- the instance loaded edition
- the instance display name
- the instance target package (VAGTemplates on Java) or the instance target application (VAGTemplates on Smalltalk)
- the instance Definition package/application
- the instance Generation Parameters package/application
- the instance target name
- the instance long target name

Note:

You can select one or more instances in the list of instances.

Creating, Modifying or Deleting Instances

You will use the Browser to create new entity instances, define and update their contents or delete them:

- *Creating an instance:* you assign it a name, and a package or application where it will be stored;
- *Defining and updating the contents of an instance:* after creating the instance, you can define its functional characteristics and generation parameters. To do so you will respectively access the Definition editor and the Generation Parameters editor that are specific to each instance entity type:

- Business Object editor;
 - Data Element editor;
 - Interface Unit editor;
 - Relational Table editor;
 - Value Style editor.
- *Deleting an instance:* a deletion removes all the descriptions of the instance.

These editors are described in “Chapter 3. Information Model Entities and their Editors” on page 51.

Accessing the Generation Tools

The Workbench allows you to generate entity instances.

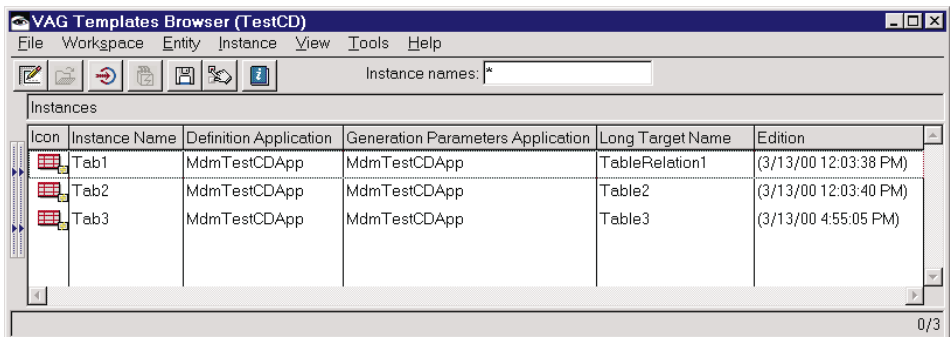
You can generate your instance with the entity specific generator or you can generate the instance and all the instances to which it is linked.

The generation is described in “Part 3. Standard Use of VAGTemplates” on page 107, “Chapter 6. Application Generation and Enhancement” on page 225 and you may practice with the examples given in “Chapter 4. Exploring VAGTemplates Basic Functions” on page 111.

The VAGTemplates Workbench

When we come to describing the specification fields, the settings and parameters that only apply to GUI client applications are highlighted with this *GUI* sign; those that only apply to TUI applications with the *TUI* sign. No sign means that the settings and parameters are common to both TUI and GUI client applications.

Presentation of the Browser



The Browser is composed of the following elements:

1. The *title bar*.

The Browser *title bar* displays the "VAG Templates Browser" label followed by the name of the current Workspace in brackets.

2. The *menu bar*.

It is described in detail in the following sections.

3. The *action bar*.

The *action bar* is the set of buttons placed below the menu bar. It displays icons representing frequently performed actions. The icons have hover help to display action names.

4. The *Entities* area.

The *Entities* area displays the list of the Information Model entities you manipulate when developing an application. They are described in detail in "Chapter 3. Information Model Entities and their Editors" on page 51.

These entities are:

- Business Object
- Data Element
- Interface Unit
- Relational Table
- Value Style

When you select one or more entities in the list, the instances that are defined for these entities are displayed in the *Instances* area.

5. The *Instances* area.

The *Instances* area displays the list of all the instances that are defined for the selected entities. Once you select an instance, you can consult or update its description *via* the Definition or Generation Parameters editors, and generate it.

Note: You can also select several instances at the same time by using the Shift and Ctrl keys and selecting instances with your mouse. You can then open all at once the Definition editors or the Generation Parameters editors for the selected instances.

The small square symbol on the bottom right of an instance icon indicates that generation parameters have been redefined for this instance in the current Workspace.

The Browser as well as all VAGTemplates editors have the following properties:

- they have a *system menu*;
- they can be moved by dragging their title bar or selecting the *Move* choice from the system menu;

- they can be reduced to an icon *via* the *Minimize* button or the *Reduce* choice from the *System* menu;
- they can be maximized *via* the *Maximize* button or the *Maximize* choice from the *System* menu ;
- they can be resized via the zoom button or the *Resize* choice from the *System* menu.

You can activate *pop-up menus* anywhere in the Browser.

Select an entity and right click in the *Entities* area. A pop-up menu is displayed offering the following choices:

- *Default Generation Parameters*

TIP: See also the *Entity* menu in “**Entity Menu**” on page 36.

Select an instance and right click in the *Instances* area. A pop-up menu is displayed offering the following choices:

- *New...*
- *Definition*
- *Generation Parameters*
- *Load*
 - >*Previous Edition*
 - >*Another Edition*
- *Editions*
- *References*
- *Associates*
- *Referenced Workspaces*
- *Copy Generation Parameters from...*
- *Set Generation Parameters to Default*
- *Move...*
- *Copy...*
- *Delete*
- *Create Business Object from RT...*
- *Create Interface Unit from BO...*
- *Generate...*

TIP: These choices can also be activated from the *Instance* menu of the menu bar, as described in “**Instance Menu**” on page 37.

File Menu

The File menu allows you to close VAGTemplates Workbench.

Table 1. File Menu

Exit VisualAge Templates...

File menu / Exit VisualAge Templates choice

Allows you to exit the VAGTemplates Workbench.

Workspace Menu

The Workspace menu allows you to choose the Workspace that you want to access, to define various parameters for this Workspace and to save your settings.

Table 2. Workspace Menu

Open

Close

Load

> Previous edition

> Another edition

Editions...

Move...

Copy...

Delete...

Generate

Definition

Workspace menu / Open... choice

Opens the *Open workspace* window which enables you to create a Workspace or select an existing Workspace from the list box.

- To create a Workspace
 1. select the *Create a new workspace* radio-button
 2. enter a name for the Workspace in the edit field
 3. specify a package or an application to store your Workspace: select an existing package or application in the combo box or enter a new name
 4. click *OK* to create the package/application and open the Workspace.
- To open an existing Workspace
 1. select the *Open an existing workspace* radio-button
 2. select a Workspace in the list
 3. click *OK* to open the Workspace.

Workspace menu / Close choice

This choice enables you to close the current Workspace.

Workspace menu / Load choice

This choice enables you to load an edition of a Workspace.

Previous Edition: This choice enables you to load the previous edition of the selected Workspace.

Another Edition: This choice enables you to load another edition of the selected Workspace. You must select a Workspace in the list that is displayed.

Workspace menu / Editions choice

This choice displays the list of the selected Workspace's editions.

Workspace menu / Move... choice

This choice enables you to move the current Workspace into another package or application. When selecting the instance you wish to move, a window with a multi-line edit appears displaying the available packages or applications where the selected Workspace instance can be moved.

Workspace menu / Copy ... choice

Opens the *Copy Workspace* window that enables you to save the contents of the current Workspace into another Workspace.

The *New workspace name* field allows you to enter the new name of the Workspace.

The *Existing workspace names* list allows you to select an existing Workspace to save your current Workspace as the selected one, and therefore replace the specifications of the latter with those of the current one.

The *Select package/application* combo box allows you to select a package or an application in which to store the new Workspace or to enter a new package or application name.

Workspace menu / Delete ... choice

Opens the *Delete Workspace(s)* window allowing you to select the Workspace you want to delete from the list. Click *OK* to delete the selected Workspace.

Workspace menu / Generate choice

VAGTemplates on Java: The *Generate* choice opens the *Generate (instance)* window that enables you to generate the application components for the selected Workspace.

Select available generator(s): This area lists the generators available for the selected entity instances. At least one generator must be selected in the list.

Instance(s) to be generated: This area displays the current Workspace instance.

Store Options:

Description of the options: The *Store Options* area allows you to specify a store option for the *builders* instantiated by the generation process.

Note: A *builder* is a Java class instance produced by the generation process. These classes model VisualAge for Java components, manages the store and load processes of the generated components into the VisualAge for Java Library and manages traceability information. Each instantiated builder, once stored, will correspond to one component in the VisualAge for Java Library.

Normal

This option stores the new specifications, preserve the RAD specifications and those that have been added using the VisualAge for Java environment.

Overwrite

This option is identical to the *Normal* option, but additionally forces the storage of RAD specifications.

Reset

This option is identical to the *Overwrite* option, but deletes the specifications added by the user.

Customize

When this mode is checked, the builders are instantiated but not immediately stored as Java components. When all builders are generated, the *Customize Generation* window opens so that you can customize some storage parameters for the Java components to be produced from these builders.

Customize Generation window: This window contains a table in which each row corresponds to one generated builder. The builder table can be customized by:

- reordering the columns by drag and drop
- showing/hiding a column by right clicking once on a column header
- resizing a column
- sorting the table using any column as sort criterion
 - To sort a column in increasing alphabetical order, left click on a column header.
 - To sort a column in decreasing alphabetical order, press the Shift key and left click on a column header.

For each builder, the following information can be displayed in the corresponding columns:

- The first column contains check boxes allowing you to confirm whether you want the builder to be stored. These check boxes are designated below as the *store management* fields. A builder will be stored only if the corresponding *store management* field is checked.
- The *Type* column displays the builder type.
- The *Name* column displays the builder name.
- The *Store Option* column allows you to modify the store option. Left click on the cell corresponding to the builder, then select a store option in the combo box.
- The *Trace Category* column allows you to modify the trace category of generated builder. Left click on the cell corresponding to the builder, then select a trace category in the combo box.
- The *Context* column displays the name of the target project or package where the generated builder will be stored.
 - If the generated builder is a project, it will directly be stored in the VisualAge for Java Library. The *Context* column is empty.
 - A package is stored in a project. The *Context* column displays the target project name.
 - A class is stored in a package. The *Context* column displays the target package name.

The table also contains two hidden columns. To view them, right click in the table and select *Show all columns* from the pop-up menu.

- The *VAGT name* column displays the name of the VAGTemplates instance from which the builder has been generated.
- The *VAGT type* column displays the type of this instance.

Customize Generation window Pop-Up menu: The *Customize Generation* window provides a pop-up menu which allows you to handle several builders at the same time, show all columns and restore the initial display of the builder table if it has been customized.

Note: To add a builder to the selection, press Ctrl key and right click on the row you want to add.

Mark

This choice allows you to check at once the *store management* fields of a set of selected builders. These builders will be stored as Java components in the VisualAge for Java Library.

Unmark

This choice allows you to uncheck at once the *store management* fields of a set of selected builders. These builders will not be stored.

Store Option

This choice opens the *Store Options* dialog allowing you to modify at once the store options for a set of builders. Check the radio-button corresponding to the desired option, then click *OK*. This option is applied to all builders you selected.

Trace Category

This choice opens the *Trace Category* dialog allowing you to modify at once the trace categories for a set of builders. Check the radio-button corresponding to the desired trace category, then click *OK*. This trace category is applied to all builders you selected.

Mark all

This choice allows you to check the store management fields of all generated builders in order to store them.

Unmark all

This choice allows you to uncheck the store management fields of all generated builders in order not to store them.

Show all columns

This choice allows you to view all the columns of the builder table.

Reset

This choice allows you to restore the initial display of the table column.

Other Functionalities: You can get help from the *Customize Generation* window by clicking on the *Help* push-button.

The *OK* button closes the *Customize Generation* window and triggers the storage of the generated builders.

Generation Scope: The *Instance only* option which allows you to generate the Workspace components, that is the predefined beans, is automatically selected. The *With associates* and *With associates and predefined beans* options are not available for a Workspace instance generation.

Note: If you use a generator more often than the others, you can select the *Save as default* check box, which will keep your generator selected for the next time you generate.

Client & Server:

Visuals

Generates the visual part of the application.

Client

Generates the client part of the application.

Server

Generates the server part of the application.

For details on the generation, refer to “Part 3. Standard Use of VAGTemplates” on page 107, “Chapter 6. Application Generation and Enhancement” on page 225, “**Standard Generation**” on page 225; for information on the predefined beans, refer to “**Generated Architecture and Principles**” on page 240, “**Components Generated From a Workspace: Predefined Beans/Parts**” on page 312 in the same chapter.

VAGTemplates on Smalltalk: The *Generate* choice opens the *Generate (instance)* window that enables you to generate the application components for the selected Workspace.

The *instance generation* option which allows you to generate the Workspace components, that is the predefined parts, is automatically selected. The *cascaded generation* and *cascaded generation with predefined parts* options are not available for a Workspace instance generation.

Select the *Override existing parts* check box so that the generated parts override the parts previously generated.

If you use a generator more often than the others, you can select the *Save as default* check box, which will keep your generator selected for the next time you generate.

For details on the generation, refer to “Part 3. Standard Use of VAGTemplates” on page 107, “Chapter 6. Application Generation and Enhancement” on page 225, “**Standard Generation**” on page 225; for information on the predefined parts, refer to “**Generated Architecture and Principles**” on page 240, “**Components Generated From a Workspace: Predefined Beans/Parts**” on page 312 in the same chapter.

Workspace menu / Definition choice

Opens the Workspace editor allowing you to customize the current Workspace and define generation parameters.

General Panel: The settings in the *General* panel enable you to define generation preferences and take particular target tool’s requirements into account.

Package/Application:

Package/Application

This area indicates the name of the package or application where all the specifications are stored.

Default value: The package or application where the Workspace is stored.

Names:

Display name

The field allows you to enter a display name for the Workspace. The display name is used in the list of called instances in help panels.

Common Parameters:

Target name

This field allows you to enter a character string that will be used as the prefix of the generated predefined beans/parts.

Default value: name of the Workspace truncated to 5 characters

You will find a description of the predefined beans/parts in “Part 3. Standard Use of VAGTemplates” on page 107, chapter “Chapter 6. Application Generation and Enhancement” on page 225, “**Generated Architecture and Principles**” on page 240, “**Components Generated From a Workspace: Predefined Beans/Parts**” on page 312.

Long target name

This field allows you to enter a character string (64 max.) that will be used as prefix of the generated Java and Smalltalk classes.

Default value: name of the Workspace

Generation Parameters:

Generation directory

This field is used to specify the directory where all the generated help files will be stored.

Default value: c:\VAGT\build

DBMS

Select a DBMS in the dropdown list.

Target Packages/Applications Panel: This panel allows you to specify the project (VAGTemplates on Java only) and the packages/applications where the components generated from the Workspace will be stored.

Target Packages/Applications:

JAVA Target project

This field is used to specify the name of the project that includes all the target packages associated with the Workspace.

Visual package/application

This field is used to indicate the package/application where the generated visual components will be stored.

Logic package/application

This field is used to indicate the package/application where the generated logic components will be stored.

Services package/application

This field is used to indicate the package/application where the generated service components will be stored.

Client Panel: This panel groups the parameters that define the general layout of the generated application's user interface.

User Interface:

Normal color

This drop-down list enables you to specify the color of fields that are in a valid state.

Default value: light yellow

Error color

This drop-down list enables you to specify the color of the fields in error.

Default value: red

Read only color

This drop-down list allows you to specify the color of read-only fields.

Default value: grey

Note: Some colors cannot be displayed in TUI applications: *light yellow* is changed into yellow in the final application, and grey into the standard color of the system where the application runs.

Horizontal orientation

The *Horizontal orientation* area enables you to define the orientation of the user interface. You can choose between left to right and right to left. You can define the horizontal orientation for a Workspace or for a Data Element.

Default value: left to right

GUI Panel: This panel groups the parameters that manage GUI client application behavior.

Layout Setup:

Horizontal margin

This drop-down list allows you to specify the horizontal space between the generated components in a layout.

Default value: medium

Vertical margin

The drop-down list allows you to specify the vertical space between the generated components in a layout.

Default value: medium

Label to value horizontal gap

This drop-down list allows you to specify the horizontal space between the labels and their values when presented horizontally.

Default value: medium

Label to value vertical gap

This drop-down list allows you to specify the vertical space between the labels and their values when presented vertically.

Default value: medium

Letter width

This drop-down list allows you to specify the character reference width.

Default value: average

Fields selected on focus

This checkbox enables you to specify whether the contents of a field are selected when the field gets the focus.

Default value: checked

Screen resolution

This field allows you to specify the resolution of the screen where the generated application will run.

Default value: 1024 x 768

Help Management:

General Help File

This field is used to specify the name of the file where the generated on-line help will be stored. The extension (.IPF, .RTF, or .HPJ) will be added automatically according to the target environment (OS/2 or Microsoft Windows).

Default value: HELP

Update Policy:

Update GUI policy

This drop-down list enables you to choose whether to provide your generated applications with two actions (*create/update*) - one creating rows and one updating rows - or with a single action (*save*) that handles both create and update functions.

Default value: create/update

Web Panel:

Application Pattern:

Web application pattern

This parameter allows to choose between two ways of transferring data to/from the browser.

Default value: Xfer

For more details, see “**Web Client**” on page 247.

TUI Panel:

Error Management:

Max navigation stack number

This spin edit enables you to specify the maximum number of navigation paths that are memorized. These paths are used to know the location of each map in the navigation tree. When the maximum number is reached, the navigation stack is re-initialized and the navigation paths and parameters are lost.

The navigation paths are used by the application to navigate from error messages to the maps where errors have been detected, for example.

Default value: 50

Messages per page

This spin edit enables you to specify the maximum number of error messages that will be displayed on each “page” in the error message map. To see the following messages, the end user can scroll down the list with the arrow keys.

Default value: 4

Note: This parameter is relevant if the *Messages display* attribute is set to *specific map* (see below).

Messages display

This spin edit allows you to specify if error messages are displayed on a separate map in a scrollable list (*specific map*), or on one line at the bottom

of the current map (*current map*). In the latter case, the total number of error messages is displayed; the end user will have to trigger an action to reach the following messages.

Default value: specific map

For information on error message management, refer to “Part 3. Standard Use of VAGTemplates” on page 107, “Chapter 6. Application Generation and Enhancement” on page 225, “**Components Generated From a Workspace: Predefined Beans/Parts**” on page 312.

User Interface:

Update TUI policy

This drop-down list enables you to specify whether updates are taken into account when the end user explicitly requests them by performing an action (explicit), or by pressing ENTER (implicit).

Default value: explicit

Standard TUI device

This drop-down list enables you to specify the output display device (printer, touch-sensitive screen, etc.). This affects the number of rows and columns that are available in the map.

Default value: 3278-2

Note: The parameter values are the IBM values (see your IBM documentation).

Entry default value

This drop-down list enables you to specify the character that will mark the fields where values can be entered.

Default value: underscore

Display popup

This checkbox enables you to specify whether help panels, help lists, value table lists and messages are laid out in a frame displayed within the current map.

Default value: checked

Function Keys Panel: This panel groups the parameters that allow you to customize the ergonomics of function keys and action codes in TUI applications.

Display Policies:

Display function keys

This checkbox enables you to specify whether the function keys will be displayed on the generated map or not. In the latter case, the function keys can still be used.

Default value: checked

Display actions

This checkbox enables you to specify whether the actions will be displayed on the generated map or not. In the latter case, the actions can still be used.

Default value: checked

Function Keys:

Help

These fields allow you to select the function key that will be used to call the on-line help, and to assign a label to it.

Default value: 01 HELP

Exit

These fields allow you to select the function key that will be used to exit the current map and return to the previous one, and to assign a label to it.

Default value: 03 EXIT

Lookup

These fields allow you to select the function key that will be used to display help lists, and to assign a label to it.

Default value: 04 LOOKUP

Cancel

These fields allow you to select the function key that will be used to cancel an action and return to the parent map, and to assign a label to it.

Default value: 12 CANCEL

Left

These fields allow you to select the function key that will be used to navigate towards the left map, and to assign a label to it.

Default value: 05 LEFT

Right

These fields allow you to select the function key that will be used to navigate towards the right map, and to assign a label to it.

Default value: 06 RIGHT

Previous page

The fields allow you to select the function key that will be used to navigate upwards to the previous page in a list, and to assign a label to it.

Default value: 07 PREV

Next page

These fields allow you to select the function key that will be used to navigate downwards to the next page in a list, and to assign a label to it.

Default value: 08 NEXT

Top

These fields allow you to select the function key that will be used to reach the first data in a list, and to assign a label to it.

Default value: 17 TOP

Refresh

These fields allow you to select the function key that will be used to retrieve the data as it was before the last action was triggered, and to assign a label to it.

Default value: 09 UNDO

Create

These fields allow you to select the function key that will be used to create a new row in the database, and to assign a label to it.

Default value: 13 INSERT

Note: This parameter is ignored if the *Update policy* parameter is set to *implicit*.

Read

The fields allow you to select the function key that will be used to access data, and to assign a label to it.

Default value: 14 SELECT

Update

These fields allow you to select the function key that will be used to update data, and to assign a label to it.

Default value: 15 UPDATE

Note: This parameter is ignored if the *Update policy* parameter is set to *implicit*.

Delete

These fields allow you to select the function key that will be used to delete data, and to assign a label to it.

Default value: 16 DELETE

Previous message

These fields allow you to select the function key that will be used to reach the previous error messages in the current map, and to assign it a label.

Default value: 10 MSG -

Note: This parameter is relevant when the *Message display* parameter is set to *current map*; otherwise it is ignored.

Next message

These fields allow you to select the function key that will be used to reach the next error messages in the current map, and to assign a label to it.

Default value: 11 MSG +

Note: This parameter is relevant when the *Message display* parameter is set to *current map*; otherwise it is ignored.

For information on these actions, refer to “Part 3. Standard Use of VAGTemplates” on page 107, “Chapter 5. Standard Functions and Layouts of Generated Applications” on page 145, “**Standard Functions**” on page 146.

Colors Panel: This panel groups the parameters that manage the ergonomics of TUI applications.

TUI Colors:

Function key label color

This drop-down list allows you to specify the color of the function key labels.

Default value: turquoise

Title color

This drop-down list allows you to specify the color of the map titles.

Default value: white

Error message color

This drop-down list allows you to specify the color of the error messages.

Default value: red

Warning message color

This drop-down list allows you to specify the color of the warning messages.

Default value: pink

Information message color

This drop-down list allows you to specify the color of the information messages.

Default value: yellow

Help text color

This drop-down list allows you to specify the color of the help texts.

Default value: white

Server Panel:

Server:

Common Area

This drop-down list allows you to specify whether or not you want the communication Record and its associated Table to be generated.

Default value: record and table

For information on the communication Record, refer to “Part 3. Standard Use of VAGTemplates” on page 107, “Chapter 6. Application Generation and Enhancement” on page 225, “**Generated Architecture and Principles**” on page 240, “**Components Generated From a Workspace: Predefined Beans/Parts**” on page 312, “**GUI and TUI Components**” on page 313.

Server layers

This drop-down list allows you to specify the type of client/server architecture you want for the generated application: *two-tier*, the architecture is based on a client and a server; *three-tier*, the architecture is based on a client, an umbrella server and as many atomic servers as actions.

Default value: two-tier

Null managed

This checkbox enables you to specify whether the null value will be globally managed for all the fields in the application or only for the fields that accept the null value in the database.

Default value: checked

Internal Formats:

Decimal separator

This field enables you to specify the character that will be used to separate the integer part from the decimal part of an integer when the value is stored in the database.

Default value: . (period)

Date Format

This drop-down list allows you to choose whether the internal format for storing dates in the database is *dmy* (<Day> <Month> <Year>), *mdy* (<Month> <Day> <Year>), or *ymd* (<Year> <Month> <Day>).

Default value: ymd

Caution: For VAGTemplates on Smalltalk, make sure that the value you choose here is compatible with the date and time format parameters in the VisualAge Smalltalk Enterprise abt.ini file.

Extractions Panel: This panel groups parameters that allow you to adapt the generation to VisualAge Generator requirements.

Alphanumeric Extractions:

SQL High value

This field allows you to specify the SQL high value that will be used when extracting alphanumeric or numeric values using the "less than" or "greater than" extraction operator.

Default value: Z

SQL Low value

This field allows you to specify the SQL low value that will be used when extracting alphanumeric or numeric values using the "less than" or "greater than" extraction operator.

Default value: <no default value

The push-button on the right enables you to retrieve from the Database the SQL high values and the SQL low values of the alphanumeric extractions corresponding to the current configuration of the database in use.

This push-button first opens the *Settings* dialog box. Once you closed this dialog box, the *Extraction Results* window opens up and prompts you to confirm that you want the Workspace alphanumeric extractions to be updated.

Date Extractions:

SQL High value

This field allows you to specify the SQL high value that will be used when extracting dates using the "less than" or "greater than" extraction operator.

Default value: 12-31-2999

SQL Low value

This field allows you to specify the SQL low value that will be used when extracting dates using the "less than" or "greater than" extraction operator.

Default value: 01-01-1901

Timestamp Extractions:

SQL High value

This field allows you to specify the SQL high value that will be used when extracting timestamps using the "less than" or "greater than" extraction operator.

Default value: 2999-12-31-00.00.00.000000

SQL Low value

This field allows you to specify the SQL low value that will be used when extracting timestamps using the "less than" or "greater than" extraction operator.

Default value: 1901-01-01-00.00.00.000000

DBCS Extractions: This area groups parameters that enable you to adapt the generation to DBCS configuration.

SQL high value

This field allows you to specify the SQL high value that will be used when extracting DBCS values using the "less than" or "greater than" extraction operator.

Default value: <no default value>

SQL low value

This field allows you to specify the SQL low value that will be used when extracting DBCS values using the "less than" or "greater than" extraction operator.

Default value: <no default value>

Error Management Panel:

Server Error Management:

Max number of messages

This spin edit allows you to specify the maximum number of stored error messages to be displayed in the error window (GUI) or map (TUI).

Default value: 10

Max number of variables

This spin edit allows you to specify the maximum number of variables that can be used to build the error messages.

Default value: 3

Max size of variables

This spin edit allows you to specify the maximum size that a variable can have in the error message.

Default value: 30

For example, if you want to include Data Element names in your error messages, you should set this parameter to 30 since Data Element names can be up to 30 characters long.

Client/Server Control Panel: This panel groups the parameters that define the controls implemented in the generated application.

Client/Server Control:

Control location

This drop-down list allows you to specify whether the checks are managed by the Client, the Server, or both by the Client and Server.

Default value: client and server

For example, assuming that you specified a check by value table on a Data Element, if this parameter is set to client, the check will be performed on the client only; the end user will be able to enter any value that is not in the value table. If you set this parameter to server or client and server, the check will be performed on the server and only the values that are defined in the value table will be authorized.

Depending on this parameter value, the controls are generated as follows:

	GUI Generation		WEB/TUI Generation	
	Client	Server	Client	Server
<i>Controls on value type and format</i>	Controls on client		Controls on client (numeric data)	
<i>Controls on value presence for required field</i>	Controls on client	Database	Controls on client	Database
<i>Controls on value in defined interval or value table</i>	Controls on client	Controls on server	Controls on client	Controls on server

Note: When the *Control location* parameter is set to *Server* or *Client and server*, the controls on the value presence are implemented by both the generated servers and the database.

See also “**Error Handling in GUI Client applications**” on page 153 and “**Error Handling in TUI Applications**” on page 156.

LUW mode

This drop-down list allows you to specify whether the commit is managed by the Client or by the Server.

Default value: server

Note: The commit is only performed when the end user navigates from one map group to the other, not among the maps of the same map group.

For example, let us assume that the Staff Business Object displayed on two maps because it is too large to be displayed on one. If the end user modifies the value of the Name field in map 1 and triggers the update action, the modification will not be taken into account in the database, even when he/she moves to the second map. Whereas if he/she returns to the main menu of the application, the modification is stored in the database.

Client business logic style

This drop-down list allows you to specify whether the generators have to generate the Business Objects with VisualAge Generator records or not. It enables you to customize the generated parts.

Default value: default

Information messages

This drop-down list enables you to handle information messages that will be displayed within the interface in an optional information bar. You can raise messages, that will not stop the execution of the current action, and will be displayed within the interface in an optional information bar.

- **Raising a message on the server**

There is a dedicated section in the XX-ERROR-LST Record: Applicative-Information contains info-code and 3 info-variables.

The code is similar to the one used in the errors, translated according to the MSGTBL table. Control-information is a flag indicating whether a message has been raised or not. Messages can be raised in the hooks of the server. Messages are caught on the client. They are displayed in the optional information bar.

- **Raising a message on the client**

If the value of this parameter is set to *bar with standard messages*, the message from the server will be displayed in an information bar. If no message has been raised by the server during an action, the client will raise a message indicating that the action has succeeded. This message is reset when another action starts. If you specify the parameter with the value *no standard messages*, the client will never raise any message. And the information bar is not displayed at the bottom of the windows. However, the API to raise, propagate and translate messages is still present and can be used.

Default value: bar with standard messages

Naming Rules Panel: This panel groups parameters that manage the naming rules for the generated parts.

Naming Policy: The *identifier first* and *type first* radio-buttons allow you to specify whether you want the names of the generated parts to begin with the identifier or with the type of the instance.

Default value: identifier first

Source Entities:

Workspace

This field allows you to specify a mnemonic that will be used to name the parts that are generated from a Workspace instance.

Default value: S

Interface unit

This field allows you to specify a mnemonic that will be used to name the parts generated from an Interface Unit instance.

Default value: W

Business object

This field allows you to specify a mnemonic that will be used to name the parts generated from a Business Object instance.

Default value: O

Relational table

This field allows you to specify a mnemonic that will be used to name the parts generated from a Relational Table instance.

Default value: T

Data element

This field allows you to specify a mnemonic that will be used to name the parts generated from a Data Element instance.

Default value: E

Part Types: These fields allow you to specify a letter that will be used to build the names of the generated parts, in addition to the identifier of the part and the Information Model mnemonic. It indicates the type of the generated part.

Program

This field allows you to specify a mnemonic that will be used to name a generated Program part.

Default value: A

Function

This field allows you to specify a mnemonic that will be used to name a generated Function.

Default value: F

Record

This field allows you to specify a mnemonic that will be used to name a generated Record part.

Default value: R

Data Item

This field allows you to specify a mnemonic that will be used to name a generated Data-Item part.

Default value: D

Table

This field allows you to specify a mnemonic that will be used to name a generated Table part.

Default value: T

TUI Map

The field allows you to specify a mnemonic that will be used to name a generated Map part.

Default value: M

TUI Map group

The field allows you to specify a mnemonic that will be used to name a generated Map Group part.

Default value: B

For information on generated parts and generated part naming, refer to “Part 3. Standard Use of VAGTemplates” on page 107, “Chapter 6. Application Generation and Enhancement” on page 225, “**Generated Architecture and Principles**” on page 240, “**Introduction**” on page 240, “**Generated Components Naming Policy**” on page 241.

TUI PSB

This field allows you to specify a mnemonic that will be used to name a generated Program Specification Block part.

Default value: Z

Obsolete Part Types:

Smalltalk only Process

This field allows you to specify a mnemonic that will be used to name a generated Process part.

Default value: P

Smalltalk only Statement Group

This field allows you to specify a mnemonic that will be used to name a generated StatementGroup part.

Default value: S

Smalltalk only GUI Application¹

This field allows you to specify a mnemonic that will be used to name a generated GUI Application part.

Default value: G

Entity Menu

The Entity menu allows you to define default parameters for every type of entity. These default parameters will be used to pre-fill the instance parameter specifications. This menu also allows you to extend the Information Model entities according to your specific needs.

Table 3. Entity Menu

Default Generation Parameters

> Shared

> Non Shared

Show Duplicate Generation Parameters

Entity menu / Default Generation Parameters choice

Displays the Default Generation Parameters editor specific to the entity you selected in the *Entities* area. This window enables you to set default values for all parameters of the entity's instances.

If you change the default values of the parameters here, at entity level, they will be the default parameters for all the entity's instances, except for those that you may have previously redefined, at instance level.

To apply the entity default parameters to redefined instances, select the instance(s), then select the *Set Generation Parameters to Default* from the *Instance* menu.

Note: The values you specify for an entity are specific to the Workspace; they do not apply to the same entity used in another Workspace.

TIP: Selecting this choice is the same as right clicking in the *Entities* area once the entity is selected and selecting the *Default Generation Parameters* choice from the pop-up menu.

Note: There is no default parameter for the Value Style entity.

To know the default parameters of each entity in the Workbench, refer to chapter "Chapter 3. Information Model Entities and their Editors" on page 51, the subchapter specific to the entity you are interested in, "**Default Generation Parameters**" on page 57.

1. This parameter ensures retrieval of legacy applications developed with the V2.2 Refresh and lesser versions. When generating an application with the 4GL oriented generators, the GUI Application part is converted into a Smalltalk visual part.

Entity menu / Show Duplicate Generation Parameters choice

Displays the duplicate instances according to the *Target name*, *Long target name* and *Fastpath* (Interface Unit only) parameters.

Instance Menu

The Instance menu allows you to manipulate all the entity instances: creating new instances, updating or deleting existing instances, filtering through the displayed instances, managing editions, and generating completed instances.

Table 4. Instance Menu

New ...

Definition

Generation Parameters

Load

> Previous edition

> Another edition

Editions...

References...

Associates

Referenced Workspaces

> Tree View

> Detail List View

Copy Generation Parameters from ...

Set Generation Parameters to Default

Move

Copy

Delete

Create Business Object from RT ...

Create Interface Unit from BO ...

Generate

Instance menu / New ... choice

Opens the *New VAGT Instance* window allowing you to create a new instance of any entity type.

TIP: Selecting this option is the same as right clicking in the *Instances* area, and selecting the *New...* choice from the pop-up menu.

The *Instance name* field is used to specify a name for the new entity instance.

The *Instance type* area allows you to select the entity type of the instance to be created.

The *Package/Application* combo box is used to select or specify a package or an application where the new instance will be stored.

Note: The Workspace package/application is always selected by default.

If you select the *Open now* check box, the instance will be saved and its Definition editor will open immediately after saving.

Instance menu / Definition choice

Opens the instance Definition editor allowing you to edit the functional description of the selected entity instance.

TIP: Selecting this option is the same as right clicking in the *Instances* area once the entity instance is selected, and selecting **Definition**. It is also the same as double-clicking on the instance you want to describe.

For more information, refer to chapter “Chapter 3. Information Model Entities and their Editors” on page 51, subchapter specific to the instance you selected, section *How to define a ...* (For example, if you selected an instance of Business Object, refer to “Chapter 3. Information Model Entities and their Editors” on page 51, “**Business Object**” on page 56, “**How to Define a Business Object**” on page 67.

Instance menu / Generation Parameters choice

Opens the instance Generation Parameters editor allowing you to specify the generation parameters of the selected entity instance.

TIP: Selecting this option is the same as right clicking in the *Instances* area once the entity instance is selected, and selecting **Generation Parameters**.

For more information, refer to chapter “Chapter 3. Information Model Entities and their Editors” on page 51, subchapter specific to the instance you selected, section *How to define a ...* (For example, if you selected an instance of Business Object, refer to “Chapter 3. Information Model Entities and their Editors” on page 51, “**Business Object**” on page 56, “**How to Define a Business Object**” on page 67.

Instance menu / Load choice

This choice allows you to load an edition of the selected instance.

Previous Edition: This choice allows you to load the previous edition of the selected instance.

TIP: Selecting this choice is the same as right clicking in the *Instances* area after selecting an instance, and selecting the **Load > Previous Edition** choice from the pop-up menu.

Another Edition: This choice allows you to load another edition of the selected instance. You must select an edition in the list that is displayed.

TIP: Selecting this choice is the same as right clicking in the *Instances* area after selecting an instance, and selecting the **Load > Another Edition** choice from the pop-up menu.

Instance menu / Editions choice

This choice displays the list of the selected instance's editions.

TIP: Selecting this choice is the same as right clicking in the *Instances* area after selecting an instance, and selecting the **Editions...** choice.

Instance menu / References ... choice

This choice opens the *References* window, allowing you to search for the instances that calls the selected instance.

Select one entity or more among the displayed entities to set your search scope.

Checking the *Filter* option will limit the search to the packages/applications you selected in the VAGTemplates Browser.

Instance menu / Associates choice

Tree View: This choice opens the *Associates Tree View* window that shows the instances called by the selected instance.

Detail List View: This choice opens the *Associates Detail List View* window that displays the list of instances called by the selected instances.

For each called instance, the following information is displayed in the list area: the icon associated with the corresponding entity, the instance name and display name, the target logic package/application and target ID associated with the instance and the instance current edition.

Instance menu / Referenced Workspaces choice

Instance menu / Copy Generation Parameters from... choice

Opens the *Copy Instance Parameters from...* window which allows you to copy the current instance's parameters defined in another Workspace to the current instance in the current Workspace.

Select the Workspace in which the parameters you want to copy are defined and click **OK**.

*For example: Let's assume that you have imported a database in the **MyApplication Workspace**. The import initialized the name of the STAFF Table in the Generation Parameters window of the Relational Table. If you generate a Business Object using the STAFF table columns from the **MyNewApplication Workspace**, there will be*

errors at the generation because the name of the table is not defined in the parameters of the STAFF table in the **MyNewApplication Workspace**. You cannot enter the name of the table since the Table name field is read-only and initialized by the import. However, if you copy the STAFF table parameters from the **MyApplication Workspace** to the STAFF table in the **MyNewApplication Workspace** there will be no problems generating or executing the application.

Instance menu / Set Generation Parameters to Default choice

This choice automatically sets the generation parameters to the entity's default values for the selected instance(s).

Instance menu / Move ... choice

Opens the *Move Instances* window allowing you to move an instance definition or generation parameters or both to an available package/application.

Check the *Move Definition* option or the *Move Generation Parameters* option or both.

In the *Select parameter workspaces* list, select a Workspace in which you want to move the instance generation parameters.

Instance menu / Copy choice

Opens the *Copy Instances* window that allows you to save the contents of the opened instance into another instance.

The *New instance name* field allows you to enter the new name of the instance.

The *Existing instance names* list allows you to select existing instances to save your current instance as the selected one, and therefore replace the specifications of the latter with those of the current instance.

By default, the instance is copied to the current package/application.

Instance menu / Delete choice

Opens a confirmation box requiring you to confirm the deletion of the selected entity instance(s).

Instance menu / Create Business Object from RT ... choice

Opens a dialog allowing you to create a Business Object from the selected Relational Table.

To create a Business Object from a given Relational Table:

1. Select a Relational Table in the list from the VAGTemplates Browser.
2. Select the *Create Business Object from RT ...* choice from the *Instance* menu.

3. In the dialog that opens up, enter a name for the new Business Object.
4. Select a package/application in the list, then click **OK**.

You can also select more than one Relational Tables from the VAGTemplates Browser. In this case, a Business Object is created for each selected Relational Table. The dialog box remains open until the last Business Object corresponding to the last Relational Table selected is created. The names of the selected Relational Table instances are automatically displayed one after another. For each of them, enter a name for the Business Object to be created.

Instance menu / Create Interface Unit from BO... choice

Opens a dialog allowing you to create an Interface Unit from the selected Business Object.

To create an Interface Unit from a Business Object:

1. Select a Business Object in the list from the VAGTemplates Browser.
2. Select the *Create Interface Unit from BO ...* choice from the *Instance* menu.
3. In the dialog that opens up, enter a name for the new Interface Unit.
4. Select a package/application in the list, then click **OK**.

You can also select more than one Business Objects from the VAGTemplates Browser. In this case, an Interface Unit is created for each selected Business Object. The dialog box remains open until the last Interface Unit corresponding to the last Business Object selected is created. The names of the selected Business Object instances are automatically displayed one after another. For each of them, enter a name for the Interface Unit to be created.

Instance menu / Generate choice

VAGTemplates on Java: The *Generate* choice opens the *Generate (instance(s))* window that enables you to generate the application components for the selected instances.

Select available generator(s): This area lists the generators available for the selected entity instances. At least one generator must be selected in the list.

Instance(s) to be generated: This area displays the instances you selected from the Browser.

Store Options:

Description of the options: The *Store Options* area allows you to specify a store option for the *builders* instantiated by the generation process.

Note: A *builder* is a Java class instance produced by the generation process. These classes model VisualAge for Java components, manages the store and load processes of the generated components into the VisualAge for Java Library and manages traceability information. Each instantiated builder, once stored, will correspond to one component in the VisualAge for Java Library.

Normal

This option stores the new specifications, preserve the RAD specifications and those that have been added using the VisualAge for Java environment.

Overwrite

This option is identical to the *Normal* option, but additionally forces the storage of RAD specifications.

Reset

This option is identical to the *Overwrite* option, but delete the specifications added by the user.

Customize

When this mode is checked, the builders are instantiated but not immediately stored as Java components. When all builders are generated, the *Customize Generation* window opens so that you can customize some storage parameters for the Java components to be produced from these builders.

Customize Generation window: This window contains a table in which each row corresponds to one generated builder. The builder table can be customized by:

- reordering the columns by drag and drop
- showing/hiding a column by right clicking once on a column header
- resizing a column
- sorting the table using any column as sort criterion
 - To sort a column in increasing alphabetical order, left click on a column header.
 - To sort a column in decreasing alphabetical order, press the Shift key and left click on a column header.

For each builder, the following information can be displayed in the corresponding columns:

- The first column contains check boxes allowing you to confirm whether you want the builder to be stored. These check boxes are designated below as the *store management* fields. A builder will be stored only if the corresponding *store management* field is checked.
- The *Type* column displays the builder type.

- The *Name* column displays the builder name.
- The *Store Option* column allows you to modify the store option. Left click on the cell corresponding to the builder, then select a store option in the combo box.
- The *Trace Category* column allows you to modify the trace category of generated builder. Left click on the cell corresponding to the builder, then select a trace category in the combo box.
- The *Context* column displays the name of the target project or package where the generated builder will be stored.
 - If the generated builder is a project, it will directly be stored in the VisualAge for Java Library. The *Context* column is empty.
 - A package is stored in a project. The *Context* column displays the target project name.
 - A class is stored in a package. The *Context* column displays the target package name.

The table also contains two hidden columns. To view them, right click in the table and select *Show all columns* from the pop-up menu.

- The *VAGT name* column displays the name of the VAGTemplates instance from which the builder has been generated.
- The *VAGT type* column displays the type of this instance.

Customize Generation window Pop-Up menu: The *Customize Generation* window provides a pop-up menu which allows you to handle several builders at the same time, show all columns and restore the initial display of the builder table if it has been customized.

Note: To add a builder to the selection, press Ctrl key and right click on the row you want to add.

Mark

This choice allows you to check at once the *store management* fields of a set of selected builders. These builders will be stored as Java components in the VisualAge for Java Library.

Unmark

This choice allows you to uncheck at once the *store management* fields of a set of selected builders. These builders will not be stored.

Store Option

This choice opens the *Store Options* dialog allowing you to modify at once the store options for a set of builders. Check the radio-button corresponding to the desired option, then click *OK*. This option is applied to all builders you selected.

Trace Category

This choice opens the *Trace Category* dialog allowing you to modify at once the trace categories for a set of builders. Check the radio-button corresponding to the desired trace category, then click *OK*. This trace category is applied to all builders you selected.

Mark all

This choice allows you to check the store management fields of all generated builders in order to store them.

Unmark all

This choice allows you to uncheck the store management fields of all generated builders in order not to store them.

Show all columns

This choice allows you to view all the columns of the builder table.

Reset

This choice allows you to restore the initial display of the table column.

Other Functionalities: You can get help from the *Customize Generation* window by clicking on the *Help* push-button.

The *OK* button closes the *Customize Generation* window and triggers the storage of the generated builders.

Generation Scope:

Instance only

This option only generates the selected instance(s).

With associates

This option successively generates the selected instance(s) and all the instances it calls.

With associates and predefined beans

This option successively generates the selected instance(s), all the instances it calls and the predefined beans.

Client & Server:

Visuals

Generates the visual part of the application.

Client

Generates the client part of the application.

Server

Generates the server part of the application.

For details on the generation, refer to “Part 3. Standard Use of VAGTemplates” on page 107, “Chapter 6. Application Generation and Enhancement” on page 225, “**Standard Generation**” on page 225; for information on the predefined beans, refer to “**Generated Architecture and Principles**” on page 240, “**Components Generated From a Workspace: Predefined Beans/Parts**” on page 312 in the same chapter.

View Menu

Table 5. View Menu

Refresh Now
Select All
Deselect All
Sort by
> Sort by Instance Name
>Sort by Display Name
> Sort by Target Package/Application
>Sort by Target ID
>Sort by Edition
Reorder Columns...
Reorder Status Bar Text...

View menu / Refresh Now choice

Refreshes the content of the *Instances* area in the Browser.

View menu / Select All choice

Allows you to select all the instances from the Browser *Instances* area.

View menu / Deselect All choice

Allows you to deselect all the instances from the Browser *Instances* area.

View menu / Sort by choice

Allows you to sort the instances displayed in the Browser *Instances* area according to the following criteria:

- Sort by Instance Name
- Sort by Display Name
- Sort by Target Package/Application
- Sort by Target ID
- Sort by Edition

View menu / Reorder Columns choice

Opens a window allowing you to reorder the columns of the *Instances* area.

View menu / Reorder Status Bar Text choice

Opens a window allowing you to reorder the data displayed in the status bar.

Tools Menu

The Tools menu allows you to activate the import of a database schema into VAGTemplates, to create Relational Table instances and to create Data Element instances. For VAGTemplates on Java only, it also allows you to enable the Show Duplicate Instances choice.

Table 6. Tools Menu

Import from Database...

Show Duplicate Instances

Tools menu / Import from Database choice

Opens the *Import from Database* window allowing you to import a relational database into the current Workspace.

- **Settings area**

Database name

By default, this *Database name* field is filled in with the database name specified in the VisualAge Generator Preferences (see the VisualAge for Java or VisualAge Smalltalk Enterprise documentation).

To modify the database name displayed, click on the *Change...* push button. The *Settings* window opens. All the fields in the window are pre-filled with the VisualAge Generator Preferences settings.

Userid

You can modify the default value by entering a new user ID in the edit field. The user ID is necessary to connect to the database you want to import.

Password

You can modify the default value by entering a new password in the edit field. The password is necessary to connect to the database you want to import.

DBMS

You can modify the default DBMS (Database Management System) using this drop-down list which allows you to select among all the available Database management systems. The drop-down list also allow you to choose to import the database via IXF FILE (NT or MVS) instead of using CLI access.

IXF is a format used by DB2 to import/export information from a relational database. To use the IXF FILE import, create a directory (by default: MdlROOT) in a DB2 command window session, and put the files required for the import. To do so, follow these commands:

- DB2 commands for a DB2/NT or OS2 database:

```
DB2 CONNECT TO %1 USER %2 USING %3
DB2 EXPORT TO SYSTABLES OF IXF SELECT*FROM
SYSIBM.SYSTABLES
DB2 EXPORT TO SYSCOLUMNS OF IXF SELECT*FROM
SYSIBM.SYSCOLUMNS
DB2 EXPORT TO SYSVIEWS OF IXF SELECT*FROM
SYSIBM.SYSVIEWS
DB2 EXPORT TO SYSVIEWDEPS OF IXF SELECT*FROM
SYSIBM.SYSVIEWDEP
DB2 EXPORT TO SYSRELS OF IXF SELECT*FROM
SYSIBM.SYSRELS
DB2CONNECT RESET
```

- DB2 commands for a DB2/MVS database:

```
DB2 CONNECT TO %1 USER %2 USING %3
DB2 EXPORT TO SYSTABLES OF IXF SELECT*FROM
SYSIBM.SYSTABLES
DB2 EXPORT TO SYSCOLUMNS OF IXF SELECT*FROM
SYSIBM.SYSCOLUMNS
DB2 EXPORT TO SYSVIEWS OF IXF SELECT*FROM
SYSIBM.SYSVIEWS
DB2 EXPORT TO SYSVIEWDEPS OF IXF SELECT*FROM
SYSIBM.SYSVIEWDEP
DB2 EXPORT TO SYSRELS OF IXF SELECT DISTINCT*FROM
SYSIBM.SYSRELS T1 , SYSIBM.SYSFOREIGNKEYS T2 WHERE
T2 RELNAME = T1.RELNAME AND T2 TBNAME = T1
TBNAME AND T2 CREATOR = T1 CREATOR
DB2 CONNECT RESET
```

Catalog

For Oracle DBMS, the *Catalog* push button opens the ODBC database manager allowing you to catalog an Oracle database.

Database name

This drop-down list allows you to modify the name of the database that will be imported into your Workspace.

Default

The *Default* push button allows you to either save the values specified in the *Settings* window for further import processes or to retrieve the VisualAge Generator default values.

Target Package/Application

This combo box allows you to select an application in which the Relational Table and Data Element instances that will be created by the

import will be stored. You can also enter a new package/application name in the field, creating thus a new package/application as you start the import.

- **Search Criteria area**

By default, the import facility imports all the tables and views in the database. The *Search Criteria* area allows you to select only the tables that you wish to import.

- The *Qualifiers* edit box allows you to choose the SQL qualifier for the tables you want to import.

For example, if some tables in the database have your user Id as a qualifier and others have that of another user, you can use this field to select only the tables with your user Id qualifier.

- The *Names* edit box allows you to extract Tables according to their names.
- The *Tables* and *Views* check boxes allow you to import only Tables, only Views or Tables and Views.

- **Build List push button**

When you clicked on *Build List* once having specified your search criteria, the *Available tables* area is filled with tables and/or views matching these criteria that have been extracted from the database.

- **Selected tables**

Select the tables you want to import in your Workspace and click the >> push-button. These tables will pass to the *Selected tables* list. These are the tables that will be imported. Click *OK* to trigger the import.

- **Reuse data elements**

This check box allows you to specify whether you want the import function to create one Data Element for each unique column in the database or if you want one Data Element for every column in the database. Identical columns that appear in more than one table will have a Data Element instance created for each column. These Data Element instances will be created with unique names (NAME and NAME1 for example).

Note to ORACLE users:

The first columns lists the ORACLE numeric formats and the second lists the corresponding formats imported into VAGTemplates.

ORACLE (precision, scale)	VAGT (capacity, precision)
NUMBER	FLOAT
NUMBER (p) p>0	NUMERIC (p, 0)
NUMBER (p, s) p>=s s>0	NUMERIC (p-s, s)

NUMBER (p, s) s<0	NUMERIC (p, 0)
NUMBER (p, s) p<s	NUMERIC (0, s)

Tools menu / Show Duplicate Instances choice

This choice opens a window that displays the duplicate instances according to given filters.

- In the *Packages* or *Applications* list, select one or more packages/applications.
- In the *Entities and Parameters* list, select one or more items.
- You can then delete the instances you want among the duplicate instances that are displayed in the *Instances and Workspaces* area, pressing the *Delete selected* push-button.

Options Menu

Table 7. Options Menu

Save Settings as Default

Options menu / Save Settings as Default choice

Saves the current work context for the next work sessions, such as the Workspace used, the columns displayed in the Workbench and their order, etc.

Help Menu

Table 8. Help Menu

Help index

General help

Using help

Product information

Help menu / Help Index choice

Displays an index of Help topics.

Help menu / General Help choice

Displays general help about the active Definition editor or Generation Parameters editor.

For more information, see topic “**How to Use VAGTemplates On-Line Help**” on page 50.

Help menu / Using Help choice

Displays a window explaining how to use the Help facility.

Help menu / Product Information choice

Displays a window with VAGTemplates product information.

How to Use VAGTemplates On-Line Help

You can get help at three different levels:

- Help is available for each entity Definition editor and Generation Parameters editor:
 - **Interface Unit** editors,
 - **Business Object** editors,
 - **Data Element** editors,
 - **Value Style** editors,
 - **Relational Table** editors.
- Help is available for each panel within an editor.
- From each of these editor panels, you can get help on fields.
 1. To get general help on any Definition editor or Generation Parameters editor follow this procedure:
 - Left click anywhere in the editor's panel,
 - Press **F1**, or click the *Help* push button in the editor.

A help window opens describing the active panel and presenting a list of hypertext items corresponding to the different panels of this particular editor.
 2. To get help on the contents of a panel in any editor:
 - From the editor general help window, click on the corresponding hypertext item.
 - Or, left click anywhere in the panel outside a field and press **F1**.

A second help window opens describing the active panel and presenting a list of hypertext items corresponding to the fields contained in the editor's panel.
 3. To get help on a field:
 - From the panel help, click on the corresponding hypertext item.
 - Or left click in the field, then press **F1**.

To close the on-line help facility, use the combination of 'Alt+F4'.

Chapter 3. Information Model Entities and their Editors

Introduction

This chapter presents the Information Model entities and describes the different editors you will use to define the entity instances. To define these instances, you have to specify two types of information: the instance generation parameters and the instance logical or functional description.

- **The generation parameters**

An instance generation parameters can be defined:

1. at entity level using the entity Default Generation Parameters editor. These parameters will be the default parameters for all the entity instances within the Workspace in which the parameters are defined.
2. at instance level using the instance Generation Parameters editor. This editor is used to modify the default parameters specified at entity level if necessary and also to define parameters that are specific to a particular instance and that cannot be defined at entity level.

Note: These editors are not available for the Value Style entity whose instances are not used as generation input.

- **The logical description**

The instance logical properties are defined using the instance Definition editor.

In each subchapter dedicated to a specific entity, we describe in the following order:

1. the generation parameters that are found in the Default Generation Parameters editor.
See the *Default Generation Parameters* section corresponding to the entity.
2. the functional properties that are found in the Definition editor.
See the section entitled *How to Define a... (Entity Name)* corresponding to the entity.
3. if any, the additional generation parameters that are specific to a particular instance, and as such definable in the instance Generation Parameters editor only.
See the section entitled *How to specify the... (Entity Name) Parameters* corresponding to the entity.

Note: When we come to describing the specification fields, the settings and parameters that only apply to GUI client applications are highlighted

with this **GUI** sign; those that only apply to TUI applications with the **TUI** sign. No sign means that the settings and parameters are common to both TUI and GUI client applications.

The Information Model Entities

The definition of business applications with VAGTemplates is based on an Information Model. The **Information Model** has two aspects:

- it is an inventory of all the elements (entities and their attributes, links between entities, etc.) you will use to design your application.
- it defines the hierarchical organization and the structure of the elements required to design your application.

This model-driven approach lets you define your applications in an exhaustive, non-redundant, and systematic way, regardless of the platform for which the developed applications will be generated.

The Information Model comprises the following entities:

- *Business Object*
- *Data Element*
- *Interface Unit*
- *Relational Table*
- *Value Style*

For information on these entities, refer to the subchapters that deal with each entity in this chapter.

The Definition Editor

The Definition editor is used to describe the functional characteristics of an entity instance.

To open the Definition editor for an entity instance:

- From the VAGTemplates Browser, double click on an instance in the *Instances* area.
- Or, left click on the instance to select it, then right click on it and select **Definition** from the pop-up menu.

The Definition editor includes several panels allowing you to enter the instance logical specifications, which do not vary from one generation to the other.

*For example, if you have created a STAFF Business Object to present the information related to a staff member, this Business Object always maps to the STAFF Table. This is the kind of specification you do using the **Definition** editor.*

The Default generation Parameters editor and the Generation Parameters Editor

Both editors are used to define the generation parameters for entity instances.

The Default Generation Parameters editor is used to define the generation parameters that are common to all the entity instances within a particular Workspace.

Note: The parameters you specify within a Workspace apply to all the instances defined in this Workspace, they do not apply to the same instance used in another Workspace.

To open an entity Default Generation Parameters editor:

- From the VAGTemplates Browser, select an entity in the *Entities* area, and from the *Entity* menu, choose *Default Generation Parameters*.
- Or select an entity, right click on it and choose *Default Generation Parameters* from the pop-up menu.

The Generation Parameters editor allows you to redefine a selected instance's default parameters set at entity level at initialization of the Workspace or the default parameters you set using the *Default Generation Parameters* choice from the *Entity* menu.

To open the Generation Parameters editor for an entity instance:

- From the VAGTemplates Browser, left click on the instance to select it, then right click on it and select *Generation Parameters* from the pop-up menu.

The Default Generation Parameters editor and the Generation Parameters editor include several panels that allow you to define instance generation parameters.

Each panel in the Default Generation Parameters editor and its corresponding panel in the Generation Parameters editor contain exactly the same fields except the *General* panel. The *General* panel in the Generation Parameters editor contains additional fields that are specific to a particular instance: typically, an instance *Target name* and *Help panel ID* fields.

When you open the Generation Parameters editor the fields displayed in the different panels are already filled in with the default parameters that have been set at entity level. This is a way of standardizing and quickly developing all the instances of your application.

In the Generation Parameters editor panels, a *Redefined* checkbox is associated with each parameter, indicating if the parameter value is the default one or not. When you modify default values, *Redefined* is automatically checked. You

can manually uncheck it, thus changing the current value to the default one (defined on the entity level), or check it even when the value is the default one, thus protecting the value against a change on the entity level.

Specifications entered in this editor panels may be changed according to each of your generation needs.

For example, the number of lines in a list presenting the STAFF Business Object can vary from one generation to the next: the list can have 5, 8, 15, ... lines. This is the kind of specifications you can enter using the Generation Parameters editor.

Editor General Characteristics

All the editors used to specify entity instances have a number of common characteristics and behaviors.

Navigation within an Editor

As seen before, three types of editors are available to define the generation parameters as well as the functional properties for the instances used in your application:

- the **Default Generation Parameters** editor
- the **Generation Parameters** editor
- the **Definition** editor

From an instance Definition editor, a dedicated push-button allows you to switch quickly to the instance Generation Parameters editor. The same navigation function is available from an instance Generation Parameters editor to the corresponding Definition editor.

For each type of editor, two parts can be identified as in the example below.

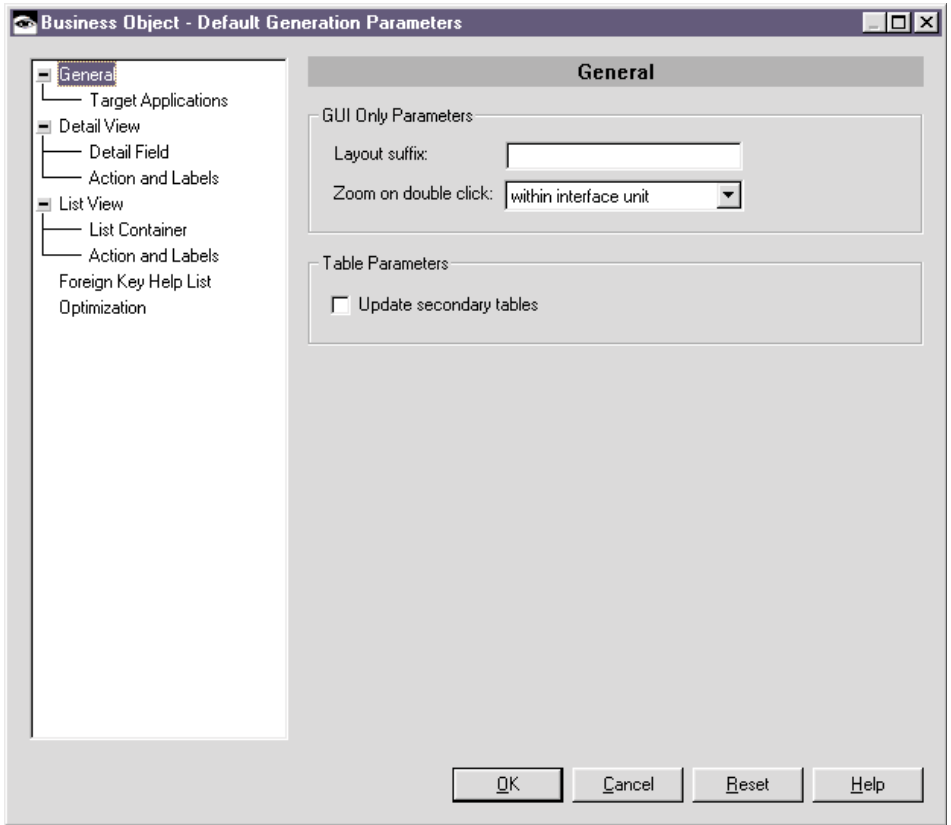


Figure 2. Business Object Default Generation Parameters Editor

- The left part called the **tree view** displays a list of items that correspond to the titles of the different panels making up the editor. To navigate from one panel to another, just left click on one of these items. Some items have sub-items; to access the corresponding panels, you have first to click on the + sign to display the sub-items.
- The right part makes up what we call an **editor panel**. The top part of each panel displays the panel title.

Push Buttons

There are four push buttons in each editor:

- the *OK* push button closes the editor and saves all the specifications you entered.
- the *Cancel* push button closes the editor and none of the specifications you entered are saved.
- the *Reset* push button clears the current specifications you entered and retrieves the last saved specifications.

- the *Help* push button opens the help panel that describes the current editor.

Pop-up Menus

You can activate pop-up menus by *right clicking* in an editor. Depending on the instance you are specifying, these pop-up menus will contain some of the following choices:

- **Add** - All editors - this choice adds an item to an updatable list.
- **Remove** - All editors - this choice removes an item from an updatable list.
- **Open <entity>** - Interface Unit, Business Object and Relational Table editors - when you select an entity that is called by the entity you are specifying (Interface Unit, Business Object or Relational Table), this choice opens its Definition editor allowing you to consult or edit the instance functional description.

For example, you are specifying a Business Object instance that calls a particular Relational Table. When you select this Relational Table in the Business Object Definition editor, this choice opens the Definition editor of the Relational Table.

- **Add volatile** - Business Object Definition editor - use this choice when you want to add a field for displaying non-constant data in a Business Object.

Editor Extensions Panels

Restricted to use of customized generators involving Information Model extensions.

Business Object

What is a Business Object?

A **Business Object** is an object *via* which consultation and update of constant data can be performed. The Business Object groups a set of fields to match the needs for a specific application to access, present, and manipulate constant data. The Business Object's fields represent Data Element calls that map to the columns of one or more Relational Tables.

For example, if the application requires a detail presenting the name of a staff member, his/her job, and his/her salary, Name, Job, and Salary would be three fields of the corresponding Business Object.

A Business Object can also present non-constant data by using volatile fields.

The Business Object maps to Relational Tables. Among these, the first table mapped is designated as the Business Object's primary table. Only the primary table's fields can be updated; the fields of the other tables are read-only.

Default Generation Parameters

The *Default Generation Parameters* choice from the *Entity* menu opens the Default Generation Parameters editor that allows you to define several parameters at entity level that will be default parameters for all the entity instances defined within a Workspace. When defining an instance, you only have to set the remaining parameters. You can of course modify these default parameters as you wish at instance level.

Default Generation Parameters Editor

General Panel:

GUI Only Parameters:

Layout Suffix

This field allows you to enter a 2 characters suffix that will be used for the creation of a Business Object layout at generation time.

For example, if you want to generate several detail layouts for the same Business Object instance, you should change this suffix each time you generate a new layout. No default value

Zoom on double click

This drop-down list allows you to specify the generation of the zoom in function from a list in a window to a detail in the same window (*within interface unit or zoom off*).

Default value: within interface unit

Table Parameters:

Update secondary tables

When this option is checked, all the tables referenced by the Business Object can potentially be modified. When unchecked, only the primary table referenced by the Business Object can potentially be modified.

Default value: unchecked

Target Packages/Applications Panel: This panel allows you to specify the project (VAGTemplates on Java only) and the packages/applications where the components generated from the Business Object will be stored.

Target Packages/Applications:

JAVA Target project

This field is used to specify the name of the project that includes all the target packages associated with the Business Object.

Visual package/application

This field is used to indicate the package/application where the visual components generated from the Business Object will be stored.

Logic package/application

This field is used to indicate the package/application where the logic components generated from the Business Object will be stored.

Services package/application

This field is used to indicate the package/application where the service components generated from the Business Object will be stored.

Detail View Panel: This panel groups parameters that manage the presentation of the details and of the fields in a detail.

For information on the presentations provided by the following parameters, refer to “Part 3. Standard Use of VAGTemplates” on page 107, “Chapter 5. Standard Functions and Layouts of Generated Applications” on page 145, “**Standard Layouts of GUI Client applications**” on page 169 and “**Standard Layouts of TUI Applications**” on page 199.

Common Parameters:

Display

This drop-down list allows you to choose the graphical aspect you want for the detail Business Object. You can choose among *form*, *form with border*, *notebook²*, *Windows notebook*, *PM notebook*.

Default value: form

Label

This field allows you to assign a label to the detail Business Object.

No default value

Note: This parameter is relevant when the *Display* parameter is set to *groupbox*; otherwise it is ignored.

Number of columns

This spin-edit allows you to specify how many columns the fields will be placed in.

Default value: 2

Number of lines

This spin-edit allows you to specify how many lines the fields will be placed on.

2. The *notebook* parameter is used to ensure ascending compatibility with applications developed with previous versions of the product. Use one of the two other notebook values instead. The *notebook* parameter is equal to *PM notebook*.

Default value: 8

GUI Only Parameters:

Notebook field location

This drop-down list allows you to specify whether you want the number of lines and columns fixed for each notebook page (*fixed number of lines and columns*), or the number of fields (*fixed number of fields*).

Default value: fixed number of lines and columns

Note: This parameter is relevant when the *Display* parameter is set to *notebook*; otherwise it is ignored.

Number of fields per page

This spin-edit allows you to define how many fields will be presented in each page of the notebook. Anytime fields do not fit in the page because of their presentation (fields too long, for example) a page is added to the notebook.

Default value: 10

Note: This parameter is relevant when the *Display* parameter is set to *notebook* and the *Notebook field location* parameter is set to *fixed number of fields*; otherwise it is ignored.

Detail Field Panel:

Detail Field:

Alignment

This drop-down list allows you to define a common alignment for the fields and their labels.

Default value: left aligned labels / left aligned values

Note: If you have specified *right-to-left* for horizontal orientation in the Data Element, values will be:

Default value: right aligned labels / right aligned values

Other values: left aligned labels / right aligned values
right aligned joined labels

Layout

This drop-down list allows you to specify the order in which the fields will be arranged. You can choose between a vertical arrangement, where the fields are positioned from *top to bottom* then from left to right, or a horizontal arrangement where the fields are positioned from *left to right* then from top to bottom.

Default value: top to bottom

Note: If you have specified *right-to-left* in the horizontal orientation arrangement, fields are positioned from *right to left* then from top to bottom. For the vertical arrangement, fields are positioned from *top to bottom* then from right to left.

GUI Label and value display

This drop-down list allows you to specify whether the label will be located above the value (*vertical*) or beside it (*horizontal*).

Default value: horizontal

Note: With *right to left* orientation, the label will be positioned on the right and the value on the left.

Sizing

This drop-down list allows you to specify the size of the fields' presentations according to their contents. You can choose between *adjusted* (the size of the field is adjusted to the contents), or *equalized* (the size of the field is equalized with the size of the largest field).

Default value: adjusted

Note: The size of the field presentations never varies with the size of the Business Object layout.

Detail Actions and Labels Panel: This panel groups the parameters that manage the presentation of the actions available in a detail Business Object. These parameters only apply to GUI client applications.

For information on the function of these actions, refer to "Part 3. Standard Use of VAGTemplates" on page 107, "Chapter 5. Standard Functions and Layouts of Generated Applications" on page 145, "**Standard Functions**" on page 146, "**Management of Persistent Data**" on page 147; for information on the layout of these actions, refer to "**Standard Layouts of GUI Client applications**" on page 169, "**Detail Business Objects**" on page 181 in the same chapter.

Detail Actions:

GUI Display

This drop-down list allows you to specify the layout of actions: *menu, push buttons, pop-up menu, menu and popup menu* or *menu and push buttons*.

Default value: menu

GUI Sizing

This drop-down list allows you to specify the size of the action presentations according to the size of their labels. The size of the presentation can be *adjusted* to the length of the label, or *equalized* to the length of the largest action label.

Default value: adjusted

Note: The size of the presentations never varies with the size of the Business Object layout.

This parameter is relevant when the value of the *Display* parameter is set to *push button*; otherwise it is ignored.

GUI *Position*

This drop-down list allows you to specify whether the push buttons are located in the layout from left to right (*left*) or from right to left (*right*).

Default value: left

Note: With the *right to left* orientation, the alignment of push buttons is right oriented:

Default value: right

GUI *CRUD Activation Control*

This check box allows you to specify a control on when the create, read, update and delete actions are active. If you check this option, the action menus or push buttons are enabled when the Business Object's logical key is present; when the key is not present they are disabled. Otherwise, the action menus or push buttons are always enabled.

Default value: true

Labels:

GUI *Check label*

This field allows you to specify the label of the action that checks the Business Object fields.

Default value: Check

GUI *New label*

This field allows you to specify the label of the action that initializes fields in a detail.

Default value: New

GUI *Create label*

This field allows you to specify the label of the action that creates a database row.

Default value: Create

Note: This parameter is disabled when the *Update GUI policy* parameter is set to *save*.

GUI *Read label*

This field allows you to specify the action that reads database rows.

Default value: Read

GUI Save label

This field allows you to specify the label of the action that saves database rows.

Default value: Save

Note: This parameter is disabled when the *Update GUI policy* parameter is set to *create/update*.

GUI Update label

This field allows you to specify the label of the action that updates a database row.

Default value: Update

Note: This parameter is disabled when the *Update GUI policy* parameter is set to *save*.

GUI Delete label

This field allows you to specify the label of the action that deletes a database row.

Default value: Delete

GUI Show message label

This field allows you to specify the label of the action that opens the window displaying error messages.

Default value: Messages

Note: This action is not provided in the standard generated applications.

GUI Menu label

This field allows you to specify the label of the menu bar item which contains the detail actions menu choices (Update, Delete...)

List View Panel: This panel groups the parameters that manage the presentation of the list Business Object and the presentation of the fields in the list.

For information on the presentations provided by the following parameters, refer to “Part 3. Standard Use of VAGTemplates” on page 107, “Chapter 5. Standard Functions and Layouts of Generated Applications” on page 145, “**Standard Layouts of GUI Client applications**” on page 169 and “**Standard Layouts of TUI Applications**” on page 199.

Common Parameters:

Number of rows to fetch

This spin edit allows you to define how many rows will be fetched from the database and stored in memory. If the number of data rows is greater

than the size of the list (GUI), the end user will have to scroll the list to view the following rows. When all the data in memory have been displayed, the end user will have to activate a new database access.

Default value: 20

GUI Display

This drop-down list allows you to define the graphical presentation of the list. You can choose among *form*, *form with border* and *groupbox*.

Default value: form

Label

This field allows you to specify a label for the list.

No default value

Note: This parameter is relevant when the *Display* parameter is set to *groupbox*.

List prefilled

This check box allows you to define whether you want the lists to be pre-filled when the window opens or not. In the latter case, the end user will have to activate an action to fill the list.

Default value: false

GUI Only Parameters:

GUI Page display

This drop-down list allows you to specify whether the list displays all the pages read in the database (*display all read pages*) or only the page being read (*display current page*). In the latter case, the end user will have to trigger an action to view the pages that have already been read.

Default value: Display current page

Note: This parameter is available when the *Page display* parameter is set to *Display all read pages*.

GUI Paging policy

This drop-down list allows you to specify whether the paging in a list is performed by actions (*explicit paging*) or using only the scroll bar (*auto-scrolling*).

Default value: explicit paging

Note: This parameter is available when the *Page display* parameter is set to *display all read pages*.

GUI Extraction criteria displayed

This check box allows you to define whether you want the extract criteria to be laid out or not. In the latter case, the extract criterion fields will be laid out only in the list container as the other fields are.

Default value: false

TUI Only Parameters:

TUI List pages stack number

This spin edit allows you to specify the maximum number of the read pages' keys that must be stored, which condition the navigation possibilities.

For example, if you set the parameter to 5, the end user will be able to re-view only the last five pages with which he/she has been working.

Note: The "keys" here are only pointers that reference a row in each page. They are used to retrieve where each page begins and ends.

Default value: 30

List Container Panel:

Common Parameters:

Number of lines

This spin edit allows you to specify the number of lines in the list, which condition the number of rows displayed simultaneously on screen. If the number of data rows is greater than the size of the list (GUI), the end user will have to scroll the list to view the following rows.

Default value: 5

Caution: When you define TUI applications, the number of lines that you specify must not be greater than the maximum number of lines that can be displayed on screen (size of the 3270 screen minus header, trailer, and lines dedicated to displaying actions codes); otherwise there will be warnings at generation time.

GUI Only Parameters:

GUI Number of columns

This spin edit allows you to specify the number of columns in the list.

Default value: 8

NOTE: In the generated application, if the number of columns is too large for the width of the window a scroll bar will be generated to allow the end user to scroll the list to see the columns that are not displayed.

Actions and Labels Panel: This panel groups parameters that manage the presentation of the actions available in a list Business Object. These parameters apply to GUI client applications.

For information on the function of these actions, refer to “Part 3. Standard Use of VAGTemplates” on page 107, “Chapter 5. Standard Functions and Layouts of Generated Applications” on page 145, “**Standard Functions**” on page 146, “**Management of Persistent Data**” on page 147; for information on the layout of these actions, refer to “**Standard Layouts of GUI Client applications**” on page 169, “**List Business Objects**” on page 191 in the same chapter.

List Actions:

GUI Display

This drop-down list allows you to specify the layout of actions: *menu*, *push buttons*, *pop-up menu*, *menu and popup menu* or *menu and push buttons*.

Default value: menu

GUI Sizing

This drop-down list allows you to specify the size of the action presentations according to the size of their labels. The size of the presentation can be *adjusted* to the length of the label, or *equalized* to the length of the largest action label.

Default value: adjusted

Note: The size of the presentations never varies with the size of the Business Object layout.

This parameter is relevant when the value of the *Display* parameter is set to *push button*; otherwise it is ignored.

GUI Position

This drop-down list allows you to specify whether the push buttons are located in the layout from left to right (*left*) or from right to left (*right*).

Default value: left

Note: With the *right to left* orientation, the alignment of push buttons is right oriented:

Default value: right

List Labels:

GUI Top label

This field allows you to specify the label of the action that reads the first rows in the database.

Default value: Top

GUI Refresh label

This field allows you to specify the label of the action that refreshes the list with the last saved data.

Default value: Refresh

GUI Extract label

This field allows you to specify the label of the action that reads the rows that match the extract criteria.

Default value: Extract

For information on extract criteria, refer to topic “**Entity Menu**” on page 36.

GUI Forward label

This field allows you to specify the label of the action that reads the next rows in the database.

Default value: Next

GUI Backward label

This field allows you to specify the label of the action that reads the previous rows in the database.

Default value: Previous

GUI Submit label

This field allows you to specify the label of the action that saves the transactions that the end user performed on updatable list lines.

Default value: Submit

Note: This parameter is relevant when the value of the *Layout type* parameter (Interface Unit Definition editor) is set to *updatable list*; otherwise it is ignored.

GUI Menu label

This field allows you to specify the label of the menu bar item which contains the list actions menu choices (Top, Extract...).

Foreign Key Help List Panel:

Foreign Key Help List:

Help list for all foreign keys

This check box allows you to define whether you want a help list generated for all the foreign keys of the primary table or not.

Default value: checked

A **help list** is a list of values available for a field that are offered to the end user as an input aid. Help lists are only available on fields that are foreign keys of the primary table.

A **foreign key** is a field or a set of fields, used to identify or access particular rows in a table. Foreign key values must correspond with primary key values in the primary table.

Help list prefilled

This check box allows you to define whether you want the help lists to be pre-filled when the window opens or not. In the latter case, the end user will have to trigger an action on the help list to view its contents.

Default value: unchecked

Help list page display

This drop-down list allows you to define whether the help list displays all the pages read in the database (*display all read pages*) or only the page being read (*display current page*). In the latter case, the end user will have to trigger an action to view the pages that have already been read.

Default value: display current page

GUI *Help list size*

This spin-edit allows you to specify the size of the help lists, i.e. how many values can be viewed without having to scroll the help list.

Default value: 4

Optimization Panel:

Service Level:

Service level

These radio-buttons allow you to choose to generate only the Business Object layouts you will need for each of your generated applications: *detail and updatable list*, *detail and read-only list*, *detail only*, *updatable list only*, *read-only list only*.

Default value: *detail and updatable list*

How to Define a Business Object

Use the Business Object **Definition** editor to define a Business Object.

To create a new Business Object instance and open its Definition editor:

1. Select the Business Object entity, select *New...* from the *Instance* menu,
2. Enter the name of the instance, making the first 5 characters significant.
3. In the *Package/Application* combo box, select a package or an application where the instance functional description will be stored,
4. Select the *Open now* check box,

5. Click *OK*.

To open the Definition editor of an already existing instance of Business Object:

1. Select the Business Object entity,
2. Select the instance to edit,
3. Select *Definition* from the *Instance* menu.

Definition Editor

This editor groups the specification fields that manage the logical description of the Business Object.

General Panel:

Package/Application:

Package/Application

This field indicates the package or the application where the the functional description of the Business Object instance is stored.

Names:

Default use name

This field allows you to enter the default name that is to be automatically proposed when this instance is used by another instance in the Workbench. It appears in the *Business Objects* list when you add a Business Object call to an Interface Unit.

Value: an alphanumeric string of 1 to 32 characters.

Display name

This field allows you to enter a display name. It will be displayed, in the generated application, in the list of Business Objects in the window (GUI) or the map (TUI) help panel where it is called.

Value: an alphanumeric string of 1 to 64 characters.

Descriptions Panel:

Descriptions:

Textual description

This multi-line edit allows you to enter a comment corresponding to the instance.

Value: an alphanumeric string of 1 to 1023 characters.

TIP: This field is not used in generating in the final application. You can use it to communicate technical information about the instance to other developers.

GUI On line help description

This multi-line edit allows you to enter a help text that will appear when the end user requests help on the Business Object instance.

Value: an alphanumeric string of 1 to 1023 characters.

For information on generated on-line help, refer to “Part 3. Standard Use of VAGTemplates” on page 107, “Chapter 5. Standard Functions and Layouts of Generated Applications” on page 145, “**Standard Functions**” on page 146, “**On-Line Help**” on page 160 On-Line Help.

Tables and Fields Panel: This panel groups the specification fields that define the Relational Tables mapped by the Business Object, and the Data Elements called in the Business Object.

Tables: This list allows you to specify the tables that you want mapped by the Business Object. By doing this, you can then define the Data Elements that you want presented in the Business Object.

Table

The *Table* column displays the name of the Relational Tables. The first table mapped is called the primary table and is the first appearing in the *Tables* list. The other tables are called secondary tables. The fields of the Business Object mapping to the primary table can be updated; the fields mapping to secondary tables are read-only.

If you right click in this column, a pop-up menu displays.

- Select the *Add* choice to select a table and add it to the list.
To do this, you can also click on the *Add Table...* push-button.
- Select the *Remove* choice to remove the selected table from the list.
- Select the *Open Table* choice to open the Definition editor of the Relational Table allowing you to consult or edit its functional description.

Mapping

The *Mapping* column displays the default use name of the Relational Table. By default it is its name. You can modify it by clicking on it and typing a new default use name.

Columns:

Column

This column displays all the columns of the table selected in the *Tables* list. You can select one or more columns of a Table and transfer it to the *Fields* list by clicking the >> button.

Note: You cannot transfer the secondary table's column that is a primary key if the linked primary table's column has already been called as a Business Object's field.

Key

The *Key* column indicates which of the columns are primary or foreign keys.

Fields: This list presents all the fields making up the Business Object. You fill it by transferring columns from the mapped tables (click >>) or by adding non constant fields.

Data Element

The *Data Element* column displays the Data Element used by the field.

Field Name

The *Field Name* column displays the default use name of the Data Element if it exists, otherwise the name of the column. You can modify the default use name here by clicking on it and typing the new name.

To add a non constant field, right click in the *Fields* list area and select the **Add volatile** choice or click on the *Add Volatile* push-button. The *Add Volatile Fields* window opens up. Select a field in the list, then click *OK*.

Note: If you check the *Filter* option, the instances in the *Add Volatile Fields* window are displayed according to the filter you set using the **Set Filter** or the **Quick Filters** choices from the VAGTemplates Browser *View* menu.

You can also add to the list, a non constant field that is not defined yet. In the *Add Volatile Fields* window, click on the *New Data Element* push-button. The *New VAGT Instance* window opens up, allowing you to create a Data Element. As the creation finishes, the *New VAGT Instance* window closes. Select the new Data Element in the *Add Volatile Fields* window to add it to the list.

Key and Criteria Panel: This panel groups the specification fields that define the Business Object's logical key as well as extraction and search criteria.

Fields: This list reflects the fields you selected in the *Tables and Fields* panel (see above). It allows you to select the field that you want to be used as logical key, extraction criteria and sort criteria. To do so, select a field in the list and click the >> push button corresponding to the *Logical Key*, *Extraction Criteria*, and/or *Sort Criteria* list.

Logical Key:

Field Name

This list presents the field(s) of the Business Object used as logical key. You select one or several fields from the *Fields* list and transfer them to the *Logical key* list by clicking the corresponding >> button.

The **logical key** of a Business Object is the key field used when the end user performs an elementary action (create, read, update, delete) on the data from the primary table.

For example, let's assume you specified the Name field as the Business Object's logical key. When the end user wants to create a new Staff member, the create action processing will check if the row already exists in the database by checking whether the name that the end user entered corresponds to an existing name.

TUI: The logical key is also used to identify the maps of one map group when a Business Object is displayed on several maps. This is the case when the Business Object's width is greater than that of a map.

Note: The field defined as logical key cannot be left empty by the end user; a value is always required.

Extraction Criteria: This list presents the field(s) of the Business Object used as extraction criteria. You select one or several fields in the *Fields* list and transfer them to the *Extraction Criteria* list by clicking the corresponding >> button.

Extract Field

The *Extract Field* column displays the field used as extract criterion, according to which the end user will be able to extract data.

Retrieve Policy

The *Retrieve Policy* drop—down list allows you to specify a comparison operator that will condition the field extraction.

For example, if you want to allow the end user to extract staff members according to their names, you will define the Name field as extract field and specify the match comparison operator. The end user searching for a name will type it in the corresponding field and the rows where the name matches the typed name will be extracted and displayed to the end user.

Sort Criteria: This list presents the field(s) of the Business Object used as sort criteria. You select one or more fields and transfer them to the *Sort Criteria* list by clicking the corresponding >> button.

Sort Field

The *Sort Field* column displays the field used as sort criterion, according to which the end user will be able to sort data.

Sort Direction

The *Sort direction* drop-down list allows you to specify an order (*ascending* or *descending*) that will condition the sort of data.

Field Attributes Panel: This panel groups the specification fields that define links between a Business Object's fields and a Relational Table's columns.

Fields:

Field Name

This column displays the name of the fields presented in a Business Object. This name corresponds to the default use name you specified for the Data Element.

Table

This column indicates, for each field of the Business Object, the name of the Relational Table to which it maps.

Note: If a field is volatile - if it does not map to any table column- this column displays "*no mapping*".

Required

This drop-down list allows you to specify, for each field of the Business Object if:

- it is *required*: The end user must always enter a value. The field is initialized empty when possible.
For example, the Name field is alphanumeric and it is the logical key of the Business Object. To trigger an action, the end user must enter a name in the field. However, when opening the application, the field can be initialized empty.
- it is *optional*: the end user does not need necessarily to enter a value. The field is initialized with its default value when it has one, otherwise it is initialized empty.
- it is *required with default*: in the generated user interface, the field must always be filled in. It is initialized with a default value: with the *null* value for numeric, date, time, timestamp Data Element for which no default value has been defined, with the default value of the Data Element otherwise.

Note: For an alphanumeric Data Element, the field's default value is an empty string if no value has been defined for the Data Element.

Access Level

This drop-down list allows you to specify, for each field of the Business Object, whether it is accessed anytime an action is activated (*always*), or only when the Business Object is presented as a detail (*on details only*). This parameter is used to optimize access times.

NOTE:: The value specified for the *Access Level* parameter must be consistent with the value of the *Laid Out* parameter.

Laid Out

This drop-down list allows you to specify, for each field of the Business Object, whether the field is presented in the list and in the detail (*always*), or in neither case (*never*), only when the Business Object is presented as a detail (*on details only*) or only when the Business Object is presented as a list (*on list only*).

Updatable

This column indicates, for each field of the Business Object, whether it can be updated or not.

SQL Join Conditions Panel: This panel groups the specification fields that allow you to define join conditions between the primary and secondary tables to which the Business Object maps. The bottom part of the panel varies according to the type of WHERE clause you choose in the *Type* area.

Note: Several join conditions can be created using the same source column and target column provided that their source tables and/or target tables are different.

Type: The *Type* area radio-buttons allow you to choose whether the join condition is expressed as a *standard WHERE clause* or as a *customized WHERE clause*.

A WHERE clause is the expression in SQL language of the links that join one column of a source table and one or more columns of a target table.

The VAGTemplates Workbench provides a default initialization of a join condition:

- when you request it explicitly using the *Compute* push-button which enables you to automatically compute a join condition from the first two tables mapped.
- when you exit the Business Object Definition editor, if at least two tables have been mapped and no condition has been defined.

The following fields are displayed if you have checked the *standard WHERE clause radio-button*.

Source Table: This drop-down list allows you to select the source table of the join condition.

Source Columns: This multi-line list allows you to select the source table's column that is the source of the join condition.

Target Table: This drop-down list allows you to select the target table of the join condition.

Target Columns: This multi-line list allows you to select the target table's column that is the target of the join condition.

Add Join Field: This push button allows you to set the join condition when you have selected one source field and one target field in the *Source Columns* and the *Target Columns* lists. The condition is then added to the list below.

- The *Source field* column displays the identifier of the source of the join condition.
- The *Target field* column displays the identifier of the target of the join condition.

The following multi-line edit box is displayed if you have checked the *customized WHERE clause* radio-button.

Enter a join condition: This multi-line edit box allows you to write complex join conditions in SQL language. For example, you can create a join condition between more than two tables, or a join having less than or greater than condition between tables.

To write a complex join condition in this panel, you must write the WHERE-clause part of an SQL statement with the VisualAge Generator syntax, otherwise it will not be recognized in VisualAge Generator.

For example: We want to write a join condition between the ORG and the STAFF tables, where the Manager column of STAFF corresponds to the ID column of ORG. This simple condition can typically be defined as a standard WHERE clause. If we wanted to write it as a customized WHERE clause, the syntax would be:

T1. MANAGER = T2. ID ³

The corresponding standard SQL clause, as you would write in DB2 is:

```
(...)  
FROM STAFF, ORG  
WHERE MANAGER = ID
```

The **Compute** push-button enables you to automatically compute a join condition from the first two tables mapped.

How to Specify the Business Object Parameters

To specify the generation parameters for a Business Object, open its Generation Parameters editor from the instance Definition editor clicking on

3. T1. and T2. are the VisualAge Generator aliases for the first and the second tables called respectively in the join condition.

the *Generation Parameters* push-button, or from the VAGTemplates Browser (*Instance menu* or pop-up menu, *Generation Parameters* choice).

Generation Parameters Editor

The editor panels are already filled in with the default parameters set at initialization of the Workspace or the default parameters you set for the Business Object entity using the *Default Generation Parameters* choice from the *Entity* menu.

If you need to modify the parameters for a specific Business Object instance, you can change them here.

Note: We will not document all the parameters here but only those parameters that cannot be specified using the *Default Generation Parameters* choice from the *Entity* menu. For a detailed description of the Business Object parameters, refer to “**Default Generation Parameters**” on page 57.

General Panel:

Package/Application

This field indicates the package or the application where the generation parameters defined for the Business Object instance will be stored.

Target name

This parameter is filled in with the instance name. It is used to build the name of the components generated from this instance. If this name exceeds 5 characters, it will be truncated. Therefore you should enter a target name beginning with 5 significant characters so that it is differentiated from all other target names of the same entity.

Long target name

This field allows you to enter a character string (64 max.) that will be used as prefix of the generated Java and Smalltalk classes.

Default value: instance name

GUI Help panel ID

This parameter is filled in with the default identifier for the panel that will display help on the Business Object instance. This identifier will be used, in the final GUI client application, to call the help panel corresponding to the Business Object on which the end user requests help. The values of the Business Object panel identifiers can range from 5,000 to 9,999.

How to Specify the Business Object Extensions

Restricted to use of customized generators involving Information Model extensions.

Data Element

What is a Data Element?

A **Data Element** represents an information element stored in a Relational Table column or manipulated as a Business Object field. Its description includes a type, a length, labels, on-line help text for end-users, value checks, etc.

VAGTemplates manages five types of Data Element:

- *Alphanumeric*: a character string-type data;
- *Numeric*: an integer or decimal, signed or unsigned numeric value;
- *Date*: a date value with year, month, and day;
- *Time*: a time value with hours, minutes, and seconds;
- *Timestamp*: a concatenated date and time value used to manage versioning.

Data Element instances are automatically created by the import of the database.

*For example, if you import a database that manipulates data such as dates of birth or names, VAGTemplates will create Data Element instances such as **BIRTHDATE** (date type) and **DEPTNAME** (alphanumeric type).*

Depending on the import option you chose, VAGTemplates will either create one Data Element instance per column of the database tables or reuse the same instance whenever identical columns are common to several tables.

During the import, VAGTemplates automatically fills in some of the Data Element input fields with information from the database, like the value type and the format. Labels are also filled in by VAGTemplates. You can then complete the description of the imported Data Elements. If a field is already filled in, information coming from a new import will not overwrite the content of the field.

When creating new Data Elements, the minimum fields to be described are the *Value type* and the *Size* fields.

Default Generation Parameters

VAGTemplates includes a facility that enables you to redefine the default parameters at initialization of the Workspace.

The *Default Generation Parameters* choice from the *Entity* menu allows you to define several parameters at entity level that will be default parameters for all the entity instances within a Workspace. When defining an instance, you will only have to set the remaining parameters. You can of course modify these default parameters as you wish at instance level.

You can specify the following parameters by selecting the *Default Generation Parameters* choice from the *Entity* menu.

Default Generation Parameters Editor

General Panel:

Common Parameters:

SQL type

This drop-down list allows you to define the fine type of the Data Element. This fine type describes data more precisely. It is required by VisualAge Generator. The possible values correspond to the data types managed by VisualAge Generator (for information, refer to your VisualAge Generator documentation).

Default value: character

Target Packages/Applications Panel: This panel allows you to specify the project (VAGTemplates on Java only) and the packages/applications where the components generated from the Data Element will be stored.

Target Packages/Applications:

JAVA Target project

This field is used to specify the name of the project that includes all the target packages associated with the Data Element.

Visual package/application

This field is used to indicate the package/application where the visual components generated from the Data Element will be stored.

Logic package/application

This field is used to indicate the package/application where the logic components generated from the Data Element will be stored.

Services package/application

This field is used to indicate the package/application where the service components generated from the Data Element will be stored.

Labels Panel:

Labels:

Default label

This field allows you to specify a default label for the field that presents the Data Element in the generated application.

The default label is a label displayed next to the field in the generated application. It helps the end user identify the nature of the displayed data.

Default value: <no default value>

Column label

This field allows you to specify a label for the field that presents the Data Element in a list column.

Default value: <no default value>

Note: If you do not specify a default value, the display name will be used as the field's label, or the Data Element's name if the display name is not defined. If you do not specify a column label, the default value or the display name will be used as the field's label, or the instance name if neither are defined.

Display Panel:

Common Parameters:

Comment display

This drop-down list allows you to choose if you want the *native* value of the Data Element displayed - that is the value itself - or the *textual* value of the Data Element - that is the comment associated with the value.

Default value: native

Note: This parameter is enabled when you specify a value display that supports textual values, i.e. combos, drop-down lists and column lists.

GUI Only Parameters:

Value display

This drop-down list allows you to choose the graphical presentation you want for the Data Elements. You can choose among the following presentations:

- *horizontal radio button, vertical radio button,*
- *drop-down list,*
- *combo, dropped down combo,*
- *horizontal scale, vertical scale,*
- *static,*
- *edit, read-only edit, multi-line edit, read-only multi-line edit, formatted edit,*
- *password.*

Default value: edit

For more details on the presentations produced by these values, refer to "Part 3. Standard Use of VAGTemplates" on page 107, "Chapter 5. Standard Functions and Layouts of Generated Applications" on page 145, "Standard Layouts of GUI Client applications" on page 169, "Fields" on page 169.

Pattern

This field allows you to define a data input mask in formatted edit boxes.

For example, if the end user has to input product reference numbers, you can specify a formatted edit for this field and define a pattern such as a999"-9999"-a. The end user will be allowed to enter data that respect this format: 1 character, 3 digits, 4 digits, 1 character.

For information on the valid characters recognized by VisualAge Generator for coding an input mask in formatted edits, refer to your VisualAge Generator documentation.

Note: This parameter is enabled if the *display* parameter is set to *formatted edit*.

Max display size

This spin-edit allows you to determine a maximum width for the fields in your layouts regardless of the logical size of the Data Element. You can therefore optimize the number of fields in one layout.

If you set the Max display size parameter to 5, all the fields using this Data Element will be displayed in a 5-character long edit box, for example. To see the end of a data field value, the end-user will have to scroll the field with the right arrow key. Default value: 99999

Note: If the logical size of the Data Element (*size* parameter for alphanumeric Data Elements, or *capacity* and *precision* parameters for numeric Data Elements) is smaller than the maximum display size, the latter is ignored and the presentation adjusts to the logical size.

Horizontal Orientation

The *Horizontal orientation* drop-down list allows you to define the orientation of the graphic interface, with two possible values: *left-to-right* and *right-to-left*.

You can define the horizontal orientation for a Workspace or for a Data Element.

Default value: left to right

How to Define a Data Element

Use the Data Element Definition editor to update the definition of a Data Element pre-filled during the import or to define a Data Element that you created.

To create a new Data Element instance and open its Definition editor:

1. Select the Data Element entity, select *New...* from the *Instance* menu,
2. Enter the name of the instance, making the first 5 characters significant,

3. In the *Package/Application* combo box, select a package or an application where the instance functional description will be stored,
4. Select the *Open now* check box,
5. Click *OK*.

To open the Definition editor of an already existing instance of Data Element:

1. Select the Data Element entity,
2. Select the instance to edit,
3. Select *Definition* from the *Instance* menu.

Definition Editor

This editor groups the specification fields that manage the logical description of a Data Element instance.

General Panel:

Package/Application:

Package/Application

This field indicates the package or application where the functional description of the Data Element instance is stored.

Names:

Default use name

This field allows you to enter the default name which will be automatically proposed when this instance is used by another instance. It appears in the *Fields* list in the *Tables and Fields* panel when you add a field to a Business Object, for example.

Value: an alphanumeric string of 1 to 32 characters.

Display name

This field allows you to enter a display name. It will appear, in the generated applications, in the list of Data Elements in the Business Object help panel if this Data Element is called in a Business Object.

Value: an alphanumeric string of 1 to 64 characters.

Note: The display name will be used as the field's label in the generated application if you do not specify a default label.

For information on default labels, refer to the description of the 77.

Type: The *Type* area varies according to the type of the Data Element, that is the value set for the *Value type* field.

Value type

All Data Elements — This drop-down list allows you to specify whether the value is an *alphanumeric*, a *numeric*, a *date*, a *time* or a *timestamp* value.

Default value: alphanumeric

TUI: In TUI applications, the fields can only be numeric or alphanumeric. The other formats are considered alphanumeric in VisualAge Generator.

Size

For alphanumeric Data Elements — The *Size* spin edit allows you to define the length (in number of characters) of the Data Element.

Default value: 1

Case control

For alphanumeric Data Elements — The *Case control* drop-down list allows you to specify whether the values that the end user enters in this field will be changed into uppercase (*uppercase*), lowercase (*lowercase*) or left as inputted (*none*).

Default value: none

TUI: Only the *none* and *uppercase* values are taken into account. *Lowercase* is considered *none*.

Capacity

For numeric Data Elements — The *Capacity* spin edit allows you to define the number of digits allowed before the decimal point.

Default value: 1

Value: 0 to 99 999 999 999

Precision

For numeric Data Elements — The *Precision* spin edit allows you to define the number of digits allowed after the decimal point.

Default value: 0

Value: 0 to 99 999 999 999

Value style

For all Data Elements but alphanumeric ones — This drop-down list allows you to attach an instance of the Value Style entity to the Data Element.

Note: This parameter is disabled if no Value Style instance has been defined (for information, refer to “**Value Style**” on page 102).

Descriptions Panel:

Descriptions:

Textual description

This multi-line edit allows you to enter a short description of the instance.
Value: an alphanumeric string of 1 to 1023 characters.

TIP: This field is not used in generating in the final application. You can use it to communicate technical information about the instance to other developers.

On line help description

This multi-line edit allows you to enter a text that will appear when the end user requests help on a field using this Data Element instance.

Value: an alphanumeric string of 1 to 1023 characters.

For information on generated on-line help, refer to “Part 3. Standard Use of VAGTemplates” on page 107, “Chapter 5. Standard Functions and Layouts of Generated Applications” on page 145, “**Standard Functions**” on page 146, “**On-Line Help**” on page 160.

Default Value Panel:

Default Value:

Default value mode

This drop-down list allows you to initialize a default value for the Data Element or no default value (*none*). The default value can be a *constant* value or a *system* value (system date, time or timestamp).

If a default value is specified for a field in the imported database, the *default value mode* parameter is set accordingly.

Default value: none

Note: The default value that will be used for a field in the generated application depends on its type:

- The Data Element is *alphanumeric*:
 - The default value mode is set to *none*; the field’s default value is an empty string.
 - The default value mode is set to *constant*; the field’s default value is the default value you specified when customizing the Data Element.
 - The *system* value is irrelevant for alphanumeric fields.
- The Data Element is *numeric*:
 - The *system* value is irrelevant for numeric fields.
 - The default value mode is set to *constant*; the field’s default value is the default value you specified when customizing the Data Element.

- The default value mode is set to *none*; the field’s default value is an empty string.
- The Data Element is *date*, *time*, or *timestamp*:
 - The default value mode is set to *none*; the field’s default value is an empty string.
 - The default value mode is set to *constant*; the field’s default value is the default value you specified when customizing the Data Element.
 - The default value mode is set to *system*; the field’s default value is the system date, time or timestamp.

Value

The *Value* field allows you to enter a value that will be used as the default value at initialization of a new field in the generated application.

Note: A default value is required for numeric, date, time and timestamp Data Elements.

Note: This parameter depends on the value specified for the *Default value mode* parameter. It is ignored if the *Default value mode* parameter is set to *none* or *system*.

Comment

The *Comment* field allows you to enter a comment that describes the value.

Check Type Panel: The panel contains only the *Check type* field when the value specified for this field is *no check* or *customized check*.

It additionally contains the *Value Table* area if the value specified for the *Check type* field is *value table* or the *Interval* area if the value specified for the *Check type* field is *interval*.

Check type

This drop-down list allows you to specify whether the value is checked by an *interval*, a *value table*, a *customized check* or if *no check* is defined on the value.

If you specify a customized check, at generation time, insertion points are created in the generated parts for you to insert specific code to implement your customized check (see “Part 3. Standard Use of VAGTemplates” on page 107, “Chapter 6. Application Generation and Enhancement” on page 225, “Generated Architecture and Principles” on page 240, “Hook on the Client Side of an Application” on page 262).

Default value: no check

Value Table: This area allows you to customize the value table.

To add a value or a comment, right click in the corresponding column and select *Add* from the pop-up menu.

To remove a value or a comment, right click in the corresponding column and select *Remove* from the pop-up menu.

Value

This column allows you to enter the authorized values.

Comment

This column allows you to enter comment for each authorized value.

Interval: This area is used to customize the interval.

Type

The *Type* drop-down list allows you to define whether the interval includes its bounds or not.

Default value: closed-closed

Minimum

The *Minimum* field allows you to enter the minimum value of the interval.

Default value: <no value> (alphanumeric value type)

0.0 (numeric value type)

<current system date> (date value type)

<current system time> (time value type)

<current system timestamp> (timestamp value type)

Maximum

The *Maximum* field allows you to enter the maximum value of the interval.

Default value: <no value> (alphanumeric value type)

0.0 (numeric value type)

<current system date> (date value type)

<current system time> (time value type)

<current system timestamp> (timestamp value type)

How to Specify the Data Element Parameters

To specify the generation parameters for a Data Element, open its Generation Parameters editor from the instance Definition editor clicking on the *Generation Parameters* push-button, or from the VAGTemplates Browser (*Instance menu* or pop-up menu, *Generation Parameters* choice).

Generation Parameters Editor

The Generation Parameters editor panels are already filled in with the default parameters that were set at initialization of the Workspace, or that you set for the Data Element entity using the *Default Generation Parameters* choice from the *Entity* menu.

However, if you need to modify the parameters for a specific Data Element, you can change them here.

Note: We will not document all the parameters here but only those parameters that cannot be specified using the *Default Generation Parameters* choice from the *Entity* menu. For a detailed description of the Data Element parameters, refer to “**Default Generation Parameters**” on page 76.

General Panel:

Package/Application

This field indicates the package or the application where the generation parameters of the Data Element instance will be stored.

Target name

This parameter is filled in with the instance name. It is used to build the name of the components generated from this instance. If this name exceeds 10 characters, it will be truncated. Therefore you should enter a target name beginning with 10 significant characters so that it is differentiated from all other target names.

Long target name

This field allows you to enter a character string (64 max.) that will be used as prefix of the generated Java and Smalltalk classes.

Default value: instance name

GUI Help panel ID

The parameter is filled in with the default identifier of the panel that will display help on the Data Element instance. This identifier will be used in the final GUI client application to call the help panel corresponding to the field on which the end user requests help. The values of the Data Element panel identifiers can range from 10,000 to 30,000.

How to Specify the Data Element Extensions

Restricted to use of customized generators involving Information Model extensions.

Interface Unit

What is an Interface Unit?

An **Interface Unit** is the interface that presents Business Objects and specifies whether they are used as *details* or as *lists*.

- A detail Business Object is used to represent one row from one (or more) table.
- A list Business Object is used to represent many rows from one (or more) table.

The Interface Unit is also used to define navigation between Interface Units.

In the generated GUI client applications, the Interface Unit will produce a graphical window; in the generated TUI applications, it will produce a 3270 screen.

To define an Interface Unit you have to:

- create an Interface Unit,
- indicate which Business Objects the Interface Unit contains, and indicate if it is used as a detail or as a list for each Business Object,
- indicate which other Interface Unit(s) your Interface Unit can display.

Default Generation Parameters

VAGTemplates includes a facility that enables you to redefine the default parameters set at initialization of the Workspace.

The *Default Generation Parameters* choice from the *Entity* menu allows you to define several parameters at entity level that will be default parameters for all the entity instances. When defining an instance, you only have to set the remaining parameters. You can of course modify these default parameters as you wish at instance level.

Default Generation Parameters Editor

For information on the presentations produced by the following parameters, refer to “Part 3. Standard Use of VAGTemplates” on page 107, “Chapter 5. Standard Functions and Layouts of Generated Applications” on page 145, “Standard Layouts of GUI Client applications” on page 169, “Windows” on page 196.

General Panel:

Common Parameters:

Title

This field allows you to specify a title for the generated window (GUI) or map (TUI).

Note: If you do not specify a title, the Interface Unit name will be used as the generated window (GUI) or map (TUI) title.

TUI Only Parameters:

Fastpath

This field allows you to specify a code for the generated map that the end user can use to reach it directly from another map. In the generated application, this code will appear in the root map next to the title of the child map and on top of the current map.

Default value: <first six characters of the Interface Unit's name>

Caution: Be careful not to give the same fastpath to different maps.

Target Packages/Applications Panel: This panel allows you to specify the project (VAGTemplates on Java only) and the packages/applications where the components generated from the Interface Unit will be stored.

Target Packages/Applications:

JAVA Target project

This field is used to specify the name of the project that includes all the target packages associated with the Interface Unit.

Visual package/application

This field is used to indicate the package/application where the visual components generated from the Interface Unit will be stored.

Logic package/application

This field is used to indicate the package/application where the logic components generated from the Interface Unit will be stored.

Services package/application

This field is used to indicate the package/application where the service components generated from the Interface Unit will be stored.

GUI Window Panel:

GUI Window: This area groups the parameters that define the presentation of a GUI Interface Unit and the layout of the Business Objects called in this Interface Unit.

Interface unit display

This drop-down list allows you to define the graphical aspect of the generated window: *notebook²*, *Windows notebook*, *PM notebook* or *normal*.

Default value: normal

Business object layout

This drop-down list allows you to define the arrangement of the Business Object layouts in the generated window.

You can choose between a vertical arrangement where the Business Objects are positioned from *top to bottom* then from left to right, or a horizontal arrangement where the Business Objects are positioned from *left to right* then from top to bottom.

Default value: top to bottom

NOTE: With *right to left* orientation, you can choose between a vertical arrangement where the Business Objects are positioned from *top to*

bottom then from right to left, or a horizontal arrangement where the Business Objects are positioned from *right to left* then from top to bottom.

Minimize button

If this option is checked, it will be possible to minimize the generated window.

Default value: true

Maximize button

If this option is checked, it will be possible to maximize the generated window.

Default value: true

Resize button

If this option is checked, it will be possible to resize the generated window.

Default value: true

Menu Titles Panel: This panel groups the parameters that define the labels of menus that depend on the Interface Unit.

Menu Titles:

Menu label

This field allows you to specify the menu label that the end user selects to display the target Interface Unit.

Windows menu title

This field allows you to specify a title for the menu item from which the end user will display open windows.

Default value: Windows

Navigation menu title

This field allows you to specify a title for the menu item that the end user will use to display target windows.

Default value: Navigation

Help menu title

This field allows you to specify a title for the menu item from which the end user will call on-line help.

Default value: Help

Edit menu title

This field allows you to specify a title for the menu item from which the end user will activate Copy, Cut and Paste actions on input areas.

Default value: Edit

Edit Menu Panel: This panel groups the parameters that define the Edit menu actions' labels.

Edit Menu:

Cut label

This field allows you to specify a title for the menu option from which the end user will activate a cut action.

Default value: Cut

Copy label

This field allows you to specify a title for the menu option from which the end user will trigger a copy action.

Default value: Copy

Paste label

This field allows you to define the title of the menu option from which the end user will trigger a paste action.

Default value: Paste

How to Define an Interface Unit

Use the Interface Unit Definition editor to define an Interface Unit.

To create a new instance of Interface Unit and open its Definition editor:

1. Select the Interface Unit entity, select *New...* from the *Instance* menu,
2. Enter the name of the instance, making the first 5 characters significant,
3. In the *Package/Application* combo box, select a package or an application where the instance functional description will be stored,
4. Select the *Open now* check box,
5. Click *OK*.

To open the Definition editor of an already existing Interface Unit instance:

1. Select the Interface Unit entity,
2. Select the instance to edit,
3. Select *Definition* from the *Instance* menu.

Definition Editor

This editor groups the specification fields that manage the logical description of the Interface Unit.

General Panel:

Package/Application:

Package/Application

This field indicates the package or the application where the Interface Unit instance functional description is stored.

Names:

Default use name

This field allows you to enter the default name that is to be automatically proposed when this instance is used by another instance. It appears when you add this Interface Unit to another Interface Unit's list of Target Interface Units.

Value: an alphanumeric string of 1 to 32 characters.

Display name

This field allows you to enter a display name. It will be displayed, in the generated application, in the list of target windows (GUI) or maps (TUI) in the help panel of the parent window or map.

Value: an alphanumeric string of 1 to 64 characters.

Type:

Type

These radio buttons allow you to specify whether the generated window (GUI) or map (TUI) will be the entry point of the generated application from which the end user will navigate (*root*) or a standard window or map (*simple*).

Default value: simple.

Descriptions Panel:

Descriptions:

Textual description

This multi-line edit allows you to enter a comment on the instance.

Value: an alphanumeric string of 1 to 1023 characters.

TIP: This field is not used when generating the final application. You can use it to communicate technical information about the instance to other developers.

On line help description

This multi-line edit allows you to enter a help text that will appear when the end user requests help on the Interface Unit instance.

For information on generated on-line help, refer to "Part 3. Standard Use of VAGTemplates" on page 107, "Chapter 5. Standard Functions and Layouts of Generated Applications" on page 145, "**Standard Functions**" on page 146, "**On-Line Help**" on page 160.

Business Objects Panel: This panel allows you to define the Business Objects called by the Interface Unit and their presentation, and to specify Target Interface Units.

Business Objects:

Business Object

This column allows you to add the Business Object instance(s) called in the Interface Unit.

Layout Type

This column indicates whether a Business Object is used as a *detail*, as a *list*, or as an *updatable list*.

Component Name

This column indicates the name of the Business Object used in the Interface Unit. It corresponds to the default use name you specified when defining the Business Object. You can modify it here by clicking on it and typing the new name.

For example, if you use the same Business Object twice in an Interface Unit, you can assign a different name to each use.

To add a Business Object to the list of Business Objects, right click in the *Business Objects* list area and select **Add** from the pop-up menu or click on the *Add...* push-button. In the *Add Business Objects* window, select the Business Object you want to add from the drop-down list then click **OK**.

Note: If you check the *Filter* option, the instances in the *Add Business Objects* window are displayed according to the packages/applications you selected in the VAGTemplates Browser.

You can also add to the list, a Business Object that is not defined yet. In the *Add Business Objects* window, click on the *New Business Object* push-button. The *New VAGT Instance* window opens up, allowing you to create a Business Object. As the creation finishes, the *New VAGT Instance* window closes. Select the new Business Object in the *Add Business Objects* window to add it to the *Business Objects* list.

To remove a Business Object from the list, select the Business Object, right click in the *Business Objects* list area and select **Remove** from the pop-up menu or click on the *Remove* push-button.

To view the description of an Business Object, right click on its name in the column and select the **Open Business Object** choice from the pop-up menu.

Target Interface Units Panel:

Target Interface Units:

Interface Unit

This column allows you to add the Interface Unit instances that are called by the current Interface Unit. This defines the navigation from one generated window (GUI) or map (TUI) to the next.

Note: The current Interface Unit instance cannot be specified as target Interface Unit of itself.

Component Name

This column indicates the default use name of the Interface Unit. It corresponds to the default use name you specified when defining the Interface Unit. You can modify it here by clicking on it and typing the new use name.

To add an Interface Unit in the list of Interface Units, right click in the *Interface Units* list area and select **Add** from the pop-up menu or click on the *Add...* push-button. In the *Add Interface Units* window, select the Interface Unit you want to add from the drop-down list and select **OK**.

Note: If you check the *Filter* option, the instances in the *Add Interface Units* window are displayed according to the packages/applications you selected in the VAGTemplates Browser.

You can also add to the list, an Interface Unit that is not defined yet. In the *Add Interface Units* window, click on the *New Interface Unit* push-button. The *New VAGT Instance* window opens up, allowing you to create an Interface Unit. As the creation finishes, the *New VAGT Instance* window closes. Select the new Interface Unit in the *Add Interface Units* window to add it to the *Interface Units* list.

To remove an Interface Unit from the list, select the Interface Unit, right click in the *Interface Units* list area and select **Remove** from the pop-up menu or click on the *Remove* push-button.

To view the description of a target Interface Unit, right click on its name in the column and select the **Open Interface Unit** choice.

How to Specify the Interface Unit Parameters

To specify the generation parameters for an Interface Unit, open its Generation Parameters editor from the instance Definition editor clicking on the *Generation Parameters* push-button, or from the VAGTemplates Browser (**Instance menu** or pop-up menu, **Generation Parameters** choice).

Generation Parameters Editor

The **Generation Parameters** editor panels are already filled in with the default parameters set at initialization of the Workspace or the default parameters you set for the Interface Unit entity using the *Default Generation Parameters* choice from the *Entity* menu.

If you need to modify the parameters for a specific Interface Unit, you can change them here.

Note: We will not document all the parameters here but only those parameters that cannot be specified using the *Default Generation Parameters* choice from the *Entity* menu. For a detailed description of all Interface Unit parameters, refer to “**Default Generation Parameters**” on page 86.

General Panel:

Package/Application

This field indicates the package or the application where the generation parameters defined for the Interface Unit instance will be stored.

Target name

This parameter is filled in with the instance name. It is used to build the name of the components generated from this instance. If this name exceeds 5 characters, it will be truncated. Therefore you should enter a target name beginning with 5 significant characters so that it is differentiated from all other target names.

Long target name

This field allows you to enter a character string (64 max.) that will be used as prefix of the generated Java and Smalltalk classes.

Default value: instance name

GUI Help panel ID

This parameter is filled in with the default identifier for the panel that will display help on the Interface Unit instance. This identifier will be used, in the final application, to call the help panel corresponding to the GUI window on which the end user requests help. The values of the Interface Unit panel identifiers can range from 2,000 to 4,999.

How to Specify the Interface Unit Extensions

Restricted to use of customized generators involving Information Model extensions.

Relational Table

What is a Relational Table?

A **Relational Table** is either a *Table*, comprising a list of columns, or a *View*, comprising several columns extracted from one or more Tables.

Relational Table instances are automatically created by the import function. You cannot create Relational Table instances in the Workbench.

The created Relational Table instances have the same structure as the imported database tables from which they originated. You can consult their definition and update the description of the Data Element instances called in the table's columns.

The Relational Table entity is also used to generate help lists in the final application.

For information on help lists, refer to "Part 3. Standard Use of VAGTemplates" on page 107, "Chapter 5. Standard Functions and Layouts of Generated Applications" on page 145, "**Standard Functions**" on page 146, "**Management of Persistent Data**" on page 147, "**Actions Available for Help Lists**" on page 151.

Default Generation Parameters

VAGTemplates includes a facility that enables you to redefine the default parameters set at initialization of the Workspace.

The *Default Generation Parameters* choice from the *Entity* menu allows you to define several parameters at entity level that will be default parameters for all the entity instances within a Workspace. When defining an instance, you only have to set the remaining parameters. You can of course modify these default parameters as you wish at instance level.

Default Generation Parameters Editor

General Panel:

Common Parameters:

Concurrency management

This drop-down list allows you to specify a control on the concurrent accesses to a database. Values can be:

- *column*: the consistency check is based on the value you specified for the *Concurrency management column* parameter in the Relational Table Definition.

- *read*: the update is performed only if all read fields are identical with those in the database, whatever value you specified for the *Concurrency management column* parameter.
- *none*: no concurrency management. The value specified in the *Concurrency management column* parameter in the Relational Table Definition is then not taken into account.

Default value: column

SQL Qualified

This check box allows you to specify whether you want the generated table names to be prefixed with a qualifier code or not. The *SQL Qualifier* must not be empty if *isSqlQualifier* is set to *true* for a table.

Default value: unchecked

Note: The qualifier and the name are defined in the database.

Table qualifier

This field allows you to specify the qualifier code that prefixes the table name in the database.

Default value: Table qualifier in the database

Note: This parameter is enabled when the *SQL Qualified* parameter is checked.

Target Packages/Applications Panel: This panel allows you to specify the project (VAGTemplates on Java only) and the packages/applications where the components generated from the Relational Table will be stored.

Target Packages/Applications:

JAVA Target project

This field is used to specify the name of the project that includes all the target packages associated with the Relational Table.

Visual package/application

This field is used to indicate the package/application where the visual components generated from the Relational Table will be stored.

Logic package/application

This field is used to indicate the package/application where the logic components generated from the Relational Table will be stored.

Services package/application

This field is used to indicate the package/application where the service components generated from the Relational Table will be stored.

Foreign Key Help List Panel:

Foreign Key Help List:

Number of rows to fetch

This spin edit allows you to define how many rows per page the help list generated from the table will display.

Default value: 20

GUI *Top label*

This field allows you to specify the label of the menu item or push button that triggers an action to reach the top of the help list.

Default value: Top

GUI *Forward label*

This field allows you to specify the label of the menu item or push button that triggers an action to reach the next page of the help list.

Default value: Forward

GUI *Backward label*

This field allows you to specify the label of the menu item or push button that triggers an action to reach the previous page of the help list.

Default value: Backward

How to Define a Relational Table

Use the Relational Table Definition editor to define a Relational Table. To open the Definition editor of a Relational Table instance:

1. Select the Relational Table entity,
2. Select the instance to edit,
3. Select *Definition* from the *Instance* menu.

Definition Editor

This editor groups the specification fields that manage the logical description of the Relational Table.

General Panel:

Package/Application:

Package/Application

This field indicates the application where the Relational Table instance functional description is stored.

Names:

Default use name

This field allows you to enter the default name that is to be automatically proposed when this instance is used by another instance. It appears when you add this Interface Unit to another Interface Unit's list of Target Interface Units.

Value: an alphanumeric string of 1 to 32 characters.

Display name

This field allows you to enter a display name. It will appear, in the generated application, in the list of target windows (GUI) or maps (TUI) in the help panel of the parent window or map.

Value: an alphanumeric string of 1 to 64 characters.

Note: This field is not use by the standard generators.

Table name

The *Table name* field display the table's name according to its generated code.

Type:

Table type

The *Type* field indicates whether the instance is a *table* or a *view*.

Descriptions Panel:

Textual description

This multi-line edit allows you to enter a short description of the instance.

Value: an alphanumeric string of 1 to 1023 characters.

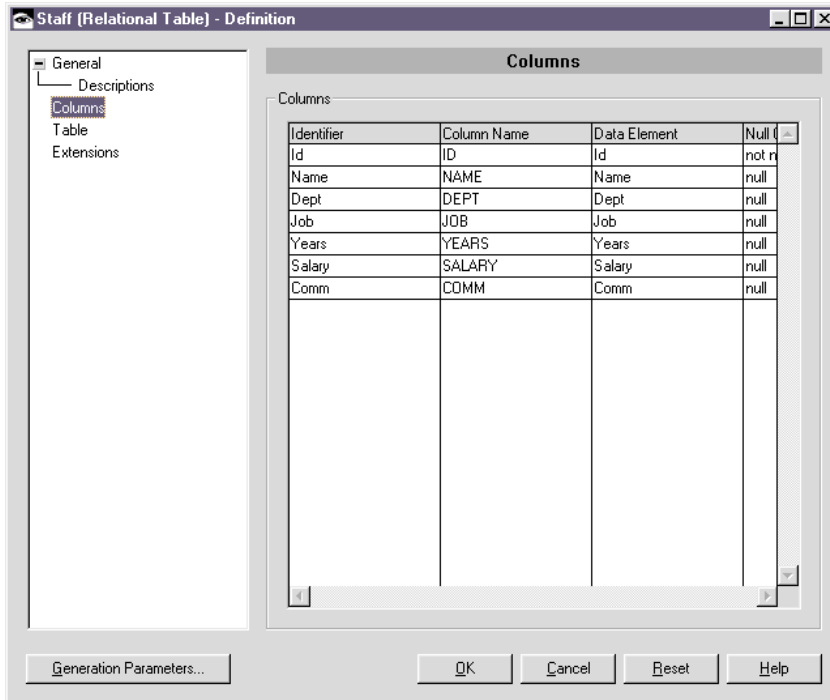
TIP: This field is not used in generating in the final application. You can use it to communicate technical information about the instance to other developers.

On-line help description

This multi-line edit cannot be inputted because no specific on-line help will be generated for Relational Table instances.

For information on generated on-line help, refer to "Part 3. Standard Use of VAGTemplates" on page 107, "Chapter 5. Standard Functions and Layouts of Generated Applications" on page 145, "**Standard Functions**" on page 146, "**On-Line Help**" on page 160.

Columns Panel: This panel groups the specification fields that define the Relational Table composition.



Columns: The *Columns* list allows you to visualize the columns of the imported relational table.

Identifier

The *Identifier* column displays the identifiers of the table's column.

Column Name

The *Column Name* column displays the name of the columns according to its generated code.

Data Element

The *Data Element* column displays the Data Elements called by the table's columns.

Null Control

The *Null Control* column displays the value defined in the Data Element. Values can be: *null*, *not null* and *not null with default*.

Updatable

This column indicates whether the Data Element can be updated or not.

Table Panel: This panel is available when the Relational Table type is *table*. If the Relational Table type is *view*, you find the *View* panel instead.

Concurrency Management Column:

Column

The *Column* drop-down list allows you to specify the column that will be used to manage concurrent accesses to the database. The column selected as *concurrency column* must not be mapped in the Business Object.

Keys:

Keys

The *Keys* list allows you to visualize the keys of the imported Relational Table.

When the table has a primary key, the '*primary*' label appears in the list. If you click on this label, the Data Element that is the table's primary key will be highlighted in the *Columns* list.

When the table has foreign keys, the name of the foreign key appears in the list.

When you right click in the *Keys* list area, a popup menu opens up, allowing you to add a primary or a secondary key, or to remove the selected key.

Note:

1. When you create a key, the corresponding link is automatically created and displayed in the *Links* area. By default, the link source key is the key you have just created.
2. When you remove a key, all the links that have this key as source key are automatically deleted. You can select several keys to remove them all at once.

Columns:

Identifier

The *Identifier* column displays the identifiers of the table's columns.

Data Element

The *Data Element* column displays the Data Elements called by the table's columns.

When you right click in the *Columns* list area, a popup menu opens up, allowing you to add or to remove a column to/from the key selected in the *Keys* list area.

- To remove one or more columns, just select them in the list, then select the ***Remove from Key*** choice from the popup menu.
- To add a column:
 1. Select one or more columns.
 2. From the popup menu, select the ***Add to Key*** choice. The ***Key - Add Columns*** window opens up.

3. Select the columns you want in the list of the Table's columns, then press *OK*.

Links: The *Links* list allows you to visualize the links that exist between the imported relational tables. The values here stem from the database import.

Identifier

The *Identifier* column lists the identifiers of the links.

Source key

The *Source key* column lists the name of the link's source key.

Target table

The *Target table* column lists the name of the link's target table.

Update rule

The *Update rule* column lists the rules that manage the update of the link's source and target.

Delete rule

The *Delete rule* column lists the rules that manage the deletion of the link's source and target.

Note: The relational link is always defined at the primary table level (the primary table being the table where the foreign key is defined). No source table is indicated as the source of the link because it is always a foreign key of the primary table. No target key is indicated as the target of the link because it is always the primary key of the secondary table.

If you right click in the Links list area, a popup menu opens up, allowing you to add or remove a link, or to open the selected Target Table.

Note: When you create a link, the source key column is automatically filled in with the last key displayed in the *Keys* list.

View Panel: This panel is available when the Relational Table type is *view*. If the Relational Table type is *table*, you find the *Table* panel instead.

CRUD Properties: The *CRUD Properties* check boxes allow you to specify whether the fields in the table columns can be modified (*Insertable*), updated (*Updatable*) or deleted (*Deletable*).

Mapping: The *Mapping* list allows you to visualize the tables that are linked to compose the view.

Mapper

The *Mapper* column lists the identifier of the current view's column.

Mapped table

The *Mapped table* column lists the identifier of the linked table.

Mapped column

The *Mapped column* column lists the identifier of the linked table's column.

Join Condition: The *Join condition* multi-line edit describes the join condition that links the tables to make up a view. This join condition is expressed as an SQL standard WHERE clause.

For information on join conditions, refer to topic "How to Define a Business Object" on page 67.

How to Specify the Relational Table Parameters

To specify the generation parameters for a Relational Table, open its Generation Parameters editor from the instance Definition editor clicking on the *Generation Parameters* push-button, or from the VAGTemplates Browser (*Instance menu* or pop-up menu, *Generation Parameters* choice).

Generation Parameters Editor

The editor panels are already filled in with the default parameters set at initialization of the Workspace or the default parameters you set for the Relational Table entity using the *Default Generation Parameters* choice from the *Entity* menu. If you need to modify the parameters for a specific Relational Table, you can change them here.

Note: We will not document all the parameters here but only those parameters that cannot be specified using the *Default Generation Parameters* choice from the *Entity* menu. For a detailed description of the other Relational Table parameters, see topic "Default Generation Parameters" on page 94.

General Panel:

Package/Application

This field indicates the package or the application where the generation parameters defined for the Relational Table instance will be stored.

Target name

The parameter is filled in with the instance name. It is used to build the name of the components generated from this instance. If this name exceeds 5 characters, it will be truncated. Therefore you should enter a target name beginning with 5 significant characters so that it is differentiated from all other target names.

Long target name

This field allows you to enter a character string (64 max.) that will be used as prefix of the generated Java and Smalltalk classes.

Default value: instance name

Foreign Key Help List Panel:

Additional help field name

The parameter indicates the name of a field that should be taken into account when generating help list service.

For Standard Generators, when you specify the name of a column in this parameter, the help list server and client parts will take into account this field. It will be read in the database and handled on the client side (new entry in the WPage record (list data), new getter, etc.). This parameter should be used when you do not wish to display the keys corresponding to the foreign key, but a comment from another column. The generated parts will contain the access for this field. You will then be able to customize your layouts in order to display this extra field as you wish.

How to Specify the Relational Table Extensions

Restricted to use of customized generators involving Information Model extensions.

Value Style

What is a Value Style?

A **Value Style** is a set of characteristics defining a presentation style for a numeric, a date, a time, or a timestamp value. It holds the display and input characteristics that such values can have in the generated applications. These characteristics can be shared by several Data Elements regardless of their logical descriptions, like the capacity and the precision for numeric Data Elements.

The same Value Style allows you to standardize the presentation of several Data Elements of the same type. VAGTemplates manages four types of Value Styles (Numeric, Date, Time, Timestamp), one for each type of Data Element likely to have a Value Style.

For example, you can define an instance of Value Style associating the \$ sign to numeric values, and another instance associating the £ sign to numeric values. All the values of a Data Element instance calling the former Value Style will be presented with a \$ sign, whereas all the values of a Data Element instance calling the latter Value Style will be presented with a £ sign.

How to Define a Value Style

To create a new instance of Value Style and open its Definition editor:

1. Select the Value Style entity, select *New...* from the **Instance** menu,
2. In the *Package/Application* combo box, select a package or an application where the instance functional description will be stored,
3. Enter the name of the instance, making the first 5 characters significant,
4. Select the *Open now* check box,

5. Click *OK*.

To open the editor of an already existing instance of Value Style:

1. Select the Value Style entity,
2. Select the instance to edit,
3. Select *Definition* from the *Instance* menu.

Note: As it is not possible to specify generation parameters for the Value Style entity or instances, the Default Generation Parameters and Generation Parameters editors are not available for this entity.

Definition Editor

This editor groups the specification fields that manage the logical description of the Value Style.

General Panel:

Package/Application:

Package/Application

This field indicates the package or application where the Value Style instance functional description is stored.

Names:

Default use name

This field allows you to enter the default name which will be automatically proposed when this instance is used by another instance.

Value: an alphanumeric string of 1 to 32 characters.

Display name

This field allows you to enter a display name for the Value Style.

Value: an alphanumeric string of 1 to 64 characters.

Note: This field is not use by the standard generators.

Type:

Value type

The *Type* drop-down list allows you to specify whether the value style is defined for *numeric*, *date*, *time* or *timestamp* values.

Descriptions Panel:

Textual description

This multi-line edit allows you to enter a comment corresponding to the instance.

Value: an alphanumeric string of 1 to 1023 characters.

TIP: This field is not used in generating in the final application. You can use it to communicate technical information about the instance to other developers.

On line help description

The multi-line edit field cannot be input as no specific On-line Help will be generated for Value Style instances.

For information on generated on-line help, refer to “Part 3. Standard Use of VAGTemplates” on page 107, “Chapter 5. Standard Functions and Layouts of Generated Applications” on page 145, “**Standard Functions**” on page 146, “**On-Line Help**” on page 160.

Value Style Panel: This panel groups the specification fields that define the Value Style type. The panel varies according to the value specified for the *Value type* field from the *General* panel.

1. The *Value type* is *numeric*. The *Value Style* panel displays the following parameters:
 - **GUI** The *Decimal separator* drop-down list allows you to specify either a period or a comma to separate the integer part from the decimal part in a decimal number.
Default value: . (period)
 - **GUI** The *Thousand separator* drop-down list allows you to specify either a comma, a period, a blank character [space] or no character [none] to indicate the thousands in a number.
Default value: , (comma)
 - **GUI** The *Positive sign* field allows you to specify what sign will indicate that a number is positive.
No default value
 - **GUI** The *Negative sign* field allows you to specify what sign will indicate that a number is negative.
No default value
 - The *Sign position* drop-down list allows you to choose whether the sign is to the left or to the right of the value, or if no sign is displayed.
Default value: left
 - **GUI** The *Unit* field allows you to specify what symbol will indicate the units of the displayed data.
No default value
 - **GUI** The *Unit position* drop-down list allows you to choose whether the units is to the *left*, or to the *right* of the value.
Default value: right

- **GUI** The *Unit and sign alignment* drop-down list allows you to choose whether the sign and units both stick to the value (*no separation*), or if only the units is separated from the value (*units separation*), or if sign and units are both separated from the value (*units and sign separation*).

*Default value:*unit and sign separation

2. The *Value Style type* is *date*. The *Value Style* panel displays the following parameters:

- The *Mask* drop-down list allows you to specify the order in which day, month and year must be presented. You can choose between *dmy* (<Day> <Month> <Year>), *mdy* (<Month> <Day> <Year>), *ymd* (<Year> <Month> <Day>).

*Default value:*mdy

- The *Separator* field allows you to indicate what character separates the different elements of the date.

No default value

- The *Year style* field allows you to determine whether the year includes the century (*full*) or not (*short*).

*Default value:*full

For example, if you choose the mdy mask, - as separator and the full year style, the dates will be displayed as follows: "11-24-1997".

3. The *Value Style* is a *time*. The *Value Style* panel opens presenting the following choices:

- **GUI** The *Mask* drop-down list allows you to specify the presentation of dates. You can choose between *full time* (<Hours> <minutes> <seconds>) or *no seconds* (<Hours> <minutes>).

*Default value:*full time

- **GUI** The *Cycle* radio buttons allow you to determine whether the time cycle will be *military* (24 hours) or *AM/PM* (12 hours). In the latter case, the time is followed by text indicating the half day (e.g. "AM" or "PM"). The presentation of this text depends on the *Am string* and *Pm string* parameters (see below).

*Default value:*military

- **GUI** The *Separator* field allows you to indicate what character separates the different elements of the time.

No default value

- **GUI** The *Am string* field allows you to specify the characters that will indicate the half day before midday (ante meridian).

*Default value:*AM

Note: This parameter is available if you choose *AM/PM* for the *Cycle* parameter.

- **GUI** The *Pm string* field allows you to specify the characters that will indicate the half day after midday (post meridian).

Default value: PM

Note: This parameter is relevant if you choose *AM/PM* for the *Cycle* parameter.

For example, if you choose the no seconds mask, the AM/PM cycle, the colon as separator, and AM and PM as half day indicators, the times will be displayed as follows: "10:45 AM".

•

TUI: By default VisualAge Generator presents the time values according to the standards in the language and country for which it is installed.

For example, if VisualAge Generator is installed for France, it will display times as: 18:35:20.

4. **GUI** The Value Style type is *timestamp*. The *Value Style* panel displays the following parameters:

- **Date identifier**

The *Date identifier* drop-down list allows you to choose the date Value Style that is to be concatenated with a time Value Style to form the timestamp.

- **Date Style**

The *Mask*, *Separator* and *Year style* fields in the *Date Style* area are read-only.

- The *Separator* field allows you to specify the character that will separate the date from the time.

No default value

- **Time Identifier**

The *Time Identifier* drop-down list allows you to choose the time Value Style that is to be concatenated with a date Value Style to form the timestamp. The characteristics of the chosen time Value Style are read-only.

- **Time Style**

The fields contained in the *Time Style* area are read-only.

How to Specify the Value Style Extensions

Restricted to use of customized generators involving Information Model extensions.

Part 3. Standard Use of VAGTemplates

Chapter 4. Exploring VAGTemplates Basic

Functions	111
Presenting a List and a Detail in the Same Window (GUI) or Map (TUI).	112
Creating an Interface Unit Presenting a List and a Detail	114
Defining the Interface Unit	114
Definition Editor	114
Setting the Interface Unit Generation Parameters	116
Generation Parameters Editor	116
Modifying the Business Object	116
Definition Editor	116
Generation Parameters Editor	116
TUI Only: Modifying the Presentation of the Business Object	117
TUI Only: Modifying the MainMenu Interface Unit	117
Generating your Application.	117
Generating your GUI Client Application.	118
Generating your TUI Application	118
Enhancing the GUI Client Application	118
Testing your Application	119
Testing the GUI Client Application	119
Testing the TUI application	121
Presenting a List and Detail in Two Different Windows (GUI) or Maps (TUI)	122
Creating an Interface Unit Presenting the List	124
Definition Editor	124
Generation Parameters Editor	125
GUI Only: Modifying the Detail Interface Unit	126
GUI Only: Modifying the Business Object's Behavior	126
TUI Only: Modifying the MainMenu Interface Unit	126
Generating your Application.	127
Testing your Application	127
Testing the GUI Client Application	127
Testing the TUI Application	127
Generating On-Line Help (VAGTemplates on Smalltalk Example)	128

Generating On-Line Help for the GUI Client Application	128
Generating On-Line Help for the TUI Application.	129
Testing On-Line Help	129
Testing On-Line Help for the GUI Client Application	129
Testing On-Line Help for the TUI Application.	130
Enhancing the GUI On-Line Help (IPF file)	131
Enhancing the GUI On-Line Help (RTF file)	135
Enhancing the TUI On-Line Help	138
Using Foreign Keys to Provide a Help List	139
Help List Principles	139
Help Lists in GUI Client Applications	139
Help Lists in TUI Applications	139
Help List Specification	140
Testing your Help List	142
Testing the Help List of the GUI Client Application	142
Testing the Help List of the TUI Application.	143

Chapter 5. Standard Functions and

Layouts of Generated Applications	145
Standard Functions	146
Management of Persistent Data	147
Actions Available for Detail Business Objects	147
Actions Available for List Business Objects	148
Actions Available for Help Lists	151
Automatic Zoom in (GUI).	152
Data Transfer between TUI Maps	152
Error Handling in GUI Client applications	153
Unitary Check: Errors in Input Fields	153
Global Check: Field Consistency	154
Server Check: Access Errors	155
Error Handling in TUI Applications	156
Unitary Check: Format Errors in Input Fields	156
Global Check: Access Errors	156

Management of the Navigation	157	Default Layout	212
Navigating Throughout a GUI Client		Layout Parameters	212
application	157	Field Size and Presentation	214
Navigating Throughout a TUI		Detail Business Objects	217
Application.	158	Default Layout	217
On-Line Help	160	Layout Parameters	218
GUI On-Line Help	160	Help Lists	220
TUI On-Line Help	163	List Business Objects	221
Edition Functions (GUI)	165	Default Layout	221
VAGTemplates on Java:		Layout Parameters	222
Implementation Principle	166		
Prompt on close	167	Chapter 6. Application Generation and	
VAGTemplates on Java:		Enhancement.	225
Implementation Principle	167	Standard Generation	225
Windows Menu	167	List of Available Generators	225
VAGTemplates on Java:		VAGTemplates on Smalltalk 3.1	
Implementation Principle	167	Generators	225
BiDi Applications	168	VAGTemplates on Smalltalk 4.0	
VAGTemplates on Java:		Generators	226
Implementation Principle	169	VAGTemplates on Java Generators	226
Standard Layouts of GUI Client		Instance Only / Instance Generation	
applications	169	Option	226
Fields.	169	Business Object Generation	227
Default Layout	169	Data Element Generation	228
Layout Parameters	169	Interface Unit Generation	228
Input Mask.	177	Relational Table Generation	229
Detail Business Objects	181	Workspace Generation	229
Default Layout	181	VAGTemplates on Smalltalk: Help	
Layout Parameters	183	Generation	230
Help Lists	190	With associates/Cascaded Generation	
List Business Objects	191	Option	231
Default Layout	192	With Associates and Predefined Beans /	
Layout Parameters	193	Cascaded Generation With Predefined	
Extraction Criteria Layout	195	Parts Option	232
Windows	196	Application Storage	232
Default Layout	196	Specification Storage	233
Layout Parameters	197	Generated Components Storage.	233
Standard Layouts of TUI Applications	199	VAGTemplates on Smalltalk:	
Maps	199	Generated Help Files Storage	233
Root Map Default Layout	199	Enhancements and Re-generation	233
Simple Map Default Layout	201	Traceability Information	234
Application Error Map Default		Traceability Categories	234
Layout	203	Generated Part Documentation	235
Management of Messages	204	VAGTemplates on Java: Traceability	
Help Map Default Layout.	204	in Comments	235
Help List Map Default Layout	204	How the Generators Use the Traceability	
Layout Parameters	205	Information	235
Help List, Help Map, Error Map		What Generator Do You Use When	
Presentation	211	Re-generating	237
Fields.	212	VAGTemplates on Java.	237

VAGTemplates on Smalltalk	238	Components Generated From a	
Generated Architecture and Principles	240	Workspace: Predefined Beans/Parts	312
Introduction	240	GUI and TUI Components	313
Generated Components Naming Policy	241	GUI Components.	317
Long Name Structures	242	TUI Components	321
VisualAge for Java/VisualAge		Application Enhancement: Public Interface	
Smalltalk Enterprise Components		of GUI Generated Components	324
Naming	243	Resource Object Bean/Part Interface	324
Predefined Beans/Parts.	244	API for Managing Detail Data	324
Server Architecture	244	API for Managing List Data	325
Server types	245	API for Managing Updatable List Data	325
Generated Servers by Entity Type	246	Business Object Bean/Part Interface	325
Client Architecture	247	List Manager Bean/Part Interface	327
Web Client	247	API for Managing List Data	327
GUI Client	248	Additional API for Managing	
TUI Client	253	Updatable List Data	327
Overview of Generated Code	253		
Servers and their Hooks	253		
Hooks	253		
Server Common Functions	263		
Two- or Three-Tier Layers.	264		
Generation of Atomic Detail Servers	264		
Generation of Help List Servers.	264		
Clients	265		
Web Client	265		
GUI Client	267		
TUI Client	286		
Components Generated by Entities	286		
Components Generated from a Data			
Element	286		
GUI and TUI Components	286		
GUI Components: Java/Smalltalk			
Class	287		
Components Generated from a Business			
Object	287		
WEB Components	287		
GUI and TUI Components	287		
GUI Components.	292		
TUI Component: Additional Server			
Program	301		
Components Generated from a			
Relational Table	302		
GUI and TUI components.	302		
GUI components: Non-Visual			
components	304		
Components Generated from an			
Interface Unit	306		
WEB Components	306		
GUI Components.	308		
TUI Parts	309		

Chapter 4. Exploring VAGTemplates Basic Functions

The development cycle of an application with VAGTemplates is comprised of 4 steps as presented in the *Introducing VAGTemplates* book:

1. **Import of a database structure into VAGTemplates:** VAGTemplates creates pre-filled instances of the Information Model entities.
2. **Specification of the application with the VAGTemplates Workbench:** you create, delete, and/or update instances of the Information Model entities to prepare the generation of your application, and specify the functional aspects of each instance.
3. **Generation of the application with VAGTemplates:** the process converts application specifications into VisualAge for Java or VisualAge Smalltalk Enterprise and VisualAge Generator operational components, and OS/2 or Windows help files.
4. **Customize the generated application** if required with the VisualAge Generator workstation.

In this chapter, you will learn how to handle the standard VAGTemplates functions and you will create operational GUI client and TUI applications using the VAGTemplates standard generators.

For these examples, we use the same ORG and STAFF tables we imported from the DB2 SAMPLE database. Before developing the examples presented in this chapter, make sure that you have completed the steps described in the *Introcucing VAGTemplates* book. The development environment is also **VAGTemplates on Smalltalk**.

Note to VAGTemplates on Java users: Before beginning to work with VAGTemplates, you must first create and project and a package to store your specifications.

VAGTemplates focuses on re-using components. In the Workbench, most of the specification fields are common to GUIs and TUIs. The description of a GUI client and a TUI application is similar when creating simple applications.

Note: When we come to describing the development processes that are specific to developing GUI client or TUI applications, the procedures that only apply to GUI clients are highlighted with this **GUI** sign; those that only apply to TUI applications with the **TUI** sign. No sign means that they are common to both GUI client and TUI applications.

Presenting a List and a Detail in the Same Window (GUI) or Map (TUI)

In this subchapter you will learn how to create an Interface Unit that uses the same Business Object twice, once as a list and once as a detail. The generated window (GUI) and map (TUI) will allow the end user to visualize a list of staff members, and the details of the selected staff member.

GUI

When the end user double-clicks on a staff member row in the list, the detail will display more information concerning the selected staff member.

TUI

When the end user enters in the access parameter area the id of a staff member appearing in the list, the detail will display more information concerning the selected staff member.

This behavior is called a *zoom in* (GUI) or a *data transfer* (TUI), and it is a standard function provided by VAGTemplates.

With this example, you will also learn how to use extract criteria.

For information on the automatic zoom in function, refer to topic “**Automatic Zoom in (GUI)**” on page 152, refer to “**Data Transfer between TUI Maps**” on page 152.

In the *Introducing VAGTemplates* book, you did the following:

- you imported the SAMPLE tables
- created the Sample Business Object.

In this chapter, you will re-use this Business Object.

Caution: Before you continue, make sure that you have completed the example presented in the *Introducing VAGTemplates* book.

TIP: In this second example, several specifications and generated parts will override what has been done previously in the *Introducing VAGTemplates* book. In order to be able to re-load the corresponding specifications and the generated application, you can version the applications created in the first example - MdmExample1App, MyVAGEntitiesApp, MyWorkspacePartsApp.

The completed window will look like this:

Staff List and Detail [Window Controls]

ListSample DetailSample Edit Windows Help

Id No	Name	Dept No
260	Jones	10.0
12	Jung	66.0
170	Kermisch	15.0
90	Koonitz	15.0
270	Lea	66.0

LOCATION

ID NUMBER

NAME

JOB TITLE

YEARS

SALARY

DEPARTMENT NUMBER

DEPARTMENT NAME

LOCATION

The completed map will look like this:

```

SYST: OS2CICS                MYAPPLICATION                FASTPATH: LSTDET
USER: PC user                 List and Detail                05-29-1997 18:28:59

ACTIONS: I INSERT U UPDATE D DELETE
  Id No   Name           Dept No
    360 Arent              20
    80 James              20
    20 Pernal             20
    10 Sanders            20

ACTION: _
ID NUMBER      360
NAME           Arent          DEPTNUMB      20
DEPARTMENT NUMBER 20        LOCATION Washington
JOB TITLE      Sales
YEARS          8
SALARY         18357

FASTPATH: _____ WASHINGTON/360 _____

PF01: HELP    PF02:          PF03: EXIT    PF04: LOOKUP  PF05:          PF06:
PF07:          PF08: NEXT    PF09:          PF10:         PF11:          PF12: CANCEL

```

Creating an Interface Unit Presenting a List and a Detail

To create the Interface Unit:

1. Select the Interface Unit entity from the list of entities,
2. Select *New...* from the *Instance* drop-down menu (or right click and select *New...* from the pop-up menu).

The *New VAGT Instance* window opens.

3. Enter *ListAndDetail* in the *Instance name* field, and specify an application in the *Application* combo box.

For this sample application, enter *MdmExample2App*. The new Interface Unit's functional specifications will be stored in a separate application.

4. Check the *Open now* check box to open the Interface Unit Definition editor once the Interface Unit is created.
5. Click *OK*.

You can now define the logical specifications of the Interface Unit.

Defining the Interface Unit

Definition Editor

The Interface Unit Definition editor allows you to specify the functional description of your Interface Unit.

General Panel:

1. In the *Default use name* field type *ListAndDetail*.
2. In the *Display name* field type *List and Detail*.

Descriptions Panel:

- In the *Textual Description* field type *Interface Unit calling list and detail BO*.
- In the *On line help description* field type:
 - If you are developing a GUI client application, enter: *This window presents Staff members in a list. To see detailed information about one staff member, double-click on a row in the list. The detail fills in with the corresponding information.*
 - If you are developing a TUI application, enter: *This map presents staff members in a list. To see detailed information about one staff member, enter his/her Id number in the access parameter area, and press enter. The detail fills in with the corresponding information.*
 - **GUI** Select the *root* radio-button from the *Type* area to indicate that the Interface Unit is the first in the application, and that it is not called by any other Interface Unit.
 - **TUI** Do not modify the selected radio-button from the *Type* area as we will create another Interface Unit that calls this one. The Detail Interface Unit is a *simple* Interface Unit

Business Objects Panel: You need to use the Sample Business Object first as a list then as a detail.

1. To call the list *BusinesObject* in the Interface Unit, right click on the *Business Object* list area.
2. Select *Add* from the pop-up menu or click on the *Add* push button, then select *Sample* in the *Add Business Objects* window.
3. Click *OK*.
4. By default, the presentation type for the Business Object is *detail*. You need to change it to a list presentation. To do so, click on the detail cell, in the *Layout type* column. Select *list*.
5. To call the detail Business Object in the Interface Unit, right click on the *Business Object* list area.
6. Select *Add* from the pop-up menu or click on the *Add* push button, then select *Sample* in the *Add Business Objects* window.
7. Click *OK*.
8. Because the Business Object is used twice, you need to differentiate these two uses. In the *Component Name* column, enter *Samplelist* in the cell corresponding to the first use of the Business Object, then enter *SampleDetail* in the cell corresponding to the second use of your Business Object..

TIP: If you called the wrong Business Object, you can remove it from the list. Select it in the list, right click and select **Remove** from the pop-up menu or click on the *Remove* push button.

You do not have to specify any target Interface Unit.

Setting the Interface Unit Generation Parameters

Generation Parameters Editor

The Generation Parameters editor allows you to specify the generation parameters for your Interface Unit.

General Panel:

1. In the *Title* field type *Staff List and Detail*.
2. **TUI** In the *Fastpath* field type *LSTDET*

Modifying the Business Object

Definition Editor

In our final application we only want the Id Number, Name and Dept fields to be generated in the list. We do not want to modify the detail Business Object.

1. Select the Business Object entity from the list of entities,
2. Double-click on the *Sample* Business Object from the list of instances.

TIP: You can also open this Business Object from the ListAndDetail Interface Unit Definition editor by right-clicking on the Sample Business Object in the *Business Objects* list (*Business Objects* panel) and selecting ***Open Business Object*** from the pop-up menu.

3. From the tree view, select *Field Attributes*. In the *Fied Attributes* panel, click in the *Laid out* column of the Job Data Element and select *on detail only* in the drop-down list.

This field will only appear in the detail, not in the list.

4. Repeat step 3 for the Years, Salary, Deptname and Location Data Elements.

Generation Parameters Editor

Your Business Object is defined with an extract criterion: Location. All the database data will be in the list if the end user does not filter these data using an extract criterion.

To make the extract criteria and the Extract action available:

1. From the editor tree view, click on *List View*.
2. In the *List View* panel, check the *Extraction criteria displayed* parameter.
3. *OK*.

In the *Optimization* panel, set the *Service level* parameter to *detail and read-only list*.

Only the parts required for the detail and the read-only list will be generated.

TUI Only: Modifying the Presentation of the Business Object

Because we want to have the same Business Object used twice in the same map, we need to adjust the number of lines in the list and the number of lines in the detail in order to have the whole information at a glance, without changing the map.

1. From the editor tree view, select *Detail View*.
2. In the *Detail View* panel, select 5 in the *Number of lines* spin edit.
The Business Object fields in the detail will be placed on five lines; the remaining fields will be placed in the following "column".
3. From the editor tree view, select *List Container*. In the *List Container* panel, select 4 in the *Number of lines* spin edit.
The list will only present 4 rows at a time.
4. Click *OK*.

TUI Only: Modifying the MainMenu Interface Unit

Because we defined a new Interface Unit, we need to call it in the MainMenu Interface Unit we created in the first example (see the *Introducing VAGTemplates* book).

1. Double-click on the *MainMenu* Interface Unit in the list of Interface Unit instances.
The Definition editor opens.
2. From the editor tree view, select *Target Interface Units*. In the *Target Interface Units* panel, right click on the *Interface Unit* column, then select **Add** from the pop-up menu, or click on the *Add* push button.
3. Select *ListAndDetail* in the *Add Interface Units* window.
4. Click *OK*.

TUI: Before you generate make sure that the *Control location* parameter (Workspace editor) is set to *server*, otherwise errors will be detected at execution time.

Generating your Application

Before generating, you can specify a new application where the new generated parts will be stored.

To do so:

1. Select the Interface Unit entity in the list of entities.
2. Select the **Default Parameters** choice from the **Entity** menu.
3. Change the value of the *Target Application* parameter.
4. Click *OK*.

Do the same for the Business Object, and the Data Element entities.

Generating your GUI Client Application

To generate your GUI client application, follow these steps:

1. Select the *ListAndDetail* Interface Unit from the list of Interface Unit instances.
2. Select *Generate...* from the *Instance* drop-down menu.
3. Select the *GUI Interface Unit - Smalltalk oriented* generator from the list of generators,
4. Select the *cascaded generation* radio-button.

All the instances you have just defined will be generated from the Interface Unit as they are directly or indirectly called by the Interface Unit.

Note: You do not need to choose *cascaded generation with predefined parts* since we have already generated the predefined parts when we generated the Detail Interface Unit in the example presented in the *Introducing VAGTemplates* book.

5. Click OK.

Generating your TUI Application

To generate your TUI application, follow these steps:

1. Select the *MainMenu* Interface Unit from the list of Interface Unit instances.
2. Select *Generate...* from the *Instance* drop-down menu.
3. Select the *TUI Interface Unit Logic* generator from the list of generators.
4. Select the *cascaded generation* radio-button.

All the instances you have just defined will be generated from the Interface Unit since they are called directly or indirectly by the Interface Unit.

Note: You do not need to choose *cascaded generation with predefined parts* since we have already generated the predefined parts when we generated the Detail Interface Unit in the example presented in the *Introducing VAGTemplates* book.

5. Click OK.

Enhancing the GUI Client Application

The menu bar of the generated Staff List and Detail window has *SampleList* and *SampleDetail* drop-down menus. *SampleList* presents the actions available for the list Business Object and *SampleDetail* presents the actions available for the detail Business Object. These menu choices can also be displayed as push-buttons or *via* pop-up menus. You can change these presentations by modifying the value of the *Display* parameter from the *Detail Action and Labels* panel of the Business Object Generation Parameters editor.

Testing your Application

Testing the GUI Client Application

You need to execute the `ListInterfaceUnitView` visual part (see the test procedure in the second part of the *Introducing VAGTemplates* book).

To test the **zoom in** function

1. Select *Top* from the *List* drop-down menu to fill the list.
2. Double-click on the row with staff member Daniels;

The detail displays the corresponding data.

TIP: A parameter in the Workbench specifies that the list must be filled at the window opening without activating the *Top* action: *List prefilled* parameter in the *List View* panel of the Business Object Generation Parameters editor.

To test the **extract function**

1. Enter *New York* in the Location field.
2. Select *Extract* from the *List* drop-down menu.

Only the staff members located in New York are extracted from the database.

Staff list and details

SampleList SampleDetail Edit Windows Help

Id No	Name	dept No	
240	Daniels	10	
260	Jones	10	
210	Lu	10	
160	Molinare	10	

Loc

ID NUMBER

NAME

DEPT NUMBER

JOB TITLE

YEARS

SALARY

DEPARTMENT NAME

LOCATION

Testing the TUI application

You need to execute the MAINMAM Program (see the test procedure in the second part of the *Introducing VAGTemplates* book.)

1. Type *B* in the Selection area to open the List and Detail map.
The list is pre-filled with all the database data.
2. To test the **data transfer** function, that is see the detail of a particular row in the list, type the Id of staff member Daniels in the access parameter area as follows: */240*, and press enter.
The corresponding information is displayed in the detail.

TIP: If the list were an updatable list, you could modify the *Daniels* row, and trigger the *Update* action; the detail would be automatically refreshed with the modified data. To change the presentation to updatable list, modify the value of the *Layout type* column in the Interface Unit Definition editor, *Business Objects* panel.

3. To test the **extract function**, enter *New York* in the access parameter area and press enter.
Only the staff members located in New York are extracted from the database and displayed in the list.

You can also use the extract criteria and the id in the same command line to extract data and display them in the list while seeing the detail of a staff member corresponding to the extract criteria.

For example, enter *New York/160* in the access parameter area and press enter.

Only the staff members located in New York are extracted from the database and displayed in the list, while the information on staff member Molinare is displayed in the detail.

```

SYST: OS2CICS                MYAPPLICATION                FASTPATH: LSTDET
USER: PC user                 List and Detail                06-18-1997 16:12:59

ACTIONS: I INSERT U UPDATE D DELETE
  Id No   Name           Dept No
    240   Daniels         10
    260   Jones           10
    210   Lu              10
    160   Molinare        10

ACTION: _
ID NUMBER      160
NAME           Molinare   DEPTNUMB      10
DEPARTMENT NUMBER 10      LOCATION     New York
JOB TITLE      Mgr
YEARS          7
SALARY         22959

FASTPATH: _____ NEW_YORK/160

PF01: HELP   PF02:          PF03: EXIT   PF04: LOOKUP PF05:          PF06:
PF07:          PF08:          PF09:          PF10:          PF11:          PF12: CANCEL

```

TIP: You can also select data using the extract criteria and the logical key directly from the Main Menu, which opens the List and Detail map and displays the requested data.

Presenting a List and Detail in Two Different Windows (GUI) or Maps (TUI)

In this subchapter you will learn how to create two windows (GUI) or maps (TUI) presenting the same Business Object. In the first one the Business Object is presented as a list and in the second as a detail. When selecting an item in the list, the second window (GUI) or map (TUI) opens and displays the details of the selected item.

In the previous examples, you did the following:

- imported the DB2 SAMPLE database,
- created the Sample Business Object,
- created the Detail Interface Unit,
- *TUI* created the Main Menu Interface Unit.

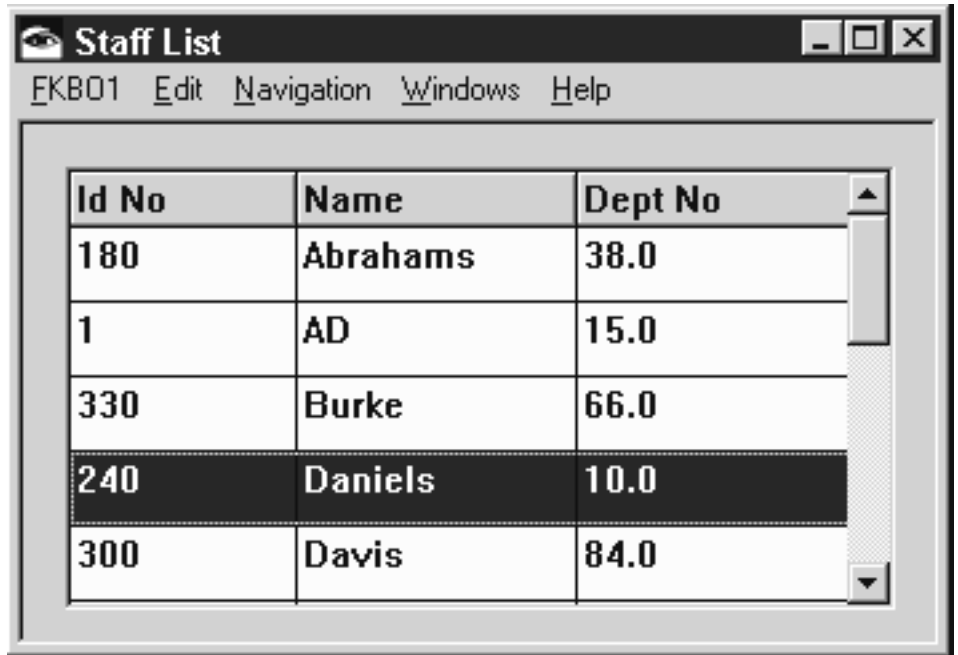
In this chapter, you will need to use these elements again.

Caution: Before you continue, make sure that you have completed the example presented in “**Presenting a List and a Detail in the Same Window (GUI) or Map (TUI)**” on page 112.

TIP: In this third example, several specifications and generated parts will override what has been done previously. In order to be able to re-load

the corresponding specifications and the generated application, you can version the applications created in the second example.

The completed GUI client application will look like this:



The completed TUI application will look like this:

```

SYST: OS2CICS          MYAPPLICATION          FASTPATH: LIST
USER: PC user          Staff list          06-18-1997 18:17:41

ACTIONS: 1 DETAIL
A Id No   Name      Dept No
1   180 Abrahams   38
-   360 Arent     20
-   330 Burke     66
-   240 Daniels   10

FASTPATH: _____

PF01: HELP   PF02:         PF03: EXIT    PF04:         PF05:         PF06:
PF07:         PF08: NEXT    PF09:         PF10:         PF11:         PF12: CANCEL

```

```

SYST: OS2CICS          MYAPPLICATION          FASTPATH: DETAIL
USER: PC user          Staff details          06-18-1997 17:16:42

ACTIONS: I INSERT U UPDATE D DELETE
ACTION: _
ID NUMBER      180
NAME           Abrahams   DEPTNUMB      38
DEPARTMENT NUMBER 38      LOCATION Atlanta
JOB TITLE      CLERK
YEARS          3
SALARY         12009

FASTPATH: _____ 0000180

PF01: HELP   PF02:         PF03: EXIT    PF04: LOOKUP  PF05:         PF06:
PF07:         PF08:         PF09:         PF10:         PF11:         PF12: CANCEL

```

Creating an Interface Unit Presenting the List

Create the List Interface Unit.

To do so, follow the procedure as in “Creating an Interface Unit Presenting a List and a Detail” on page 114.

Definition Editor

General Panel:

1. In the *Default use name* field type *List*.

2. In the *Display name* field type *List*.
3. **GUI** Select the *root* radio-button from the *Type* area to indicate that the Interface Unit is the first in the application, and that it is not called by any other Interface Unit.
4. **TUI** Do not modify the selected radio-button from the *Type* area as we will create another Interface Unit that calls this one. The List Interface Unit is a *simple* Interface Unit.
5. **TUI** In the *Fastpath* field type *LIST*.

Descriptions Panel:

1. In the *Textual Description* field type *Interface Unit used to show Staff list*.
2. In the *On line help description* field:
 - If you are developing a GUI client application enter: *This window presents a list of staff members*.
 - If you are developing a TUI application enter: *This map presents a list of staff members*.

Business Objects Panel:

1. To indicate which Business Object is called in the Interface Unit, right click on the *Business Object* column in the *Business Objects* panel,
2. Select *Add* from the pop-up menu or click on the *Add* push button,
3. Select *Sample* in the *Add Business Objects* window,
4. Click *OK*.
5. By default, the presentation type for the Business Object is detail. You need to change it to a list presentation. To do so, select *list* from the *Layout Type* column.

Target Interface Units Panel:

1. To indicate which Interface Unit is called by this Interface Unit, right click on the *Interface Unit* column in the *Target Interface Units* panel,
2. Select *Add* from the pop-up menu or click on the *Add* push button,
3. Select *Detail* in the *Add Interface Units* window,
4. Click *OK*.

Generation Parameters Editor

General Panel:

1. Enter a name in the *Target name* field.
 This target name is used to build the name of the components generated from the Sample Business Object. If this name exceeds 5 characters, it will be truncated. Therefore, you should enter a target name with 5 significant characters if the instance name is longer than 5 characters.

In order to distinguish the components generated from the ListAndDetail Interface Unit (the default target name is Lista) and those generated from the List Interface Unit (the default target name is List) you can modify the target name of the List Interface Unit here.

2. In the *Title* field type *Staff List*.

For more information on the naming policy of the generated components, refer to topic “**Generated Components Naming Policy**” on page 241.

GUI Only: Modifying the Detail Interface Unit

In the first example presented in the *Introducing VAGTemplates* book, the Detail Interface Unit was defined as a root Interface Unit. Now that this Interface Unit is called by the List Interface Unit, we must modify the corresponding parameter.

1. Open the Detail Interface Unit Definition editor.

TIP: You can open this editor directly from the List Interface Unit Definition editor by right-clicking on the Detail Interface Unit in the *Target Interface Units* panel and selecting *Open Interface Unit* from the pop-up menu.

2. Select the *simple* radio-button from the *Type* area.

GUI Only: Modifying the Business Object’s Behavior

In the previous generated application, when we opened the List window, we had to activate the *Top* action to fill in the list. To make the list fill at window opening, follow these steps:

1. Open the Business Object Generation Parameters editor.
2. From the editor tree view, select *List View*.
3. In the *List View* panel, check the *List prefilled* parameter.
4. Click OK.

TUI Only: Modifying the MainMenu Interface Unit

Because we have defined a new Interface Unit, we need to call it in the MainMenu Interface Unit.

1. Double-click on the MainMenu Interface Unit in the list of Interface Unit instances.
The Interface Unit Definition editor opens.
2. From the editor tree view, select *Target Interface Units*. Right click on the *Interface Unit* column, then select *Add* from the pop-up menu, or click on the *Add* push button.
3. Select *List* in the *Add Interface Units* drop-down list.
4. Click OK.

TUI: Before you generate make sure that the *Control location* parameter (Workspace editor) is set to *server*, otherwise errors will be detected at execution time.

Note: We do not need to modify the presentation of the Sample Business Object's fields in the list since we re-use the Sample Business Object instance we defined in the second example. In that example, we have specified the only fields that we want to appear in the detail (see "**Modifying the Business Object**" on page 116).

Generating your Application

Before generating, you can specify a new application where the generated parts will be stored (see "**Generating your Application**" on page 117).

GUI: To generate your application, follow the procedure as in "**Generating your GUI Client Application**" on page 118.

TUI: To generate your application, follow the procedure as in "**Generating your TUI Application**" on page 118.

Testing your Application

Testing the GUI Client Application

Execute the ListInterfaceUnitView visual part (see the procedure described in the second part of the *Introducing VAGTemplates* book). The List window opens and the list is pre-filled with data.

To open the detail displaying the detail of a selected row in the list, double-click on the line with staff member *Daniels*.

The Detail window opens and displays the requested information.

Testing the TUI Application

Execute the MAINMAM Program (see the procedure described in the second part of the *Introducing VAGTemplates* book), and open the List map. The list is pre-filled with data.

To open the map displaying the detail of a selected row, in the list, type *1* (the action code for the Detail map) in front of staff member *Daniels* row and press enter.

The Detail map opens and displays the requested information.

Generating On-Line Help (VAGTemplates on Smalltalk Example)

Reminder: For the current version of VAGTemplates, the help generators are available for the Smalltalk version of the product only.

The on-line help generation is a standard function provided by VAGTemplates. It is organized hierarchically: from the Interface Unit help to the called Interface Unit and Business Object help, to the called Data Element help.

For more information on generated on-line help, refer to topic “**On-Line Help**” on page 160.

In the present example, we will generate on-line help for the ListAndDetail application and we will re-use the ListAndDetail Interface Unit and the MainMenu Interface Unit (TUI) we defined in the second example (see “**Presenting a List and a Detail in the Same Window (GUI) or Map (TUI)**” on page 112)

To generate an operational on-line help, we need to enter the help text in the *On-line help description* field (*Definitioneditor*, *Descriptions* panel of each entity). This text will be used by the help generators to fill the help panels.

In the previous examples, when creating the Detail, the ListAndDetail, the List, and the MainMenu Interface Units, the Sample Business Objects and the Data Elements it calls, we filled in this field.

Now we only need to generate the on-line help.

Generating On-Line Help for the GUI Client Application

Generating the on-line help will create one file in IPF format and one in RTF and in HPJ format, which are the help formats used in the OS/2 and Windows environments respectively.

To generate the on-line help for the GUI client application, follow these steps:

1. Select the ListAndDetail Interface Unit from the list of Interface Unit instances,
2. Select *Generate...* in the *Instance* drop-down menu.
3. Select the *GUI Interface Unit Help* generator from the list of generators,
4. Select the *cascaded generation* radio-button.

The help panels of all the instances you have defined will be generated from the ListAndDetail Interface Unit since they are directly or indirectly called by this Interface Unit.

5. Click *OK*.

The generation produces a Help.IPF file, which is the default name for the help file. This file is the source file for the OS/2 help facility.

It also produces a Help.RTF and a Help.HPJ file, which are the source files for the Windows Help facility.

Note: By default these files are stored in the Build subdirectory under the VAGTemplates root directory (*Generation* parameter, Workspace editor).

6. In order for the on-line help to be used in the final application, you need to compile the source file to obtain an operational help file named Help.HLP (for information on the OS/2 help compiler, refer to your IBM documentation, for information on the Windows help compiler, refer to your Microsoft documentation).
7. Copy the resulting file in the Help subdirectory of your VisualAge Smalltalk Enterprise root directory.

Generating On-Line Help for the TUI Application

Generating on-line help will generate a VisualAge Generator table part containing on-line help.

To generate on-line help for the TUI application, follow these steps:

1. Select the MainMenu Interface Unit from the list of Interface Unit instances,
2. In the *Instance* drop-down menu, select *Generate...*,
3. Select the *TUI Interface Unit Help* generator from the list of generators.
4. Select the *cascaded generation* radio-button.

The help panels of all the instances you have defined will be generated from the MainMenu Interface Unit as they are directly or indirectly called by the Interface Unit.

5. Click *OK*.

Testing On-Line Help

Testing On-Line Help for the GUI Client Application

Execute the ListInterfaceUnitView visual part (see the procedure described in the second part of the *Introducing VAGTemplates* book).

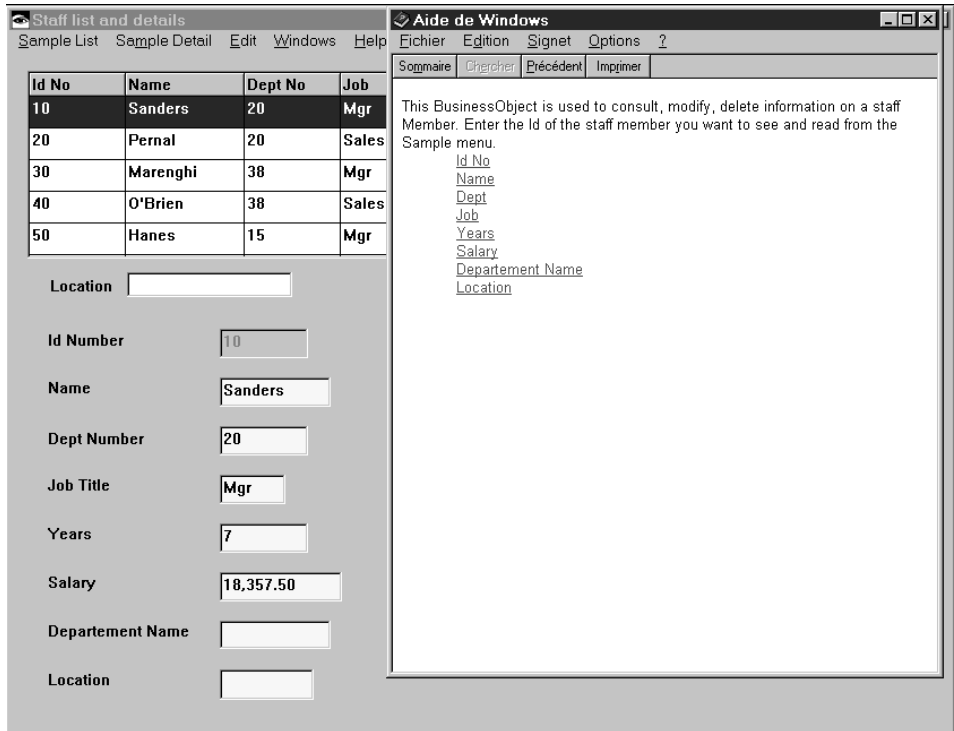
To get general help for the window, follow these steps:

1. Press F1 or click the *Help* menu.

The help panel presenting the help for the List and Details window opens. This panel also provides a hypertext link to the Sample Business Object's help panel.

2. Double-click on *Sample*.

The help panel presenting the help for the Sample Business Object opens. This panel also provides a hypertext link to the field help panels.



3. Double-click on *IdNo*.

The help panel presenting the help for the IdNo field opens.

To get help directly for a particular field, follow these steps:

1. Click in a field.
2. Press F01 or click the *Help* menu.

The help panel presenting the help on the field opens.

Testing On-Line Help for the TUI Application

Execute the MAINMAM Program (see the procedure described in the second part of the *Introducing VAGTemplates* book).

1. When the Main Menu map opens, press the F01 key.

A pop-up help displays the on-line help for the Main Menu map.

2. Type *B* in the Selection area to open the List and Detail map.
3. Press the F01 key.

A pop-up help displays the on-line help for the List and Detail map, for the Sample Business Object used as a list, for the fields in the list, and then for the Sample Business Object used as a detail and for its fields.

TIP: The presentation of the on-line help in a pop-up map is the default presentation. You can modify it by changing the default value of the *Popup policy* parameter (Workspace editor, *Parameters* tab - page 6).

```
SYST: OS2CICS          MYAPPLICATION          FASTPATH: LSTDET
USER: PC user          List and Detail          06-19-1997 17:55:38

ACTIONS: I INSERT U UPDATE D DELETE
  Id No   Name      Dept No
    180   Abrahams   38
    360   Arent      20
    330   Burke       66

HELP PANEL

List and Detail
This map presents the staff members in a list and the details of a
selected member of staff.
Sample
This BusinessObject is used to consult, modify, or delete information
on a staff member. Enter the ID of the staff member you want to see
and press enter.
IdNo
Enter an integer value different from 0.
Name
PF03: EXIT  PF07:          PF08: NEXT
```

Enhancing the GUI On-Line Help (IPF file)

In our application, the same Sample Business Object is called twice with two different presentations in the same map. Therefore, the standard generated help displays the same help text twice, once for the list, once for the detail, and presents all the fields called in the Business Object twice whereas the list only presents three of them.

If you want to modify this behavior, you need to modify the IPF file.

For information on IPF file tags, refer to your IBM documentation.

1. Open the file and modify it as shown below for example.

Note: Deleted code appears struck through, added text appears underlined, modified text appears in bold type. Our explanation comments are in double quotes.

```
:userdoc.
:h1 name=1 res=1 global.Interface Units
:font facename=Helvetica size=16x10.
:ul.
:li.
:link reftype=hd res=4963.
```

```

Staff list and details
:elink.
:eu1.
:h1 name=4963 res=4963 global.Staff list and detail
:font facename=Helvetica size=16x10.
:p.
This window presents the staff members in a list. To see
detailed information about one staff member, double-click on a
row in the list. The detail fills in with the corresponding
information&..
:ul.
:li.
:link reftype=hd res=9543. "This number is the
identifier of the Sample Business Object help panel as defined in VAGTemplates"
Sample list
:elink.
:li.
:link reftype=hd res=9544. "This number is the identifier of
the Sample detail help panel"
Sample detail
:elink.
:eu1.

:h1 name=9543 res=9543 global.Sample list
:font facename=Helvetica size=16x10.
:p.
This Business Object is used to consult
information about a staff member&..
Select Top from the SampleList menu to fill
in the list&.. Double-click on a row
to fill in the detail with the corresponding data&..
:ul.
:li.
:link reftype=hd res=19378.
IdNo
:elink.
:li.
:link reftype=hd res=29231.
Name
:elink.
:li.
:link reftype=hd res=15875.
Dept
:elink.
:link reftype=hd res=27013.
"We delete the extra items from the list of hypertext links"
Job
:elink.
:li.
:link reftype=hd res=26469.
YEARS
:elink.
:li.
:link reftype=hd res=25653.
Salary
:elink.

```

```

:li.
:link reftype=hd res=29439.
Deptname
:elink.
:li.
:link reftype=hd res=12953.
Location
:elink.
:eul.
:h1 name=27013 res=27013 global.Job
:font facename=Helvetica size=16x10.
:p.
Enter a string value with 5 characters maximum
:h1 name=29231 res=29231 global.Name
:font facename=Helvetica size=16x10.
:p.
Enter a string value with 9 characters maximum
:h1 name=19378 res=19378 global.IdNo
:font facename=Helvetica size=16x10.
:p.
Enter an integer value different from 0
:h1 name=15875 res=15875 global.Dept
:font facename=Helvetica size=16x10.
:p.
Enter an integer value
:h1 name=26469 res=26469 global.Years
"We delete the references to the help panels of the fields
that are not in the list"
:font facename=Helvetica size=16x10.
:p.
Enter an integer value
:h1 name=29439 res=29439 global.Deptname
:font facename=Helvetica size=16x10.
:p.
Enter a string value with 14 characters maximum
:h1 name=25653 res=25653 global.Salary
:font facename=Helvetica size=16x10.
:p.
Enter a decimal value (7,2)
:h1 name=12953 res=12953 global.Location
:font facename=Helvetica size=16x10.
:p.
Enter a string value with 13 characters maximum
:h1 name=9544 res=9544 global.Sample
detail "We insert the help for the detail Business Object"
:font facename=Helvetica size=16x10.
:p.
This Business Object is used to consult,
modify, or delete information about a staff member&.. Enter the Id of
the staff member you want to see in the ID Number field and select Read from
the SampleDetail menu, or double-click on a row in the list.&..
:ul.
:li.
:link reftype=hd res=19378.

```

```

IdNo
:elink.
:li.
:link reftype=hd res=29231.
Name
:elink.
:li.
:link reftype=hd res=15875.
Dept
:elink.
:li.
:link reftype=hd res=27013.
Job
:elink.
:li.
:link reftype=hd res=26469.
Years
:elink.
:li.
:link reftype=hd res=25653.
Salary
:elink.
:li.
:link reftype=hd res=29439.
Deptname
:elink.
:li.
:link reftype=hd res=12953.
Location
:elink.
:eu1.
:h1 name=27013 res=27013 global.Job
:font facename=Helvetica size=16x10.
:p.
Enter a string value with 5 characters maximum
:h1 name=29231 res=29231 global.Name
:font facename=Helvetica size=16x10.
:p.
Enter a string value with 9 characters maximum
:h1 name=19378 res=19378 global.IdNo
:font facename=Helvetica size=16x10.
:p.
Enter an integer value different from 0
:h1 name=15875 res=15875 global.Dept
:font facename=Helvetica size=16x10.
:p.
Enter an integer value
:h1 name=26469 res=26469 global.years
:font facename=Helvetica size=16x10.
:p.
Enter an integer value
:h1 name=29439 res=29439 global.Deptname
:font facename=Helvetica size=16x10.
:p.
Enter a string value with 14 characters maximum

```

```

:hl name=25653 res=25653 global.Salary
:font facename=Helvetica size=16x10.
:p.
Enter a decimal value (7,2)
:hl name=12953 res=12953 global.Location
:font facename=Helvetica size=16x10.
:p.
Enter a string value with 13 characters maximum
:userdoc.

```

2. Compile the IPF file.
3. Copy the resulting file in the Help subdirectory of your VisualAge Smalltalk Enterprise root directory.
4. Test your on-line help again.

Enhancing the GUI On-Line Help (RTF file)

In our application, the same Sample Business Object is called twice with two different presentations in the same map. Therefore, the standard generated help displays the same help text twice, once for the list, once for the detail, and presents all the fields called in the Business Object twice whereas the list only presents three of them.

If you want to modify this behavior, you need to modify the RTF and HPJ files.

For information on RTF files tagging, refer to your Microsoft documentation.

1. Open the RTF file and modify it as shown below for example.

Note: Deleted code appears struck through, added text appears underlined, modified text appears in bold type. Our explanation comments are in double quotes.

```

{\rtf1
{\fonttbl \deff1
{\f0\froman Times New Roman;}{\f1\fdcor Courier New;}{\f2\fswiss Arial;}{\f3\fswiss
Arial;}}
#{\footnote 1}
\par\tab {\u1db \f3 \fs20 Staff List and Detail}{\v \f3 \fs20 4963}
\page
#{\footnote 4963}
\par \f3 \fs20 This window presents the staff members in a list. To view
the details of a staff member, double-click in the corresponding row in the
list. The detail fills in with the requested information.
\par\tab {\u1db \f3 \fs20 Sample}{\v \f3 \fs20 5622}
"This number is the identifier panel as defined in VAGTemplates"
\page of the Sample Business Object help
#{\footnote 5622}
\keepn\f2\fs20 Sample List
"We insert an unscrollable title for the help panel"
\par\pard \f3 \fs20 This
Business Object is used to consult, modify, or delete information about a
staff member. Select Top from the SampleList menu to fill in the list. Double-click
on a row to fill in the detail with the corresponding data.
\par \f3 \fs20
\par\tab {\u1db \f3 \fs20 Id no}{\v \f3 \fs20 19378}
\par\tab {\u1db \f3 \fs20 Name}{\v \f3 \fs20 29231}

```

```

\par\tab {\uldb \f3 \fs20 Job}{\v \f3 \fs20 27013}
"We delete the extra items from the list of hypertext links
\par\tab {\uldb \f3 \fs20 Years}{\v \f3 \fs20 26469}
\par\tab {\uldb \f3 \fs20 Salary}{\v \f3 \fs20 25653}
\par\tab {\uldb \f3 \fs20 Dept}{\v \f3 \fs20 15875}
\par\tab {\uldb \f3 \fs20 DeptName}{\v \f3 \fs20 29439}
\par\tab {\uldb \f3 \fs20 Location1}{\v \f3 \fs20 12953}
\page
#{\footnote 27013}tab;
"We delete the references to the help
panels of the fields that are not in the list"
\par \f3 \fs20
Enter a string value with 5 characters maximum
\page
#{\footnote 29231}
\keepn\f2\fs20 Name
" We insert an unscrollable title
for the help panel"
\par\pard \fs20 Enter a string value with 9
characters maximum
\page
#{\footnote 19378}
\keepn\f2\fs20 Id Number
\par\pard \f3 \fs20 Enter an integer value
different from 0
\page
#{\footnote 15875}
\keepn\f2\fs20 Department Number
\par\pard \f3 \fs20 Enter an integer value
\page
#{\footnote 26469}
\par \f3 \fs20 Enter an integer value
\page
#{\footnote 29439}
\par \f3 \fs20 Enter a string value with 14 characters maximum
\page
#{\footnote 25653}
\par \f3 \fs20 Enter a decimal value with a 7-digit capacity
and a 2-digit precision
\page
#{\footnote 12953}
\par \f3 \fs20 Enter a string value with 13 characters maximum
\page

#{\footnote 5623} "We insert the help for the detail
Business Object"
\keepn\f2\fs20 Sample Detail
" We insert an unscrollable title for the help panel"
\par\pard \f3 \fs20
This Business Object is used to consult,
modify, or delete information about a staff member.
Select Top from the SampleList menu to fill in the list.
Double-click on a row to fill in the detail with the
corresponding data.
\par \f3 \fs20

```



```

\par\tab {\u1db \f3 \fs20 Id no}{\v \f3 \fs20 19378}
\par\tab {\u1db \f3 \fs20 Name}{\v \f3 \fs20 29231}
\par\tab {\u1db \f3 \fs20 Job}{\v \f3 \fs20 27013}
\par\tab {\u1db \f3 \fs20 Years}{\v \f3 \fs20 26469}
\par\tab {\u1db \f3 \fs20 Salary}{\v \f3 \fs20 25653}
\par\tab {\u1db \f3 \fs20 Dept}{\v \f3 \fs20 15875}
\par\tab {\u1db \f3 \fs20 DeptName}{\v \f3 \fs20
29439}
\par\tab {\u1db \f3 \fs20 Location1}{\v \f3 \fs20
12953}
\page
#\footnote 27013}
\keepn\f2\fs20 Job " We insert
an unscrollable title for the help panel"
\par\pard \f3 \fs20 Enter a string value with 5
characters maximum
\page
#\footnote 29231}
\keepn\f2\fs20 Name
\par\pard\f3 \fs20 Enter a string value with 9 characters
maximum
\page
#\footnote 19378}
\keepn\f2\fs20 Id Number
\par\pard\f3 \fs20 Enter an integer value different
from 0
\page
#\footnote 15875}
\keepn\f2\fs20 Department Number
\par\pard\f3 \fs20 Enter an integer value
\page
#\footnote 26469}
\keepn\f2\fs20 Years
\par\pard\f3 \fs20 Enter an integer value
\page
#\footnote 29439}
\keepn\f2\fs20 Department Name
\par\pard\fs20 Enter a string value with 14 characters
maximum
\page

```

(RTF file modifications cont'd)

```

#\footnote 25653}
\keepn\f2\fs20 Salary
\par\pard\f3 \fs20 Enter a decimal value with a
7-digit capacity and a 2-digit precision
\page
#\footnote 12953}
\keepn\f2\fs20 Location
\par\pard\fs20 Enter a string value with 13 characters
maximum
\page
}

```

2. Modify the HPJ file as shown below to add a title to the help window and to add the new help panel number for the detail Business Object.

```
[ WINDOWS ] "We add a title to the help window"  
Main="Help for MyApplication"  
[ FILES ] "This is the call to the help file"  
HELP.RTF  
[MAP] "These are the calls to the help panels."  
    "These panel Ids are defined in the VAGTemplates Workbench"  
4963 4963  
5622 5622  
5623 5623 "We add the panel  
Id for the new detail Business Object's panel"  
27013 27013  
29231 29231  
19378 19378  
15875 15875  
26469 26469  
29439 29439  
25653 25653  
12953 12953
```

3. Compile the HPJ file.
4. Copy the resulting file in the Help subdirectory of your VisualAge Smalltalk Enterprise root directory.
5. Test your on-line help again.

Enhancing the TUI On-Line Help

In our application, the same Sample Business Object is called twice with two different presentations in the same map. Therefore, the standard generated help displays the same help text twice, once for the list, once for the detail, and presents all the fields called in the Business Object twice whereas the list only presents three of them.

VAG: If you want to modify this behavior, you need to do it using the VisualAge Generator workstation.

The TUI on-line help text and items are stored in a Table Part. You only need to modify the text and the items in this table.

1. Select the *ListatH* Table Part from the VAGen parts list in the MyVAGTWorkspaceEntitiesApp.

The table description opens presenting three DataItems:

- the type of the VAGTemplates instances for which help will be displayed (*W* for Interface Unit, *O* for Business Object, *E* for Data Element);
- the identifier of the instance;
- the help text you wrote using the VAGTemplates Workbench.

2. Select *Table contents...* from the *Define* menu.

The contents of the previous DataItems are displayed.

3. Modify the help text in the HELP-TEXT Data-Items.

For example, select the first Sample occurrence and type Sample List in the edit box. The change is taken into account when you click on another cell.

4. Delete the extra items that appear in the help panel of the list Business Object.

Select the lines from JOB1 to LOCATION1 - skip DEPT- below the List item (hold down left mouse button and drag).

5. Press Del or select *Delete* from the *Edit* menu.
6. Test your on-line help again.

Using Foreign Keys to Provide a Help List

In this subchapter, you will learn how to develop a window (GUI) or a map (TUI) presenting the details of a staff member and using a help list.

Help List Principles

In the applications developed with VAGTemplates, you can provide the end user with a function called **help list**.

Note: You can only use the help list function if the imported database contains tables with foreign keys.

Help Lists in GUI Client Applications

A **help list** opens when the end user clicks the combo box button. In this help list, the end user can select a value by clicking on it. The list closes and replaces the current value with the selected value.

If the end user knows the first character of the value he/she is looking for, he/she can type it in. The first item that begins with this character is then selected (normal list box operation).

Help Lists in TUI Applications

A **help list** panel opens when the end user presses the *Lookup* key after positioning the cursor in a field that offers help lists. Such fields are marked with an asterisk (*). In this help list, the end user can select a value by typing a selection code in front of the value. The help list closes and replaces the current value of the field with the selected value.



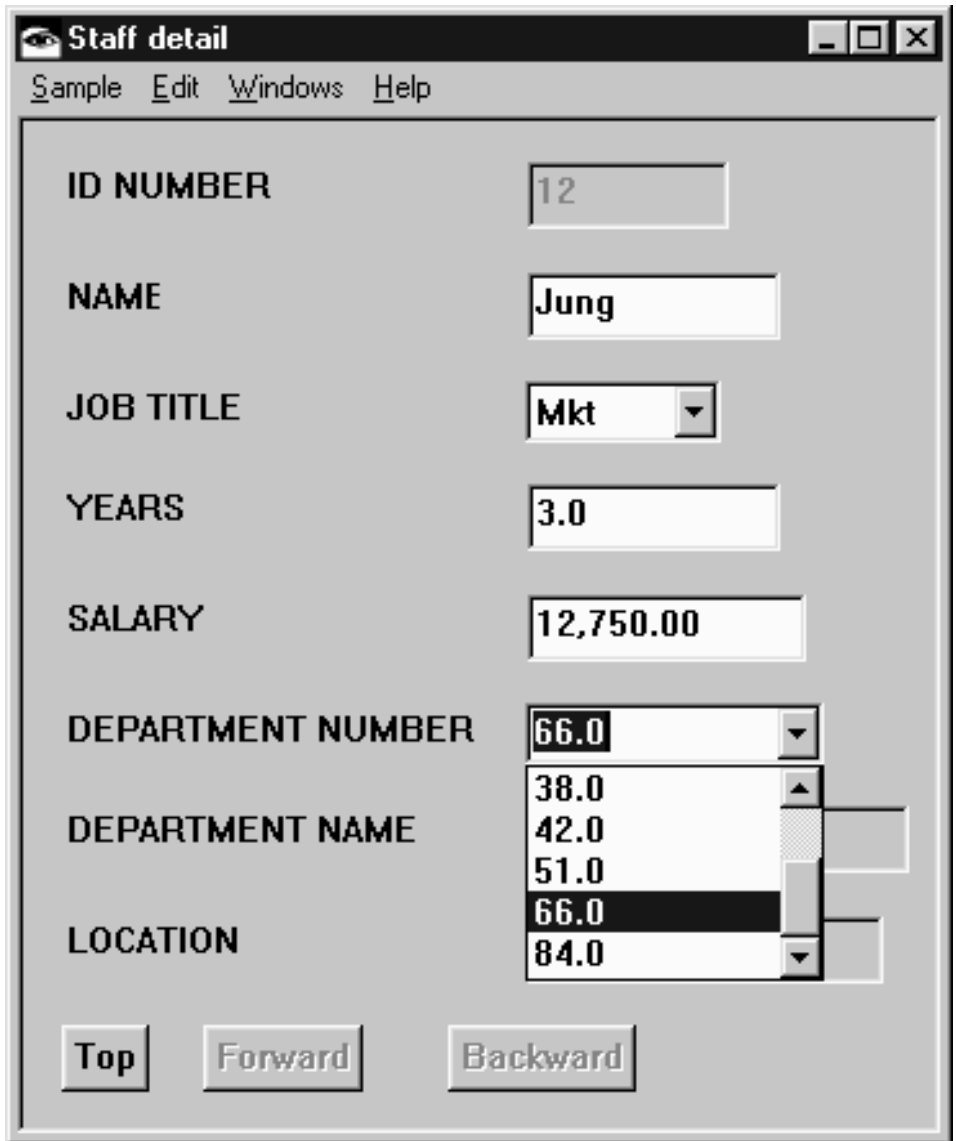
Help List Specification

Caution: The DB2 SAMPLE database we used so far does not contain any keys. For this example, you need to duplicate the SAMPLE database to create a SAMPLE2 database. Then, in SAMPLE2, you need to create a primary key for the ORG table - DEPTNUMB - and a foreign key - DEPTSTOR - between the ORG table (DEPTNUMB column) and the STAFF table (DEPT column).

For assistance, see the IBM DB2 Documentation.

To do this example, create a new Workspace and import the SAMPLE2 database in this new Workspace. And re-do the first example (topic detailed in the second part of the *Introducing VAGTemplates* book).

The GUI client application will look like this:



The TUI application will look like this:

```

SYST: OS2CICS          MYAPPLICATION          FASTPATH: Detail
USER: PC user          Staff Details          06-20-1997 15:47:10

ACTIONS: I INSERT U UPDATE D DELETE
ACTION:  _
ID:      240
NAME:    D
DEPTNUMB*:
JOB TITLE*: M
YEARS:
SALARY:
DEPARTMENT: H
LOCATION:  N

FASTPATH: _____ 240

PF01: HELP  PF02:          PF03: EXIT  PF04: LOOKUP PF05:          PF06:
PF07:          PF08:          PF09:          PF10:          PF11:          PF12: CANCEL

```

```

ORG HELP LIST PANEL
_ 0010
_ 0015
_ 0020
_ 0038
_ 0042
_ 0051
_ 0066
_ 0084

PF03: EXIT  PF07:          PF08:

```

Testing your Help List

Testing the Help List of the GUI Client Application

Execute the `DetailInterfaceUnitView` visual part (see the procedure described in the second part of the *Introducing VAGTemplates* book).

Once the Staff Detail window opens:

1. In the *Id* field, type *10*.
2. Select *Read* from the *Sample* drop-down menu.
3. Click the Top push-button at the bottom of the window to fill in the help list.

TIP: A parameter in the Workbench triggers the filling of the help list at the window opening: *Help list prefilled* parameter, Business Object editor.

4. Click on the Department Number field combo-box.
The help list displays all the department numbers that the ORG table contains.
5. Click on one of the values in the help list.
The current value is replaced with the selected one.

Note: If you try to enter a value that is not in the help list, the background color of the field turns red to indicate that you made an error. If you open the error message window, by selecting *Check* from the *Sample* menu, the following message is displayed "*Invalid value for foreign key*"

TIP: In this example, all the department numbers in the ORG table can be displayed in the help list. However, when the number of values is too big, the end user can move a page forward (with the *Forward* push-button) and a page backward (with the *Backward* push-button) in this list.

Testing the Help List of the TUI Application

VAST: Execute the MainMAM Program (see the procedure described in the second part of the *Introducing VAGTemplates* book).

Once the Staff Detail map opens:

1. Type 240 in the access parameter area and press enter to fill in the detail.
2. Position the cursor in the Department Number field and press the F04 function key.

A pop-up map appears and displays all the department numbers that the ORG table contains.

3. Type a selection code in front of a value to select it, and press enter.

The current value is replaced with the selected one.

Note: If you try to enter a value that is not in the help list, the background color of the field turns red to indicate that you made an error and an error message is displayed at the bottom of the map: *"Invalid value for foreign key"*

TIP: In this example, all the department numbers in the ORG table can be displayed in the help list. However, when the number of values is too big, the end user can move a page forward (with the F08 function key) and a page backward (with the function key) in this list.

Chapter 5. Standard Functions and Layouts of Generated Applications

GUI: The generated GUI client applications are window dialogs which allow the end user to navigate from one window to the other by following the links that make up the window tree. The windows present data in details or lists. There are various actions for managing data in details or in lists, as well as navigation actions. The generated GUI client application also includes error management, on-line help and help lists on foreign keys.

TUI: The generated TUI applications are map dialogs. The end user can navigate through the application by following the links that make up the map tree or by using a fastpath to reach any application main map directly, wherever it is located in the map tree. The navigation between maps can integrate access parameters. The generated maps can be menus that provide access to other maps, details and/or lists. The generated TUI applications are provided with various actions to access data, as well as error management, on-line help and help lists on foreign keys.

This chapter offers you a preview on the functions that VAGTemplates generates for your applications and the graphical presentations available according to the parameters you specified (or did not specify) with the Workbench.

Note: Paragraphs that only apply to GUI client applications are signaled with this **GUI** mark; paragraphs that only apply to TUI applications are signaled with this **TUI** mark.

When you can modify the default generation parameters in the VAGTemplates Workbench, this **VAGT** mark precedes the explanation paragraph.

Paragraphs that only apply to Java client applications are signaled with this **JAVA** mark; paragraphs that only apply to Smalltalk applications are signaled with this **SMTK** mark.

When you can modify your application using the VisualAge Generator workstation only, after the generation, this **VAG** mark precedes the explanation paragraph.

Servers: The standard generators provided by VAGTemplates generate source code to implement the functional processing identified in the Information Model definitions. The standard generators support the implementation of the following types of systems:

- GUI client/server application system
- TUI application system
- Web application system

All these different system types generated by VAGTemplates rely on the same server programs. This means that you can generate GUI clients and TUI applications that will share the same server programs. If you specify the required business logic at the server level, it will apply to all the clients generated with VAGTemplates.

Three types of server are generated for each Business Object:

- Detail
- List
- Updatable list

Two additional server types are generated when required:

- Non I/O check server
- Foreign key lookup server

For more details on servers, refer to “**Server Architecture**” on page 244.

Standard Functions

The VAGTemplates RAD (Rapid Application Development) generators provide the generated applications with standard functions and layouts.

The generated applications’ standard functions are the following:

- management of persistent data (creations, updates, etc.),
- error handling,
- management of the navigation,
- on-line help,

Note: For VAGTemplates on Java, the principle and methodology for implementing GUI on-line help is presented.

- edition functions (*GUI*),

Note: For VAGTemplates on Java, the principle and methodology for implementing each edition function in the VisualAge for Java environment is detailed.

- prompt on close,

Note: For VAGTemplates on Java, the principle and methodology for implementing this function is presented.

- **Windows** menu,

Note: For VAGTemplates on Java, the principle and methodology for implementing this menu is presented.

- Bi-directional ability.

Note: For VAGTemplates on Java, the principle and methodology for implementing this function is presented.

The following window, generated with VAGTempaltes on Java and modified in this target environment, is an example showing some of these standard functions.

Management of Persistent Data

In VAGTemplates, **persistent data**, i.e. data stored in a database, and non-persistent data are presented through Business Objects. VAGTemplates generates various actions helping the end user to manage persistent data.

TUI: These actions are available through function keys or action codes. By default, the elementary actions are triggered by action codes - I, Insert, D, Delete, U, Update - but they are also defined in the Information Model as action keys. If you prefer to use the function keys instead, you can set the VisualAge Generator PFEQUATE setting correspondingly or modify the values of the function keys in the action code table (see "*Action Code Table*" on page 311).

Actions Available for Detail Business Objects

VAGTemplates provides your generated applications with the following default services for accessing the database:

- creating rows;
- reading a row in the database, using one or more fields as extract criteria;
- updating rows;
- deleting rows;
- checking input values.

GUI: VAGTemplates allows you to choose between two update policies:

- Two actions manage the creation and the update of rows on the server: the *Create* and *Update* actions.
- One action manages the creation and update of rows at a time on the server: the *Save* action. All creations and updates are sent to the server when this action is activated. To insert an empty detail, the *New* action is added; the *Save* action updates it on the server.

These update policies are set *via* the *Update GUI policy* parameter, Workspace Definition editor, *GUI* panel.

The connections between actions in the menu and the server that accesses the database, and the associated processing are automatically created at generation time. These connections are customizable after the generation and are RAD.

TUI: VAGTemplates allows you to choose between two update policies:

- Rows are created and updated on the server when the end user triggers an action (explicit creation and update).
- Rows are created and updated on the server when the end user presses enter (implicit creation and update).

These update policies are set via the *Update TUI policy* parameter, Workspace Definition editor, *TUI* panel.

For information on action layouts in GUI client applications, refer to “*Actions*” on page 189.

For information on action layouts in TUI applications, refer to “*Function Keys and Actions*” on page 208.

Actions Available for List Business Objects

VAGTemplates allows generation of read-only lists and updatable lists.

Actions Available for Read-Only Lists: VAGTemplates provides read-only lists with default paging services for the massive consultation of data extracted according to specified extract criteria, that is to say either in *native* or textual value.

For information on how to specify extract criteria, refer to Part 1, “Part 2. The VAGTemplates Workbench” on page 7, “Chapter 3. Information Model Entities and their Editors” on page 51, “**Business Object**” on page 56, “**How to Define a Business Object**” on page 67.

Note: In this documentation, the word **selection** means the set of data verifying the extract criteria, **memory page** the selection available

without a new database access, and **graphical page** the number of data displayed simultaneously on screen, in the list.

Paging services allow the end user to:

- display the current memory page of a selection,
- refresh the current memory page of a selection
- display the memory page preceding the current page of a selection,
- display the memory page following the current page of a selection,
- define a new selection.

VAGT: A parameter in the Workbench allows you to specify the size of the memory page: *Number of rows to fetch* parameter, Business Object Generation Parameters editor, *List View* panel.

TUI: If the memory page and the graphical page are not the same size in TUI applications, the smaller size between the two is taken into account. The end user will trigger an action to access the next data from the database.

GUI: VAGTemplates allows you to choose among three paging policies:

1. The list displays only the current page of a selection (*Page display* parameter, set to *display current page*, Business Object Generation Parameters editor, *List View* panel).

The action initiates another database access when the last data in the memory page is reached. This changes the contents of the memory page accordingly.

The generated actions are *Top*, to reach the first data in the database, *Next*, to reach the next page of a selection, *Previous*, to reach the previous page of a selection, *Refresh*, to refresh the page displayed, *Extract*, to define a new selection.

When the memory page is greater than the graphical page, the end user moves forwards and backwards throughout the memory page, using the vertical scroll bar, without a new database access.

2. All the pages read in the database are displayed in the list (*Page display* parameter, set to *display all read pages*, Business Object Generation Parameters editor, *List View* panel).

- The paging is requested *explicitly* (*Paging policy* parameter, set to *explicit paging*, Business Object editor).

The action initiates another database access when the last data in the memory page is reached. This changes the contents of the memory page accordingly.

The generated paging actions are *Top*, to reach the first data in the database, *Next*, to reach the next page of a selection, *Extract* to d

efine a new selection. The end user uses the scroll bar to move forwards and backwards throughout the read pages.

When the memory page is greater than the graphical page, the end user moves forwards and backwards throughout the memory page, using the vertical scroll bar, without a new database access.

- The paging services are requested by the scroll bar (*Paging policy* parameter, set to *auto-scrolling*, Business Object Generation Parameters editor, *List View* panel).

The end user activates the generated *Top* action to reach the first data read from the database, and the *Extract* action to define a new selection. He/she uses the vertical scroll bar to page backwards and forwards in the list.

Actually, the first memory page is displayed in the list; when the last data in the memory page is reached with the scroll bar, a new database access is triggered but invisible to the end user: he/she only scrolls the list.

For information on action layouts in GUI client applications, refer to topic "*Actions*" on page 195.

For information on action layouts in TUI applications, refer to topic "*Function Keys and Actions*" on page 208.

Actions Available for Updatable Lists: VAGTemplates provides updatable lists with default paging services (see "*Actions Available for Read-Only Lists*" on page 148) and services for the creation, deletion or modification of rows. The requested actions are stored then sent together at the same time.

Note: The auto-scrolling function is not available for updatable lists.

In this documentation, the term **movement** means a creation, an update or a deletion. The term **list of movements** is the set of movements passed to the server at the same time.

The size of the list of movements and the maximum number of movements passed during one client-server exchange are limited to the number of rows read in the database at the same time (*Number of rows to fetch* parameter, Business Object Generation Parameters editor, *List View* panel).

The available movements are implemented as actions that allow:

- the creation of a row: a blank row is added after the selected row (*New* action) and created on the server (*Create* action);
- the update of a row: the selected row's fields are edited (*Update* action).

Note: The end user can only modify the key of a row added with the *New* action. Once the row is created on the server its key is read-only.

- the deletion of a row: the selected row is removed from the list (*Delete* action) then the deletion is sent to the server (*Update* action).

When a *Top* action is selected on a foreign key, a combo box is edited with several values from the database, these values being either *native* or *textual*. You can easier input or select a value.

When the updatable list displays only the current page, the *Next* and *Previous* paging actions submit the non-committed movements to the server; the *Refresh* action cancels the non-committed movements.

When the updatable list displays all the read pages, the paging actions have no consequences on the commit.

For information on action layouts in GUI client applications, refer to topic “*Actions*” on page 195.

For information on action layouts in TUI applications, refer to topic “*Function Keys and Actions*” on page 208.

Actions Available for Help Lists

A **help list** is a list of values available for a field that are offered to the end user as an input aid. These values are retrieved from the primary key of the BusinessObject’s target table linked to the mono-field foreign key of a Business Object’s primary table. The end user can choose among the values in the list. This helps prevent input errors.

The available actions are paging actions allowing the end user to go to the top of the list, to move forwards and backwards in the list, and to refresh or extract data in the help list (VAGTemplates does not provide these two latter actions in standard applications).

GUI: VAGTemplates allows you to specify two behaviors for help lists:

- all the pages read in the database are stored and displayed in the help list. In this case, the *Top* action fills the help list with the first data in the database; the *Next* action reads the next data in the database and memorizes it. To see the previous read page, the end user uses the scroll bar. The *Previous* action is not generated.
- only the current read page is displayed. In this case, the *Top* action fills the help list with the first data in the database; The *Next* and *Previous* actions access the next and previous data pages.

This behavior depends on the value set for the *Help list page display* parameter, Business Object Generation Parameters editor, *Foreign Key Help List* panel.

Note: Help lists are provided for details and updatable lists, they are not provided for lists.

TUI: The fields for which a help list is provided are signaled with an asterisk (*).

For information on help lists' action layout in GUI client applications, refer to "*Help List Action Layout*" on page 191.

For information on help list's action layout in TUI applications, refer to "*Function Keys and Actions*" on page 208.

Automatic Zoom in (GUI)

VAGTemplates provides your GUI client applications with a function for automatically zooming in on a particular row in a list and presenting the corresponding data in a detail. This function is generated when you call the same Business Object twice in a window: once as a list, once as a detail. When the end user double-clicks on a row in the list, the detail is refreshed with the corresponding data.

In Java/Smalltalk applications, the same behavior is generated when a Business Object is called once as a list and once as a detail in two windows that have a parent-to-child link.

A practical example of intra-window zoom in is provided in "**Presenting a List and a Detail in the Same Window (GUI) or Map (TUI)**" on page 112. A practical example of inter-window zoom in is provided in "**Presenting a List and Detail in Two Different Windows (GUI) or Maps (TUI)**" on page 122.

Data Transfer between TUI Maps

VAGTemplates provides your TUI applications with a function for automatically transferring the information of a selected row in a list to a detail in another map. This function is generated when a list calls a detail as child map. The end user only has to enter the action code of the detail in the action field of the selected row and press enter; the detail map opens displaying the corresponding data.

An example of data transfer is given in "**Presenting a List and Detail in Two Different Windows (GUI) or Maps (TUI)**" on page 122.

Another standard function allows transfer of updates from an updatable list to a detail called in the same map. For example, if the detail displays

information on a staff member that is also displayed in the list, and if the end user updates the data in the list, the detail is automatically refreshed with the updated data.

An example of data update transfer is given in “**Presenting a List and a Detail in the Same Window (GUI) or Map (TUI)**” on page 112.

Error Handling in GUI Client applications

The following subchapter describes generated Standard Error Handling functions and provide a GUI for error management.

By default, the generated GUI client applications provide three check levels:

- unitary checks handling errors in input fields;
- global checks handling error in field consistency,
- server checks handling errors during a server access.

These checks are conditioned by the value of the *Control location* parameter, Workspace Definition editor, *Client/Server Control* panel.

Unitary Check: Errors in Input Fields

• Check

If the *Control location* parameter is set to *client* or *client and server*, user input validation is automatically performed by the client, at input time, on each field losing the focus, according to the following Data Element specifications:

- *format and type*: check whether or not the value input for a numeric field is a numeric value;
- *requirement*: check whether or not the required field is empty;
- *defined interval or value table check*: check whether or not an input value appears in a value table.

If the *Control location* parameter is set to *server* only the format and type controls are made by the client.

• Feedback

When an error is detected, the field in error background color turns red (default error color). When the focus is back in the field, to correct the error, the field’s color changes back to normal.

VAGT: A parameter in the Workbench allows you to modify the color of fields in error: *Error color* parameter, Workspace Definition editor, *Client* panel.

For information on how to specify a Data Element’s type and format, refer to Part 1, “Part 2. The VAGTemplates Workbench” on page 7, “Chapter 3.

Information Model Entities and their Editors” on page 51, “**Data Element**” on page 76 , “**How to Define a Data Element**” on page 79.

Global Check: Field Consistency

- **Check**

If the *Control location* parameter is set to *client* or *client and server*, the global check is performed any time an update or a create action is triggered or when the end user requests it by triggering the *Check* action.

Note: The *Check* action performs a series of comparative unitary checks on all Business Object fields within all the Business Object’s layouts.

For example, if the Mr radio button is selected, and the value input in the Sex field is F, the global check will detect the error.

This action is originated from the Business Object part (see “*Business Object Bean/Part*” on page 293)

The global check consists of the following check sequence:

- performing a series of comparative unitary checks, *via* the ErrorHandling part (see “**Unitary Check: Errors in Input Fields**” on page 153), calling possible inter-field checks.

VAGTemplates provides your generated application with empty parts - the -CHK-HOOK Function (TUI) called by the server and the **additionalChecks** method (GUI) called by the client - that let you implement inter-field checking. You can add error checking code in these hooks. By default, these hooks are called by this global check, even if they are empty.

If the *Control location* parameter is set to *client*, only the format is checked.

This global check originates from the Business Object part (see “*Business Object Bean/Part*” on page 293)

- **Feedback**

If an error is detected, the background color of fields in error turns red (default error color) and an error message window opens displaying the list of the associated error messages.

Each message consists of:

- a short sentence describing the nature of the error,
For example, "YEARS: value is not in value table"
- an "OK" push-button, which closes the window,
- a "Help" push-button, which calls the on-line help for the window calling the Business Object on which the action was performed, if you implement this behavior in the final application.

When the focus is back in the field, to correct the error, the field's color changes back to normal

VAGT: You can parameterize the fields in error background colors in the Workbench: *Error color* parameter, Workspace Definition editor, *Client* panel.

Server Check: Access Errors

The server detects errors on persistent data that result from the activation of an erroneous create or update action (creation of an existing row, for example).

If the *Control location* parameter is set to *server* or *client and server*, the server checks are the following:

- checking whether or not required fields are empty;
- checking whether or not fields accept the NULL value;
- checking whether or not inputted values appear in the defined value table, for example.
- calling possible additional checks.

VAGTemplates provides your generated application with empty parts, called hooks, that let you implement additional checks (see “**Hooks**” on page 253). You can add error checking code in these hooks. By default, these hooks are called by this server check, even if they are empty.

If the *Control location* parameter is not set to *server* nor *client and server*, only the required and NULL fields are checked.

- **Feedback**

When the end user activates an update action, the server automatically checks whether the action can be fulfilled. If an error is detected, the user is informed whether he/she can correct the error (in case of an application error), or not (in case of a system error).

- Activating an action causes an *application error*:
 - the background color of fields in error turns red (default error color),
 - an error message window opens displaying the list of the associated error messages. Each message consists of:
 - a short sentence describing the nature of the error;
 - an “OK” push-button, which closes the window,
 - In the generated components you can add a “Help” push-button, which calls the on-line help for the window that calls the Business Object in which the action was performed, if you implement this behavior in the generated application.

- Activating an action causes a *system error*. A window opens presenting technical data on the error that the end user can then report to the system administrator.

VAGT: You can parameterize the erroneous field's color in the Workbench: *Error color* parameter, Workspace definition editor, *Client* panel.

Error Handling in TUI Applications

The following subchapter describes generated Standard Error Handling functions that provide a TUI for error management.

By default, the generated TUI applications provide two check levels:

- unitary checks handling format errors in numeric input fields;
- global checks handling errors during a server access.

These checks are implemented if the value of the *Control location* parameter is set to *server*.

Unitary Check: Format Errors in Input Fields

- **Check**

User input validation is automatically performed, at input time, on each numeric field, according to the format and the type specified for the Data Element in the Workbench. The end user is not allowed to enter values other than numeric values in numeric fields.

- *format and type*: non-numeric values cannot be entered in numeric fields;
- *requirement*: check whether or not the required field is empty;
- *defined interval or value table check*: check whether or not an input value appears in a value table.

- **Feedback**

When an error is detected, the fields in error background color turns red and an error map is displayed explaining the error.

For information on how to specify a Data Element's type and format, refer to Part 1, "Part 2. The VAGTemplates Workbench" on page 7, "Chapter 3. Information Model Entities and their Editors" on page 51, "**Data Element**" on page 76, "**How to Define a Data Element**" on page 79.

Global Check: Access Errors

Errors on persistent data can result from the input of invalid data or the activation of an erroneous creation action (creation of an existing row, for example). The check on invalid data is performed on all fields when the end user activates an elementary action.

- **Feedback**

The check invalid creation action is performed when the action is activated. The control automatically checks whether the action can be fulfilled or not. If an error is detected, the user is informed whether he/she can correct the error (in case of an application error), or not (in case of a system error).

- Activating an action causes an *application error*.

The color of the fields in error turns red (default color), and an error message map opens displaying the list of the associated error messages. Each message consists of a short sentence describing the nature of the error, "*Row not found - Check key field*", for example.

A Workspace parameter allows you to specify the presentation of error messages in the trailer of the current map (*Messages display* parameter, Workspace Definition editor, TUI panel). In this case, the first message is displayed, along with the total number of messages and its range among them. The end user can scroll the following error messages using the appropriate function key. When the errors are corrected, he/she can press enter to validate the input. The server will check it anew.

- Activating an action causes a *system error*.

The application exits the current map and opens the system error map that presents technical data on the error that the end user can report to the system team.

VAGT: You can parameterize the fields in error background color in the Workbench: *Error color* parameter, Workspace Definition editor, *Client* panel.

Management of the Navigation

Navigating Throughout a GUI Client application

By default, VAGTemplates generates navigation actions within your application. These actions are calls to child windows and make up a window dialog.

A **window dialog** is initialized by a window, defined as a root window. This window is the entry point to the final application.

The navigation possibilities from a window are the following:

- *upward navigation* by closing the window;
- *downward navigation* by opening child windows using the *Navigation* menu.
- *backward and forward navigation* to the open windows by using the *Windows* menu.

VAGT: A parameter in the Workbench allows you to modify the default *Navigation* menu title: *Navigation menu title* parameter, Interface Unit Generation Parameters editor, *Menu Titles* panel. A parameter in the

Workbench allows you to modify the default *Windows* menu title: *Windows menu title* parameter, Interface Unit Generation Parameters editor, *Menu Titles* panel.

The default labels of the items in the *Windows* menu are the titles of the open windows.

If there are more than 9 open windows, click on the *More* choice to view the whole list of open windows.

Navigating Throughout a TUI Application

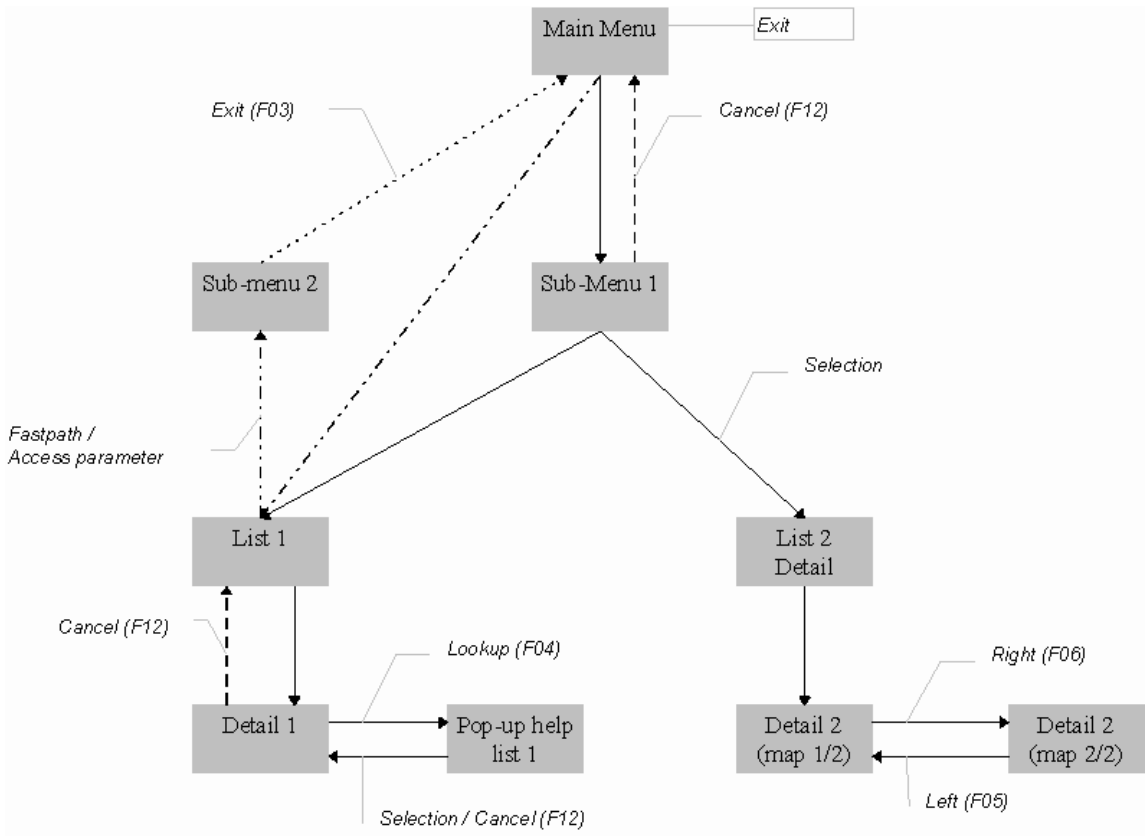
By default, VAGTemplates generates navigation actions within your application. These actions are calls to child maps and make up a navigation tree. In addition, the TUI application allows the end user to move directly to any application main map using its fastpath, which is a kind of map identifier, whatever its location in the navigation tree.

The navigation tree is initialized by a root map, which contains the calls to its child map, but no parent map. This map is the entry point to the final application.

The root map is often used as a menu that allows access to any child map, using a fastpath, and to select data in the child map's Business Object(s), using access parameters.

Navigation Possibilities: The navigation possibilities from a map are the following:

- *upward navigation* by pressing the *Cancel* (F12) function key, which opens the previous map, or the *Exit* (F03) function key, which opens the last menu in the navigation tree.
- *downward navigation* by calling up the child maps using their associated selection codes or function keys, or by calling up any map in the application using its fastpath.
- *horizontal navigation* by calling up the linked maps within the map set using the *Left* (F05) and *Right* (F06) function keys.



Note: Parameters in the Workbench allow you to modify the function keys' codes and labels (Workspace Definition editor, *Function Keys* panel).

Access Parameters and Extract Criteria: You can use access parameters to directly access a map which displays only the data you want to view.

For example, you can access directly from the main menu: the list of the staff members that work in Washington, if the Location field is defined as extract criteria, the detail of the staff member no. 360, if the Id Number field is the Business Object key. You only have to enter Washington/360 in the fastpath line of the application menu.

```

SYST: OS2CICS                MYAPPLICATION                FASTPATH: MENU
USER: PC user                 Main menu                    07-29-1997 12:54:59

S FASTPATH                    ACCESS_PARAMETERS
A DETAIL Staff details        ID NUMBER
B LSTDET List and Detail      (LOCATION)/ID NUMBER
C LIST Staff list             (LOCATION)

SELECT: _

FASTPATH: _____ WASHINGTON/360 _____

PF01: HELP   PF02:          PF03: EXIT   PF04:          PF05:          PF06:
PF07:          PF08:          PF09:          PF10:          PF11:          PF12: CANCEL

```

Access parameters can be key fields and/or extract criteria. Key fields are indicated for both details and lists; these fields are required. Extract criteria are indicated only for lists as they are usually used to extract data from lists; these fields are not required, that is why they are indicated between brackets. When extract criteria and key fields are defined for a Business Object, they are always separated with a slash (/).

Note: The end user must type a slash to access data according to the extract criteria and the key. If the end user wants to access data according to two extract criteria, he/she must enter the extract criteria separated with a comma as follows:

ARENT, WASHINGTON

On-Line Help

VAGTemplates provides your applications with an on-line help function provided that you have entered the corresponding functional documentation in the Workbench (*On-line help description* field, all Definition editors of all Information Model entities except the Value Style).

GUI On-Line Help

Each window in the generated application includes a *Help* menu with the standard Help menu items (*Help index, Using help, Product information*). This help menu also offers one particular item, *General Help*, providing three levels of help: direct help for the current window, then help for Business Objects, and then again for input fields.

TIP: The on-line help for a current window or field can also be activated by pressing the F1 key.

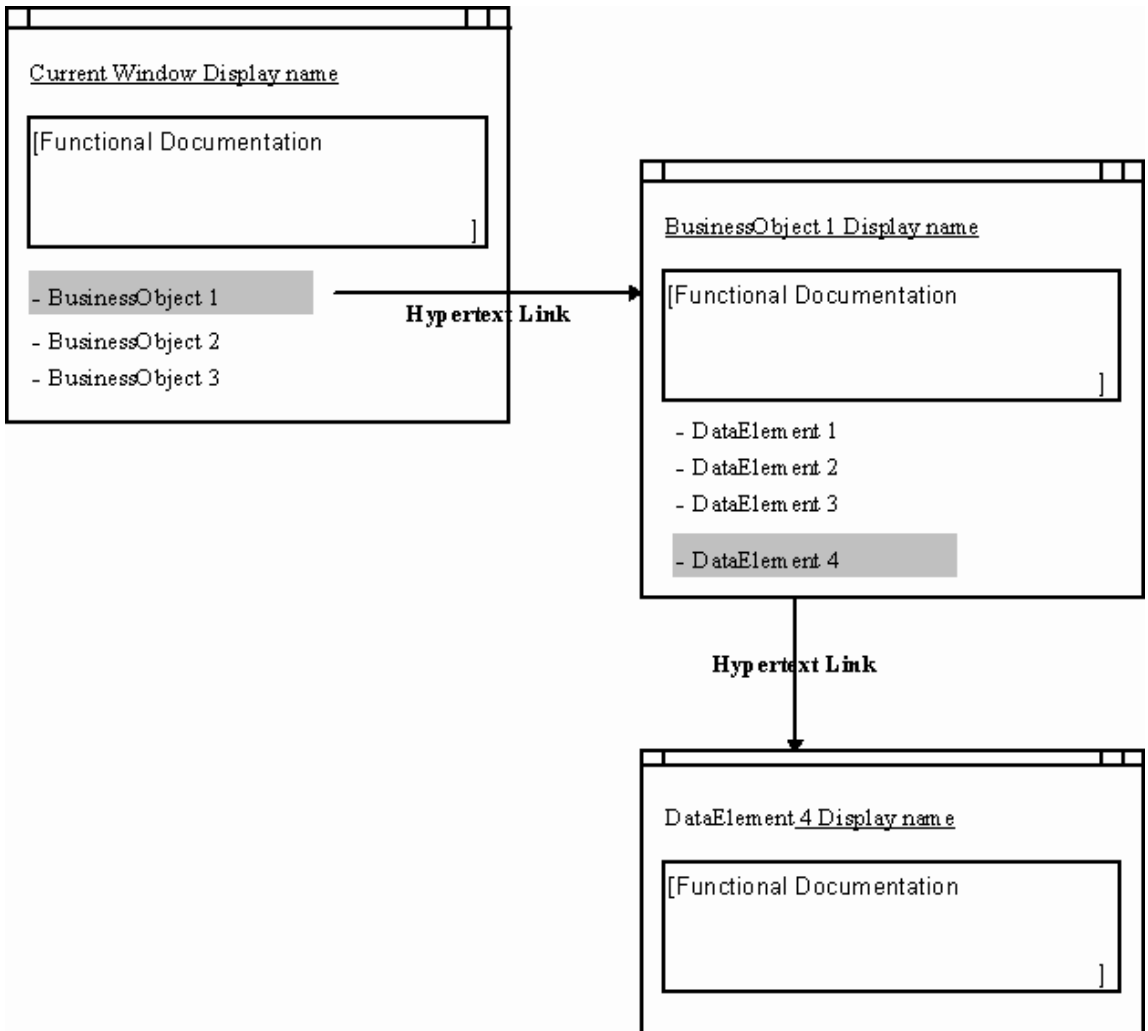


Figure 3. GUI On-Line Help Architecture

Help for Windows: By clicking on the *Help* menu or pressing F1 on a window with the focus, the end user opens a help panel. (The panel also opens by clicking the *Help* push-button in an error message window if you implement this behavior in the generated application.)

The current window help panel includes:

- a title, which is the display name that you specified in the *Display name* field when defining the Interface Unit;
- a functional documentation section, which displays the documentation you previously entered in the *On-line help description* field when defining the Interface Unit;
- a list area, displaying the display names of the Business Objects called in the window. From there, the end user can call the help panel of the selected Business Object.

TIP: By default, the list area includes all the Business Objects *called* in the current window. If you have not provided help for some of the Business Objects, you can remove items from the list directly in the help file (.IPF or .RTF and .HPJ).

Help for Business Objects: The end user activates the help for Business Objects by selecting one of the items from the list area, in the window help panel. (The panel also opens by clicking the *Help* push-button in an error message window if you implement this behavior in the generated application.)

The Business Object help panel includes:

- a title, which is the display name that you specified in the *Display name* field when defining the Business Object;
- a functional documentation section, which displays the documentation you previously entered in the *On-line help description* field when defining the Business Object;
- a list area, presenting the display names of the fields called in the Business Object. From there, the end user can call the help panel for the selected Data Element.

TIP: By default, the list area includes all the Data Element *called* in the current window. If you have not provided help for some of the Data Elements, you can remove items from the list directly in the help file (.IPF or .RTF and .HPJ).

Help for Fields: By pressing F1 on a field with the focus, in a window, the end user opens a help panel about the focused field. The panel also opens when selecting the corresponding item in the Business Object help panel (or clicking the *Help* push-button in an error message window if you implement this behavior in the generated application).

The field help panel includes:

- a title, which is the display name that you specified in the *Display name* field when defining the field;

- a documentation section, displaying the documentation you entered in the *On-line help description* field while defining your Data Element.

An example of on-line help generation and customization is given in “**Generating On-Line Help (VAGTemplates on Smalltalk Example)**” on page 128.

For information on how to define an entity, refer to Part 1, “Part 2. The VAGTemplates Workbench” on page 7, “Chapter 3. Information Model Entities and their Editors” on page 51.

VAGTemplates on Java: Implementation Principle: Two types of help are available on request in the generated application:

- help on field (selected field)
- help on window (selected window)

To be aware of a request, all the generated windows must be listening to the **KeyEvents**. When a request is emitted, it is transferred to the **MdlCommonServices** class which is in charge of the actual action on the browser. At this stage, if the request fails (because no help is available, or the system cannot retrieve a browser to display the help, or for an unknown reason), a message box opens to indicate what has happened. A request to the **MdlCommonServices** class is made using a **JFrame** (the window from which the request has been made) and a **Resource Bundle** from where the selected component can be retrieved and the associated page the browser must display.

The **help(JFrame, String)** method must be defined in the **MdlCommonServices** class.

The following components must be defined for every window:

- the **helpResourceBundle** field
- the **processKeyEvent(KeyEvent)** method
- the **help()** method
- the connection between the help menu item and the **help()** method

TUI On-Line Help

Caution: In order that the end user gets on-line help in the final application, you must also generate the application with the *TUI help* generator.

The TUI applications are provided with on-line help for each map, which is activated by pressing the F01 key. This on-line help provides three levels of help: direct help for the current map, then help for Business Objects, and then again for input fields.

VAGT: A parameter in the Workbench allows you to modify the default key's code and label: *Help* parameter, Workspace Definition editor, *Function Keys* panel.

The on-line help is presented by default as a pop-up map that opens at the bottom of the current map. A parameter in the Workbench allows you to display the on-line help on the full screen in a separate map: *Display popup* parameter, Workspace definition editor, *TUI* panel.

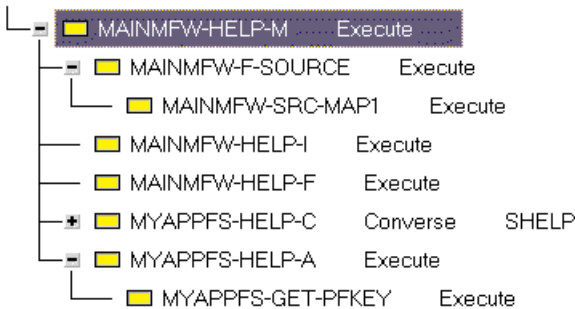


Figure 4. TUI On-Line Help Architecture

Help for Maps: To get help for the current map, the end user must position the cursor anywhere in the map, except on fields and on action codes, and then press F01.

The current map's help panel includes:

- the display name that you specified in the *Display name* field when defining the Interface Unit;
- a functional documentation section, which displays the documentation you previously entered in the *On-line help description* field when defining the Interface Unit;
- the display name(s) of the Business Object(s) called in the map, and its(their) functional documentation;
- the display names of the fields called in the Business Object(s), and their functional documentation.

TIP: By default, the help panel presents the documentation of all the Business Objects *called* in the current map. If you have not provided help for some of the Business Objects, you can remove their documentation directly in the Table where the help texts are stored.

Help for Business Objects: To get help directly on the Business Object, the end user must position the cursor in an action code and press F01.

The Business Object's help map includes:

- the display name that you specified in the *Display name* field when defining the Business Object;
- a functional documentation section, which displays the documentation you previously entered in the *On-line help description* field when defining your Business Object;
- the display names of the fields called in the Business Object(s), and their functional documentation.

TIP: By default, the help panel presents the documentation of all the Data Elements *called* in the current map. If you have not provided help for some of the Data Elements, you can remove their documentation directly in the Table where the help texts are stored.

Help for Fields: By pressing F01 on a field with the focus, in a map, the end user opens a help panel about the focused field.

The field's help panel includes:

- the display name that you specified in the *Display name* field when defining the Data Element;
- a documentation section, displaying the documentation you entered in the *On-line help description* field while defining the Data Element.

Note: The end user can request help on updatable fields only.

An example of on-line help generation and customization is given in “**Generating On-Line Help (VAGTemplates on Smalltalk Example)**” on page 128.

For information on how to define an entity, refer to Part 1, “Part 2. The VAGTemplates Workbench” on page 7, “Chapter 3. Information Model Entities and their Editors” on page 51.

Edition Functions (GUI)

The VAGTemplates generated GUI client applications provide standard Copy, Cut and Paste functions that can be activated *via* the *Copy/Cut/Paste* choices in an *Edit* menu or *via* keyboard shortcuts (shift+del, ctrl+ins, shift+ins).

The text selected by the end user is placed in the clipboard, and may be pasted in another input field.

VAGT: A parameter in the Workbench allows you to modify the label of the *Edit* menu and edition actions: *Edit menu title*, *Copy label*, *Cut label*, and *Paste label* parameters, Interface Unit GenerationParameters editor, *Edit Menu* panel.

VAGTemplates on Java: Implementation Principle

The following methods must be implemented and associated with their corresponding function:

- the **copy()** method

```
public void copy() {
    /* copy */
    Component c = this.getFocusOwner();
    if (c instanceof JTextComponent) {
        JTextComponent text = (JTextComponent) c;
        // JTextFields can't save the selection on loosing focus
        if (text.getSelectedText() == null) {
            text.selectAll();
        }
        text.copy();
    }
}
```

- the **cut()** method

```
public void cut() {
    /* cut */
    Component c = this.getFocusOwner();
    if (c instanceof JTextComponent) {
        JTextComponent text = (JTextComponent) c;
        // JTextFields can't save the selection on loosing focus
        if (text.getSelectedText() == null) {
            text.selectAll();
        }
        text.cut();
    }
}
```

- the **paste()** method

```
public void paste() {
    /* paste */
    Component c = this.getFocusOwner();
    if (c instanceof JTextComponent) {
        ((JTextComponent) c).paste();
    }
}
```

When the **Edit** menu is selected, the **calculateEditActionAvailability()** method is called to enable/disable the paste menu. This method is thus called on the menu event.

```
public void calculateEditActionsAvailability() {

    // Enable Paste action according to Clipboard
    Clipboard clipboard = Toolkit.getDefaultToolkit().getSystemClipboard();
    Transferable content = clipboard.getContents(this);

    // Check if the clipboard is empty
    if (content == null) {
        getPasteJMenuItem().setEnabled(false);
    }
}
```

```
} else {  
getPasteJMenuItem().setEnabled(true);  
}  
}
```

Prompt on close

If a window contains a detail that displays a persistent Business Object instance that has been modified by the user, then a prompt pops up when the window is being closed, asking if you want to really exit or not.

VAGTemplates on Java: Implementation Principle

All the VAGTemplates generated windows are children of the JFrame class. Therefore they emit all the window's events (on opening, closing, ...) that can be intercepted through the processWindowEvent(WindowEvent) method. The idea is to redefine this method, check if the window is closing through the event and prompt the user if necessary.

Note:

Only the detail updates are handled by this system.

The message displayed in the prompt from the CONFIRM_ON_EXIT_MESSAGE static field defined in the MdlCommonServices class.

Windows Menu

This menu is generated for every window and displays the list of all the open windows.

When you select a window item in the list, the corresponding window is shown.

When you close a window, the **Windows** menu from all the open window is automatically updated.

VAGTemplates on Java: Implementation Principle

Every window is responsible for its registration/unregistration in the MdlWindowsRegistry.

Each time a window is subjected to an open request, it self-registers (**registerWindow()**>> **addWindow()** in the **MdlWindowsRegistry**). During a close request, the windows self-unregisters (**unregisterWindow()** >> **removeWindow()** in the **MdlWindowsRegistry**). If these two actions are performed at the same time, the MdlWindowsRegistry **windows** attribute emits a property change event, listened by all the active windows. They immediately change their **Windows** menu list content, using the common method **updateMenu()** in the **MdlWindowsRegistry** according to the

windows attribute. Thanks to the ActionListener defined for the **MdlWindowsRegistry** , when the user selects a menu item in the **Windows** menu list , the system retrieves the selected window and show it, using the **showSelectedView(int)** method. Remark:

Note: The **MdlWindowsRegistry** is a singleton, instantiated once when the first window is executed.

The following components must be defined for every window:

- the **getWindowsRegistry()** method to retrieve a singleton of the **MdlWindowsRegistry** class.
- the **register/unregisterWindow()** methods.
- the **updateWindowsMenu()** method.
- the **show[Window]View()** method (because of the Garbage Collector).
- the **ProcessWindowEvent(WindowEvent)** method.
- the **WindowsRegistry.windows(propertyChangeEvent)** connection to call the **updateWindowMenu()** method.

The following components must be defined for the MdlWindowsRegistry class:

- the **ActionEvent** listener to listen to actions on menu items.
- the **actionPerformed(ActionEvent)** method.
- the **addWindow(JFrame)** method.
- the **removeWindow(JFrame)** method.
- the **isWindowRegistered(JFrame)** method.
- the **updateMenu(JMenu)** method.
- the **showSelectedView(int)** method.

BiDi Applications

The applications generated by VAGTemplates have bi-directional ability:

- The text components allow the user to enter either arabic (from right to left) or latin (from left to right) characters.
- The visual components can be laid-out from left to right or from right to left.

To select the orientation of the generated components, the VAGTemplates Workbench provides the following two BiDi parameters:

- the *horizontal orientation* in the Workspace Definition editor
- the *horizontal orientation* in the Data Element Generation Parameters editor

The first parameter is used to define the global orientation layout of the visual components. The second is used for data edition and display.

VAGTemplates on Java: Implementation Principle

The VAGTemplates on Java generators set the Swing property `java.awt.ComponentOrientation` to `LEFT_TO_RIGHT` or `RIGHT_TO_LEFT` and layout the visual components according to the values specified for the Information Model BiDi parameters.

Note: The `ComponentOrientation` Swing property exists only in Java 2 (since JDK 1.2). This JDK is not stabilized in VisualAge for Java; that explains that some visual components (e.g. `JTable`, `JFrame`) are not reversed.

Standard Layouts of GUI Client applications

The VAGTemplates RAD generators know a set of layouts for Data Elements, Business Objects, Interface Units and actions. They use the VAGTemplates default parameters to generate a default layout for each entity instance that you define. However, you can modify these parameters in the Workbench to customize the provided layouts.

This subchapter presents:

- the default layout generated with the default VAGTemplates parameters,
- the parameters you can modify to customize these layouts.

Fields

Default Layout

The fields' layout is linked to the check implemented on the Data Elements they present. By default, VAGTemplates assumes that a Data Element is not subjected to a check. The generated default presentation for fields is thus an edit box.

Layout Parameters

If you want other graphical presentations for your fields, you need to specify the corresponding parameters in the Workbench when defining your Data Element. When you implemented checks on the Data Element, only some of the presentations are available.

We indicate in the paragraphs below the presentations that are dedicated to certain check types. This does not imply that each check type is limited to these presentations; for example, you could have an edit box for a field controlled by an interval.

Fields Controlled by a Value Table:

VAGT: *Check type* parameter, *value table* value, Data Element Definition editor, *Check Type* panel.

The fields that are controlled by a value table can provide the end user with a help list displaying the contents of the value table.

Radio-button

A radio-button allows the end user to choose an exclusive value. There is one radio button per value in the value table. Radio buttons can be presented vertically or horizontally.



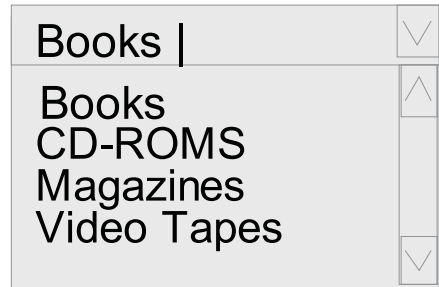
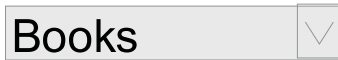
The text displayed with the radio buttons corresponds to the comments documenting the values you define in the value table.

VAGT: *Value display* parameter, *horizontal radio button*, or *vertical radio button* values, Data Element Generation Parameters editor, *Display* panel.

Drop-down list

A drop-down list represents the list of values available for a field. It comprises a read-only edit box and a list. The contents of the edit box changes when the end user selects a value in the list. The list part appears when the end user clicks on the arrow button of the edit box.

VAGT: *Value display* parameter, *drop-down list* value, Data Element Generation Parameters editor, *Display* panel.



A drop-down list can present the value of the field, it is termed *native drop-down list*, or the comment of the value, thus termed *textual drop-down list*.

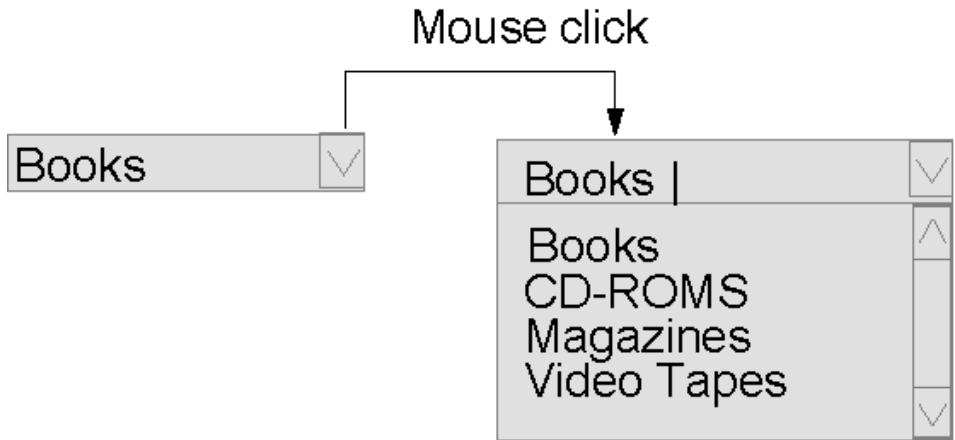
VAGT: *Comment display* parameter, *native* or *textual* values, Data Element Generation Parameters editor, *Display* panel.

Drop-down combo

A drop-down combo represents the list of values available for a field. It comprises an input edit box and a list. The end user can input a value in the edit box or select an item in the list by clicking on it.

The drop-down combo can be folded, i.e. the list part only appears when the end user clicks on the arrow button of the edit box, or unfolded.

VAGT: *Value display* parameter, *combo*, or *dropped-down combo* values, Data Element Generation Parameters editor, *Display* panel.



A drop-down combo can present the value of the field, it is termed *native drop-down combo*, or the comment of the value, thus termed *textual dropped-down combo*.

VAGT: *Comment display* parameter, *native* or *textual* values, Data Element Generation Parameters editor, *Display* panel.

Fields Controlled by an Interval:

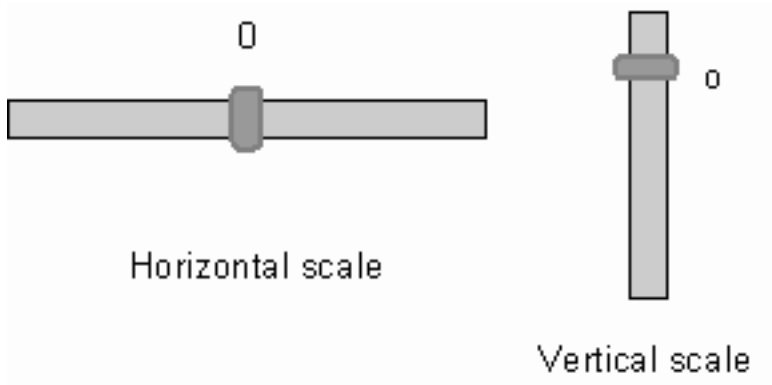
VAGT: *Check type* parameter, *interval* value, Data Element Definition editor, *Check Type* panel.

The fields that are controlled by an interval can provide the end user with an input aid; the interval of authorized values defined for the field can be displayed.

Scale

A scale is used to display the interval of values available for a numeric field. It enables the end user to move rapidly in a large interval.

A scale can be horizontal or vertical.



VAGT: *Value display* parameter, *vertical scale*, or *horizontal scale* values, Data Element Generation Parameters editor, *Display* panel.

Scales only display current values.

Other Fields:

VAGT: *Check type* parameter, *no check* value, Data Element Definition editor, *Check Type* panel.

These other fields do not offer a real input aid. However, an implicit check is activated according to the nature and/or the format of the field.

Edit

An edit can display information that was specified by an application or input by the end user.



A *read-only edit* displays information specified by an application but does not accept end user inputs. However, its contents can be selected and copied to the clipboard for further re-use.



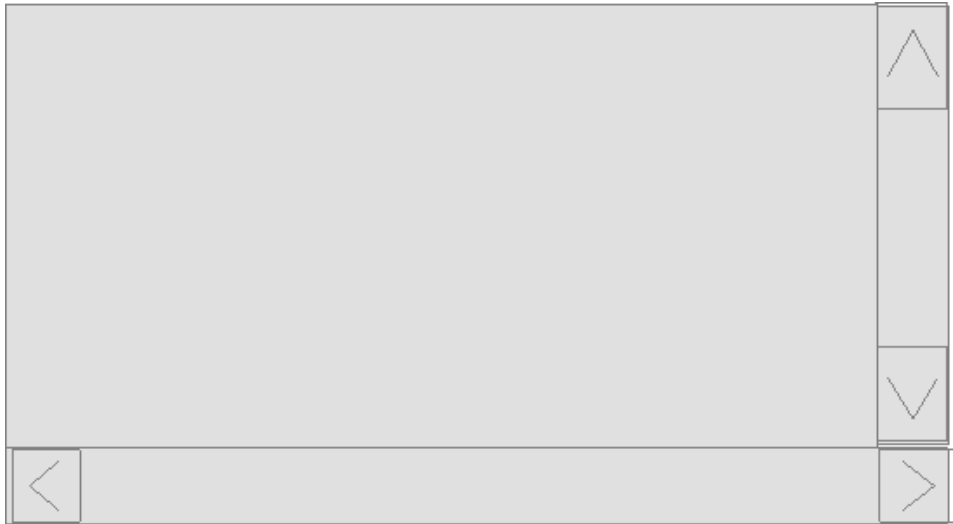
VAGT: *Value display* parameter, *edit*, or *read-only edit* values, Data Element Generation Parameters editor, *Display* panel.

Multi-line edit

A multi-line edit is useful for displaying a large amount of textual

information. The multi-line edit provides the end user with edition functions, such as insert mode, cut, copy, and paste.

The multi-line edit can be *read-only*.



VAGT: *Value display* parameter, *multi-line edit*, or *read-only multi-line edit* values, Data Element Generation Parameters editor, *Display* panel.

By default, the multi-line edit contains 4 lines, but you can modify it (*List size* parameter, Business Object editor).

Formatted edit (VAGTemplates on Smalltalk only)

A formatted edit is an alphanumeric or numeric input field that guides the end user's input *via* an edit mask.

VAGT: *Value display* parameter, *formatted edit* value, Data Element Generation Parameters editor, *Display* panel.

The edit mask is implemented by a regular expression made of particular characters that symbolize the input pattern. Enter the regular expression in the *Pattern* parameter, Data Element Generation Parameters editor, *Display* panel.

The valid characters for alphanumeric field edit mask are:

- B, for a blank character
- 9, for a number
- text between double quotes, for literals
- A, for a letter

- X, for any character

For example, a field displaying invoice references could have the following edit mask: A"-9999"-99"-9999 The authorized end user input could be:
E-1997-07-0124

The valid characters for a numeric field edit mask are:

- B, for a blank character
- 9, for a number
- text between double quotes, for literal values
- C, for currency
- D, for decimal separator
- T, for thousands separator
- S, for sign

For example, for a field displaying dividends the regular expression could be: S999T999D99C The authorized end user input could be: + 120,547.15\$

TIP: If the presentation of the numeric Data Element is defined by a Value Style, the value of the *Unit and sign alignment* parameter is used to define whether the unit and the sign are floating or not.

Note to VAGTemplates on Smalltalk users: The currency, decimal separator and thousands separator symbols are fixed by VisualAge Smalltalk Enterprise.

Password

A password field allows the end user to enter his/her confidential password, whose characters appear as asterisks (*).



VAGT: *Value display* parameter, *password* value, Data Data Element Generation Parameters editor, *Display* panel.

Static

A static displays the value of a field. It is read-only.



VAGT: *Value display* parameter, *static* value, Data Element Generation Parameters editor, *Display* panel.

Note: The above graphical presentations display the values of the fields, not their possible associated comments.

Presentation Label:

Note: A **presentation label** is a character string that can be displayed next to the graphical presentation of the Data Element. It helps the end user identify the nature of the displayed data.

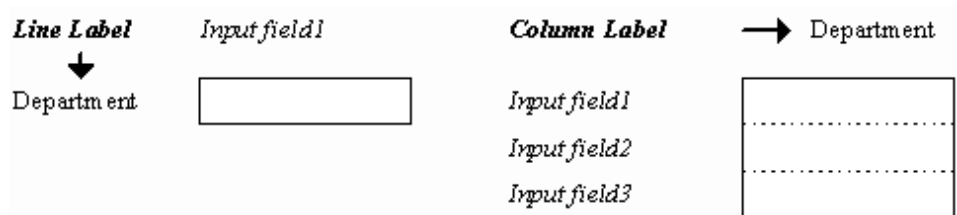
Two types of presentation labels are available:

- **line label:** This label is displayed with the graphical presentation of a Data Element, when a single occurrence of it is provided in a Business Object detail layout.

VAGT: *Default label* parameter, Data Element Generation Parameters editor, *Labels* panel.

- **column label:** This label is displayed with the graphical presentation of a Data Element, when several occurrences of it are provided, as in a container in Business Object list layouts. The column label can only be defined on one line.

VAGT: *Column label* parameter, Data Element Generation Parameters editor, *Labels* panel.



Field Layout Width:

VAGT: *Max display size* parameter, Data Element Generation Parameters editor, *Display* panel.

You can specify a maximum width for displaying fields in your layouts.

The maximum width will be the smaller size of the logical size of the displayed data - maximum number of characters for alphanumeric data,

capacity and precision for numeric data, etc. (see “**Input Mask**”) - and the size specified in the *Max display size* parameter.

For example, if you set the size of alphanumeric fields to 10 characters and the Max display size parameter to 5, the displayed field will be 5-characters long. To see the rest of the text, the end user can move the cursor to the end of the text in the field.

Input Mask

According to the type you specify for your Data Element - alphanumeric, numeric, date, time, timestamp - in the Workbench, end user input will be guided in some way, thanks to inherent input checks.

Alphanumeric Fields: The field accepts any character string. Checks apply to the following:

Size

The field does not accept additional characters when the maximum size is reached.

VAGT: *Size* parameter, Data Element Definition editor, *General* panel.

Letter case

Depending on whether you selected *uppercase*, *lowercase*, or *none*, the control modifies the input according to the letter case you specified, whatever the case used by the end user.

VAGT: *Case control* parameter, Data Element definition editor, *General* panel.

By default, the size of the field is one character; no case control is applied.

Numeric Fields: The field only accepts integer or decimal, signed or unsigned values. Checks apply to the following:

Decimal separator

You can specify a character - "." (*period*) or "," (*comma*) - as the decimal separator. If the end user omits to type the separator or types a wrong separator, it is corrected according to your parameter. The default separator is "." (*period*).

VAGT: *Decimal separator* setting, Value Style Definition editor, *Value Style* panel.

Thousands separator

You can specify a character - "," (*comma*), "." (*period*), or " " (*blank character*)- as the thousands separator, or no separator. If the end user

omits to type the separator or types a wrong separator, it is corrected according to your parameter. The default separator is "," (*comma*).

VAGT: *Thousands separator* setting, Value Style Definition editor, *Value Style* panel..

Positive sign

You can specify a character as the positive (plus) sign. If the end user omits to type the sign or types a wrong sign, it is corrected according to your parameter. The default positive sign is " " (blank character).

VAGT: *Positive sign* setting, Value Style Definition editor, *Value Style* panel.

Negative sign

You can specify a character as the negative (minus) sign. If the end user omits to type the sign or types a wrong sign, it is corrected according to your parameter. The default negative sign is "-" (minus sign).

VAGT: *Negative sign* setting, Value Style Definition editor, *Value Style* panel.

Sign position

You can specify whether or not a sign will be displayed, and where it will be positioned relatively to the value. If the end user's input does not correspond to your specification, it is corrected according to your parameter. The default value is *no sign*.

VAGT: *Sign position* setting, Value Style Definition editor, *Value Style* panel.

Unit

You can specify where the units will be positioned relative to the value. If the end user's input does not correspond to your specification, it is corrected according to your parameter.

VAGT: *Units* settings, Value Style Definition editor, *Value Style* panel.

Unit position

You can specify whether or not units will be displayed, and where they will be positioned relative to the value. If the end user's input does not correspond to your specification, it is corrected according to your parameter. The default value is *right*.

VAGT: *Unit position* parameter, Value Style Definition editor, *Value Style* panel.

Unit and sign alignment

You can specify where the units and the sign will be positioned relative to the value. If the end user's input does not correspond to your specification, it is corrected according to your parameter. The default value is *unit and sign separation*.

VAGT: *Unit and sign alignment* setting, Value Style Definition editor, *Value Style* panel.

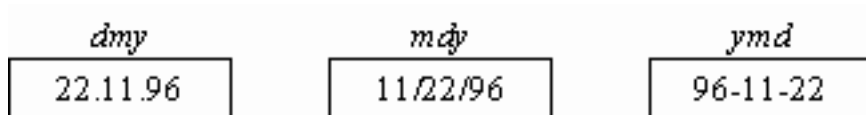
Note: Checks jointly apply to the capacity - number of digits before the decimal separator - and to the precision - number of digits after the decimal separator -you defined for numeric values.

For example, if the capacity equals 5 and the precision 2, the integer part of the value cannot exceed 5 digits and the decimal part 2 digits.

Date Fields: The field only accepts date values. Checks apply to the following:

Mask

The end user is required to type day, month and year values according to the order you specified - *dmy*, *mdy*, *ymd*. The default value is *mdy*.



VAGT: *Mask* setting, Value Style Definition editor, *Value Style* panel.

Separator

You can specify any character as a separator for days, months and years. If the end user omits to type the separator or types a wrong separator, it is automatically corrected according to your parameter.

VAGT: *Separator* setting, Value Style Definition editor, *Value Style* panel.

Year style

According to the style you specified - *short* or *full* - the date value will be displayed with 2 or 4 characters, the century characters being added by VisualAge Smalltalk Enterprise. The default value is *full*.

VAGT: *Year style* setting, Value Style Definition editor, *Value Style* panel.

Note: A default value is required for date fields.

Caution: Make sure the values you specify for these parameters are compatible with the date and time format parameters in the

VisualAge Smalltalk Enterprise hpt.ini file. VAGTemplates uses a VisualAge Generator method that uses parameters defined in this file.

VAGT: The *Date Format* parameter (Workspace Definition editor, *Server* panel) is responsible for hard-coding the date in the 4GL code. It ensures consistency between the end user's input and the internal date format required by VisualAge Generator. For VAGtemplates on Java, it is also responsible for retrieving data for Java clients.

Time Fields: The field only accepts time values. Checks apply to the following:

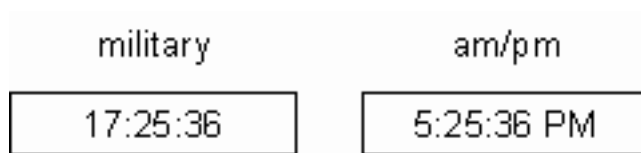
Mask

According to the mask you specified - *full time, no seconds* - the time value will or will not present the seconds. The default value is full time.

VAGT: *Mask* setting, Value Style Definition editor, *Value Style* panel.

Cycle

According to the cycle you specified - *military, AM/PM* - the end user is or is not allowed to type hours above 12. The default value is *military*.



VAGT: *Cycle* setting, Value Style Definition editor, *Value Style* panel.

Separator

You can specify any character as a separator for hours, minutes and seconds. If the end user omits to type the separator or types a wrong separator, it is automatically corrected according to your parameter.

VAGT: *Separator* setting, Value Style Definition editor, *Value Style* panel.

AM string

The field will display the characters you specified for morning times in AM/PM cycle - *AM, am, A, a*. The default value is *AM*.

VAGT: *Am string* setting, Value Style Definition editor, *Value Style* panel.

PM string

The field will display the characters you specified for evening times in AM/PM cycle - *PM, pm, P, p*. The default value is *PM*.

VAGT: *Pm string* setting, Value Style Definition editor, *Value Style* panel.

Note: A default value is required for time fields.

Timestamp Fields:

VAGT: *Time style* and *Separator* settings, Value Style Definition editor, *Value Style* panel.

Caution: The timestamp type is not recognized in VisualAge Generator, which considers it alphanumeric. Therefore, no check is insured at the end user's input.

The timestamp type is the concatenation of a date type and a time type.

You only have to specify the date and time styles you want associated to build your timestamp type, and a character for separating date and time values.

Note: A default value is required for timestamp fields.

For more information on how errors in input fields are handled, refer to "Unitary Check: Errors in Input Fields" on page 153.

To Display or Not To Display:

VAGT: *Display* parameter, Business Object Definition editor, *Field Attributes* panel.

By default, all the Data Elements *called* by a Business Object appear in the final application. However, the *Display* column allows you to choose the Business Object layouts for each Data Element:

- in the lists and the details (*always* value)
- neither in lists nor in details (*never* value)
- in details but not in lists (*on detail only* value)
- in lists but not in details (*on list only* value)

Detail Business Objects

Business Objects can appear as details or lists. In a detail, each individual field (except extract fields) can have a different layout. In a list, the layout affects all the fields presented in the list.

Default Layout

A Business Object appearing as a detail represents the columns from the relational table(s) that it calls as independent graphical fields.

The default Business Object detail layout generated with VAGTemplates comprises:

- a [label, value] pair, for each Data Element it calls,
- elementary actions (create, read, update, delete, check).

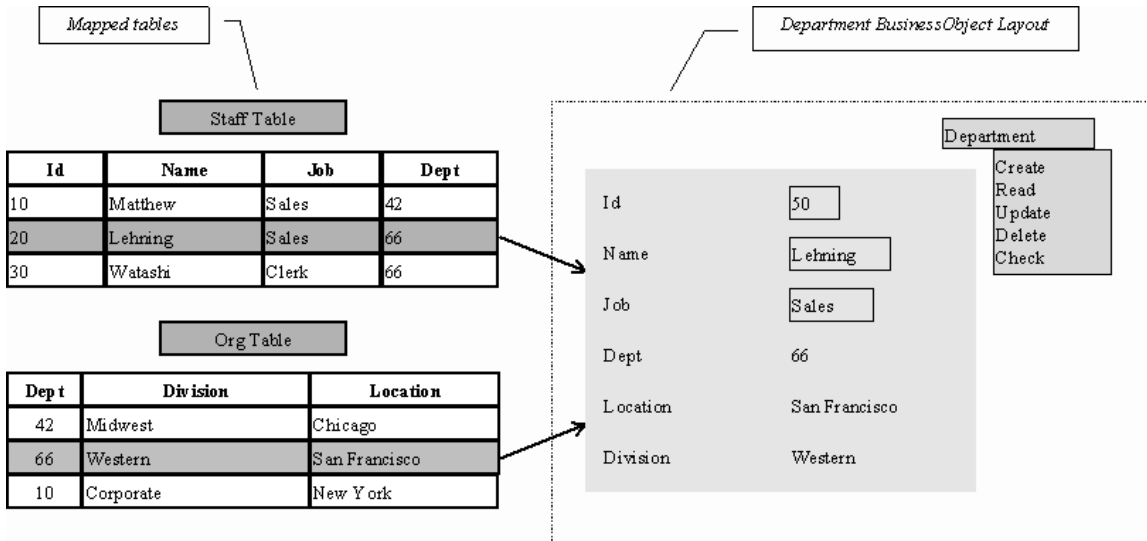


Figure 5. Default Business Object Detail Layout

Fields - labels and values - are arranged across the Business Object layout in the order in which they were specified in the Business Object, and grouped into a form with no frame.

Fields labels and values are horizontally aligned, the label being on the left of the value, and arranged from top to bottom then from left to right, on 8 rows.

The size of the fields is fixed, i.e. it does not vary with the size of the Business Object layout, and adjusted to its contents; the size of the labels is always fixed and adjusted.

Actions appear as standard menu items and the menu title corresponds to the Business Object identifier. They are labeled: Create, Read, Update, Delete, Check, New, Save.

Note: Actions are specified at the Business Object level and one action corresponds to one Business Object at a time.

For example, if two detail Business Objects are presented in the same window, the Update action will appear twice.

By default, all the Table's fields *mapped* by a Business Object appear automatically in the Business Object layout, but you can choose not to layout some of them (see "*To Display or Not To Display*" on page 181).

Layout Parameters

Anytime the default layout does not fit your particular needs, you can set several parameters to specify other graphical presentations. We describe, in the below subchapters, the sizing and arrangement modes available for the Business Object detail layout.

For information on how to specify Business Object parameters, refer to Part 1, "Part 2. The VAGTemplates Workbench" on page 7, "Chapter 3. Information Model Entities and their Editors" on page 51, "**Business Object**" on page 56, "**How to Define a Business Object**" on page 67.

Note: In this documentation we call a **line** several horizontally aligned [label+value] pairs, and a **column** one of these pairs.

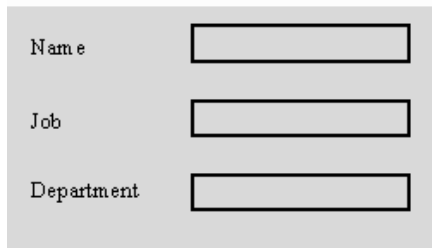
Detail Sizing: The size of the detail is always deduced from its contents (see the paragraphs below).

Presentation:

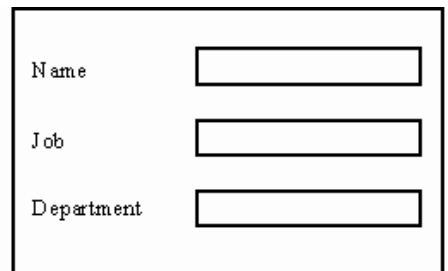
VAGT: *Display* parameter, Business Object Generation Parameters editor, *Detail View* panel.

Form with or without a border

Fields are grouped together in a limited area. This area can be either framed or not framed.



A form layout with three rows of labels and input fields. The labels are "Name", "Job", and "Department". Each label is followed by a rectangular input field. The entire form is set against a light gray background and does not have a border.



A form layout with three rows of labels and input fields, identical to the previous one. The labels are "Name", "Job", and "Department". Each label is followed by a rectangular input field. The entire form is enclosed in a black rectangular border.

Groupbox

Fields are grouped together in a labeled frame.

Employee

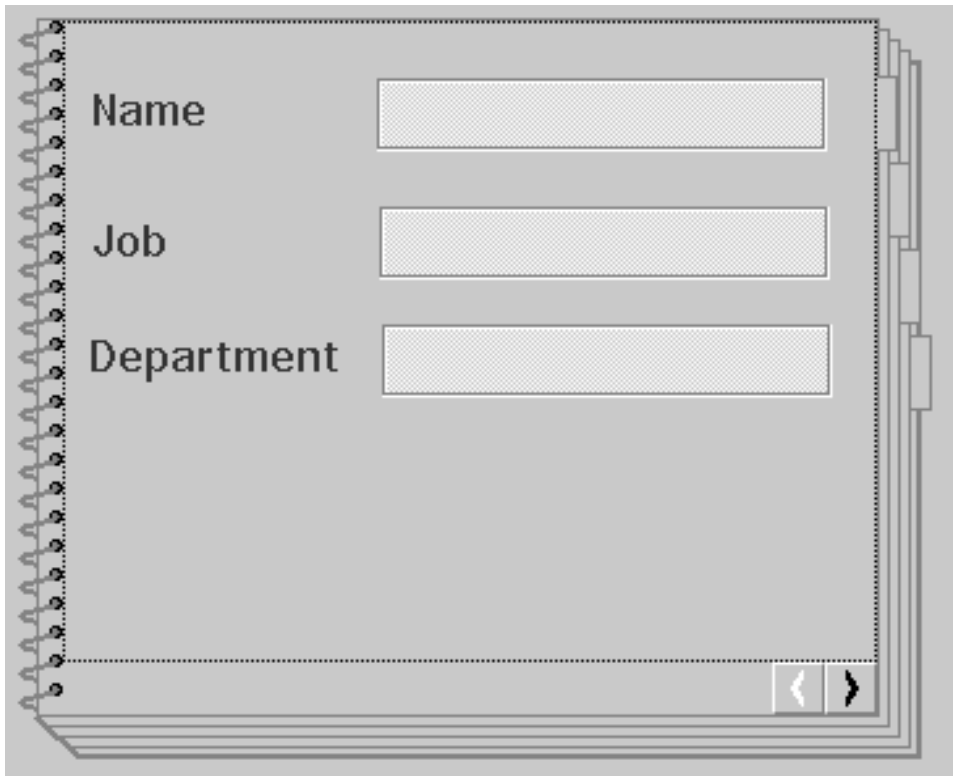
Name	<input type="text"/>
Job	<input type="text"/>
Department	<input type="text"/>

VAGT: You can specify the label of the frame in the Workbench: *Label* parameter, Business Object Generation Parameters editor, *Detail View* panel.

The above presentations allow you to specify the number of lines or columns available for displaying the fields, whether you choose a *vertical arrangement* or an *horizontal arrangement* respectively for the fields (see "*Arrangement*" on page 187).

Notebook

This presentation consists of one or more pages, identified with tabs.



This presentation allows you to choose the distribution of the fields on the page(s).

You can choose between a *notebook*, *Windows notebook*, and *PM notebook*.

You can specify:

- the number of lines and the number of columns available for displaying the fields on each page - *Number of columns*, *Number of lines* parameters, Business Object Generation parameters editor, *Detail View* panel.
- the number of fields per page and the number of lines or the number of columns (whether you chose a *vertical arrangement* or a *horizontal arrangement* respectively for the fields - see "**Arrangement**" on page 187) - *Number of fields per page*, *Number of columns*, *Number of lines* parameters, Business Object Generation Parameters editor, *Detail View* panel.

Note: The maximum number of lines or columns that can be displayed on a notebook page depends on the number and the presentation of the fields (edit, combo box, etc.). If the number of fields is too big, a new page is added to the notebook.

VAST: If you want to add a label to the notebook tabs, you must do it in the generated application.

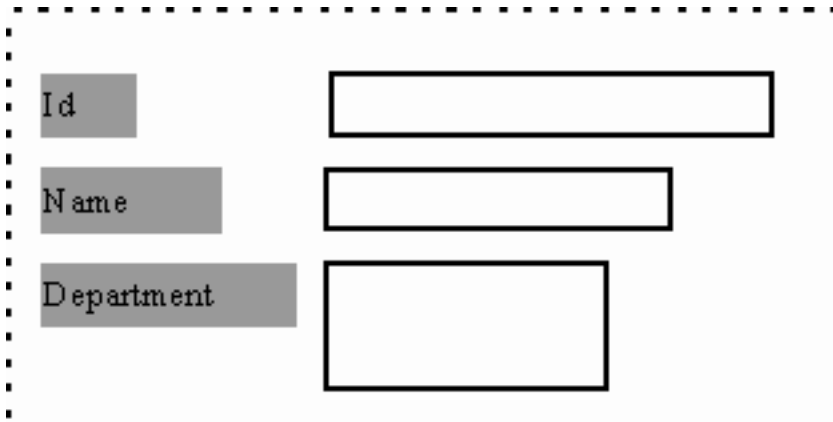
Alignment:

VAGT: *Alignment* parameter, Business Object Generation parameters editor, *Detail Field* panel.

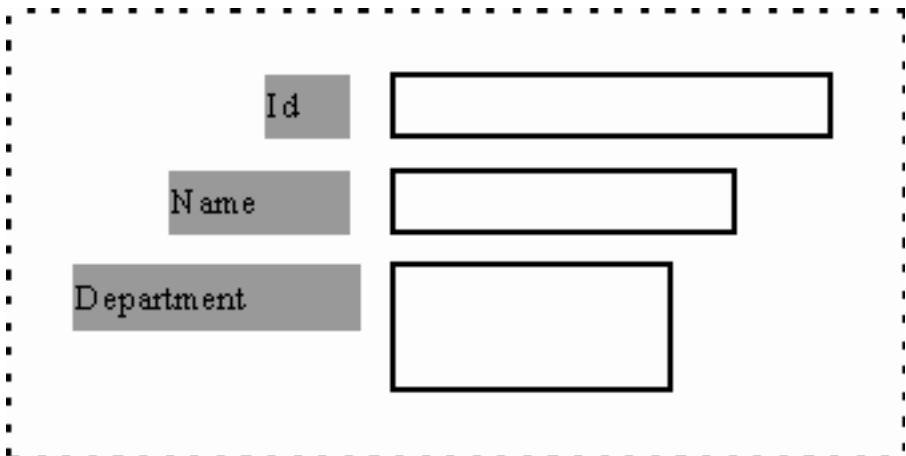
Labels are all the same height.

Horizontal alignment

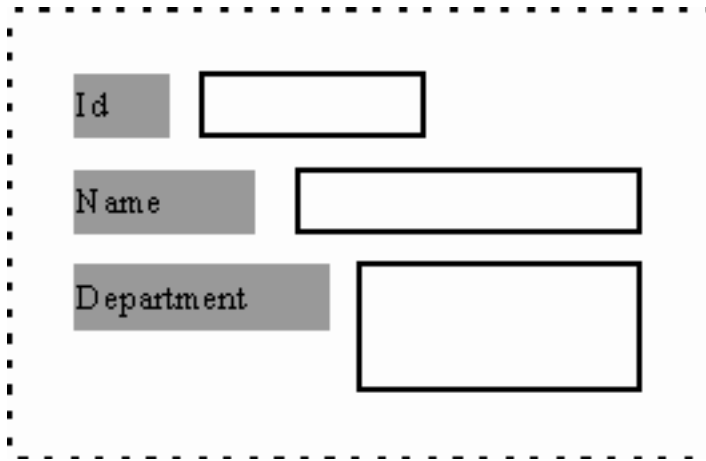
Left aligned labels / left aligned values.



Right aligned labels / left aligned values.



Left aligned joined labels and values.



VAST: If you prefer viewing your fields sprinkled across the Business Object layout, you will have to change the arrangement after the generation.

Arrangement:

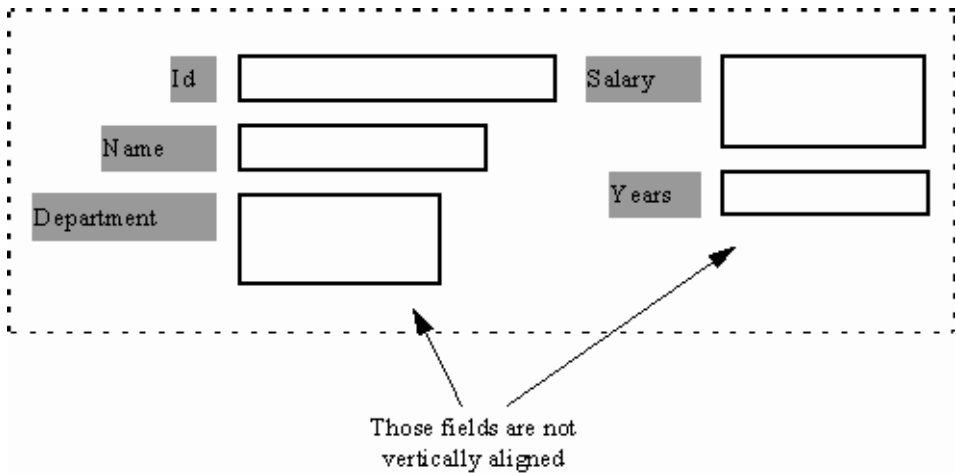
VAGT: *Layout* parameter, *top to bottom* or *left to right* values, Business Object Generation parameters editor, *Detail Field* panel.

The fields - labels+values - can be placed:

- from top to bottom, then from left to right (vertical arrangement),
- from left to right, then from top to bottom (horizontal arrangement).

You can also choose the number of columns for the horizontal arrangement, and the number of lines for the vertical arrangement (*Number of lines* and *Number of columns* parameters, Business Object Generation parameters editor, *Detail View* panel).

Note: In *vertical arrangement* mode, horizontal alignments are respected. Vertical spacing between [field+label] pairs is fixed. Therefore, vertical alignment is not guaranteed.



Sizing:

VAGT: *Sizing* parameter, Business Object Generation parameters editor, *Detail Field* panel.

The size of the labels is always fixed and adjusted.

Adjusted

The size of the fields is adjusted to the contents. It does not vary with the size of the Business Object layout.

Equalized

The size of the fields is initialized to the size of the contents, then equalized with the size of the largest field in the lines or the columns.

Label and Value Display:

VAGT: *Label and value display*, Business Object Generation parameters editor, *Detail Field* panel.

You have various possibilities for presenting field labels and values in the Business Object:

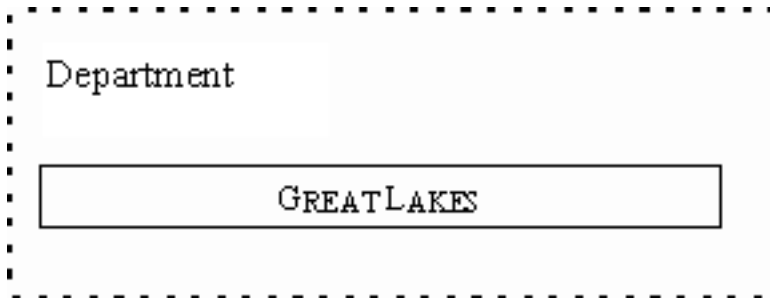
No label

The value in the field comes with no label.



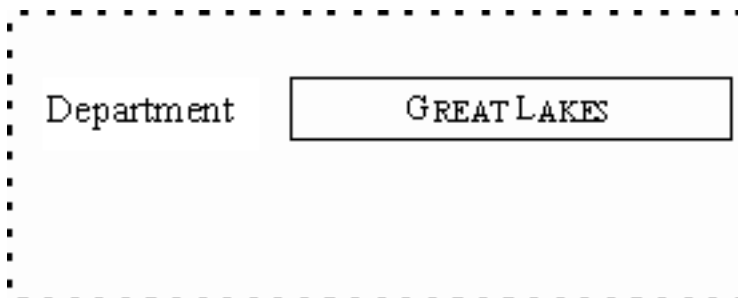
Vertical

The label appears above the value; both are left aligned.



Horizontal

The label appears to the left of the value; both are aligned on top.



Lists Sizing:

VAGT: *Help list size* parameter, Business Object Generation parameters editor, *Foreign Key Help List* panel.

You can set the number of lines in help lists. This number ranges from 1 to 20 lines. By default, lists have 4 lines.

Actions:

VAGT: *Display* parameter, Business Object Generation Parameters editor, *Actions and Labels* panel.

VAGTemplates provide the generated applications with default management actions (create, read, update, delete, check).

For information on the generated standard actions, refer to topic “**Actions Available for Detail Business Objects**” on page 147.

These elementary actions can appear as follows:

Push-button

Push-buttons are placed at the bottom of the layout. They are aligned and adjusted. (See “*Arrangement*” on page 187.)

VAGT: *Display* parameter, *push-button* value, Business Object Generation Parameters editor, *Actions and Labels* panel.

A parameter in the Workbench allows you to specify whether you want the push-buttons aligned on the left of the window, on the right or centered in the window: *Position* parameter, Business Object Generation Parameters editor, *Actions and Labels* panel.

Menu items

Actions can appear as menu items in a *menu*, or in a *pop-up menu*.

VAGT: *Display* parameter, *menu* and *pop-up menu* values, Business Object Generation Parameters editor, *Actions and Labels* panel.

VAGT: Parameters in the Workbench enable you to modify the default labels assigned to these actions: *Check label*, *New label*, *Create label*, *Read label*, *Save label*, *Update label*, *Delete label*, *Show message label* and *Menu label* parameters, Business Object Generation Parameters editor, *Actions and Labels* panel.

Help Lists

Help lists are provided for the mono-field foreign keys of a Business Object’s primary table, i.e. the first table mapped by the Business Object’s fields.

A **foreign key** is a field or a set of fields, used to identify or access particular rows in a table, whose values must correspond to one value in the primary key of the joined Table.

Help List Layout:

VAGT: *Help list for all foreign keys* parameter, Business Object Generation Parameters editor, *Foreign Key Help List* panel.

You can choose whether or not to provide help lists. If you check this option, a help list appearing as a combo box will be provided for all the mono-field foreign keys of the Business Object. This help list will be filled in with the values of the secondary tables' primary keys. Actions will also be available (see "*Help List Action Layout*"). If the option is unchecked, foreign keys will not appear as help lists but like the other fields.

Help List Action Layout: Help list paging actions are presented as push-buttons at the bottom of the Detail. The push-buttons corresponding to different foreign keys are located on separate lines. Their default labels are *Top*, *Forward*, *Backward*.

VAGT: Parameters let you modify the default label of help list actions: *Top label*, *Forward label*, *Backward label* parameters, Relational Table Generation Parameters editor, *Foreign Key Help List* panel.

For more information on help list paging actions, refer to topic "**Actions Available for Help Lists**" on page 151.

Note: The refresh and extract actions are not generated by VAGTemplates.

VAST: If you want to modify the layout of help list actions, you must do so after the generation.

Filling Help Lists:

VAGT: *Help list prefilled* parameter, Business Object Generation Parameters editor, *Foreign Key Help List* panel.

This parameter allows you to specify whether the help list will be filled with data or empty at window opening. If the parameter is unchecked, the end user will have to trigger the *Top* action to fill the help list with data.

A parameter allows you to specify whether the help list displays only the current page, or all the read pages: *Help list page display* parameter, Business Object Generation Parameters editor, *Foreign Key Help List* panel. In the first case, the end user will have to activate an action to page forwards and backwards throughout the help list. In the latter case, he/she can page backwards and forwards throughout the read pages with the scroll bar.

List Business Objects

A Business Object appearing as a list is used to represent the Business Object's fields as the columns of a unique table-type graphical component, called a **container**. The list can be read-only or updatable (*Layout type* parameter, Interface Unit Definition editor, *Business Objects* panel).

Default Layout

The default Business Object list layout generated with VAGTemplates comprises:

- a container, whose columns correspond to the fields of the Business Object,
- elementary actions.

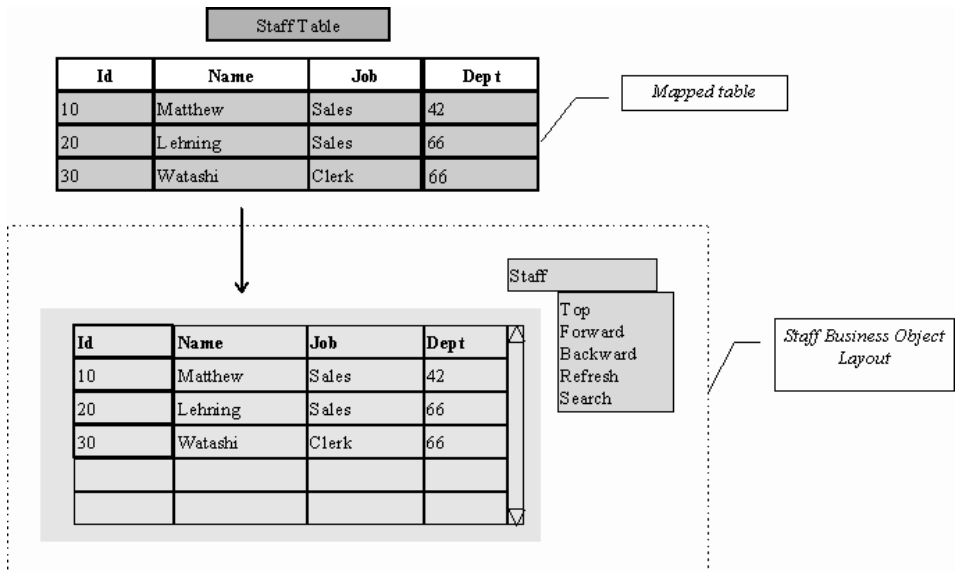


Figure 6. Default Business Object List Layout

The container is presented in a form with no frame. Its size is fixed, with 5 lines and 8 columns. It is laid out with a vertical scrollbar.

Each column comes with a heading and is adjusted to the maximum size of its largest label or field. Column headings are aligned on the top and on the left.

Actions are generated standard menu items and the menu title corresponds to the Business Object's identifier. The default action labels are: *Top*, *Forward*, *Backward*, *Refresh*, *Extract*, and *Submit* (updatable lists).

Note: Actions are specified at the Business Object level and one action corresponds to one Business Object at a time.

For example, if two list Business Objects are presented in the same window, the Refresh action will appear twice.

All the Table fields *mapped* by a Business Object appear automatically in the Business Object layout, but you can choose to not layout some of them (see “*To Display or Not To Display*” on page 181).

Layout Parameters

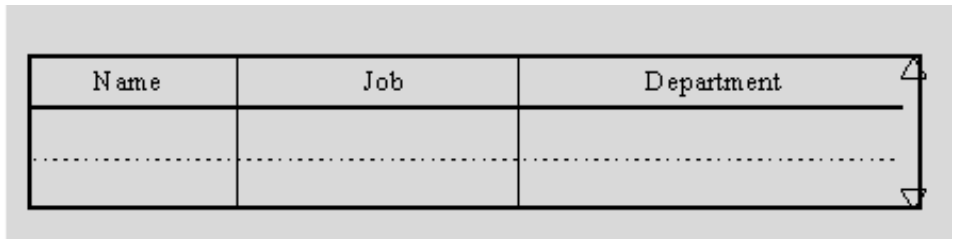
Anytime the default layout does not fit your particular needs, you can set several parameters to specify other graphical presentations. The available sizing and arrangement modes for the Business Object list layout are stated in the following subchapters.

Presentation:

VAGT: *Display* parameter, Business Object Generation Parameters editor, *List View* panel.

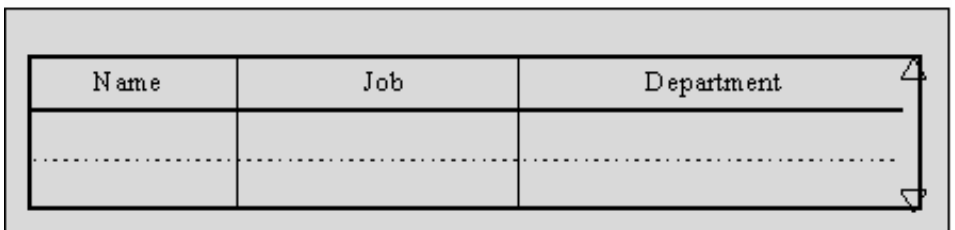
Form with or without a border

The container is presented within a limited area. The layout can either have a frame or not.



A diagram showing a rectangular form without an outer border. The form contains a table with three columns: "Name", "Job", and "Department". The table has two rows: the first row contains the column headers, and the second row is empty. A horizontal dashed line is positioned below the first row. On the right side of the table, there are two vertical arrows pointing in opposite directions, indicating a scrollable area.

Form

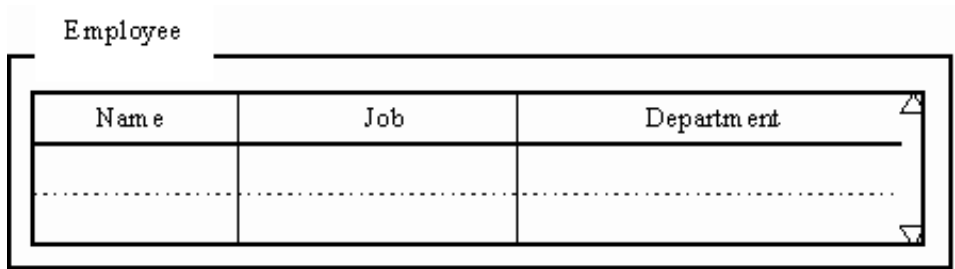


A diagram showing a rectangular form with an outer border. The form contains a table with three columns: "Name", "Job", and "Department". The table has two rows: the first row contains the column headers, and the second row is empty. A horizontal dashed line is positioned below the first row. On the right side of the table, there are two vertical arrows pointing in opposite directions, indicating a scrollable area.

Form with border

Groupbox

The container is framed in a labeled frame.



VAGT: You can specify the label of the frame in the Workbench: *Label* parameter, Business Object Generation Parameters editor, *List View* panel.

Sizing:

Number of columns

You can specify the maximum number of columns presented in the container. If there are more columns than the display possibility of the container, a horizontal scrollbar is provided.

VAGT: *Number of columns* parameter, Business Object Generation Parameters editor, *List Container* panel.

Number of lines

You can specify the number of lines you want to view in the container. This will condition the size of the graphical page.

VAGT: *Number of lines* parameter, Business Object Generation Parameters editor, *List Container* panel.

Filling the container:

VAGT: *List prefilled* parameter, Business Object Generation Parameters editor, *List View* panel.

This parameter allows you to specify whether or not the container will be filled with data at window opening.

If the parameter is unchecked, the end user will have to trigger the *Top* action to fill the container with data.

A parameter allows you to specify whether the list displays only the current page, or all the read pages, and whether paging is done by activating actions or using the scroll bar (see "*Actions Available for Read-Only Lists*" on page 148).

Actions:

VAGT: *Display* parameter, Business Object Generation Parameters editor, *Action and Labels* panel.

VAGTemplates provides the generated read-only list with default paging actions (display the first page, move back and forth to other pages, etc.), and updatable lists with default paging and data management actions (creation, update, deletion).

For information on standard generated actions refer to “**Actions Available for List Business Objects**” on page 148.

These elementary actions can appear as follows:

Push-button

Push-buttons are placed at the bottom of the layout. They are aligned and adjusted. (See “*Sizing*” on page 188, for explanations on this presentation).

VAGT: *Display* parameter, *pushbutton* value, Business Object Generation Parameters editor, *Action and Labels* panel.

A parameter in the Workbench allows you to specify whether you want the push-buttons aligned on the left of the window, on the right or centered in the window: *Position*, Business Object Generation Parameters editor, *Action and Labels* panel.

Menu item

Actions can appear as menu items in a *menu*, or in a *popup menu*.

VAGT: *Display* parameter, *menu* and *popup menu* values, Business Object Generation Parameters editor, *Action and Labels* panel.

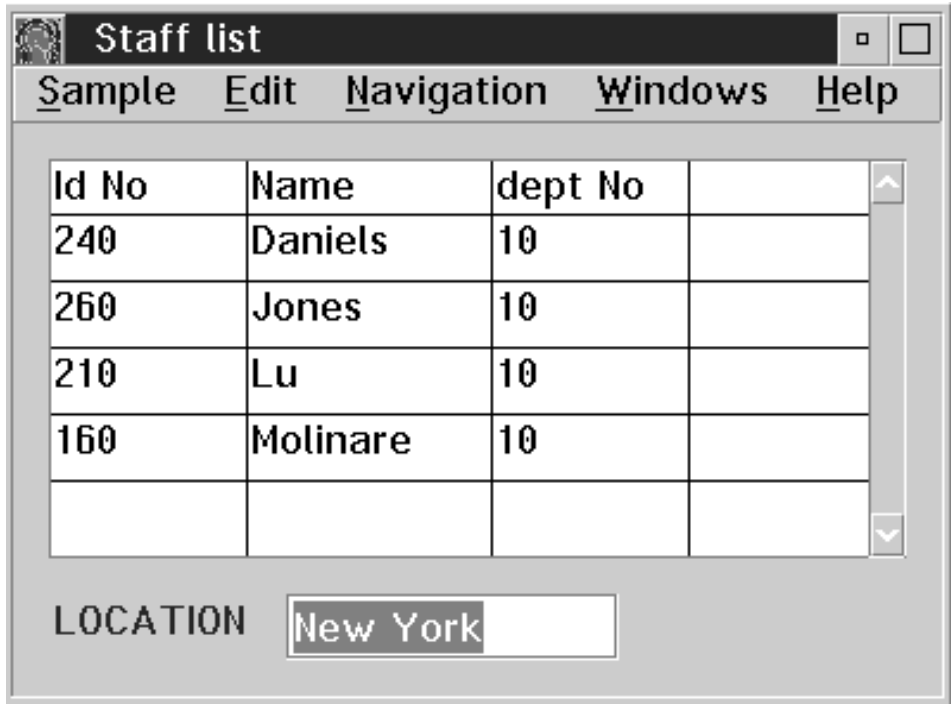
VAGT: Parameters in the Workbench allow you to modify the default labels assigned to these actions: *Top label*, *Refresh label*, *Extract label*, *Forward label*, *Backward label*, *Submit label*, *Menu label*, for updatable lists, Business Object Generation Parameters editor, *Action and Labels* panel.

Extraction Criteria Layout

VAGT: *Extraction criteria displayed* parameter, Business Object Generation Parameters editor, *List View* panel.

You can choose whether or not to provide extract criteria. If the parameter is checked, all extraction criteria will be displayed outside the container. Each

extraction criterion appears as an edit box, on a separate line. If the option is unchecked, the extraction criteria appear as the other fields.



In the above window, the *Location* field is used as an extraction criteria. It appears outside the container and allows the end user to enter a location and to retrieve the staff members that work at this location.

Windows

The generated window has two functions:

- *visualizing and grouping Business Objects*: application data are visualized in windows through the use of Business Objects, presented in details or lists, and manipulated *via* several actions.
- *defining a window dialog via its own integration in a navigation tree*: each window knows its child windows. Downward navigation is made possible by calling child windows, and upward navigation by closing the child windows. The **Windows** menu allows navigation between the open windows.

Default Layout

The default window layout generated with VAGTemplates comprises:

- zero, one, or more Business Object(s),
- navigation and standard actions.

The generated window is a standard window that comprises a title bar, a system menu, minimize and maximize buttons, resize handles, and a menu bar. The size of the window layout is adjusted to the contents. Its coordinates cannot be parameterized.

Business Object layouts are arranged from top to bottom then from left to right, and are centered in the layout.

Navigation actions appear as standard menu items and the menu title is *Navigation*. Each navigation action is labeled with the child window titles.

Edition actions appear as standard menu items under the *Edit* menu title. The edition actions' labels are: *Copy*, *Cut*, *Paste*. Help functions appear as standard menu items under the *Help* menu title.

Layout Parameters

Anytime the default layout is not suited to your particular needs, you can set several parameters to specify other graphical presentations. The available sizing and arrangement modes for the window layout are stated in the following paragraphs.

For information on how to define a window, refer to Part 1, "Part 2. The VAGTemplates Workbench" on page 7, "Chapter 3. Information Model Entities and their Editors" on page 51, "**Interface Unit**" on page 85, "**How to Define an Interface Unit**" on page 89.

Presentation:

VAGT: *Interface unit display* parameter, Interface Unit Generation Parameters editor, *GUI Window* panel.

normal

The window has a title bar, a system menu, minimize and maximize buttons, resize handles, and a menu bar. The Business Object layouts are displayed together.

notebook

The window has a title bar, a system menu, minimize and maximize buttons, resize handles, and a menu bar. It presents one or more pages identified with tabs. Each Business Object layout is presented on a different page.

You can choose between *notebook*, *Windows notebook*, and *PM notebook*.

VAST: If you want to add a label to the notebook tabs, you need to do it after the generation.

Title:

VAGT: *Title* parameter, Interface Unit Generation Parameters editor, *General* panel.

This parameter allows you to specify the title of the window, which appears in the title bar.

Sizing:

VAGT: *Maximize, Minimize, Resize* parameters, Interface Unit Generation Parameters editor, *GUI Window* panel.

These parameters allow you to provide the generated window with:

- the possibility to maximize the window layout;
- the possibility to minimize the window layout;
- the possibility to resize the window layout.

Business Object Arrangement:

VAGT: *Business Object layout* parameter, *top to bottom* and *left to right* values, Interface Unit Generation Parameters editor, *GUI Window* panel.

Business Objects layouts can be arranged:

- from top to bottom, then from left to right (vertical arrangement),
- from left to right, then from top to bottom (horizontal arrangement).

They are aligned on top and on the left whatever the arrangement is.

Business Object Sizing Modes: Business Object layouts' size and position are *fixed*, i.e. they do not vary with the size of the window layout.

Actions: VAGTemplates provides the generated window with navigation, standard edition and help actions.

These actions are laid out as menu items in the window's menu bar. Their default labels are: *Navigation, Windows, Edit, and Help*.

VAGT: The *Menu Titles* panel from the Interface Unit Generation Parameters editor enables you to modify the default labels assigned to the following:

- **Navigation** menu: *Navigation menu title* parameter
- **Windows** menu: *Windows menu title* parameter
- **Edit** menu: *Edit menu title* parameter

The *Edit Menu* panel from the Interface Unit Generation Parameters editor allows you to modify the edit actions default labels: *Cut label*, *Copy label*, *Paste label*.

For information on standard generated actions refer to “**Management of Persistent Data**” on page 147, “**Navigating Throughout a GUI Client application**” on page 157, “**GUI On-Line Help**” on page 160, and “**Edition Functions (GUI)**” on page 165.

Standard Layouts of TUI Applications

The VAGTemplates RAD generators provide a standard design for fields, Business Objects, maps and actions. They use the VAGTemplates default parameters to generate a default layout for each entity instance that you define. However, you can modify these parameters in the Workbench to customize the provided layouts.

This subchapter presents:

- the default presentations produced with the default VAGTemplates parameters, that is the generated layout if you do not modify any parameter in the Workbench;
- the parameters you can modify to customize these layouts.

Maps

VAGTemplates allows you to generate two types of maps: root maps and simple maps (*Type* parameter, Interface Unit Definition editor, *General* panel). VAGTemplates also generates error maps and help maps.

Root maps are often used as application main menus. They do not call any Business Object. They contain a list of child map calls and their access parameters (extract criteria and/or primary keys). The root map is the entry point of the application.

Simple maps can contain from 0 to n Business Objects with no layout constraints. They offer actions for manipulating Business Object data and actions for navigating from one map to the other. They also present system information such as the system type, the user id, etc.

Note: The default map size is 24 lines by 80 columns. This size is defined by the default device specified for the application (*Standard TUI device* parameter, Workspace definition editor).

Root Map Default Layout

By default, the root map is generated with:

- a header presenting system information,

- a body presenting the list of child maps and their access parameters,
- a trailer presenting the function keys and various messages.

```

SYST: DS2CICS          MYTUIAPPLICATION          FASTPATH: MENU
USER: PC user         Main menu          07-25-1997 11:45:36

S FASTPATH          ACCESS_PARAMETERS
A DETAIL Staff details  ID NUMBER
B LSTDET List and Detail (LOCATION)/ID NUMBER
C LIST Staff list      (LOCATION)

SELECT: _

FASTPATH: _____

PF01: HELP   PF02:          PF03: EXIT   PF04:          PF05:          PF06:
PF07:          PF08:          PF09:          PF10:          PF11:          PF12: CANCEL

```

The read-only field color is green; the field in error color is red; the input field color is yellow; the title color is white, and the function keys label is turquoise. The input fields are signaled with underscores.

Default Root Map Header: The header starts on the first line of the screen and comprises three lines (line 1 to 3 of 24):

- Line 1: This line displays the type of the system that runs the application, the name of the application, which is actually the name of your VAGTemplates Workspace, and the fastpath of the root map.
- Line 2: This line displays the user id, the title of the map, the current date and time.
- Line 3: This line is a blank separation line.

VAST: If you want to modify the presentation of the header, you must modify the corresponding generated Map part.

Default Root Map Body: The map's body starts on line 4 of the screen and ends on line 21. It displays the list of the child maps towards which the end user can navigate, indicating for each map:

- its selection code;
- its fastpath;

- its title;
- the access parameters that are available for the map, i.e. the primary key(s) of the Business Object(s) it calls, and the extract criteria that are defined for the Business Object(s) displayed between brackets;
- a blank separation line;
- the *Selection* field where the end user enters the selection code of the map he/she wants to open;
- a blank separation line;
- the *Fastpath* field where the end user enters the fastpath of the map he/she wants to open, and the field where he/she can enter the access parameters of the map;
- a blank separation line.

Note: If the number of lines to display the list of called maps is bigger than the available number of line in the map body, the list will be truncated and only the first lines that fit in the map will be displayed. In this case, you should define several menu maps.

Default Root Map Trailer: The trailer starts on line 22 of the screen and comprises three lines (line 22 to 24 of 24):

- Line 22: This line is a blank separation line.
- Lines 23-24: These lines display the action labels and function keys.

VAST: If you want to modify the presentation of the trailer, you must modify the corresponding generated Map part.

Simple Map Default Layout

By default, the simple map is generated with:

- a header presenting system information,
- a body presenting the Business Object(s) it calls and the available actions for the Business Object(s),
- a trailer presenting the function keys.

```

SYST: OS2CICS          MYAPPLICATION          FASTPATH: LSTDET
USER: PC user         List and Detail          05-29-1997  18:28:59

ACTIONS: I INSERT U UPDATE D DELETE
  Id No   Name      Dept No
    360 Arent        20
    80 James         20
    20 Pernal        20
    10 Sanders       20

ACTION: _
ID NUMBER      360
NAME           Arent      DEPTNUMB      20
DEPARTMENT NUMBER 20      LOCATION Washington
JOB TITLE      Sales
YEARS          8
SALARY         18357

FASTPATH: _____ WASHINGTON/360 _____

PF01: HELP    PF02:         PF03: EXIT    PF04: LOOKUP PF05:         PF06:
PF07:         PF08: NEXT    PF09:         PF10:         PF11:         PF12: CANCEL

```

The read-only field color is green; the field in error color is red; the input field color is yellow; the title color is white, and the function key label color is turquoise. The input fields are signaled with underscores.

Caution: A map cannot display **two** Business Object layouts of the same type. For example, two list layouts of the same Business Object cause the generation to be aborted.

Note: When a map displays an updatable list and a detail Business Object, the generated actions are available for both Business Objects. If the map displays a read-only list and a detail Business Object, as in the example above, the generated actions are only a available for the detail Business Object.

Default Simple Map Header: The header starts on the first line of the screen and comprises four lines (line 1 to 4 of 24):

- Line 1: This line displays the type of the system that runs the application, the name of the application, which is actually the name of your VAGTemplates Workspace, and the fastpath of the root map.
- Line 2: This line displays the user id, the title of the map, the current date and time.
- Line 3: This line is a blank separation line.
- Line 4: This line displays the available actions for the Business Object(s).

VAST: If you want to modify the presentation of the header, you must modify the corresponding generated Map part.

Default Simple Map Body: The map body starts on line 4 of the screen and ends on line 21. It displays the Business Objects called by the map, separated by a blank line. Business Objects are laid out from top to bottom then from left to right.

For updatable Business Objects, a line displays the *Action* field where the end user can enter the action code.

Line 20 displays the *Fastpath* field where the end user enters the fastpath of the map he/she wants to open, and the field where he/she can enter the access parameters of the map.

Note: If the width of a Business Object is too big to display it on a single map, it is spread on several maps. The end user can scroll horizontally the various maps using the F05 (*Left*) and F06 (*Right*) function keys. In this case, the logical key of the Business Object, the available actions and the function keys are repeated on each map.

If the number of lines of the Business Object is greater than the available number of lines in the map (24 lines minus header and trailer), the Business Object cannot be generated; an error occurs.

In a map displaying several Business Objects, if the Business Object has too many lines to be displayed entirely on the map, it will be displayed on a new map. The end user can reach it by using the F05 (*Left*) and F06 (*Right*) function keys.

When the whole map is displayed on several maps, the current map number is displayed at the end of the fastpath line.

Default Simple Map Trailer: The trailer starts on line 22 of the screen and comprises three lines (line 22 to 24 of 24):

Line 22: This line is a blank separation line.

Lines 23-24: These lines display the action labels and function keys.

VAST: If you want to modify the presentation of the trailer, you must modify the corresponding generated Map part.

Application Error Map Default Layout

By default, the application error map is a pop-up map displayed above the current map. It comprises 8 lines:

- 1 line displaying the map title: ERROR LIST
- 4 lines displaying the list of the messages;
- 1 line displaying the function keys available for the map.

The title color is white; the error message color is red; the warning message color is pink, the information message color is pink; the read-only field color is green; the function key label color is turquoise.

VAST: For information on error management, refer to “**Error Handling in TUI Applications**” on page 156.

Management of Messages

You can now raise messages that will not stop the execution of the current action. These messages will be displayed within the interface in an optional bar.

- Raising a message on the server:

The xxERROR-LST record has a flag, CONTROL-INFORMATION, which indicates whether a message has been raised or not. Also, applicative-information contains info-code and 3 info-variables. The code is similar to the one used in the errors.

Messages can be raised in the hooks of the server.

They are raised by the client and received by the server. A server message has priority.
- Raising a message on the client:

Help Map Default Layout

By default, the help map is a pop-up map displayed on top of the current map. It comprises 16 lines displaying:

- the help text;
- the function keys available for the map.

The help text color is white; the read-only field color is green; the function key label color is turquoise.

VAST: For information on on-line help, refer to “**TUI On-Line Help**” on page 163.

Help List Map Default Layout

By default, the help list map is a pop-up map displayed on top of the current map. It comprises 8 lines displaying:

- the title of the help list, for example ORG HELP LIST PANEL;
- the list of available values;
- the function keys available for the map.

The text color is white; the read-only field color is green; the function key label color is turquoise.

VAST: For information on help lists, refer to topic “**Actions Available for Help Lists**” on page 151, and “**Help Lists**” on page 220.

Layout Parameters

We describe in the paragraphs below the parameters that allow you to modify the default colors, the default title and fastpath of the map and the presentation of actions, help maps, help lists, message maps.

Colors: Several parameters allow you to modify the color of the texts displayed in the various maps:

Input fields

You can specify the default color of input fields. Such fields appear in root and simple maps and in help lists. The default color is *yellow*.

VAGT: *Normal color* parameter, Workspace Definition editor, *Client* panel.

Read-only fields

You can specify the color of read-only fields. Labels appearing in the header of the root and simple maps, function key codes appearing on all maps, fields’ labels appearing in the body of the simple maps and in the help lists are read-only. The default color is *green*.

VAGT: *Readonly color* parameter, Workspace Definition editor, *Client* panel.

Fields in error

You can specify the color of input fields when an error has been detected in them. The default color is *red*.

VAGT: *Error color* parameter, Workspace Definition editor, *Client* panel.

Function key labels

You can specify the color of the function key labels that appear on all maps. The default color is *turquoise*.

VAGT: *Function key label color* parameter, Workspace Definition editor, *Colors* panel.

Titles

You can specify the color of the map titles that appear on the simple maps, on the message maps, and on the root map in the header and in the list of called maps. The default color is *white*.

VAGT: *Title color* parameter, Workspace Definition editor, *Colors* panel.

Help text

You can specify the color of the help text that appear on the help maps and on the help list maps. The default color is *white*.

VAGT: *Help text color* parameter, Workspace Definition editor, *Colors* panel.

Error messages

You can specify the color of the error messages that appear on the message maps, or at the bottom of the root and simple maps. The default color is *red*.

VAGT: *Error message color* parameter, Workspace Definition editor, *Colors* panel.

Information messages

You can specify the color of the information messages that appear on the message maps, or at the bottom of the root and simple maps. The default color is *pink*.

VAGT: *Information message color* parameter, Workspace Definition editor, *Colors* panel.

Warning messages

You can specify the color of the warning messages that appear on the message maps. The default color is *pink*.

VAGT: *Warning message color* parameter, Workspace Definition editor, *Colors* panel.

TIP: In the standard generated TUI applications, there is no distinction between error messages, information messages and warning messages; they are all considered error messages. Therefore the *Information message color* and *Warning message color* are ignored. If you want to specify an error gravity, you can customize the generators so that they use the error-gravity Data-Item from the WERROR-LIST Record (see “*Error Data Records*” on page 315).

Note: Some colors cannot be displayed in TUI applications. If you specify one of these colors, the color applied in the generated application will be the nearest available color of the system where the application runs. For example, if you specify grey, the nearest color could be green on your system.

Title:

VAGT: *Title* parameter, Interface Unit Generation Parameters editor, *General* panel.

This parameter allows you to specify the title of the map, which appears in the header and in the main menu map.

Fastpath:

VAGT: *Fastpath* parameter, nterface Unit Generation Parameters editor, *General* panel.

This parameter allows you to specify the fastpath that will allow the end user to directly access the map by simply entering its fastpath in the *Faspath* field.

Size of Application Error Maps: You can modify the size of the application message maps by specifying the number of messages you want displayed on a page at a time. By default, 4 messages are displayed at a time.

VAGT: *Messages per page* parameter, Workspace Definition editor, *TUI* panel.

VAST: If you want to modify the size of the help maps, you must do it on the corresponding generated Map part.

Size of Map Headers: You can modify the size of the simple map header by displaying the action codes or not.

VAGT: *Display actions* parameter, Workspace Definition editor, *Function Keys* panel.

Therefore, the header will have three lines if you choose not to display the action codes (option checked).

```
#SYST: ^OS2CICS # ^MYAPPLICATION # #FASTPATH: ^LSTDET#  
#USER: ^PC USER # ^List and Detail # ^05-29-1997^18:28:59#
```

It will have four lines if you choose to display the action codes (parameter set to *true*).

```
#SYST: ^OS2CICS # ^MYAPPLICATION # #FASTPATH: ^LSTDET#  
#USER: ^PC USER # ^List and Detail # ^05-29-1997^18:28:59#
```

```
#ACTIONS: ^I^INSERT^U^UPDATE^D^DELETE^ ^ ^ #
```

Size of Map Trailers: You can modify the size of the root and simple map trailers by either displaying or not displaying the function keys and the error messages in the trailer.

VAGT: *Display function keys* parameter, and *Messages display* parameter, Workspace Definition editor, respectively *Function Keys* and *TUI* panels.

The trailer will have 0 lines if you choose not to display the function keys (*Function keys display policy* parameter set to *false*), and to display the error messages in a specific map (*Messages display* parameter set to *specific map*).

It will have 1 line if you choose to display the error messages at the bottom of the current map (*Messages display* parameter set to *current map*) but not the function keys (*Display function keys* parameter unchecked).

```
^(E)^INVALID ACTION                                     ^      #
```

It will have 3 lines if you choose to display the function keys (*Display function keys* checked) but not the error messages (*Messages display* parameter set to *specific map*).

```
#PF01:^HELP #PF02:^ #PF03:^EXIT #PF04:^LOOKUP#PF05:^ #PF06:^ #  
#PF07:^ #PF08:^NEXT #PF09:^ #PF10:^ #PF11:^ #PF12:^CANCEL#
```

It will have 4 lines if you choose to display the function keys (*Display function keys* parameter checked) and the error messages (*Messages display* parameter set to *current map*).

```
^(E)^INVALID COMMAND                                     ^      #  
#PF01:^HELP #PF02:^ #PF03:^EXIT #PF04:^LOOKUP#PF05:^ #PF06:^ #  
#PF07:^ #PF08:^NEXT #PF09:^ #PF10:^ #PF11:^ #PF12:^CANCEL#
```

Function Keys and Actions: VAGTemplates provides your applications with default navigation, data management and help actions that are activated using function keys. Several parameters allow you to modify the presentation of the function keys and their labels.

For information on standard generated actions refer to “**Management of Persistent Data**” on page 147, “**Navigating Throughout a TUI Application**” on page 158, and “**TUI On-Line Help**” on page 163.

Function key display

You can choose whether or not to display the function keys on the maps. Even if they are not displayed, the end user can still activate them. By default they are displayed.

VAGT: *Display function keys* parameter, Workspace Definition editor, *Function Keys* panel.

Action codes display

You can choose whether or not to display the action codes on the maps. Even if they are not displayed, the end user can still activate them. By default they are displayed.

VAGT: *Display actions* parameter, Workspace Definition editor, *Function Keys* panel.

Parameters in the Workbench enable you to modify the default labels assigned to the function keys, and the function key code associated with an action:

Cancel action

By default, the function key that exits the current map without saving the modifications and returns to the menu is the F12 key; its default label is *Cancel*.

VAGT: *Cancel* parameter, Workspace Definition editor, *Function Keys* panel.

Create action

By default, the function key that creates a new row in the database is the F13 key; its default label is *Create*.

VAGT: *Create* parameter, Workspace Definition editor, *Function Keys* panel.

Delete action

By default, the function key that deletes a row in the database is the F16 key; its default label is *Delete*.

VAGT: *Delete* parameter, Workspace Definition editor, *Function Keys* panel.

Exit action

By default, the function key that exits the current map and returns to the previous map is the F03 key; its default label is *Exit*.

VAGT: *Exit* parameter, Workspace Definition editor, *Function Keys* panel.

Help action

By default, the function key that opens the on-line help is the F01 key; its default label is *Help*.

VAGT: *Help* parameter, Workspace Definition editor, *Function Keys* panel.

Left map action

By default, the function key that opens the next map is the F05 key; its default label is *Left*.

VAGT: *Left page* parameter, Workspace Definition editor, *Function Keys* panel.

Look up in help list action

By default, the function key that opens the help lists is the F04 key; its default label is *Lookup*.

VAGT: *Lookup* parameter, Workspace Definition editor, *Function Keys* panel.

Next list page action

By default, the function key that displays the next data in a list is the F08 key; its default label is *Next*.

VAGT: *Next page* parameter, Workspace Definition editor, *Function Keys* panel.

Next message action

By default, the function key that displays the next error message at the bottom of the current map is the F11 key; its default label is *Msg +*.

VAGT: *Next message* parameter, Workspace Definition editor, *Function Keys* panel.

Previous list page action

By default, the function key that displays the previous data in a list is the F07 key; its default label is *Prev*.

VAGT: *Previous page* parameter, Workspace Definition editor, *Function Keys* panel.

Previous message action

By default, the function key that displays the previous error message at the bottom of the current map is the F10 key; its default label is *Msg -*.

VAGT: *Previous message* parameter, Workspace Definition editor, *Function Keys* panel.

Read action

By default, the function key that reads a row in the database is the F14 key; its default label is *Select*.

VAGT: *Read* parameter, Workspace Definition editor, *Function Keys* panel.

Refresh action

By default, the function key that displays the data as it was when last updated is the F09 key; its default label is *Undo*.

VAGT: *Refresh* parameter, Workspace Definition editor, *Function Keys* panel.

Right map action

By default, the function key that opens back the previous map is the F06 key; its default label is *Right*.

VAGT: *Right page* parameter, Workspace Definition editor, *Function Keys* panel.

Top list page action

By default, the function key that displays the first data in a list is the F17 key; its default label is *Top*.

VAGT: *Top* parameter, Workspace Definition editor, *Function Keys* panel.

Update action

By default, the function key that updates a row in the database is the F15 key; its default label is *Update*.

VAGT: *Update* parameter, Workspace Definition editor, *Function Keys* panel.

VAST: VAGTemplates provides 16 function keys associated with 16 actions (F01 and F03 to F17). By default, only the first 12 are displayed and can be used in the generated applications; the last five function keys that are associated with the elementary actions (create, read, update, delete) and the *Top* action are not used. The elementary actions are generated by default with particular action codes; the *Top* action is not used. If you want to use the function keys from 13 to 17, you can modify the default behavior in the generated application.

If you want to add function keys (from F18 to F24), you must add them in the function key Table part (see “*Action Code Table*” on page 311).

Note: If you prefer using the function keys from 13 to 24 instead of the function keys from 01 to 12, you can modify a method used by the generators, the **arePFKeysEquated:** method from the **MdIVGApplication** class (see the *VAGTemplates on Smalltalk Reference Guide*, modify the PFEQUATE VisualAge Generator attribute directly.

Help List, Help Map, Error Map Presentation

By default, Help List Maps, Help Maps, and Error Maps are presented as pop-up maps displayed within the current map.

A parameter in the Workbench allows you to present them on the full screen. In this case, the data map is no longer visible. Pressing the F03 key (*Exit*) closes the pop-up map. Selecting data in a help list map closes the pop-up map and transfers the selected value to the data map field for which input aid was requested.

VAGT: *Display popup* parameter, Workspace Definition editor, TUI panel.

Fields

Default Layout

By default, all the fields called by the Business Object are laid out in the final application. Read-only fields are green; updatable fields are yellow; fields that required a value are shown with underscores.

The size and the presentation of the fields is linked to the type of the Data Element it presents.

Layout Parameters

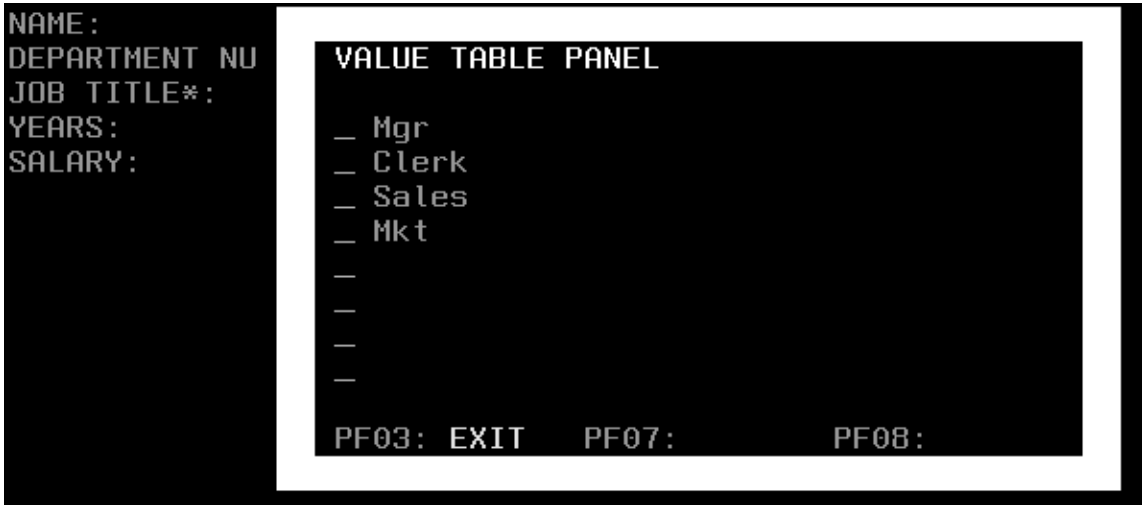
When defining your Data Elements, you can specify various parameters to modify the presentation of fields. The field offers a help list or not according to the type of value check you specified for the Data Element.

Fields Controlled by a Value Table:

VAGT: *Check type* parameter, *value table* value, Data Element Definition editor, *Check Type* panel.

The fields that are controlled by a value table provide the end user with a help list displaying the contents of the value table.

These fields are shown with an asterisk (*). When the end user requests input aid for the field, he/she presses the **Lookup** key (default key), and a pop-up map is displayed presenting the possible values for the field (default presentation). The end user can select a value in the help list.



For information on modifying function key labels and associated actions, refer to topic “*Function Keys and Actions*” on page 208.

For information on changing field colors, refer to topic “*Colors*” on page 205.

Presentation Label:

Note: A **presentation label** is a character string that can be displayed next to the graphical presentation of the Data Element. It helps the end user identify the nature of the displayed data.

Two types of presentation labels are available:

- **line label:** This label is displayed with the graphical presentation of a Data Element, when a single occurrence of it appears in a Business Object detail layout.

VAGT: *Default label* parameter, Data Element Generation Parameters editor, *Labels* panel.

- **column label:** This label is displayed with the graphical presentation of a Data Element, when several occurrences of it appear in a list Business Object. The column label can only be defined on one line.

VAGT: *Column label* parameter, Data Element Generation Parameters editor, *Labels* panel.

<i>Line label</i>	<i>Input field 1</i>	<i>Column label</i>
#DEPARTMENT:	^Washington#	#Name#

<i>Line label</i>	<i>Input field 1</i>	<i>Column label</i>	
		^Wallis	# <i>Input field 1</i>
		^Arent	# <i>Input field 2</i>

Field Size and Presentation

According to the type - alphanumeric, numeric, date, time, timestamp - you specify for your Data Element in the Workbench, end user input will be guided in some way, thanks to inherent input checks.

Alphanumeric Fields: An alphanumeric field accepts any character string.

Size

The size of the field is derived from the size that you specified when defining the Data Element it presents. The field does not accept additional characters when the maximum size is reached.

By default, the size of the field is one character.

VAGT: *Size* parameter, Data Element Definition editor, *General* panel.

Numeric Fields: A numeric field only accepts integer or decimal, signed or unsigned values.

Decimal separator

VisualAge Generator does not allow any customization of the decimal separator. The default separator will be the default system separator. If the end user enters a separator, it will be automatically reverted to the system separator.

VAGT: *Decimal separator* setting, Value Style Definition editor, *Value Style* panel.

Thousand separator

VisualAge Generator does not allow any customization of the thousands separator. However, if you specify the empty string, no separator will be generated for the numeric value, as VisualAge Generator will consider that no separator is defined. If you specify one, whatever it may be, the displayed separator will be the system default separator. If the end user enters a separator, it will be automatically reverted to the system separator.

VAGT: *Thousand separator* setting, Value Style Definition editor, *Value Style* panel.

Positive sign

VisualAge Generator does not allow any customization of the positive

(plus) sign. The default positive sign in VisualAge Generator is the blank character. If the end user enters a sign, it will be automatically reverted to the blank character.

VAGT: *Positive sign* setting, Value Style Definition editor, *Value Style* panel.

Negative sign

VisualAge Generator does not allow any customization of the negative (minus) sign. The default negative sign in VisualAge Generator is "-" (minus sign). If the end user enters a sign, it will be automatically reverted to the default sign.

VAGT: *Negative sign* setting, Value Style Definition editor, *Value Style* panel.

Sign position

You can specify whether or not a sign will be displayed, and where it will be positioned relative to the value. If the end user's input does not correspond to your specification, it is corrected according to your parameter. The default value is *no sign*.

VAGT: *Sign position* setting, Value Style Definition editor, *Value Style* panel.

Unit

VisualAge Generator does not allow any customization of the units symbol. The default unit in VisualAge Generator are the units of the system language in which the product is localized. If you specify no unit in the *Unit* field, no units will be displayed.

VAGT: *Unit* setting, Value Style Definition editor, *Value Style* panel.

Unit position

VisualAge Generator does not allow any customization of the units position. The default presentation in VisualAge Generator is the presentation implied by the system language in which the product is localized. If you specify a units position, VisualAge Generator will add units and assign it its default presentation. If you specify no units, no units will be displayed. The default value is *no units*.

VAGT: *Unit position* setting, Value Style Definition editor, *Value Style* panel.

Unit and sign alignment

VisualAge Generator does not allow any customization of the units position. The default presentation in VisualAge Generator is the presentation implied by the system language in which the product is localized.

VAGT: *Unit and sign alignment* setting, Value Style Definition editor, *Value Style* panel.

Size

The size of the field is derived from the capacity - number of digits before the decimal separator - and to the precision - number of digits after the decimal separator - that you specified when defining the Data Element it presents. The field does not accept additional characters when the maximum size is reached.

For example, if the capacity equals 5 and the precision 2, the integer part of the value cannot exceed 5 digits and the decimal part 2 digits.

By default, the capacity is 1 and the precision 0.

VAGT: *Capacity and Precision* setting, Data Element Definition editor, *General* panel.

Date Fields: The field only accepts date values.

Mask

The end user is required to type day, month and year values according to the order you specified - *dmy*, *mdy*, *ymd*. The default value is *mdy*.

<i>dmy</i>	<i>mdy</i>	<i>ymd</i>
22.11.96	11/22/96	96-11-22

VAGT: *Mask* setting, Value Style Definition editor, *Value Style* panel.

Separator

You can specify any character as a separator for days, months and years. If the end user omits to type the separator or types a wrong separator, it is automatically corrected according to your parameter.

VAGT: *Separator* setting, Value Style Definition editor, *Value Style* panel.

Year style

According to the style you specified - *short* or *full* - the date value will be displayed on 2 or 4 characters, the century characters being added by VisualAge Generator. The default value is *full*.

VAGT: *Year style* setting, Value Style Definition editor, *Value Style* panel.

Note: A default value is required for date fields.

Caution: Make sure the values you specify for these parameters are compatible with the date and time format parameters in the VisualAge Smalltalk Enterprise hpt.ini file.

Time Fields: VisualAge Generator does not allow any customization of the time presentation. The default presentation in VisualAge Generator is the presentation implied by the system language in which the product is localized.

Timestamp Fields: VisualAge Generator does not allow any customization of the time presentation. The default presentation in VisualAge Generator is the presentation implied by the system language in which the product is localized.

Caution: The timestamp type is not recognized in VisualAge Generator, which considers it alphanumeric. Therefore, no check is insured at end user's input.

For information on how errors in input fields are handled, refer to "Management of Persistent Data" on page 147.

To Display or Not To Display:

VAGT: *Display* parameter, Business Object Definition editor, *Field Attributes* panel.

By default, all the Data Elements *called* by a Business Object appear in the final application. However, the *Display* column allows you to choose the Business Object layouts for each Data Element:

- in the lists and the details (*always* value);
- neither in lists nor in details (*never* value);
- in details but not in lists (*on detail only* value).
- in lists but not in details (*on list only* value).

Detail Business Objects

Business Objects can appear as details or lists. In a detail, each individual field (except extract fields) can have a different layout. In a list, the layout affects all the fields presented in the list.

Default Layout

A Business Object appearing as a detail represents the Data Elements from the relational table(s) that it calls as independent graphical components.

The default Business Object detail layout generated with VAGTemplates comprises:

- a [label, value] pair, for each Data Element it calls,
- elementary actions (create, read, update, delete, check).

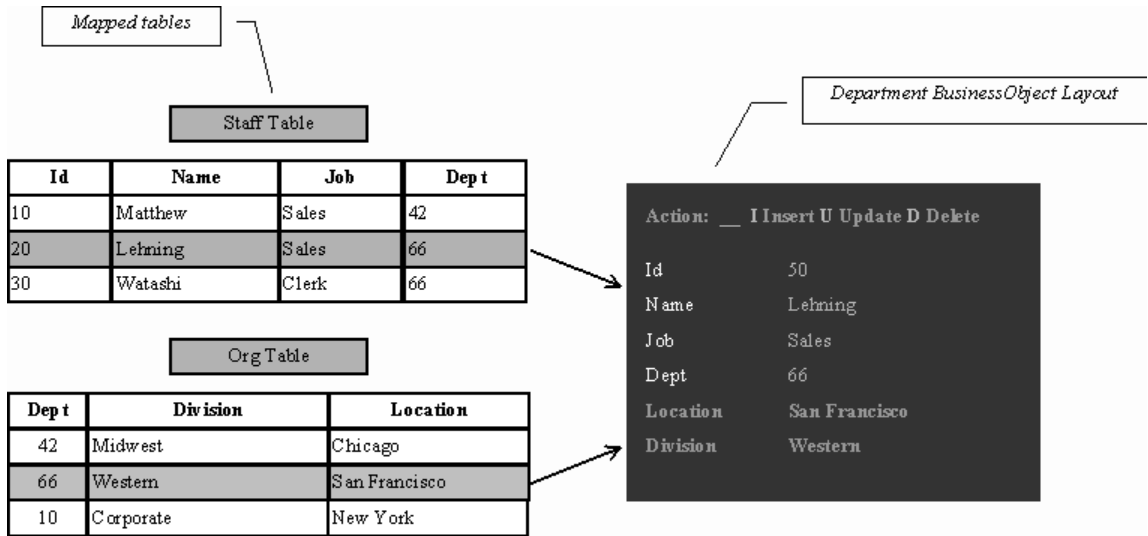


Figure 7. Default Business Object Detail Layout

Fields - labels and values - are arranged across the Business Object layout in the order in which they were specified in the Business Object.

Field labels and values are horizontally aligned, the label being on the left of the value, and arranged from top to bottom on 8 lines - except the key fields that are presented from left to right in a reserved line. Labels and values are left-aligned.

All the Table's fields *mapped* by a Business Object automatically appear in the final application, but you can choose not to provide some of them (see "*To Display or Not To Display*" on page 217).

Layout Parameters

Several parameters allow you to modify the presentation of the Business Object detail layout.

For information on how to specify Business Object parameters, refer to Part 1, "Part 2. The VAGTemplates Workbench" on page 7, "Chapter 3. Information Model Entities and their Editors" on page 51, "**Business Object**" on page 56, "**How to Define a Business Object**" on page 67.

Note: For the needs of this documentation we call a **line** several horizontally aligned [label+value] pairs, and a **column** one of these pairs.

Detail Sizing: The size of the detail is limited by the size of the map.

Arrangement:

VAGT: *Layout* parameter, *top to bottom* or *left to right* values, Business Object Generation Parameters editor, *Detail Field* panel.

The fields - labels+values - can be placed:

- from top to bottom, then from left to right (vertical arrangement),
- from left to right, then from top to bottom (horizontal arrangement).

Top to bottom presentation

#ACTION: ^_#
#ID NUMBER: ^_____#
#NAME: # ^_____#
#DEPT NUMBER: # ^_____#
#JOB TITLE*: # ^_____#
#YEARS: # ^_____#
#SALARY: # ^_____#
#DEPARTMENT NAME: ^_____#
#LOCATION: # ^_____#

You can choose the number of lines for the vertical arrangement (*Number of lines* parameter, Business Object editor), but you cannot choose the number of columns. The detail is spread on several maps if all the columns do not fit in a single map.

If the specified number of lines is too small compared with the number of Business Object fields to be displayed, the remaining fields are presented in another column.

#ACTION: ^_#
#ID NUMBER: ^_____#
#NAME: # ^_____# #DEPARTMENT NAME: ^_____#
#DEPT NUMBER: # ^_____# #LOCATION: # ^_____#
#JOB TITLE*: # ^_____#
#YEARS: # ^_____#
#SALARY: # ^_____#

Left to right presentation

```
#ACTION: ^_
#ID NUMBER: ^_____#
#NAME: ^_____#DEPT NUMBER: ^_____#JOB TITLE*: ^_____#YEARS: ^_____#
#SALARY: ^_____#DEPARTMENT NAME: ^_____#LOCATION: ^_____#
```

When not all the fields fit in a single map, the detail is spread over several maps.

You cannot choose the number of columns on which to display the fields since the layout generator optimizes the usable space to display as many fields as possible on a single map. You can see in the above example that the fields are not aligned but they are next to one another. The *Number of columns* parameter is ignored.

Help Lists

Help lists are provided for the mono-field foreign keys of a Business Object's primary table, i.e. the first table mapped by the Business Object's fields.

A **foreign key** is a field or a set of fields, used to identify or access particular rows in a table, whose values must correspond to one value in the primary key of the joined Table.



Help List Layout:

VAGT: *Help lists for all foreign keys parameter, Business Object Generation Parameters editor, Foreign Key Help List panel.*

You can choose whether or not to provide help lists. If the option is checked, help lists appearing as pop-up help list maps will be provided for the mono-field foreign keys of the Business Object. The help lists will be filled in with the values of the secondary tables' primary keys. Their available actions will also be displayed (see “**Actions Available for Help Lists**” on page 151, and “*Function Keys and Actions*” on page 208). If the option is unchecked, the foreign keys will appear like the other fields.

Filling Help Lists:

VAGT: *Help list prefilled* parameter, Business Object Generation Parameters editor, *Foreign Key Help List* panel.

This parameter allows you to specify whether the help list will be filled with data or empty at window opening. If the parameter is unchecked, the end user will have to trigger the *Top* action to fill the help list with data.

List Business Objects

A Business Object appearing as a list is used to represent the Business Object's fields as the columns of a unique table-type graphical component, called a **list**. The list can be read-only or updatable (*Layout type* parameter, Interface Unit Definition editor, *Business Objects* panel).

Default Layout

The default Business Object list layout generated with VAGTemplates comprises:

- a list, whose columns correspond to the Business Object Data Elements,
- elementary actions (if the list is updatable).

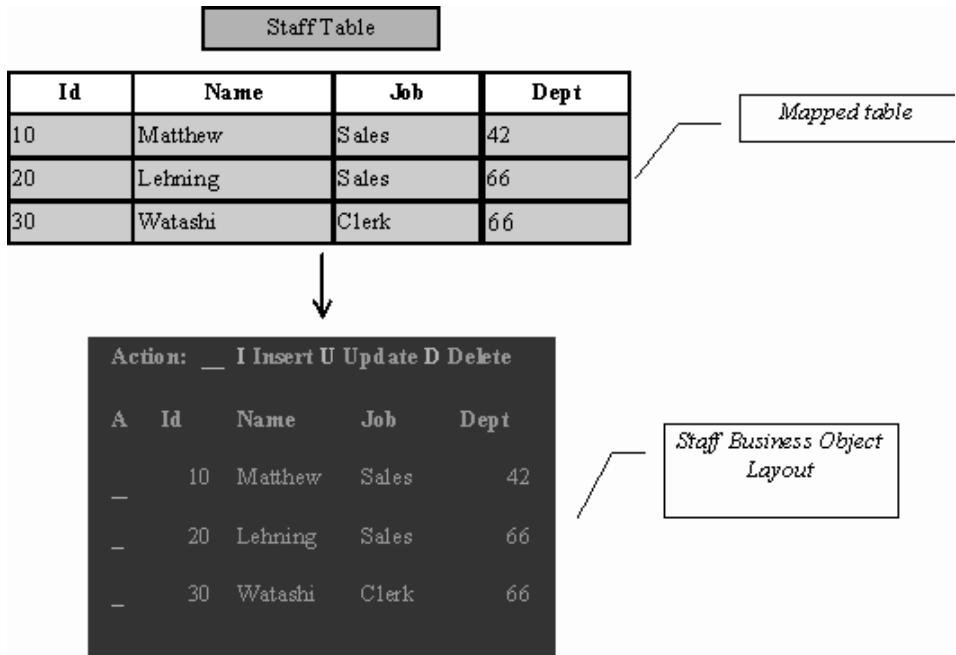


Figure 8. Default Business Object List Layout

By default, each column in the list comes with a heading and is adjusted to the size of its largest label or field. Column headings are aligned on the top and on the left. Numeric fields are right-aligned, alphanumeric fields are left-aligned.

By default, all the Table's fields *mapped* by a Business Object appear automatically in the final application, but you can choose not to include some of them (see *"To Display or Not To Display"* on page 217).

Layout Parameters

Several parameters allow you to modify the presentation of the Business Object list layout.

For information on how to specify Business Object parameters, refer to Part 1, "Part 2. The VAGTemplates Workbench" on page 7, "Chapter 3. Information Model Entities and their Editors" on page 51, "**Business Object**" on page 56, "**How to Define a Business Object**" on page 67.

Sizing:

Number of columns

You cannot specify the number of columns of a list since all called fields must be displayed. If there are too many columns for the map size, the

remaining columns are displayed on another map. The *Number of columns* parameter in the Business Object Generation Parameters (*List Container* panel) editor is ignored.

The sum of the size of the action column (in updatable lists) and that of the key field column, or the size of a value column must not be greater than the length of the map lines, otherwise the generation is aborted.

Number of lines

You can specify the number of lines you want to view in the list. The number of lines does not take the labels' line into account; it only applies to value fields.

If the number of lines is greater than the available lines in the map the generation is aborted (see "*Default Simple Map Body*" on page 203)

VAGT: *Number of lines* parameter in the Business Object Generation Parameters (*List Container* panel).

Filling the List:

VAGT: *List prefilled* parameter, Business Object Generation Parameters editor, *List View* panel.

This parameter allows you to specify whether or not the list will be filled with data at the map initialization.

If the parameter is set to *false*, the end user will have to trigger the *Top* action to fill the list with data.

Chapter 6. Application Generation and Enhancement

Standard Generation

Once you have described your application by completing the entity instance specifications with VAGTemplates, you have to build the executable application components by generating the instances.

When you generate an instance, you can either generate the selected instance only (*Instance only* option in VAGTemplates on Java, *instance generation* option in VAGTemplates on Smalltalk), or the selected instance plus all the instances called by the selected instance (respectively *With associates* and *cascaded generation* options). You can also cascade the generation and generate the components that are used by all the application's components, called predefined parts/beans (respectively *With associates and predefined beans* and *cascaded generation with predefined parts* options).

The generation produces operational VisualAge for Java or VisualAge Smalltalk Enterprise components and VisualAge Generator parts (for information, refer to “**Generated Architecture and Principles**” on page 240). These components are stored in the VisualAge Library.

You can generate instances of the following entities: *Data Element*, *Business Object*, *Interface Unit*, *Relational Table* and *Workspace*.

Note: The Value Style instances cannot be generated directly. Their instances are generated by the Business Object layout generator as they affect the *presentation* of the Data Elements within the Business Object's layout.

List of Available Generators

VAGTemplates on Smalltalk 3.1 Generators

VAGTemplates 4.0 provides you with a set of generators corresponding to VAGTemplates on Smalltalk 3.1 generators. They are available with the current version of VAGTemplates but will become obsolete in VAGTemplates 4.1. Consequently, users who customized generators with VAGTemplates 3.1 must migrate their customized generators to the 4.0 version of VAGTemplates.

The concerned generators are the following:

- 4GL GUI generators
- Smalltalk GUI generators
- TUI generators

Note: The *Object Oriented* generators generate VisualAge Smalltalk Enterprise parts and the additional required VisualAge Generator parts. The *4GL Oriented* generators generate only VisualAge Generator parts; they are maintained to ensure retrieval of legacy applications developed with VAGTemplates V2.2.

Caution: New VAGTemplates on Smalltalk developers must NOT use these generators.

VAGTemplates on Smalltalk 4.0 Generators

VAGTemplates on Smalltalk provides you with four sets of generators:

- Smalltalk GUI generators to generate the whole GUI client application including **Business Object** and **Interface Unit** layouts.
- Smalltalk TUI generators to generate the whole TUI application including **Business Objects** and **Interface Unit** layouts (organized into maps and map groups).
- Smalltalk Web generators to generate the whole Web-based application.
- Smalltalk Help GUI generators to generate online help for GUI client applications. Help is generated into RTF and HPJ files for Windows, and IPF files for OS/2.

Note: When installing VAGTemplates on Smalltalk, these generators are installed by default. To use the 3.1 generators, you must load another version of the generators applications in your VisualAge image.

VAGTemplates on Java Generators

VAGTemplates on Java provides you with three sets of generators:

- GUI generators to generate the whole GUI client application including **Business Object** and **Interface Unit** layouts.
- TUI generators to generate the whole TUI application including **Business Objects** and **Interface Unit** layouts (organized into maps and map groups).
- Web generators to generate the whole Web-based application.

Instance Only / Instance Generation Option

Note: The type of generation described below corresponds to the *Instance only* option in VAGTemplates on Java and *instance generation* option in VAGTemplates on Smalltalk.

This option allows you to generate only the selected instance. You can use it to generate *Data Element*, *Business Object*, *Interface Unit*, *Relational Table* and *Workspace* instances.

TIP: You can use this option when you only need to re-generate a few instances that you have modified.

For example, if you have already generated your application but have modified the extract criteria for the Business Object, use Instance only/instance generation option to re-generate only the Business Object.

Note: Some components are not re-generated to preserve enhancements you may have made on the generated application. For information, refer to “**Enhancements and Re-generation**” on page 233.

Business Object Generation

The description of a Business Object has two aspects, the logical description (its fields, the tables it maps to, the actions available on persistent data, etc.) and the graphical description (its layouts).

VAGTemplates on Java: In the *Generate (instance)* window, three check boxes allow you to generate the visual, client and server part of of the selected instance. You must choose at least one option.

VAGTemplates on Smalltalk: In GUI client applications, there are two Business Object generators: the *GUI Business Object Logic* generator generates the logical description; the *GUI Business Object Layout* generator generates the graphical description.

A quick way to view your graphical layout is to generate your Business Object instance with the Business Object Layout generator.

Layouts are customizable components that are not overridden by another generation. If you modify parameters that apply to a generated layout you must delete the generated component before starting the re-generation, otherwise the component will not be re-generated or select the override existing components option when you select a generator. For information, refer to “**Enhancements and Re-generation**” on page 233.

The *TUI Business Object* generator generates the logical description of the Business Object. The graphical description is taken care of by the Interface Unit layout generator.

Data Element Generation

VAGTemplates on Java: In the *Generate (instance)* window, three check boxes allow you to generate the visual, client and server part of of the selected instance. You must choose at least one option.

VAGTemplates on Smalltalk: Use the *GUI Data Element* generator to generate a Data Element instance for a GUI client application, and the *TUI Data Element* generator to generate a Data Element instance for a TUI application. Both generators generate the components that correspond to the logical description of the Data Element instance you made in the Workbench.

Interface Unit Generation

VAGTemplates on Java: In the *Generate (instance)* window, three check boxes allow you to generate the visual, client and server part of of the selected instance. You must choose at least one option.

VAGTemplates on Smalltalk:

Use the *GUI Interface Unit* generator to generate the Interface Unit layout, the navigation and edition actions, and the Business Object calls. The layout of the Business Object and the generation of data management actions are taken care of by the *GUI Business Object Logic* generator.

In TUI applications, the Interface Unit has two aspects, a logical aspect and a graphical aspect. There are two Interface Unit generators: the *TUI Interface Unit Logic* generator, which generates the logical description of the Interface Unit, and the *TUI Interface Unit Layout* generator, which generates the graphical description of the Interface Unit and of the Business Objects its calls.

Layouts are customizable components that are not overridden by another generation. If you modify parameters that apply to a generated

layout you must delete the generated component before starting the re-generation, otherwise the component will not be re-generated, or select the override existing components option when you select a generator. For information, refer to “**Enhancements and Re-generation**” on page 233.

Caution: The *TUI Interface Unit Logic* generator generates the actions available for managing persistent data. If a Business Object is added or removed from an Interface Unit, you will have to re-generate it.

Relational Table Generation

If you specify a 3-tier architecture for your application, the Relational Table generator generates atomic servers.

Note: You should only generate a Relational Table when it has a primary key.

For information on help lists, refer to “**Actions Available for Help Lists**” on page 151.

For information on 3-tier architecture, refer to “*Three-tier Architecture*” on page 290 , and “*Three-tier Architecture*” on page 304.

VAGTemplates on Java: In the *Generate (instance)* window, three check boxes allow you to generate the visual, client and server part of of the selected instance. You must choose at least one option.

VAGTemplates on Smalltalk: Use the *GUI Relational Table Generator* to generate a Relational Table instance for a GUI client application, and the *TUI Relational Table Generator* to generate a Relational Table instance for a TUI application. Both generators generate all the components that provides help lits services on foreign keys.

Workspace Generation

Use the GUI Workspace generator to generate a Workspace instance for a GUI client application, and the TUI Workspace generator to generate a Workspace

instance for a TUI application. Both generators generate the predefined parts, i.e. all the parts that provide error handling services and the communication area used by the application.

In addition, the *GUI Workspace* generator generates the *Windows* menu that allows navigation throughout the open windows in the GUI client application.

TUI: If an Interface Unit is added or removed from a Workspace, you will have to re-generate it.

For information on predefined beans/parts, refer to “**Components Generated From a Workspace: Predefined Beans/Parts**” on page 312.

For information on error handling in GUI client applications, refer to “**Error Handling in GUI Client applications**” on page 153, and on error handling in TUI applications, refer to “**Error Handling in TUI Applications**” on page 156.

VAGTemplates on Smalltalk: Help Generation

On-line help is always generated from an Interface Unit. An instance generation produces the complete help for the Interface Unit, the called Interface Units, Business Objects and Data Elements. The generation must be activated from the root Interface Unit.

GUI: Use the *GUI Interface Unit Help* generator to generate on-line help for the GUI client applications. The *GUI Interface Unit Help* generator generates an IPF file (OS/2 help format) and an RTF and an HPJ file (Windows help format) that contains the help text you entered in the *On-line help description* field when defining your instances. These files are located by default in the BUILD subdirectory of the VAGTemplates root directory. To make this help operational you must compile these files and copy the resulting file in the HELP subdirectory of the VisualAge for Java or VisualAge Smalltalk Enterprise root directory.

TUI: Use the *TUI Interface Unit Help* generator to generate on-line help for TUI applications. The *TUI Interface Unit Help* generator generates a VisualAge Generator Table part that contains the help text you entered in the *On-line help description* field when defining your instances.

For information on the generated on-line help, refer to “**On-Line Help**” on page 160.

An example of help generation is in “**Generating On-Line Help (VAGTemplates on Smalltalk Example)**” on page 128.

With associates/Cascaded Generation Option

Note: The type of generation described below corresponds to the *With associates* option in VAGTemplates on Java and *cascaded generation* option in VAGTemplates on Smalltalk.

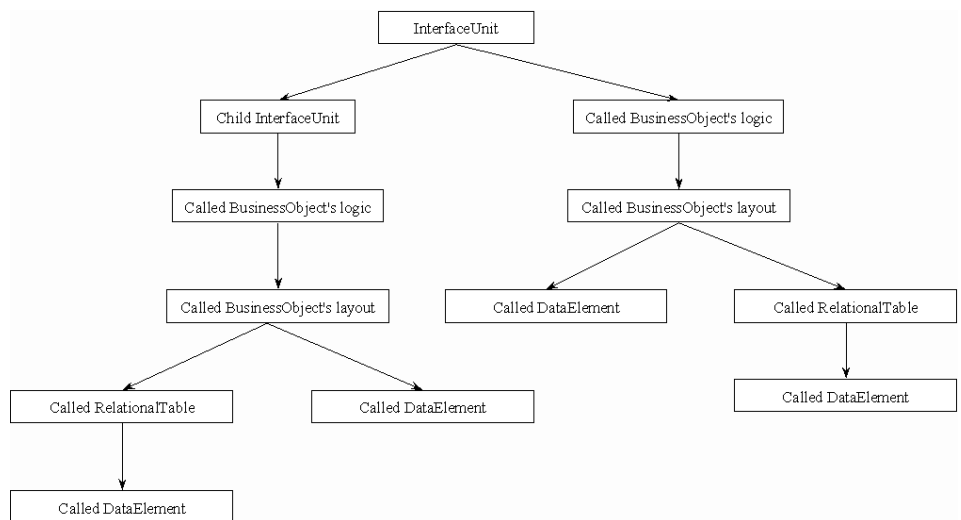
This type of generation allows you to generate the selected instance and all the instances it calls, in succession right down to the lowest level. The selected instance will be generated, as will the other instances it calls, and so on. You can use the this option to generate Business Object and Interface Unit instances.

TIP: You can use it anytime you want to generate all the instances of an application at once. Therefore, you do not need to worry about what instances to generate.

For example, if you have defined an Interface Unit calling a Business Object that calls Data Elements for which Value Styles are defined, use With associates / cascaded generation option from the Interface Unit and all the above instances will be generated.

Note: The option is not available for the Workspace generators since an instance included in a Workspace is not considered a called instance.

The generation principle is the following:



The following instances are generated sequentially:

- the selected Interface Unit instance;

- the Interface Unit instances it calls;
- the logical descriptions of the Business Object instances it calls;
- the graphical descriptions of the Business Object instances it calls;
- the Relational Tables;
- the Data Element instances used by the Business Objects;
- the Data Elements instances used by the Relational Tables.

Note: If one instance is called several times by different instances, it is generated only once.

With Associates and Predefined Beans / Cascaded Generation With Predefined Parts Option

Note: The type of generation described below corresponds to the *With associates and predefined beans* option in VAGTemplates on Java and *cascaded generation with predefined parts* option in VAGTemplates on Smalltalk.

The predefined beans/parts are shared by all the other generated parts. You need to generate them only once for the same Workspace. You do not need to re-generate them except if you modify the Workspace definition in the Workbench.

For information on the generated predefined beans/parts and other parts, refer to “**Generated Architecture and Principles**” on page 240.

The use of this option from an Interface Unit generates all the components the application needs to be operational in one generation phase. It equals the *Instance only/instance generation* of a Workspace plus the *With associates/cascaded generation* of an Interface Unit.

Note: This option is not available for the Workspace generators since the *Instance Only/instance generation* of a Workspace generates the predefined parts.

TIP: Use this option only the first time you generate all the instances of an application. Then, anytime you want to re-generate these instances, use instance generation or cascaded generation.

Application Storage

All VAGTemplates components are stored in the VisualAge Library:

- the VAGTemplates product
- the instance specifications you make with the Workbench
- the generated components.

Specification Storage

By default, the instance specifications imported into VAGTemplates from an existing database or the instance specifications you entered via the Workbench are stored in the Java package or Smalltalk application you specified when creating your Workspace.

Anytime you create an instance, you are asked to specify a storage package/application.

Generated Components Storage

By default, the generated components are stored in the MyVAGTEntitiesApp package/application and the generated predefined parts in the MyVAGTWorkspaceEntitiesApp package/application. The choice of these packages/applications is set by parameters in the *Target Package/Target Application* panel of each entity's Generation Parameters editor. You can modify these parameters.

For more information on instance parameters, refer to Part 1, "Part 2. The VAGTemplates Workbench" on page 7, "Chapter 3. Information Model Entities and their Editors" on page 51.

VAGTemplates on Smalltalk: Generated Help Files Storage

By default, generated help files are stored in the BUILD subdirectory of VAGTemplates's root directory. The choice of this directory is set by the *Generation* parameter of the Workspace instance. You can modify this parameter using the *Definition* choice from the *Workspace* menu.

To be able to use the generated help files you still have to compile the files and copy the compiled files to the Help subdirectory of the VisualAge for Java or VisualAge Smalltalk Enterprise root directory.

For more information on the Workspace settings, refer to Part 1, "Part 2. The VAGTemplates Workbench" on page 7, "Chapter 2. The Workbench" on page 9, "The VAGTemplates Workbench" on page 12, "Workspace Menu" on page 15.

Enhancements and Re-generation

Once your application is generated, you may wish to enhance some generated components. VAGTemplates ensures persistency of your applications by making a difference between components that must be described and maintained at the Information Model level, using the Workbench, and components that can be enhanced and maintained in the target tool. This difference can be seen in the traceability information generated with each component.

In this subchapter we explain the use of the traceability information, which governs the re-generation process and allows you to know in which environment you can make modifications to your application and preserve them.

Note: Before modifying generated components, make sure that VAGTemplates cannot generate the behavior or the presentation you want (see “Chapter 5. Standard Functions and Layouts of Generated Applications” on page 145).

Traceability Information

Traceability Categories

VAGTemplates introduces four traceability categories for the components generated from Information Model specifications:

RAD

- These components are developed at the Information Model level, with the VAGTemplates Workbench. They can be enhanced and maintained at the target tool level, with the VisualAge workstation. They are generated once with the first generation in order to initialize the component and allow its enhancement.

For example, generated layouts are RAD components.

HOOK

- These components are insertion points generated to let you add specific code into the generated applications. Their existence is defined and maintained at the Information Model level. Their implementation is developed and maintained at the target tool level.

For example, you can specify an additional check for Business Object fields in a Hook component: the `additionalCheck` method.

API

- These components are developed and maintained at the Information Model level, with the VAGTemplates Workbench. These components can be used by RAD and HOOK components.

For example, the methods for accessing the Business Object's data are API components.

INTERNAL

- These components are developed and maintained at the Information Model level. They implement services provided by the API components. They are likely to be modified throughout the life of the VAGTemplates product. They must not be used by RAD parts.

The traceability category is used by the generators to know whether or not they must re-generate a part.

Generated Part Documentation

The generators include a detailed documentation in the generated components' code.

This documentation indicates:

- the specification category of the generated component (RAD, HOOK, API, INTERNAL),
- the source Information Model entity form which the component has been generated,
- the version of the source entity
- the service provided by the generated component.

This complete traceability information is generated for you to trace the part's origin, its status in the final application, and to know whether the part must be modified in the target tool or with the VAGTemplates Workbench.

VAGTemplates on Java: Traceability in Comments

Traceability information is displayed in the generated components' comments.

In the headers of the classes and interfaces, the traceability category and a specific comment are generated.

For the methods, the traceability category and the method signature are generated.

For classes and interfaces, three methods are generated:

- `public static void mdlClassHeader ()`
- `public static void mdlVisualComposition ()`
- `public static void mdlPublicInterface ()`

The code of these methods consists in a comment that shows:

- the generated fields and their traceability category
- the generated beans and connections and their their traceability category
- the generated properties, methods and events and their traceability category

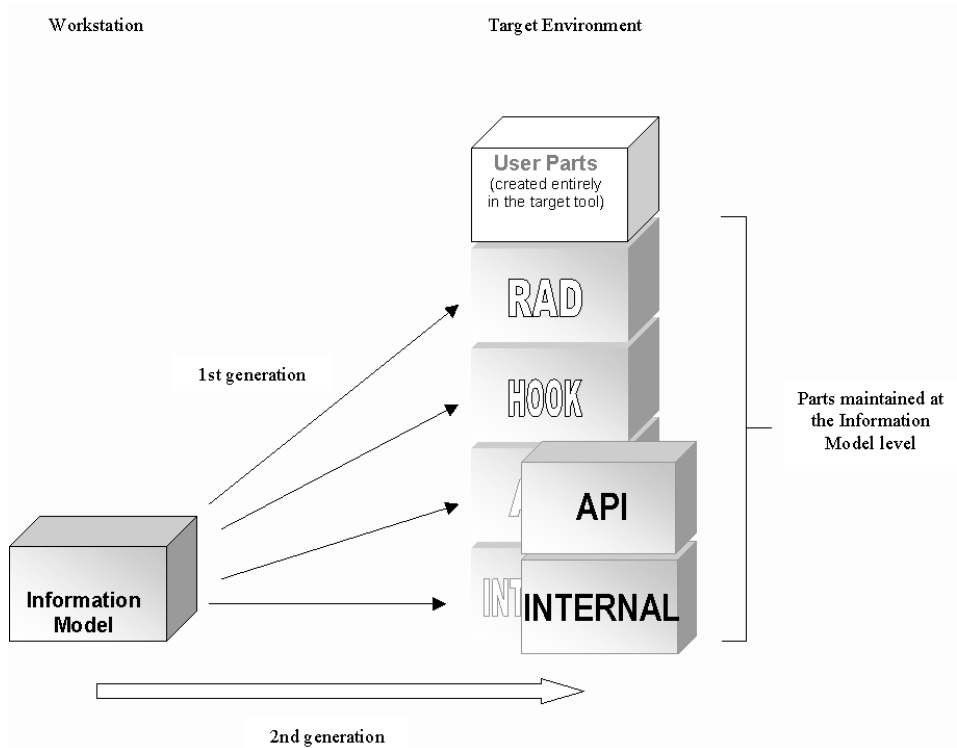
How the Generators Use the Traceability Information

When you create an application with VAGTemplates, you make your specifications using the Workbench, then you generate the specified instances. When you generate, the generators add traceability information into the code of each generated component.

The first time you generate, the generators will create the required components with their traceability information in your VisualAge image.

If you modify the specifications of an instance in the VAGTemplates Workbench and re-generate the instance, the generators will compare the new specifications to generate with the component in the image generated from the old specifications to generate or not the component:

- If the component has is not found in the image, it is generated.
- If the source instance exist and the part has been generated previously, the generators look at the traceability category:
 - if the category is RAD or HOOK, the component is not re-generated; if you had modified the component between the two generations it is not overridden, and your modifications are preserved.
 - if the category is API or INTERNAL, the component is re-generated and overrides the previously generated component.



Therefore, if you have to modify functional aspects of your application, make the modifications on the Information Model entities using the VAGTemplates Workbench and re-generate the application, otherwise your modifications will be overridden. If you want to modify graphical aspects of the application, you can do it in the target environment.

For information on RAD, HOOK, API, and INTERNAL generated components, refer to “**Generated Architecture and Principles**” on page 240.

What Generator Do You Use When Re-generating

This section provides the general rules about the choice of generators and generation option.

For information on generation options, refer to “**Standard Generation**” on page 225.

VAGTemplates on Java

GUI Re-generation: Anytime you modify presentation parameters at the Data Element level, re-generate the Business Object that calls the Data Element checking the *With associates*, *Visuals*, *Client* and *Server* options.

Anytime you modify functional parameters at the Data Element level, like format or checks, re-generate the Business Object that calls the Data Element checking the *With associates*, *Visuals*, *Client* and *Server* options.

Anytime you modify presentation parameters at the Business Object level, re-generate the Business Object checking the *Instance only*, and *Visuals* options.

Anytime you modify functional parameters at the Business Object level, re-generate the Business Object checking the *With associates*, *Client* and *Server* options.

Anytime you modify presentation parameters at the Interface Unit level, re-generate the Interface Unit checking the *Instance only*, *Visuals*, and *Client* options.

Anytime you modify functional parameters at the Interface Unit level, like the calls to other Interface Units, re-generate the Interface Unit checking the *Instance only*, *Visuals*, and *Client* options.

Anytime you modify Workspace parameters that apply to the whole application, re-generate the Interface Unit checking the *With associates and predefined beans*, *Visuals*, *Client* and *Server* options.

Anytime you modify parameters that only apply to the Workspace, re-generate the Workspace checking the *Visuals*, *Client* and *Server* options.

Note: For a Workspace generation, the *Instance only* option is automatically checked.

TUI Re-generation: Anytime you modify parameters at the Data Element level, like format or check, re-generate the Business Object that calls the Data Element checking the *Instance only* and *Client* options.

Anytime you modify presentation parameters at the Business Object level, re-generate the Interface Unit that calls the Business Object checking the *Instance only* and *Visuals* options.

Anytime you modify functional parameters at the Business Object level, like the mapping, re-generate the Interface Unit that calls the Business Object checking the *With associates*, *Client* and *Server* options.

Anytime you modify presentation parameters at the Interface Unit level, re-generate the Interface Unit checking the *Instance only*, and *Visuals* options.

Anytime you modify functional parameters at the Interface Unit level, like the calls to other Interface Units, re-generate the Interface Unit checking the *Instance only*, *Visuals*, *Client* and *Server* options.

Anytime you modify Workspace parameters that apply to the whole application, re-generate the Interface Unit checking the *With associates and predefined beans*, *Visuals*, *Client* and *Server* options.

Anytime you modify parameters that only apply to the Workspace, re-generate the Workspace checking the *Visuals*, *Client* and *Server* options.

Note: For a Workspace generation, the *Instance only* option is automatically checked.

VAGTemplates on Smalltalk

GUI Re-generation: Anytime you modify presentation parameters at the Data Element level, re-generate the Business Object that calls the Data Element with the *GUI Business Object Layout* generator and use *cascaded generation*.

Anytime you modify functional parameters at the Data Element level, like format or checks, re-generate the Business Object that calls the Data Element with the *GUI Business Object Layout* generator and use *cascaded generation*.

Anytime you modify presentation parameters at the Business Object level, re-generate the Business Object with the *GUI Business Object Layout* generator and use *instance generation*.

Anytime you modify functional parameters at the Business Object level, re-generate the Business Object with the *GUI Business Object Logic* generator and use *cascaded generation*.

Anytime you modify presentation parameters at the Interface Unit level, re-generate the Interface Unit with the *GUI Interface Unit* generator and use *instance generation*.

Anytime you modify functional parameters at the Interface Unit level, like the calls to other Interface Units, re-generate the Interface Unit with the *GUI Interface Unit* generator and use *instance generation*.

Anytime you modify Workspace parameters that apply to the whole application, re-generate your application with the *GUI Interface Unit* generator using *cascaded generation with predefined parts*.

Anytime you modify parameters that only apply to the Workspace, re-generate the Workspace with the *GUI Workspace* generator using *instance generation*.

TUI Re-generation: Anytime you modify parameters at the Data Element level, like format or check, re-generate the Business Object that calls the Data Element with the *TUI Business Object* generator and use *cascaded generation*.

Anytime you modify presentation parameters at the Business Object level, re-generate the Interface Unit that calls the Business Object with the *TUI Interface Unit Layout* generator and use *instance generation*.

Anytime you modify functional parameters at the Business Object level, like the mapping, re-generate the Interface Unit that calls the Business Object with the *TUI Interface Unit Logic* generator and use *cascaded generation*.

Anytime you modify presentation parameters at the Interface Unit level, re-generate the Interface Unit with the *TUI Interface Unit Layout* generator and use *instance generation*.

Anytime you modify functional parameters at the Interface Unit level, like the calls to other Interface Units, re-generate the Interface Unit with the *TUI Interface Unit Logic* generator and use *instance generation*.

Anytime you modify Workspace parameters that apply to the whole application, re-generate your application with the *TUI Interface Unit Logic* generator using *cascaded generation with predefined parts*.

Anytime you modify parameters that only apply to the Workspace, re-generate the Workspace with the *TUI Workspace* generator using the *instance generation*.

Generated Architecture and Principles

Introduction

Applications generated with VAGTemplates are structured like nests of dolls. The most complex components are embedded in complex components, which are embedded in less complex components and so on. When you want to customize your application, you only intervene on the simpler components, that is the graphical components in a GUI client application or the lowest level components in an application tree of a TUI application.

These components will be described more precisely in the following sections.

VAGTemplates provides two types of generators to generate these components:

- **RAD generators:** They are dedicated to generating graphical components, that is maps and map groups (TUI) and layouts and their connections (GUI):
 - *TUI Maps and MapGroups:* These components present the fields, the Business Objects, and the actions. However, the graphical aspect is closely linked with the logical aspect. For example, if you want to remove a field from a map, you must take care to remove also the corresponding logic (corresponding Data-Item, use of the Data-Item, etc.).
 - *GUI Detail Subview bean/parts:* It contains the graphical parts and actions that correspond to the Business Object's specifications, Business Object's attributes that represent the Business Object's fields, and a variable based on the Business Object to allow the Business Object to be re-used. Each graphical parts is connected to the attributes, and each elementary action to the Business Object variable.
 - *GUI List Subview bean/parts:* It contains a graphical table and graphical actions, attributes representing the Business Object's fields, and a variable based on the Business Object for filling the table. The graphical table is connected to the attributes and the actions to the variable.
 - *GUI View Parts:* It contains graphical parts - Business Object layouts, graphical actions - and organic parts. The RAD generators only generate the graphical parts.
- **Non-RAD generators:** They are dedicated to generating functional and organic components responsible for accessing data on the client or the server, managing error checks, etc.:
 - *Programs* made up of sets of related part definitions that can be generated into an executable form;

- *Data-Items* describing the characteristics of a single field in a record or a table;
- *Functions* containing a set of processing statements, for performing I/O operations, constituting the procedural logic of the application and used only in server applications;
- *Records*, which are data structures comprising collections of Data-Items;
- *Tables*, which are collections of related Data-Items that can be used to edit data, store messages and information.
- *Non-visual parts*,
- *Java/Smalltalk classes*.

Generated Components Naming Policy

The naming of the generated components follows rules so that it is easy to determine which part has been generated from which Information Model entity instance.

The VisualAge components names have no length constraints.

The naming of the generated VisualAge Generator parts must meet VisualAge Generator length requirements:

- Programs and Tables are allowed 7 characters,
- Maps are allowed 8 characters,
- Map groups are allowed 6 characters,
- Tables are allowed 7 characters,

Their names are called *short names*.

- Functions and Records are allowed 18 characters,
- Data-Items are allowed 32 characters.

Their names are called *long names*.

VAGTemplates on Smalltalk: In VisualAge Generator, Map names are always the same: HEADER, TRAILER, MAPx, etc., and they are defined in the MdivGTUIUsages poolDictionary (refer to the *VAGTemplates Reference on Smalltalk Guide*. The VAGTemplates' naming policy does not apply to these parts. However, the naming policy applies to Map groups.

To build these components' names, VAGTemplates combines the target name of the instance with mnemonics. The target name is the instance name

truncated to five characters for all entities except the Data Element whose target name is by default the instance name truncated to 10 characters.

VAGT: Each entity (except Value Style) has a target name parameter.

The mnemonics are divided into the following categories:

- *Information Model entity mnemonics:* this information is used to determine from what VAGTemplates entity the resulting component was generated;
- *Component mnemonics:* this information is used to determine the nature of the resulting component itself;
- *usage mnemonics:* this information is used to complete the previous two mnemonics when necessary and to differentiate various uses for the same type of component.

VAGT: Parameters in the Workbench allow you to modify the entity mnemonics and part mnemonics: Workspace Definition editor, *Naming Rules* panel.

There are two possible concatenations of the previous mnemonics to build the generated components' names:

- the name begins with the target name of the instance: *Naming policy* parameter, set to *identifier first*, Workspace Definition editor, *Naming Rules* panel.
 - the name begins with the part Mnemonic first: *Naming policy* parameter, set to *type first*, Workspace Definition editor, *Naming Rules* panel..
1. the *naming policy* is *identifier first*, the name structure is the following:
<Instance Target Name><Entity Mnemonic><Usage Constant>
*For example, when a Program part is generated from the SAMPLE Business Object used as a detail, its name is **SAMPLAU**:*
 - *SAMPL is the Business Object's target name*
 - *A the Program mnemonic*
 - *U the mnemonic for a detail Business Object*
 2. the *naming policy* is *type first*, the name structure is the following:
<Part Mnemonic><Usage Mnemonic><Instance Target Name>
*The name of the Program part previously mentioned is **AUSAMPL**.*

Long Name Structures

The names of a Data-Item, Function and Record parts are composed of four elements:

1. the *naming policy* is *identifier first*, the rule is:
<Instance Target Name><Part Mnemonic><Entity Mnemonic><Usage Constant>

For example, the name of a predefined part, the main function in error handling is: **MYAPPS-ERROR-MNGT**

- *MYAPP* is the Workspace's target name
 - *F* the Function mnemonic
 - *S* the Workspace mnemonic
 - *ERROR-MNGT* the usage
2. the *naming policy* is *type first*, the name structure is the following:
<Part Mnemonic>< Entity Mnemonic><Instance Target Name><Usage Constant>

The name of the Function part previously mentioned is **PSMYAPP-ERROR-MNGT**.

The usage constants are as explicit as possible to let you easily identify the role of the generated component. If you have a doubt, a comment is added in the header of the generated component's code.

For example, the usage constant *-ADD-ERR* means that the corresponding Function is responsible for managing errors occurring while performing the *ADD* action. The usage constant *-U-MAIN* means that the corresponding Function is a Main Function for a mono-Instance Business Object (*U* constant from the *MdlVGMnemos poolDictionary*).

Note: The usage in Records' names distinguishes between Working-Storage Records, prefixed with *W* (e.g.: *-WEND-KEY*), and SQL-Row Records, prefixed with *R* (e.g.: *-RINSTANCE*).

Note: The generated Data-Item names do not include a usage constant.

For example, the name of the Data-Item generated from the *DEPTNAME* Data Element will be: **DEPTNAMEDE**

- *DEPTNAME* is the Data Element's target name
- *D* the Data-Item mnemonic
- *E* the Data Element mnemonic

VisualAge for Java/VisualAge Smalltalk Enterprise Components Naming

The names of VisualAge for Java/VisualAge Smalltalk Enterprise components are built on the same principle:

1. the *naming policy* is *identifier first*, the rule is:
<Instance Target Name><Entity Mnemonic><Usage Constant>

For example, the name of the visual component generated for a detail Business Object will be: **SAMPLDetailSubview**

- *SAMPL* is the Business Object's target name
- *O* the Business Object mnemonic

- *DETAILSUBVIEW* the usage
2. the naming policy is *type first*, the name structure is the following:
<Entity Mnemonic><Instance Target Name><Usage Constant>
*The name of the Function part previously mentioned is **OSampIDetailSubview**.*

Predefined Beans/Parts

VAGTemplates generates a number of predefined components:

- Message tables storing the labels associated with error messages
- Functions storing errors
- Communication Record and Table (GUI) for sharing data within the application
- **GUI** Error windows displaying error messages
- **GUI** Windows menu displaying the list of open windows
- **GUI** Java/Smalltalk classes factoring out common services and managing errors
- **TUI** Error maps displaying error messages
- **TUI** Navigation Record and Table, including all the applications associated with the fastpaths
- **TUI** Error Program managing errors

The predefined beans/parts are all generated from the Workspace instances. They are used by all the components in an application.

For a complete description of the public predefined beans/parts, refer to “**Components Generated From a Workspace: Predefined Beans/Parts**” on page 312.

Server Architecture

The standard generators provided by VAGTemplates generate source code to implement the functional processing identified in the Information Model definitions. The standard generators support the implementation of the following types of systems:

- GUI client/server application system
- TUI application system
- Web application system

All these different system types generated by VAGTemplates rely on the same server programs. This means that you can generate GUI clients and TUI applications that will share the same server programs. If you specify the required business logic at the server level, it will apply to all the clients generated with VAGTemplates.

Server types

Three types of server are generated for each Business Object:

- Detail
- List
- Updatable list

Two additional server types are generated when required:

- Non I/O check server
- Foreign key lookup server

Detail Server

The detail server provides support for the basic database actions of Create, Read, Update and Delete (CRUD) on one row of a Business Object. The update action is managed for the primary table of a Business Object. A detail server is sometimes called an atomic server.

The detail server handles error management (for example, when a database action fails) and provides **hooks** to insert your business logic. Depending on your specifications, the detail server takes into account validation of the data, concurrence management, and joins between the tables.

List Server

The list server reads several rows of a Business Object. It answers data queries depending on the extraction and sort criteria specified in the Business Object.

The list server is designed to handle paging. The returned data is extracted from the selection depending on the specified first keys of your page. The server also answers the first set of keys of the next page to read.

Updatable List Server

The updatable list server performs a series of CRUD actions on a set of rows for a Business Object. The updatable list server sequentially processes each row by calling the appropriate detail server and stores any errors that occur, if needed.

The updatable list server can be used to perform several actions in the same logical unit of work (LUW).

Non I/O Check Server

This server, which is called only in a TUI Interface Unit, performs all the checks that can be done without accessing the database. This includes mandatory field checking and specialized edits for fields specified in the Business Objects that have selected check types (interval, value table, or customized check). Check types are specified in the Data Element definition (see “**Data Element**” on page 76).

These checks are also performed in the detail server if the *Control location* parameter is set to **client and server** or **server** in the Workspace Definition (see “**Client/Server Control**” on page 32).

Foreign Key Lookup Server

When a table has one or more foreign keys, then the value of the foreign key field must match a value of a primary key in the parent table. To support a help list function that can display a list of valid values for the foreign key, VAGTemplates generates this server when the primary table in a Business Object has a foreign key, and the foreign key is laid out in the fields defined in a Business Object definition.

When used, this server is directly called from the TUI Interface Unit program that uses a detail view for the Business Object and from the detail resource object in the GUI client Interface Unit. The list resource object generated from the Relational Table calls the help list server.

Generated Servers by Entity Type

	Two-tier Architecture	Three-tier Architecture
BusinessObject	<ul style="list-style-type: none"> • a mono-instance server composed of 4 processes making CRUD actions on the database • a multi-instance server reading a list of instances in the database 	<ul style="list-style-type: none"> • a mono-instance umbrella server calling the CRUD atomic servers generated from the primary table of the BusinessObject or an atomic server generated from the Business Object (example: a reading atomic server if the Business Object maps two tables) • a multi-instance umbrella server <ul style="list-style-type: none"> – a multi-instance atomic server reading a list of instances in the database
RelationalTable		an atomic server per each CRUD action *
RelationalTable HelpList	a mono-instance server composed of 4 processes making CRUD actions on the database; the list of instances are the primary keys of the table.	
* when possible; otherwise, the server is generated at the BusinessObject level.		

Client Architecture

Web Client

Technical Overview: VisualAge Generator and VAGTemplates provide two different architectures using the UI Record to build Web applications that transfer data to/from the browser:

- **Converse architecture** (also used for the generated TUI applications): when the entire page is sent to the browser, the application waits for an action to be done (by the end-user), i.e. it is conversing, and when an action is done, the entire page is returned to the server.
- **Xfer architecture:** sends an entire page to the browser, and when an action is done, the browser sends data back to the server (submitted action, entry data and possibly indexes). This mode — being much lighter than the Converse one — is provided as the default option in the VAGTemplates Information Model.

The graphic display 'Web read-only detail' is used to display data in a detail layout. No CRUD action is available. This page is never stored by the server.

Program Navigation: Program navigation (program A to program B) is done by transferring program (using Xfer) with or without parameters. These parameters, initialized by program A, are provided to program B (to prefill key data items for example).

- 'program link' UI Type
- whenever possible, zoom inter (List to Detail), implemented using 'link parameters'

Error Handling: Two levels of controls are provided in the generated applications.

- The first set of controls is located on the browser and directly linked to the data record definitions.
- The second set of controls is located on the server. It is a reuse of what have been implemented for TUI applications.

When the error can be linked to a data item, the message (in red) appears under it and under all the data (in standard color) otherwise.

Display: All available parameters used to layout Data Elements are not taken into account by the web generators.

Only the following parameters may be used:

- Updatable
- Read-only
- Read-only combo-box

GUI Client

Architecture Principle: The GUI client architecture is described below, from the most internal object to the most external one:

- The Resource Object encapsulates mono-access servers (detail server) and multi-access servers (list servers). There is only one instance per Resource Object class: the instance allowing access to physical data.
- The Business Object encapsulates data that is defined by a unique key, and actions for managing the data.
- The List Manager holds Business Object collections and manages paging.

Note: The Business Object and the List Manager access the server through the Resource Object.

- The detail and list subviews may display one Business Object instance or a collection of Business Object instances and trigger actions.
- The Views organize the subviews and the navigation management between data in the subviews.
- The View navigation organizes the management of common data within an application, the graphical navigation, and the management of data navigation between Views.

A couple of services are available. These services are built to be plugged to the overall design.

Logical Components: You will find here a general description of the architecture of the client side of the generated applications. Its goal is to explain the architecture, the choices made and their relevance.

Business Object as One set of data: The Business Object encapsulates a data instance. Therefore, there may be as many Business Object instances as there are rows in the read tables.

In order to be able to manipulate the same instance in various graphical objects, the offered API allows manipulation of the same Business Object instance provided that the key is identical.

The main characteristic of the architecture is in the Business Object definition.

The generated Business Object is instantiated; each instance corresponds to a table row.

Instance Uniqueness: The aim is to guarantee that **the same Business Object instance will be returned if it is being used by the application.** This mechanism is different from a cache mechanism: Instances that become unused are not memorized.

As the read instances are unique, their keys cannot be modified - because the key defines the instance. Therefore, two Business Object states are defined:

- The **persistent** state means that the instance comes from the database. In this state, instances are always unique. The Update/Delete actions are available in the persistent state.
- The **volatile** state corresponds to what would be displayed in a blank detail. The Read/Create actions are available in the volatile state. Delete is also available for convenience.

Only persistent instances are guaranteed to be unique.

Consequences: It is essential that the instances manipulated for the same key should be unique. However as soon as a Business Object is manipulated and not just simple data, various constraints appear.

Example

It is not possible to modify the key of a read instance since an instance is defined through its key. Modifying the key results in manipulating a different instance.

The status definition imposes some constraints on graphical use. Only a subset of actions is available depending on the status of the Business Object. The keys can be modified only for a volatile Business Object.

List Manager: The list manager is an object that contains the data to be displayed in a list. This data is stored as collections of Business Object's instances. Instances are sorted according to the sort criteria.

The list manager object also stores the necessary information to manage the list (extraction criteria data, paging data, automatic scrolling data).

Paging: Reading a list is done by reading one page of data after another. The list manager stores 2 collections: the full list of instances that contains all the pages that have already been read, and the list of instances of the current read page. It is possible to display either collection.

Autoscrolling: Lists can be scrolled without an explicit action, through smart scroll bar widgets. This mode is called auto-scroll.

First, the list needs to know the total row that will be displayed.

Then, the list sends a request to the system to retrieve x rows, depending on the action done on the scroll bar. To gather this amount of rows, one or more server calls may be needed. The server may also answer more instances than needed by the list during the request.

The list manager is able to handle these requests. It triggers as many calls to the server as needed and stores any extra rows that have been read. Each call from the server updates the total number of rows of the selection. The list manager is able to adjust itself if this number varies during the paging (rows added or deleted in the database).

Updatable List: Updatable list management is a special case of list management. As the list manager handles instances of Business Object, it is possible to modify a subset of them and track the changes, in order to consolidate them at once through a single server call.

Updatable lists offer the same functions as read-only lists and add some new ones. The updatable lists manager is thus a subclass of the list manager.

Note: Autoscrolling cannot be available for updatable lists.

Actions: All the actions performed on an updatable list become one of the 3 following ones:

- Create a row: a volatile Business Object is inserted,
- Update a row: a persistent Business Object is changed,
- Delete a row: a persistent Business Object is removed from the list.

Each of these actions is called a movement. The number of movements is limited, because the number of rows passed to the server is fixed.

- Creating and deleting one row cancels the movement,
- Creating and updating one row is one single create movement,
- Updating and deleting one row is one single delete movement,
- etc.

Resource Object as an Access Part: The server accesses are isolated into a part. **At run-time, there is one instance of this part per Business Object class.** This solves the problem of multiple client-server communication links; as only one Resource Object instance is manipulated, there is only one link per Business Object class.

Services:

Error Handling:

Introduction:

General Ergonomic Principles

The main ergonomic principle is to prevent the user from carrying out errors: error prevention is a major objective of the generated user interface (input help, contextual management of available actions,...).

Standard Generated Errors Typology

The suggested typology is deliberately oriented towards the " end user's " error approach and not according to the typology of the control processes underlying error detection. For example, from the " developer's " point of view, there is a difference between an error of data format and an error of contents according to a management rule (list of values, interval...). As far as the end user is concerned, the error is always an input error: therefore, for format or contents errors, the graphical process of the error must be the same, with the same error message.

Errors

Feedback is given the first time the field loses focus. This feedback includes the modification of the field background color, with selection of the non-valid value; visual feedback remains active until the error is corrected.

When explicitly requesting it (for example, activating a " Check " menu item), the user can open the standard window for presenting error messages which contains the list of messages associated with common input errors. Each message consists of :

- the laid out label of the non-valid field,
- a short sentence describing the nature of the error (standard message associated with an error, that can be easily customized).

Double clicking on an item in the message list positions the focus on the erroneous field (the first field, for a multi-field data). For Notebook presentations, double clicking also brings the page containing the erroneous data to the foreground.

The window also contains two push-buttons, the " OK " push-button is used to close the window without modifying its contents and the " Help " push-button is used to open the help panel associated with the selected message line. (help is a customization).

This window, also used when errors come from action activation, is detailed in the next section.

A system error is the only error that lets the user show the internal of the problem, so that he could contact a system administrator. The feedback associated with a " system " error must be very distinctive.

Messages: A message is one piece of information that is displayed to the user but should not stop the flow of an action.

It is displayed within the interface in an optional information bar.

Foreign Key Help List: A foreign key help list is a list of available keys from the target table of a foreign key. Help lists services are provided, as an option, only on single field foreign keys.

Sharing Data: An application (defined by the tree of Interface Unit from the root interface unit.) may need to store pieces of data and make it accessible from any place of the application. To do so, we define one instance of the class called **SharedPart**, that is known by all the views of the application.

This object is made to store data that needs to be shared by several windows. It is easy to make this object accessible by the subviews and other parts if needed. The sharedPart is generated and created with the root interface unit.

Subviews and Views: Views and subviews are the graphical places where data is displayed and used. They also are the place where the services are being plugged-in.

- The subview display one or a collection of Business Object,
- The view displays one or more subview and contains the navigation functions.

Views and subviews are able to retrieve the messages raised by the component they embed. They can also raise their own messages.

Java GUI Client Specificity:

Release of Business Object Instances: An object has to be released when it is no longer present in the Interface Unit. This happens in two different contexts:

- In a detail-type Interface Unit: when the displayed instance is replaced by another,
- In a list-type Interface Unit: when the object collection is refreshed or emptied (the latter being a particular case of refresh).

When an object has to be released, a dynamic search is performed so as to determine whether that object is still being used or not.

The Business Object sends an event when its release is requested. This event is received by detail-type Interface Units and List Managers. This event has a boolean attribute. The value of this attribute is modified when the event is being received. The Business Object is released if the boolean attribute is not modified.

Such object release is included in the runtime. As a result, customization will not cause any specific operation.

Smalltalk GUI Client Specificity:

Instance Management: Instance management provides the unicity mechanism on persistent Business Objects. It is a generic mechanism, handled by class services of the **InstanceManager** class.

The instance management has two goals:

- To register and make available the persistent instances that are in use, so that the actions can use these instances if needed,
- To remove the instances that are not in use anymore by the application.

TUI Client

There is an architectural structure to the programs and other 4GL parts used to implement the TUI system, and while a TUI system is not an object oriented system, components can still be identified. The system components identified for a TUI system are the following:

- Main program (View): Implements processing required to present a text user interface to the end user and manage the associated processing.
- Text maps (Subview): Text user interface parts used to present business data.
- Business object logic: The layer of processing logic representing the business context of a set of encapsulated physical data and database actions.
- Resource object logic: The layer of processing logic that implements server program calls for any required list or detail database processing.
- Shared component services: Common services available to other components, such as error management, messaging, and authorization processing.
- Navigation: Communication and control processing implemented using defined APIs, as required for each component.
- Server programs: Detail, list, updatable list, and help list servers to support access to the data used in each Business Object. Servers can be implemented to support either a two- or three-tier system layer structure.

For more information on these components, see below.

Overview of Generated Code

Servers and their Hooks

Hooks

Hooks are Functions or Methods generated by default and inserted in particular places within Programs to let you add specific 4GL, Smalltalk or Java treatments both on the Client and Server sides of your application.

There can be two kinds of Function hooks:

- those that are generated empty are suffixed with -HOOK.
- COMMIT Functions are generated with default code that can be modified. the **additionalChecks** method allowing to specify the inter-field checking in the generated

Method hooks are generated empty.

Hooks are standardly called in all applications where they are inserted, even when they are empty. Therefore, you need not define calls to your specific treatments.

Note: Hooks are generated for you to use and modify. They should not be deleted.

Hooks on the Server:

Note: The examples given in this paragraph are valid for a 2-tier architecture.

For one Business Object instance, VAGTemplates generates three server applications that correspond to the Business Object detail, list and updatable list. You will find a number of hooks in these servers *before* and *after* every data access Function; these hooks are all implemented as Functions.

Note: These hooks are common to GUI client and TUI applications.

Hooks Available with the Detail (mono-instance server)

- A number of hooks are available in the generated Program that corresponds to the mono-instance use of a Business Object, i.e. the detail.

In this Program, there are five types of data access:

- *select*, to read a row;
- *insert*, to create a new row;
- *update*, to modify an existing row;
- *delete*, to delete an existing row;
- *save*, to save new and modified rows at the same time;

Each access is described as a sequence of Functions, which includes hooks.

For example, if the generated Business Object is SAMPLE, the resulting generated Program is SAMPLA1 and the generated Functions are presented in the following diagram:

[-]	■	SAMPLFO-U-MAIN	Execute	Main Function	
	—	■	MYAPPF5-INIT-MAIN	Execute	
	+	■	SAMPLFO-SELECT	Execute	Select Function
	+	■	SAMPLFO-INSERT	Execute	Insert Function
	+	■	SAMPLFO-UPDATE	Execute	Update Function
	+	■	SAMPLFO-SAVE	Execute	Save Function
	+	■	SAMPLFO-DELETE	Execute	Delete Function
	—	■	SAMPLFO-DISP-NULL	Execute	Manage NULL values

– *Select Function Sequence*

[-]	■	SAMPLFO-SELECT	Execute	Select Function	
	—	■	SAMPLFO-A-SEL-HOOK	Execute	Hook: ante Select
	[-]	■	SAMPLFO-IO-SELECT	Inquiry	SAMPLRO-RINSTANCE MYAPPF5-SEL-ERR
		📄	SAMPLRO-RINSTANCE	Access Record	
		🔍	MYAPPF5-SEL-ERR	Execute	Error of the Inquiry
	—	■	SAMPLFO-E-SEL-HOOK	Execute	Hook to cancel an error
	—	■	SAMPLFO-TREAT-NULL	Execute	Manage NULL values
	—	■	SAMPLFO-P-SEL-HOOK	Execute	Hook: post Select

In this sequence, SAMPLFO-IO-SELECT reads data. The hooks included in this sequence allow you to insert specific treatment before and after the execution of the read action:

- SAMPLFO-A-SEL-HOOK is executed before (Ante) the SAMPLFO-IO-SELECT Function and can trigger a function before the data are read;
- SAMPLFO-P-SEL-HOOK is executed after (Post) the SAMPLFO-IO-SELECT Function and can trigger a function after the data are read.

– *Insert Function Sequence*

[-]	SAMPLFO-INSERT	Execute	Insert Function	
—	SAMPLFO-A-INS-HOOK	Execute	Hook: ante INSERT	
+	SAMPLFO-CONTROLS	Execute	Fields control on the server	
—	SAMPLFO-SRVCK-HOOK	Execute	Hook for the server controls	
+	SAMPLFO-I-REQ-CHK	Execute	Manage required fields	
—	SAMPLFO-DTCT-NULL	Execute	Manage the NULL values	
+	SAMPLFO-IO-INSERT	Add	SAMPLRO-RMAIN-TBL	MYAPPPFS-ADD-ERR
—	SAMPLFO-E-INS-HOOK	Execute	Hook to cancel an error	
—	SAMPLFO-P-INS-HOOK	Execute	Hook: post INSERT	

In this sequence, SAMPLFO-IO-INSERT creates data. The hooks included in this sequence allow you to insert code before and after the execution of the creation action:

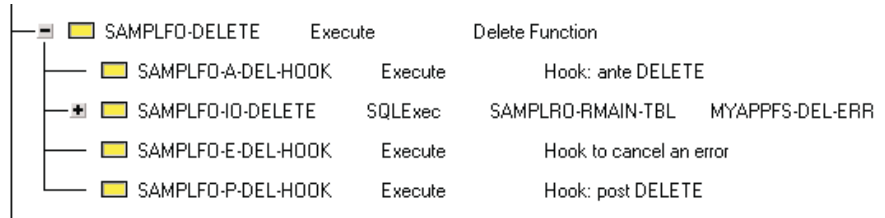
- SAMPLFO-A-INS-HOOK is executed before any other function.
 - SAMPLFO-CKSRV-HOOK is executed before the standard server checks and the SAMPLFO-IO-INSERT Function and could trigger additional checks before the data is created;
 - SAMPLFO-P-INS-HOOK is executed after the SAMPLFO-IO-INSERT Function and could trigger a function after data is created.
- *Update Function Sequence*

[-]	SAMPLFO-UPDATE	Execute	Update Function	
—	SAMPLFO-A-UPD-HOOK	Execute	Hook: ante UPDATE	
+	SAMPLFO-CONTROLS	Execute	Fields control on the server	
—	SAMPLFO-SRVCK-HOOK	Execute	Hook for the server controls	
—	SAMPLFO-DTCT-NULL	Execute	Manage the NULL values	
+	SAMPLFO-IO-UPDATE	SQLExec	SAMPLRO-RMAIN-TBL	EZERTN
+	MYAPPPFS-UPD-ERR	Execute	Error of the Update	
—	SAMPLFO-E-UPD-HOOK	Execute	Hook to cancel an error	
—	SAMPLFO-P-UPD-HOOK	Execute	Hook: post UPDATE	

In this sequence, SAMPLFO-IO-UPDATE updates data. The hooks included in this sequence allow you to insert code before and after the execution of the update action:

- SAMPLFO-A-UPD-HOOK is executed before any other function.

- SAMPLFO-CKSRV-HOOK is executed before the standard server checks and the SAMPLFO-IO-UPDATE Function and could trigger a function before the data are updated;
 - SAMPLFO-P-UPD-HOOK is executed after the SAMPLFO-IO-UPDATE Function and could trigger a function after data is updated.
- *Delete Function Sequence*



In this sequence, SAMPLFO-IO-DELETE deletes data. The hooks included in this sequence allow you to insert code before and after the execution of the delete action:

- SAMPLFO-A-DEL-HOOK is executed before the SAMPLFO-IO-DELETE function and could trigger a function before the data is deleted;
 - SAMPLFO-P-DEL-HOOK is executed after the SAMPLFO-IO-DELETE function and can trigger a function after the data is deleted.
- *Save Function Sequence*

☐	SAMPLFO-SAVE	Execute	Save Function		
☐	SAMPLFO-A-UPD-HOOK	Execute	Hook: ante UPDATE		
☒	SAMPLFO-CONTROLS	Execute	Fields control on the server		
☐	SAMPLFO-CHK-HOOK	Execute	Hook for the interfield checks		
☐	SAMPLFO-DTCT-NULL	Execute	Manage the NULL values		
☒	SAMPLFO-IO-UPDATE	SQLExec	SAMPLRO-RMAIN-TBL	EZERTN	Update IO Function
☐	SAMPLFO-A-INS-HOOK	Execute	Hook: ante INSERT		
☒	SAMPLFO-I-REQ-CHK	Execute	Manage required fields		
☒	SAMPLFO-IO-INSERT	Add	SAMPLRO-RMAIN-TBL	MYAPPF5-ADD-ERR	Insert IO Function
☒	MYAPPF5-UPD-ERR	Execute	Error of the Update		
☐	SAMPLFO-P-INS-HOOK	Execute	Hook: post INSERT		
☐	SAMPLFO-P-UPD-HOOK	Execute	Hook: post UPDATE		
☒	SAMPLFO-DELETE	Execute	Delete Function		
☐	SAMPLFO-DISP-NULL	Execute	Manage NULL values		
☐	SAMPLFO-U-COMMIT	Execute	Hook for Commit		

The -SAVE function performs creation and update actions at the same time. The Functions it calls are the same as those called by the

-UPDATE and the -INSERT functions.

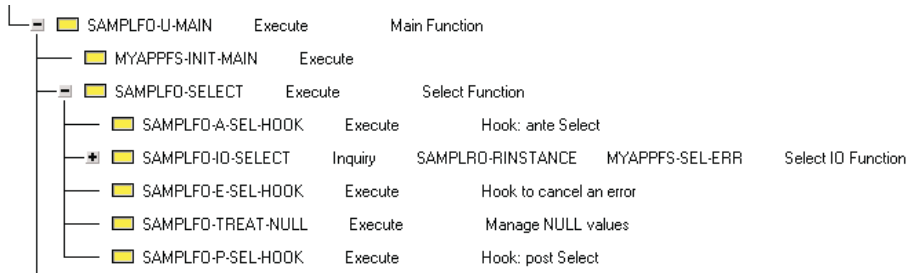
In this sequence, SAMPLFO-IO-SAVE creates and updates data. The hooks included in this sequence allow you to insert a code before and after the execution of the update and insert actions:

- SAMPLFO-A-UPD-HOOK is executed before the SAMPLFO-IO-UPDATE Function and could trigger a function before the data is updated;
- SAMPLFO-A-INS-HOOK is executed before the standard server checks and the SAMPLFO-IO-INSERT Function and could trigger a function before the data is created;
- SAMPLFO-P-INS-HOOK is executed after the SAMPLFO-IO-INSERT Function and can trigger a function after data is created.
- SAMPLFO-P-UPD-HOOK is executed after the SAMPLFO-IO-UPDATE Function and can trigger a function after data is updated.

Hooks available In a Three-tier Architecture: Select Function Example

•

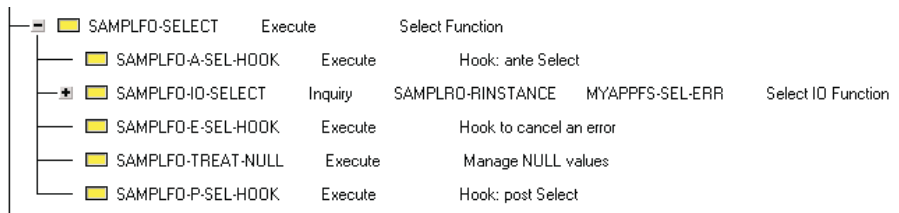
- *Umbrella server*



The above example shows only the select function.

- SAMPLFO-A-SEL-HOOK is executed before the call to the SAMPLOS atomic server that reads data from the database. It can trigger a function before the data is read;
 - SAMPLFO-P-SEL-HOOK is executed after the call to the SAMPLOS atomic server that reads data from the database. It can trigger a function after the data is read.
- *Atomic server for the selection: SAMPLOS*

The following atomic server is generated from the Sample Business Object. If it were generated from a Relational Table, it would be the same but named <Relational Table target name>TS, for example ORGTS. Its generated functions would be name d ORGPT-IO-SELECT, for example.



In this sequence, SAMPLFO-IO-SELECT reads data from the database. The hooks included in this sequence allow you to insert code before and after the execution of the read action:

- SAMPLFO-A-SEL-HOOK is executed before the SAMPLFO-IO-SELECT function and could trigger a function before the data is read;
- SAMPLFO-P-SEL-HOOK is executed after the SAMPLFO-IO-SELECT function and can trigger a function after the data is read.

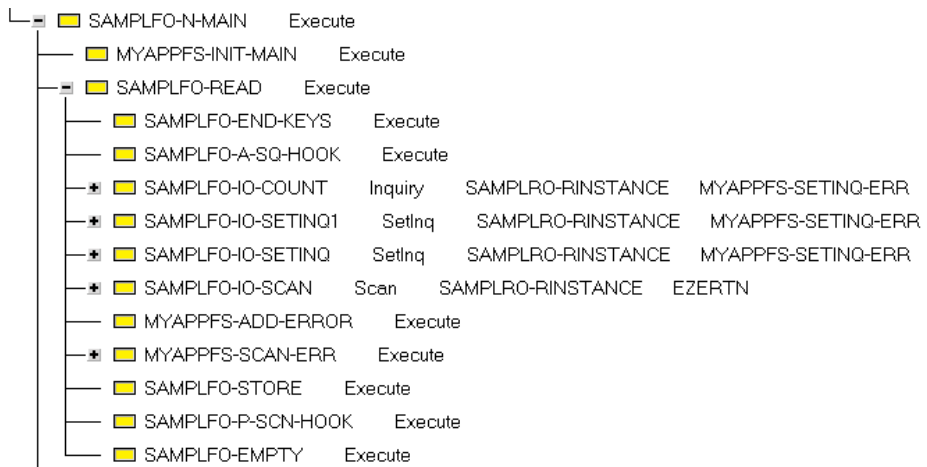
Thus, the same hooks are called on the umbrella server and on the atomic server to allow you insert code on each server.

Hooks Available with the List (multi-instance server)

- A number of hooks are available in the generated program that corresponds to the multi-instance use of a Business Object, i.e. the list.

In this program, the data access consists of reading a data page. This access is described as a sequence of functions, which include the hooks.

For example, for the generated SAMPLE Business Object, the resulting generated program is SAMPLAN and the generated functions are presented in the following diagram:



Read Function Sequence

-

In this sequence, the SAMPLFO-READ function reads data, the SAMPLFO-IO-SETINQ1 function is then executed when reading the first page, the SAMPLFO-IO-SETINQ function performs the selection for the other pages, the SAMPLFO-IO-SCAN function reads one row, and the SAMPLFO-STORE function stores this row in the Working Storage Record.

Note: With the SAMPLFO-IO-SETINQ1 function, the WHERE clause takes into account the extract criteria but not the paging criteria. The WHERE clause being simpler, the response time is shorter.

The hooks included in this sequence allow you to insert code before and after the execution of the read action:

- SAMPLFO-A-SQ-HOOK is executed before the SAMPLFO-IO-SETINQ function that performs global selections and could trigger a function before the data is read, for example to modify or add extract or paging criteria;
- SAMPLFO-P-SCN-HOOK is executed after the SAMPLFO-IO-SCAN and SAMPLFO-STORE functions, i.e. once a row has been read and stored in

the Working Storage Record. For example, it could trigger calculation of the data stored in the Working Storage Record.

Hooks Available with the Updatable List (multi-instance server)

- The actions available for an updatable list are the same as those available for a detail: creation, update, deletion, read. The difference is that these actions can be triggered on several rows at the same time, whereas in a detail, they are triggered one at a time.

Therefore, the function sequences for the multi-instance server are the same as those for the mono-instance servers.

For example, for the generated Business Object SAMPLE, the resulting generated program corresponding to the multi-instance use of the Business Object is SAMPLAL

More Hooks on the Server:

- -CHK-HOOK Function. This hook allows you to implement additional controls before a server function is activated, like inter-field checks. These checks could allow you to indicate, with 4GL expressions, the relationships that must exist between the fields presented by the Business Object.

*Let us take the example of the **Vehicle** Business Object which contains the "Average fuel consumption" and "Maximum fuel consumption" fields. You could write an inter-field check that ensures that the average fuel consumption is strictly smaller than the maximum fuel consumption.*

- -COMMIT. This hook contains default code to handle a commit or a rollback, if needed (umbrella server and error status).

For details, see also "**Commit/Rollback Processing**" on page 263.

Caution: Differentiate this check from value checks you may define when specifying your Data Elements.

TUI Only: More Hooks on the Server:

- -SEC-HOOK function. This function allows you to write security treatments. By default, it is called by the -NAVIGATE function that manages navigation. For example, you could use this hook to define user authorizations to navigate to some maps and not to others.
- -N-EXT-HOOK function. It is called by the multi-instance server application to add code to the extract criteria. You could use this hook to customize the extract criteria used as access parameters.
- -U-EXT-HOOK function. It is called by the mono-instance server application to add code to the extract criteria. You could use this hook to customize the extract criteria used as access parameters.
- -INIT-HOOK function. There is one -INIT-HOOK function per generated Interface Unit in the application. This function allows you to implement initialization.

Hook on the Client Side of an Application: If you specified a customized check for a Data Element in the Workbench, you can use a hook method to define the check.

The **additionalChecks** method from the Business Object bean/part allows you to implement additional controls before a server function is activated, like inter-field checks. These checks could allow you to indicate the relationships that must exist between the fields presented by the Business Object.

*Let us take the example of the **Vehicle** Business Object which contains the "Average fuel consumption" and "Maximum fuel consumption" fields. You could write an inter-field check that ensures that the average fuel consumption is strictly smaller than the maximum fuel consumption.*

Caution: Differentiate this check from value checks you may define when specifying your Data Elements.

The **additionalChecks** method is called by the **checkBusinessObject** method from the Business Object bean/part. Once the customize check is defined, the **additionalChecks** method can call the **putFocusOnField** method that puts the focus back on the erroneous field and the **putFieldInError:** method which changes the field's color. These are Business Object bean/part methods.

For information on the Business Object bean/part, refer to "*Business Object Bean/Part*" on page 293)

- -N-SRV-HOOK Function. It is called after response from the multi-instance server application and before error treatments.
- -U-SRV-HOOK Function. It is called after response from the mono-instance server application and before error treatments.
- -CKCLI-HOOK Function. You can use this hook to implement interfield checks on the client.
- -DEF-HOOK Function. You can use this hook to set customized values to the -WINSTANCE Record that stores data used by the mono-instance Business Object.

We indicate for each generated part its status in the generated application, preceded with this **VAST** sign. The possible statuses are the following:

- API: functional component on which an API is offered in the final application;
- RAD: mainly graphical components but also components on which an API is offered in the final application;
- HOOK: insert point for specific coding in the final application;
- INTERNAL: organic component.

The <> sign placed before part names denotes a prefix, which is the target name of the source entity plus the entity mnemonic (as specified in the Workbench).

For information, refer to “**Traceability Categories**” on page 234.

Server Common Functions

Each server performs specific tasks but is implemented with common functions that apply to all server types.

Error Management: Several levels of errors are managed in the servers:

- Warning
A message is raised but the treatments are not stopped.
- Application error
Typically, this is when a business rule fails or when an action on the database fails (as when a row already exists). Several errors can be raised. An error causes a rollback and treatments are stopped.
- System error
This is a database or communication error that cannot be understood by the server.

Any identified errors are stored in the common records passed as parameters. Warning, application errors, and status flags are kept in an ERROR-LIST record and system errors in a SYS-ERROR record.

Commit/Rollback Processing: The parameter **LUW mode** specified in the Workspace determines if client or server-base LUW management will be implemented. Server-based management is recommended.

In the generated application system, commit and rollback processing are managed by a control indicator that is passed as a parameter to the server program.

- To implement server-based LUW management, the LUW control indicator, called CONTROL-COMMIT-FLAG, is set to true when calling the server program.
This flag in the ERROR-LIST record is memorized at the server beginning in WVAR. The commit/rollback is executed at the end of the server in a hook suffixed with -COMMIT.
- Commit or rollback processing is performed prior to exiting the server program when the LUW control indicator is set to true and no errors have been detected. If an error has been detected, a rollback is performed. Code hooks can be used to customized this behavior.
- When a server program calls another server program (such as when an atomic detail server is called from an umbrella server), the called (atomic)

server does not perform commit or rollback processing. The umbrella server sets the LUW control indicator to false before calling the atomic server. Code hooks allow you to change this behavior.

Concurrency Management:

Two- or Three-Tier Layers

A parameter at the VAGTemplates Workspace level allows you to specify whether you want to generate two- or three-tier servers.

Two-tier generation will produce detail and list servers. These servers are called directly by the client. They contain business logic and database I/O in one single VisualAge Generator program.

Three-tier generation produces umbrella servers that are directly called by the client as well as several atomic servers, one for each type of action (list access, CRUD actions). The umbrella server calls the appropriate atomic servers. Business logic can be implemented on the umbrella server side or on the atomic server side, depending on the data you need to manipulate and on the level of reusability of your treatments.

Generation of Atomic Detail Servers

When you specify a three-tier implementation approach for the server programs, the generation of one Business Object will produce several servers:

- Umbrella servers for detail and list processing. These servers are generated from the Business Object entity and call several atomic detail servers.
- Atomic detail server for the list access. This server is generated from the Business Object.
- Atomic detail servers generated from the primary table of the Business Object. These servers handle the CRUD actions.
- Atomic detail server generated from the Business Object. These servers handle some CRUD actions and are generated only when the atomic detail servers generated from the primary table cannot be used for the Business Object. For example, if a Business Object maps only part of the primary table's fields, the update atomic detail server generated from the primary table cannot be used. A specific detail server is generated for the Business Object.

Generation of Help List Servers

Help list services are provided in the generated application for the foreign keys of the primary table of a Business Object. The help list displays the available keys, read from the database, of the target table of the foreign key.

Help list servers are always implemented as atomic servers that are generated from the Relational Table entity (the target table of the foreign key).

Clients

In this section, we review how the components of each client system — Web client, GUI client and TUI client — are structured to provide support for the required processing.

Web Client

UI Record Definition & Content: This record contains all the items the interface (displayed in a browser) needs to display and manipulate data and navigate between programs.

Data in XFer

The record is displayed by an Xfer without an application name. This means that no server keeps the record's data.

To avoid losing input data while transferring, a 'Form' UI Type data item is defined, including all the data item defined in the UI Record.

Acting as a Program Link, this Form reposts all of its content in input to the web server defined in its properties.

The standard generated web applications transfer to the same server program using the properties of the Form when an action (except the navigate action) is triggered.

Note: Only 'input', 'input/output', 'hidden", and 'submit' UI types values are reposted.

Data in Converse

All the data is sent back to the server in Converse. There is no special architecture needed for the Data-Item here.

Actions on Data

The actions are defined within the UI Record as 'Submit' UI Type data items. They are manipulated in the program by using their initial values (for XFer) or the current value of the data-item (for converse).

When an action is submitted, the 'submit value item' is set by it. Then it can be tested and used by the program in order to trigger the correct action.

Buttons showing their action label are grouped by use (CRUD, List actions, and FK actions).

CRUD actions are provided on a Detail and only Top/Next actions for a List (other actions can be triggered by using the browser actions).

Paging in XFer

The 'Next page' action needs to know what the next-key is in order to

give the info to the server. So each time the program is back from the foreign-key help list server, the value of the next-key is stored in a hidden UI type data item.

Paging in Converse

The paging data is memorized in the Next-Key record by the server.

Foreign Key Data

As for business object Lists, only Top/Next actions are provided on foreign key help lists.

The help list data is stored in a read-only data-item, that is displayed as a combo-box. The current value of the foreign-key is added in the list if it is not present in the current page.

For Xfer: As the read-only data is not passed to the server when an action is submitted, the help list data is also stored in a hidden data-item, which is a copy of the read-only data-item. The index of the combo-box will allow to retrieve the selected value.

For Converse: The help list data is memorized by the server in the Wpage record.

Program Definition: One VAGen program is generated from an instance of Interface Unit as an entry point. The servers used by the Web applications are the same as for all the VAGT generated applications. The program contains the following three attributes:

- Type: 'Web Transaction'
- Message table prefix: xMSG (ENU table)
- First UI Record (based on the Interface Unit content)

Navigation: Transfer between Programs: A program transfers to another one with 'Program Link' UI Type data items defined in the UI Record.

Navigation between interface units and zoom is done by posting the value of the selected key items to the first UI Record of the targeted program.

Error Handling:

Message Table

The message table is used to display error messages.

Web side controls

Each time an input field is modified, its linked control function and definitions controls are executed. If an error occurs, a message is constructed with the xMSGENU messages table; then the user must correct the error before continuing.

Server controls

Like TUI applications, these controls are done within the program xxxXOF generated at the Business Object level.

System Error Management

In case of a system error, a program, generated at the Workspace level is called. This program, using a UI record displays info about the error encountered.

GUI Client

Java GUI Client:

Instance Manager System:

Instance Manager System Diagram:

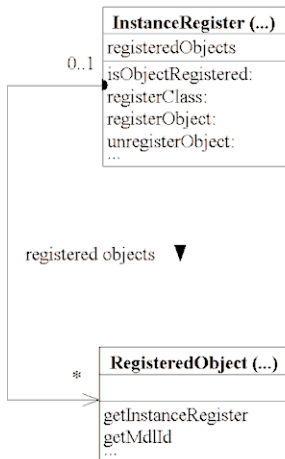


Figure 9. Instance Manager

InstanceRegister: Object managing the unicity of Business Object instances used by the generated application.

This class is managed as a singleton (unique object). Methods dedicated to the management of Business Object instances are instance methods.

RegisteredObject: Abstract class, linked to the **InstanceRegister** class, implementing the recording/unrecording mechanism on the **InstanceRegister**.

Model System: You will find below a detailed description of the different objects found in the main component of a Java client. This description starts with VisualAge Generator parts and ends with the MMI part.

Model System Diagram:

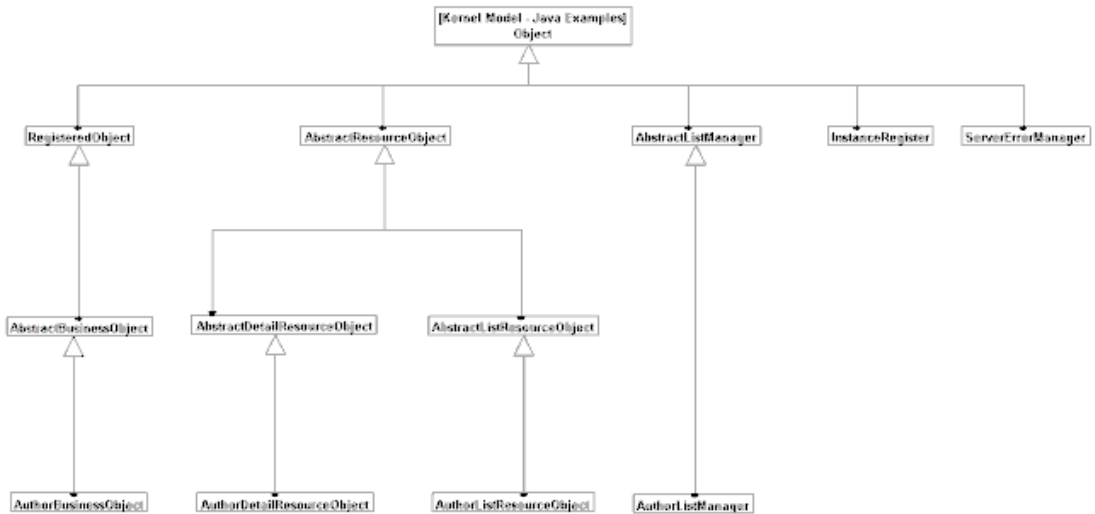


Figure 10. Class Hierarchy

This diagram presents the main Java classes generated from a Business Object named "Author".

VisualAge Generator Parts: These objects encapsulate as Java beans the VisualAge Generator "parts" and manage the Client Server communication.

ResourceObject: Technological object ensuring the link between model objects (Business Object, List Manager) and server objects.

For each ResourceObject class, one and only one instance is created (Design Pattern Singleton). In order to access the server, all instances of each type of Business Object are channeled through this ResourceObject unique instance.

The benefit of this architecture is to reduce the communication load between client and server components.

A ResourceObject is responsible for server access, for the conversion of records into Business Object.

Also, a ResourceObject detects server errors and middleware exceptions (CSOException). If an error is detected, the ResourceObject raises an exception (MdlException).

Hierarchy

For each type of Business Object, there are two ResourceObject sub-classes: a ListResourceObject and a DetailResourceObject.

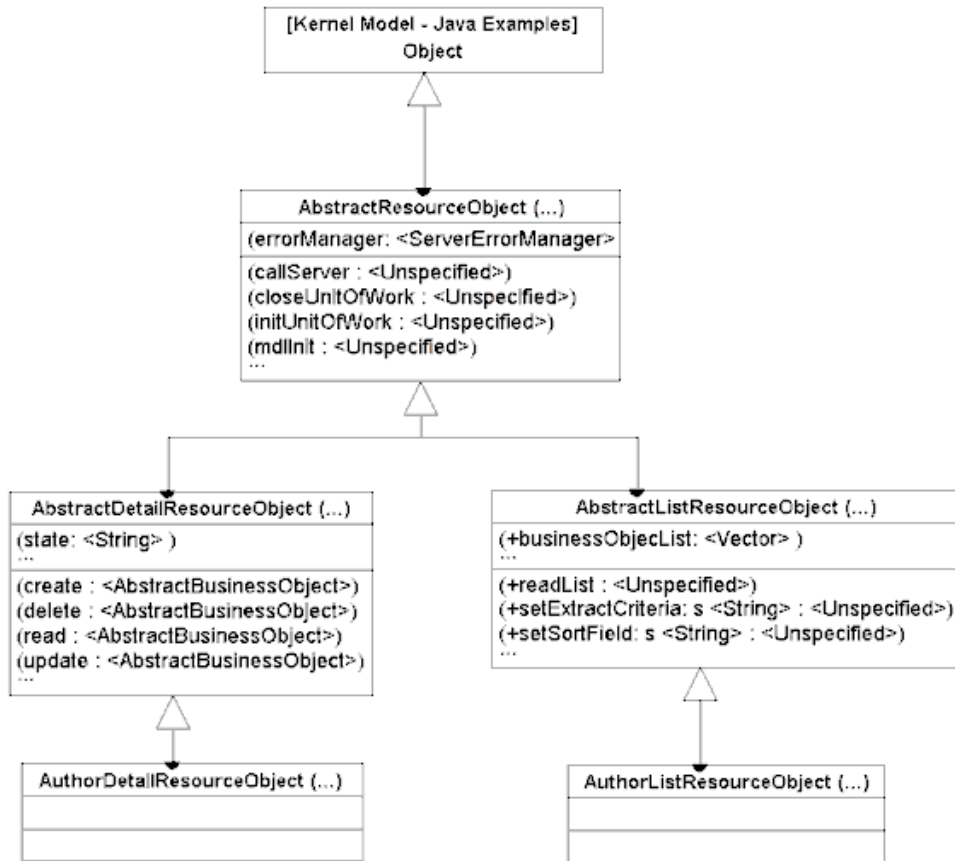


Figure 11. ResourceObject Hierarchy

AbstractResourceObject

Abstract class managing server calls and error detection.

AbstractListResourceObject

Abstract class managing list server access and implementing the list-dedicated API used by the ListManager.

<>AbstractDetailResourceObject

Abstract class managing detail server access.

Note: <> represents the prefix of the generated classes, dependent on instance parameterization. In the diagram above, <> is replaced by "Author".

<>DetailResourceObject

Concrete class implementing VisualAge Generator parts related to detail servers.

<>ListResourceObject

Concrete class implementing VisualAge Generator parts related to list servers.

BusinessObject: Object representing a Business Object described in VAGTemplates.

Its attributes procede from the DataFields of the Business Object and offers services corresponding to the actions the end user may ask: creation, modification, deletion... of the Business Object.

Each instance of BusinessObject is unique in the application, which allows the management of modifications. In order to do this, BusinessObject inherits from RegisteredObject which implements the registration mechanism to the InstanceRegister.

See also InstanceRegister and ResourceObject.

Hierarchy

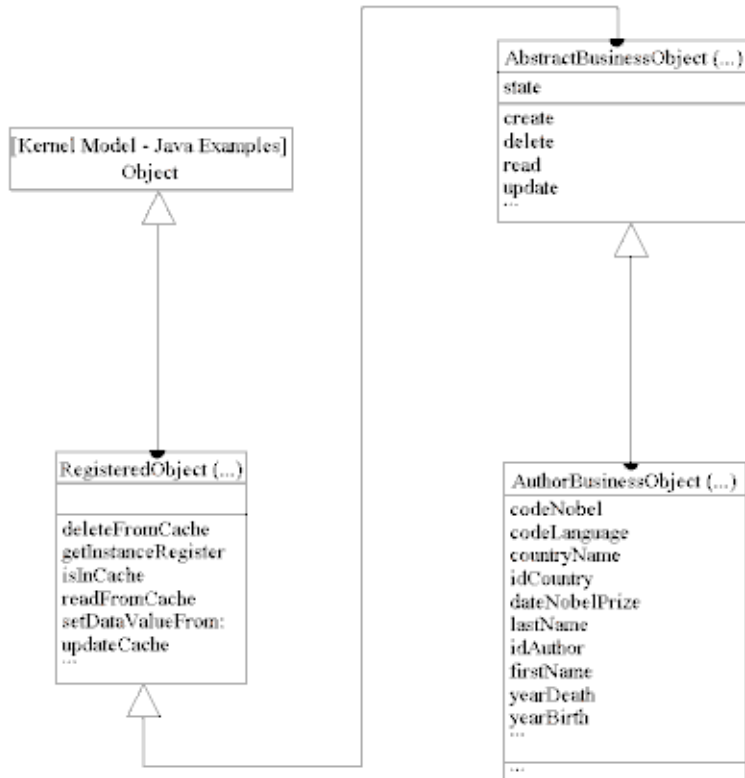


Figure 12. BusinessObject Hierarchy

RegisteredObject

See RegisteredObject above.

AbstractBusinessObject

Abstract class implementing mechanisms common to all BusinessObject objects, including corresponding messages (action failed/succeeded). This class includes the CRUD API (create, refresh, update, delete). The Business Object instance API is characterized by public methods and by a corresponding BeanInfo page, allowing to manipulate the object via connections.

<>BusinessObject

Implements the Business Object's specific data and associated methods. The field types are String, Date, Long and Double. Also, within a BusinessObject object, a reset method may be implemented and is described below.

- Reset
 - Principle

Applications must be able to retrieve the last memorized read of a persistent Business Object. In a user input context, the initial data must be saved. In a database read context, a reinitialization to null is performed.

- Implementation

Two instance variables are generated for each field: a field modification flag and a copy of the data.

Setters operate in two modes: a mode where the initial value is memorized (if changed by the end-user) and a reinitialization mode (flag and value copy set to null).

This mode toggles before and after the BusinessObject gets filled in by the ResourceObject and is calculated by the getMdlStoreCopyOfData() method which also takes into account the Business Object status.

Note: The reset method is implemented in the concrete class of the BusinessObject ; internal mechanisms are located in the parent.

- Global modification flag

A Business Object modification flag is filled in by the setters: it is positioned to true when the setters mode toggles.

List Manager: This object manages a list of Business Objects, this list proceeds from the VisualAge Generator server list.

List management is performed in relation with the ListResourceObject.

Also linked to the InstanceRegister regarding instance management and to the MdlTableModel regarding their graphical presentation.

AbstractListManager

Abstract class defining the set of methods allowing list operations: readFirst, extract, readPrevious, readNext et refresh. This class also includes error message management: data retrieved succeeded/failed.

<>ListManager

Implements getters and setters of sortCriteria and extractCriteria, also including the method evaluating the possible actions on a list (read next/previous page).

MdlTableModel

See MdlTableModel below.

Error System: Sub-system allowing for server and client error management.

CommonServices: Object allowing to manage services common to the whole dialog.

This class is managed as a singleton (unique object) shared by the whole dialog. It centralizes errors (distributed in `ErrorObject` and `SystemErrorObject`) and manages the call of error message boxes (for both application logic and system errors).

This class is instantiated by each view.

ErrorHandling: Object managing error raising services.

This class is managed as an instance. It validates the Business Objects' fields, changes their color, etc. This class is instantiated by the `DetailSubview` and implements the following event listeners:

- `MdlChecksListener`
- `MdlDetailActionListener`

Field: Object managing the Business Objects' fields.

This class is managed as an instance. It triggers the call for validation on the Business Objects' fields as well as and raises focus-related events (gain, loss).

ErrorObject: Object representing an error in the application logic. This class is managed as an instance.

SystemErrorObject: Object representing a system error and/or a CSO exception sent by the server.

ServerErrorManager: Object allowing management of errors raised by the servers.

This class is managed as an instance. It procedes from the `VisualAge Generator servers`.

According to the type of errors (application logic, system, or CSO exception), it creates the associated objects (`ErrorObject` and `SystemErrorObject`).

ErrorMessageTable: Hashtable type object dedicated to error message management.

UserMessageTable: Hashtable type object dedicated to the management of user error messages (empty by default).

ErrorView: Window presenting translated errors.

The opening of this window is controled by the `CommonServices` class.

It allows the end-user to select an error. As a result, the focus is on the corresponding field.

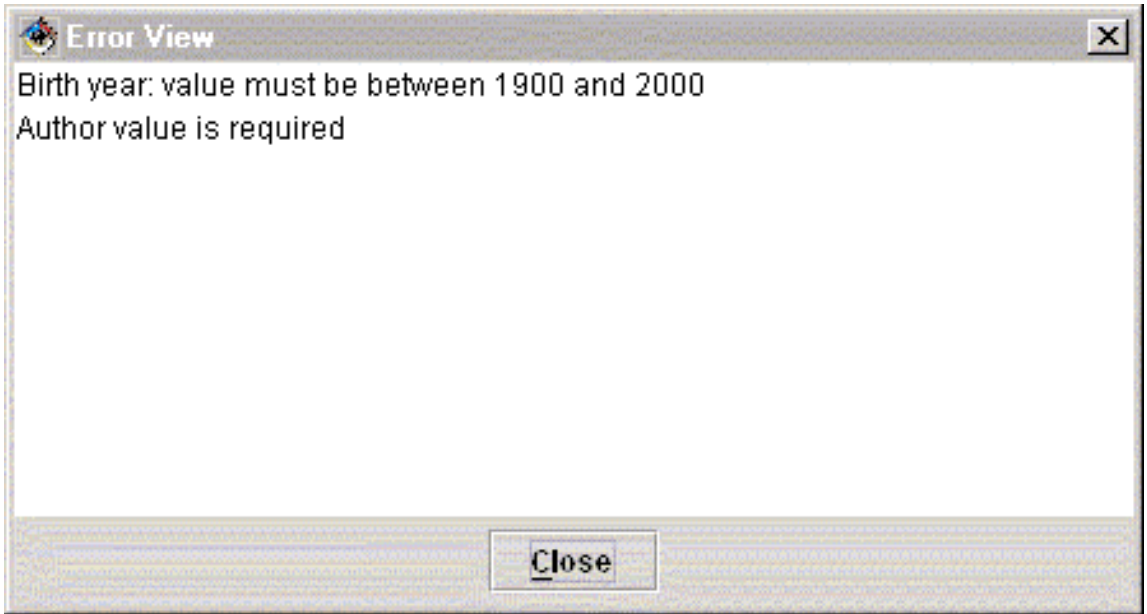


Figure 13. ErrorView

SystemErrorView: Window presenting system errors and CSO exceptions.

The opening of this window is controlled by the CommonServices class.

Interface Unit System: The Interface Unit System presents the prototype graphical components only.

All graphical windows generated with VAGTemplates use Swing components.

DetailSubview: This component represents the standard Detail view, it inherits from the JPanel class.

This view contains an instance of BusinessObject and an instance of ErrorHandler.

Representation of DataFields

With VAGTemplates, several types of graphical components may be generated:

- JTextField
- JRadioButton
- JSlide

- JComboBox

As well as a combination of Swing components in order to create:

- DroppedDownList
- DropDownList

Data Conversion

Data format and display depends on a component named MdlConverter which belongs to the runtime.

For all types of data elements but alphanumeric ones, this component formats data presentation according to the value types defined in VAGTemplates (numeric, date, TimeStamp, Time). Input and display formats are defined which control user input and data presentation.

ListSubview: Standard presentation of a Business Object list. It integrates a ListManager instance variable and an instance of MdlTableModel.

Also includes the CommonServices instance.

MdlTableModel: This object represents the list graphical component model (JTable), and therefore inherits from `com.sun.java.swing.table.AbstractTableModel`.

It can be considered as an interface between the Business Object managed by the ListManager and JTable graphical component.

The MdlTableModel manages the list of elements to display, the ListManager contains the server list.

InterfaceUnitView: JFrame-type window integrating both the BusinessObjectListSubview and BusinessObjectDetailSubview Beans. This window implements the intra zoom.

Java client classes categories: The different classes generated in a Java client may be divided into three categories according to their links with VAGTemplates.

No link : not generated

Linked to the VAGTemplates Workspace : generated with "predefined beans"

Linked to a VAGTemplates entity : always generated

Not generated: Runtime: These classes - found in the `com.ibm.mdl.runtime.vaj` package - are essentially tools used by the generated application and are entirely independent of VAGTemplates instances.

- <>MdlDocument
- <>MdlComboBoxModel
- <>MdlTableMap
- <>MdlTableSorter
- <>MdlEditor
- <>MdlTableRenderer
- <>MdlClock
- <>MdlChecksEvent
- <>MdlDetailActionEvent
- <>MdlInstanceManagementEvent
- <>MdlListActionEvent
- <>MdlConverter
- <>MdlInstanceRegister
- <>MdlRegisteredObjectAbstract
- <>MdlException
- <>MdlChecksListener
- <>MdlDetailActionListener
- <>MdlInstanceManagementListener
- <>MdlListActionListener
- <>MdlRegisteredObject

Generation linked to the VAGTemplates Workspace: Classes generated once per VAGTemplates Workspace, they mostly correspond to the parent classes of:

- <>MdlComboBoxModel
- <>MdlTableModel
- <>MdlErrorView
- <>MdlSystemErrorView
- <>AbstractBusinessObject
- <>AbstractListManager
- <>AbstractResourceObject
- <>AbstractDetailResourceObject
- <>AbstractListResourceObject
- <>MdlCommonServices
- <>MdlDataElementsChecks
- <>MdlDetailModel
- <>MdlErrorObject
- <>MdlField

- <>MdlServerErrorManager
- <>MdlSystemErrorObject

Note: <> represents the prefix of the generated classes, dependent on the Workspace parameterization.

Generation linked to VAGTemplates instances: Classes directly linked to the definition and parameterization of VAGTemplates instances. They are sorted in two different packages:

- LogicPackage
 - <>BusinessObject
 - <>ListManager
 - <>DetailResourceObject
 - <>ListResourceObject
- VisualPackage
 - <>BusinessObjectDetailSubview
 - <>BusinessObjectListSubview
 - <>WInterfaceUnitView

Note: <> represents the prefix of the generated classes, dependent on instances paratemerization.

Smalltalk GUI Client:

Logical Components: This chapter offers a general description of the architecture of the client side of the generated applications. Its goal is to explain the architecture and also to justify the choices that are made here and their relevance.

Business Object:

Data Storage

When an instance is persistent, the data that has been read from the database must be returned to, and local changes made to the Business Objects' data can be forgotten. This function should not come with a full copy of the Business Object's data: that would be too costly in term of performances. To do so, the initial piece of data is memorized only when the user changes it through the interface (the first time). To be able to distinguish from an original value that is nil and a copy that has not been set yet; we need to define a flag for each field. That flag means: 'the field has been changed since it has been read from the database, and its original value is preserved'. This data management allows to implement several functions:

- Cancel the changes made on a Business Object (reset),

- Warn the user that changes should be committed or cancelled (message box when a detail is closing),
- Test whether a Business Object has been locally changed or not (isModified)

Partial Read and Update

When an instance is read in a list, it is possible to read only a few fields in the Database and read all the fields when the data is moved to a detail. Therefore, there are 2 ways to read a Business Object :

- The Business Object read via a detail contains all its data,
- It is possible to define the Business Object so that the list would only read a subset of the Business Object's data.

A Business Object that is not fully read in a list can be defined in the Model (access level of a field). In this case, another state applies to the persistent instance : the **accessLevel**.

AccessLevel can have 2 values :

- partial, which means that the Business Object has been read only through a list. Only a subset of its fields has been read,
- complete : All the fields of the Business Object have been read.

Access Level has no meaning for a volatile Business Object. It is possible to update and delete a partial Business Object.

List Manager: There is one instance of list manager for each list, as the context (current page and extraction criteria setting) is different for each list instance.

If an application contains several lists of the same Business Object, several collections of instances will thus exist.

However, the different collections will point to the same objects, thanks to the Business Object's unicity mechanism.

The collections are duplicated, not the data that they contain. The actions available with the list manager are of two kinds: Defining a selection and paging.

Selection

One set of extraction criteria is defined for a Business Object. These extraction criteria are used by the where clause of the select order to define the selection. Once this selection has been defined, it is possible to read its rows. Most of the time, the server will not be able to answer the full selection. Then reading the selection will be done by reading one page of its data after another.

How does it work?

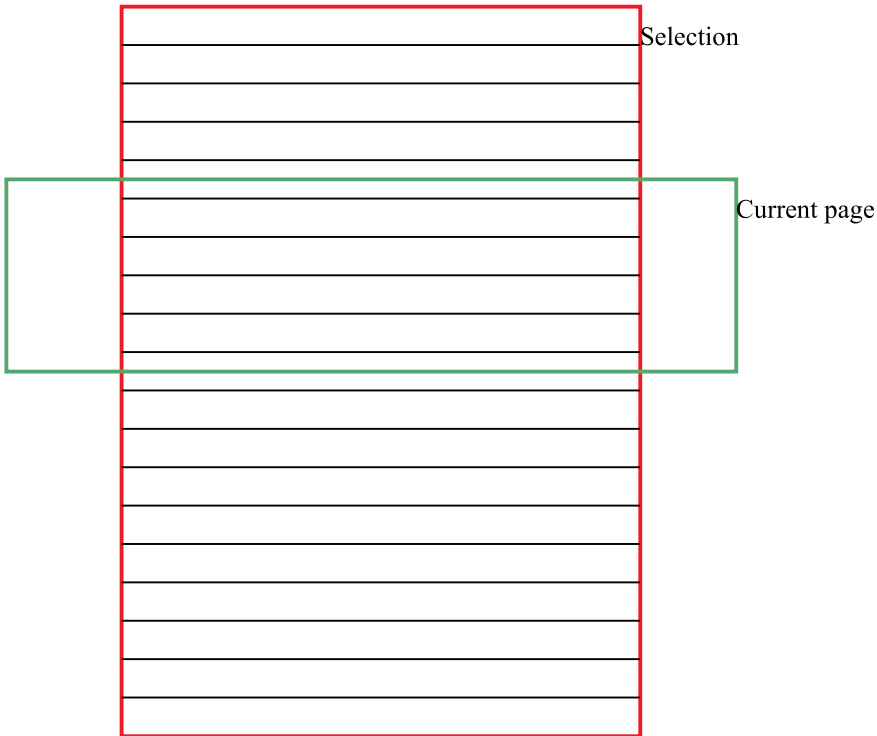
Let's examine the SQL statement that is generated to read one page. Here we define a list which criteria is based on empno >= X and that is sorted by LastName.

```
SELECT columns INTO fields FROM EMPLOYEE T1
WHERE
/* PROCESSING EXTRACT CRITERIA
T1.EMPNO >= :EMPLORO-WBEGIN-KEY.EMPNO
AND T1.EMPNO <= :EMPLORO-WEND-KEY.EMPNO

/* PROCESSING SCROLLING
AND
( T1.LASTNAME > :EMPLORO-WNEXT-KEY.LASTNAME
OR T1.LASTNAME = :EMPLORO-WNEXT-KEY.LASTNAME
AND ( T1.EMPNO > :EMPLORO-WNEXT-KEY.EMPNO
OR T1.EMPNO = :EMPLORO-WNEXT-KEY.EMPNO ) )
```

Let's see in a schema the details:

The page (green form) is a subset of the red form. It is defined by what



defines the red form, plus a starting point that needs to be unique. This

starting point must thus include the key. But the key is not enough. It needs to take into account the order too, as a higher and equal clause is issued. The page is defined by the combination of the key and the sort criteria.

Note: The generators are using an API that adds the key to the sort field if needed.

To sum-up, **the extraction criteria are used to define the selection data. The sort criteria are used to define the page data.**

Resource Object: An Access Part: As the accesses are not integrated into the Business Object, it is possible to generate a Business Object that includes more or less 4GL language. On the one hand, a Business Object could offer a pure Smalltalk implementation, independent from the technology that accesses the data. On the other hand, the Business Object can be coded in 4GL language.

The coding of the Business Object is independant from the coding of the Resource Object. We don't have to use 4GL language in the Business Object.

Detail Access

To manage detail accesses, the Resource Object API verbs take a Business Object as a parameter. The Business Object data are used during the server call and updated on the return from the server. Depending on whether the server action succeeds or fails, the Business Object is either modified or left untouched.

List Access

To manage list accesses, the Resource Object reads from the server a collection of Business Objects. The Resource Object API offers services for reading a data page according to extraction and location criteria, which locate the page in a selection.

Updatable List Access

The resource Object gets as parameters collections of movements. It calls the server with the corresponding data. It answers a collection of Business Objects whose movement failed.

Instance Management: The instance manager offers an API to register, unregister a persistent object given its class and its unique key. This API is used by all the action that has to handle persistent Business Objects.

The storage of the instances is based on Weak Collections. These particular Smalltalk objects are able to remove (thanks to Smalltalk's garbage collector) any entry in the collection that is not known by any object.

Services:

Error Handling:

Standard Errors

- Error Definition

The error is an object (class Error) that contains the following information:

- Error code
- Gravity
- Ordered Collection of variables
- Reference to the logical field
- Etc.

The error includes a reference to the field, which makes it possible to retrieve it and perform a feedback.

- Error Window

The error window is an object (class ErrorView) able to display one or several errors. It means that it will accept a collection of errors, compute them (translating the codes and variables to a text), handle the feedback on the related graphical fields when double-clicking on a line.

The error window should be unique for the whole application, so that it can be cleared when a new action is performed and collect one or several errors, from one or several sources. Thus, the error window is stored in a class attribute.

- Raising an Error

The error raising is also offered through the class API of the class CommonServices. The same kind of mechanism applies to the system errors. An error can be raised by an action or through a customization.

- Checks in a detail

We want to provide the fields of a detail with the following functions:

- Check on format
- Check on input in required fields
- Check on data element
- Inter-field check (once all the fields have been checked).

Error feedback varies depending on whether the end-user is inputting or validating data: in the first case (the user types in the detail), the field color changes, in the second case (the user asks for a check), an error (object) is built and explains the nature of the problem.

- How to implement these behaviors ?

Using converters that may be enriched via subclassing allows implementation of the requested checks. However, this means that it is not possible to dissociate checks on format, checks on required fields, and checks on data elements. Furthermore, algorithmic checks on data

elements have to be implemented via the creation of a new converter type, which implies generating a new class. Standard converters can not be used in this case.

We've also seen that the result of the check is not the same during an input or a validation: we want to control the activation of checks in order to raise or not raise an error in case the checks fail.

A dedicated part is thus necessary.

- How does it work ?

Field control is provided at 2 levels:

- At the data-element level: The controls are described in the model at the data-element level. A method is generated in a class dedicated to that purpose to perform the described check. This method is a reusable piece of code,
- At the Business Object level. Each field belongs to a business object; the business object is thus responsible to know how to check its fields. The default behavior of this checking is to call the data-element method. It is thus possible to override the data-element check or to add some Business Object specific ones for one field.

The checks are implemented on the logical layer, in a reusable way, but are able to send feedbacks to the graphical layer.

The Error detection is implemented in a generic way. A class, called `ErrorHandling`, is responsible to define the error detection and raising policy. This part handles a collection of generic objects, the `Fields`, that gives the relevant information to enforce the error policy: a pointer to its graphical widget, whether the field is a key, whether it is read-only, what method controls it in the Business Object, etc.

Thus, the error handling is handled by:

- The initialization of the `Field` objects, when the detail layout is created,
- The creation of an instance of error handling, linked to the instance of the detail subview and its current business object. This error handling is initialized by the collection of fields and has a pointer to the currently displayed business object.

A series of callbacks will trigger methods of the error handling when a check is requested on the whole business object or any individual field.

- Other functions

The Error Handling mechanism offers an API to dynamically display as read-only or updatable one given field.

The error handling knows all the graphical fields (widgets) displayed in a detail and the current displayed Business Object. We've seen that a key defines a persistent Business Object.

Therefore, a persistent Business Object's keys should not be modified through the interface. However, a volatile one should. The error handling is responsible for the following behavior: it protects the key fields of a persistent Business Object and unprotects the key fields of a volatile Business Object.

System Error

System errors are simpler than standard errors. They are defined by the **SystemError** Class. Only one system error can be raised at a time. A System error Window (**SystemErrorView**) is stored in a class attribute of **CommonServices** and opens when the **CommonServices** system error API is used.

Messages:

Raising a message on the server

Messages can be raised in the hooks of the server. They are stored in a specific section or the error record.

Messages are caught on the client. They are displayed in the optional information bar.

Raising a message on the client

If you specify the workspace parameter (parameter, page 2) "information messages" with the value "bar with standard messages", the messages from the server will be displayed in an information bar. If the server has raised no message during an action, the client will raise a message indicating that the action has succeeded. This message is reset when another action starts.

If you specify "information messages" with the value "no standard messages"; the client will never raise any message. The information bar is not displayed at the bottom of the windows. However, the API to raise, propagate and translate messages is still present and can be used.

Foreign Key Help List:

As a Business Object

Help list services are generated as a special kind of Business Object. The classes are named after the target table's name. One item of a help list comes from one instance of Business Object. The help list is handled in a List Manager.

Additional fields

By default, help lists are built from the key of a table. It is possible, though, to specify an additional field. This field will be read by the server,

stored in the corresponding Business Object and handled by the list manager. However, the detail subview will not display it. This needs to be done by customizing the detail subview.

Sharing Data: This part is a place of customization, but is also used in standard to share some pieces of data.

Window list

The sharedPart instance is accessible from every window. Therefore, it is used to register the list of opened windows. This list is then used by each window to build a menu. This menu displays a button for each current opened window of the application.

Help Lists

A help list contains data from the keys of one table. It is possible that this table would be the target of several foreign keys in several tables. However, the list of the keys would be the same, so it makes sense to share it.

Therefore, the help list's data (list manager) is stored in the sharedPart.

Subviews and Views:

Detail:

Displaying a Business Object

The current instance of Business Object is stored in a variable. This variable is never set to nil.

Several actions may answer another instance of Business Object. Then the result of this action is stored in the Variable.

The Business Object's public interface describes several events. It is important to fire these events on the current instance of Business Object and not the returned one, since, during an action, it's the current Business Object that is displayed and not the result one.

Error Handling

The detail is the place where the error handling is plugged. It is initialized from the detail's fields and knows the current instance of the Business Object.

Help List

When the detail contains a foreign key field that is candidate for a help list, then this field is displayed as a combo-box and the help list manager is accessed through a variable. This variable will know the corresponding instance of list manager stored in the shared part, through promotions.

List: A list displays the data stored in a list manager. It displays thus a collection of Business Objects. Each list contains its instance of list manager.

However, it is possible to share these instances. To make this customization easy, the list manager is stored in a variable. List can display their data in 3 ways:

- By displaying one page after another,
- By displaying all the read pages,
- Through the auto-scrolling mechanism.

The first and second choice is made by displaying one collection of the list manager or another. The third choice involves the triggering of the scrolling mechanism on the list manager.

TUI Client

This section details the structure of the different components involved in a TUI client.

Main Program (View):

Components Generated by Entities

Components Generated from a Data Element

The components stemming from a Data Element are generated to implement specific checks on fields other than the global checks generated from a Business Object. These checks need the generation of the following:

- one Data-Item,
- one function,
- **GUI** one Java/Smalltalk Class for all the Data Elements supporting one Java/Smalltalk Method per Data Element.

The VisualAge Generator parts are prefixed with the target name of the Data Element; the Java/Smalltalk class with the Workspace target name.

GUI and TUI Components

Function:

Traceability Category: INTERNAL

The generated function is responsible for managing server checks on fields. There is one function generated per Data Element instance, provided that a check has been specified in VAGTemplates. It takes a Data-Item as parameter.

For example, assuming you defined a numeric Data Element - DEPT - and specified a check by value table, the DEPTPE-CTRL function will be generated to implement the defined control on the server.

Note: You can use and modify this part to define additional checks on fields, for example. This part should not be deleted.

GUI Components: Java/Smalltalk Class

Traceability Category: API

The generated `<>DataElementChecks` Java/Smalltalk Class holds methods that manage the client checks on fields defined for the Data Elements (check by interval, check by value table). These methods are called by the Business Object part. There is one method per Data Element.

Note: The traceability category of all the methods provided by this class is RAD.

For information on the Business Object bean/part, refer to “*Business Object Bean/Part*” on page 304.

Components Generated from a Business Object

The components generated from a Business Object’s specifications are the following:

- WEB components,
- Functions,
- Records,
- Server Programs,
- **GUI** Visual components
- **GUI** Non-visual components.

These components are prefixed with the target name of the Business Object.

WEB Components

The UI Record UI-KEY: This record is never displayed. It is used as a parameter for the paging in list (in case of Xfer).

GUI and TUI Components

Functions:

Traceability Category: INTERNAL

A function is generated for each action. Each function calls:

- I/O Functions for performing the corresponding action;
- Functions for managing the NULL value;
- Functions for performing global checks, like required field checks.

Caution: Functions are generated for internal purposes only. They should not be used or deleted. They should not be modified unless you wish to customize SQL accesses.

Records:

Traceability Category: API

A Business Object is defined by several Records. These Records are Data Records and are defined for one Business Object. They reflect its composition in fields. They allow access to the Business Object fields through their own data items.

One Record is responsible for storing one type of data.

Mono-Instance Data WINSTANCE Record

This Record is responsible for storing data used in simple accesses and the current data selected during a multiple access.

The WINSTANCE Record is a Working-Storage Record comprising the following Data-Items:

- INSTANCE-FIELDS, describing the Business Object instance's structure in data fields;
 - Data-Items representing the Business Object's fields;
 - Data-Items of lower level are functional and ready to be generated with Java wrappers.
- INSTANCE-NUL-FLAGS, describing the structure of the fields managing the NULL value:
 - N-prefixed Data-Items indicating whether the field is set to NULL (Y) or not (N);
- INSTANCE-CONCURRENCY⁴, grouping the columns used for managing concurrent accesses:
 - C-prefixed Data-Items representing the fields used for managing concurrent accesses.
- INSTANCE-ACTION-CODE, grouping the codes of the actions to be performed on the instance: S (Select), I (Insert), U (Update), D (Delete).

Multi-Instance Data

WPAGE Record

This Record is responsible for storing read page data used in a multiple access.

4. This Data-Item is only generated when a column has been defined for concurrency management in the Workbench.

The WPAGE Record is a Working-Storage Record comprising the following Data-Items:

- INSTANCE-FIELDS, describing the Business Object instance's structure:
 - Data-Items representing the Business Object's fields;
- INSTANCE-NULL-FLAGS, describing the structure of the fields managing the NULL value:
 - N-prefixed fields indicating whether the field is set to NULL (Y) or not (N);
- INSTANCE-CONCURRENCY⁴, grouping the columns used for managing concurrent accesses.:
 - C-prefixed Data-Items representing the fields used for managing concurrent accesses.
- INSTANCE-ACTION-CODES, grouping the codes of the actions to be performed on the instance:
 - INSTANCE-ACTION-CODE: S (Select), I (Insert), U (Update), D (Delete);
- CONTROL-COUNT-INSTANCES, indicating whether or not instances in the selection must be counted.
- CONTROL-COUNTER, indicating the number of instances to calculate the scrollbar for the auto-scrolling function.
- CONTROL-RETURNED-ROWS-NUMBER, indicating the number of lines read in the page;

Extraction Data WBEGIN-KEY Record

This Record is responsible for storing the data selected according to extract criteria during a multiple access.

The WBEGIN-KEY Record is a Working-Storage Record comprising the following Data-Items:

- EXTRACT-CRITERIA, representing the fields in the extract criteria;
 - Data-Items representing the Business Object's fields used for extracting data.
 - Data-Items of lower level are functional and ready to be generated with Java wrappers.
- *GUI* CONTROL-SELECTED-ROW-INDEX, indicating the index of the selected item in the page.

Note: You are allowed to use these Records and modify their contents, but you should not modify their structure or delete them.

Server Programs:

Traceability Category: API

Two-tier Architecture: If you specified a 2-tier server architecture for your application (*Server layers*, Workspace Definition editor, *Server* panel), that is one client layer and one server layer, three server Programs are generated from a Business Object according to its presentation as a detail, as a list, or as an updatable list:

- a Mono-Instance Program, suffixed with 1⁵, which allows access to the data presented in a detail Business Object;
- a Multi-Instance Program, suffixed with N⁵, which allows access to the data presented in a list Business Object.
- an Updatable Multi-Instance Program, suffixed with L⁵, which allows access to the data presented in an updatable list Business Object.

Note: Not all these components are generated. Their generation depends on the value set for the *Service level* parameter (Business Object Generation Parameters editor, *Optimization* panel).

Actions in the server are implemented as follows:

1. Context initialization (Record initialization, error context initialization, etc)
2. Hook executed before any other function
3. SQL command
4. Error treatment
5. Hook executed after the other Functions

Note: You can use these components. They should not be modified or deleted.

Three-tier Architecture: If you specified a 3-tier server architecture for your application (*Server layers*, Workspace Definition editor, *Server* panel), that is one client layer and two server layers, an umbrella server and atomic servers are generated.

1. Detail Business Object

One umbrella server is generated per *detail Business Object*. The umbrella server does not access the database. It calls atomic servers to perform the accesses. When possible, the umbrella server calls the atomic server generated from the Relational Table. When it is not possible, atomic servers are generated from the Business Objects. The umbrella server calls these atomic servers.

5. This suffix is a constant from the `com.ibm.mdl.Visitors.Services.NamingServices` class (VAGTemplates on Java) or from the `MdIVGMnemos.poolDictionary` (VAGTemplates on Smalltalk).

Note: The Business Object umbrella server uses the atomic servers generated from the Relational Table or those generated from the Business Object, depending on several criteria (mostly the mapping) and on the action to perform:

- If the Relational Table has a primary key, the atomic servers are generated from the Relational Table. There is one atomic server generated per action (create, read, update, delete, read keys, etc...).
- If the Relational Table has no primary key, the atomic servers are generated from the Business Object. There is one atomic server generated per action (create, read, update, delete, read keys, etc...). In this case, the Multi-Instance Server Program is used as an atomic server.

Select action

- One atomic server for the selection is generated from the Business Object.

Create action

- The atomic server generated from the Relational Table is always used. The columns that are not mapped by the Business Object are initialized to NULL.

Update action

- The atomic server generated from the Relational Table is always used if the Business Object maps to all the Relational Table's columns. Otherwise, the atomic server generated from the Business Object is used.

Delete action

- The atomic server generated from the Relational Table is always used if the Business Object maps to the Relational Table's primary key. Otherwise, the atomic server from the Business Object will be used.

2. List Business Object

One umbrella server and one atomic server are generated per *list Business Object*.

3.

In 3-tier architecture, actions in the server are implemented as follows:

- a. Part of the context initialization is made at the umbrella server level before calling the atomic server.
- b. Hook executed before the atomic server call
- c. Call to the atomic server
- d. Hook executed before any other function
- e. SQL access
- f. Error treatment
- g. Hook executed after the Functions

- h. Hook executed by the umbrella server after return from the atomic server

Note: You can use these components. They should not be modified or deleted.

GUI Components

Visual Components:

Traceability Category: API

There is one Visual bean/part generated per type of Business Object used:

- one Subview bean/part for the Business Object used as a detail,
- one Subview bean/part for the Business Object used as a list,
- one Subview bean/part used for the Business Object used as an updatable list.

The Subview comprises the graphical components of the Business Object (graphical fields, menus, push-buttons) and their connections to data, and the menu items or push-buttons that trigger actions. It implements the filling of the graphical fields with the data from the Business Object bean/part.

Detail Subview bean/part: The Detail Subview bean/part presents the Business Object *via* connections to the Business Object's attributes (its fields). These attributes are accessed through a variable based on the Business Object.

The elementary actions (*Create, Read, Update, Delete*) are activated *via* connections to the variable based on the Business Object. The *Check* action is triggered *via* the Business Object bean/part.

The Detail Subview also present help lists (when they are generated) and activates paging actions on this list *via* connections to the variable based on the List Manager for the help list.

For information on the Business Object bean/part, refer to "*Business Object Bean/Part*" on page 293. For information on the List Manager bean/part, refer to "*List Manager bean/part*" on page 297.

List Subview bean/part: The List Subview bean/part contains a graphical data container and paging actions (appearing as menus or push-buttons). This part implements the filling of the list with the data from the List Manager bean/part. The List Subview activates paging actions on the list *via* connections to the variable based on the List Manager.

Note: You can use or modify this part. It should not be deleted.

For information on the List Manager bean/part, refer to topic “*List Manager bean/part*” on page 297. For information on paging actions, refer to topic “**Actions Available for List Business Objects**” on page 148.

Updatable List Subview bean/part: The Updatable List Subview bean/part contains a graphical data container and paging and elementary actions (appearing as menus or push-buttons). This part implements the filling of the list with the data from the List Manager bean/part. The Updatable List Subview activates actions on the list *via* connections to the variable based on the List Manager.

Note: You can use or modify this part. It should not be deleted.

For information on the Updatable List Manager bean/part, refer to topic “*Updatable List Manager bean/part*” on page 300.

Non-Visual Components: According to the type of generated Business Object, the generated beans/parts are:

- **<>BusinessObject**
- **<>ListManager** (if the Business Object is used as a list)
- **<>UpdatableListManager** (if the Business Object is used as an updatable list)
- **<>ResourceObject**

Note: We will call these components the Business Object bean/part, the List Manager bean/part, the Updatable List Manager bean/part, and the Resource Object bean/part respectively.

Business Object Bean/Part:

Traceability Category: API

Introduction: There is one Business Object bean/part generated per Business Object instance. This bean/part encapsulates the Business Object’s data and the actions that manage the data (create, read, update, delete).

The Business Object bean/part calls the ResourceObject part to trigger the actions.

The Business Object bean/part is responsible for performing global checks any time a creation action is triggered. It calls the ErrorHandler part by sending an event to perform unitary checks on fields, checks on required fields, and possible inter-field checks.

The Business Object bean/part inherits from a class generated at the Workspace level: <>**BusinessObjectAbstract**. This class is documented in “**Abstract Business Object**”.

Caution: This component should not be used, modified, or deleted.

For more information on checks, refer to topic “**Error Handling in GUI Client applications**” on page 153.

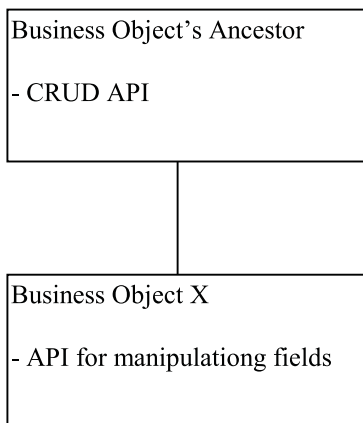
For information on the Resource Object bean/part, refer to “*Resource Object bean/part*” on page 301. For information on the Error Handling part, refer to “*ErrorHandling Part*” on page 320.

Smalltalk Business Object Instance API:

Note: Following information relevant with Smalltalk only. Java-related information will be available in a future edition.

Public methods and the corresponding public interface, which allows the object to be manipulated via connections, characterize the Business Object instance API.

Inheritance: Thanks to inheritance, processing that can be factorized (generic processing) is defined at the ancestor level.



Abstract Business Object:

Genericity

The actions defined by the abstract Business Object are the generic ones: These actions are the ones that do not need to manipulate the Business Object’s fields. It is also possible to define as virtual the whole setting of these fields, that would be implemented at the Business Object’s level.

Method description - Actions_API

	State before	State after (success)	Object after (success)	State after (failure)	Object after (failure)
Read	Volatile	Persistent	Same or different	Volatile	Same
Refresh	Volatile / Persistent	Persistent	Same or different	Volatile	Same
Create	Volatile	Persistent	Same	Persistent	Same
Update	Persistent	Persistent	Same	Persistent	Same
Save	Volatile / Persistent	Persistent	Same or different	Initial State	Same
Delete	Volatile / Persistent	Volatile	Same	Same	Same
Reset	Persistent	Persistent	Same	NA	NA
New Instance	Volatile / Persistent	Persistent	Different	NA	NA

These actions should never answer nil.

Note: Reset is implemented at the Business Object's level, and explained later.

check

Performs the client checks on self. To do so, the field that has the focus is validated and all the updatable fields are checked, through the ErrorHandling mechanism. Errors are displayed if detected. This action does not change the state of the Business Object.

read

self is volatile. read searches if an instance with the same key exists in the instance dictionary. If such an instance exists, this action returns it. Otherwise, it calls the Read action from the Resource Object. If this action succeeds, self is stored in the instance dictionary and becomes persistent.

refresh

self is volatile or persistent. refresh calls the Read action from the Resource Object. If this action succeeds, self is stored in the instance dictionary and becomes persistent. If it fails, self becomes volatile and it removes the instance of same key stored in the instance dictionary.

The difference between read and refresh is that refresh always reads data from the DB, even if a persistent instance already exists for that key.

create

self is volatile. create searches if an instance with the same key exists in the instance dictionary. If such an instance exists, an error is raised. Otherwise, this action calls the Create action from the Resource Object. If this action succeeds, self is stored in the instance dictionary and becomes persistent.

update

self is persistent. update calls the Update action of the Resource Object.

save

self is either volatile or persistent.

If self is volatile, save calls the Create action of the Resource Object. If this action succeeds, self is stored in the instance dictionary and becomes persistent.

If self is persistent, this action calls the Update action of the Resource Object.

delete

self is either persistent or volatile. delete calls the Delete action of the Resource Object, and removes the instance corresponding to the Business Object key from the instance dictionary. At the end, self is volatile.

newInstance

newInstance returns a new volatile instance initialized with the Business Object's default values.

Method description - Hooks_API**additionalChecks**

This method is called by checkBusinessObject. It is a hook that can be redefined if needed in the Business Object class. This method is made to write inter field checking or any additional checking needed in the Business Object.

Method description - Message_API**mdlIsUpdated and mdlIsUpdated: aBoolean**

Setter and getter for the mdlIsUpdated attribute. This attribute is relevant only for persistent Business Objects.

Note: In V3.1, the value is set through the setters of the Business Object and reset when the data comes from the database.

mdlState and mdlState: aSymbol

Setter and getter for the Business Object's state (volatile or persistent). The default value is volatile.

Method description - Status_Internal (private)

isCopyDisabled and isCopyDisabled: aBoolean

These methods are used in the reset mechanism. They allow to set a flag that says whether, when a field setter is called, a copy of the original field should be made or not. Typically, when the database fills the Business Object's data, a copy is undesired. However, when the end-user types a value in a detail, that copy is needed.

Business Object:

Reset

Reset rolls back the changes that have been made to the persistent Business Object. After a reset, the Business Object's field are reset to the value that has been read from the Database.

Two instance variables are generated for each field : a flag to monitor if the field has been changed and a copy of the data : testing if the copy's data is not enough, since nil is a possible value for a field.

Field setters work in two modes. These modes are available only for persistent Business Objects.

- One mode where the original value is copied if needed (when the user changes the contents of the Business Object). This is the regular mode.
- One mode to set the value. The copy and the flag are set to nil. This happens when the Resource Object is filling the Business Object.

Toggling from one mode to another is made at the Resource Object level.

Due to this mechanism, it is possible to precisely know whether a Business Object has been modified since it has been read from the DB. This is the isModified method. Its algorithm is as follows:

- Check if the field's flag indicates that a change has occurred. If so, compare the value of the copy and the current field, to see if they are the same or different.
- Exit once a change has been found.

List Manager bean/part:

Traceability Category: API

Introduction: The List Manager bean/part manipulates collections of data. It manages the paging actions and memorizes the read pages. It calls the Resource Object bean/part to trigger actions and retrieve data.

The same part is used to manage actions available on help lists. However, it is generated from the Relational Table whose mono-field primary key is linked to the foreign keys of the Business Object's primary table.

The List Manager bean/part inherits from a class generated at the Workspace level: <>**ListManagerAbstract**This class is documented in “Abstract List Manager”.

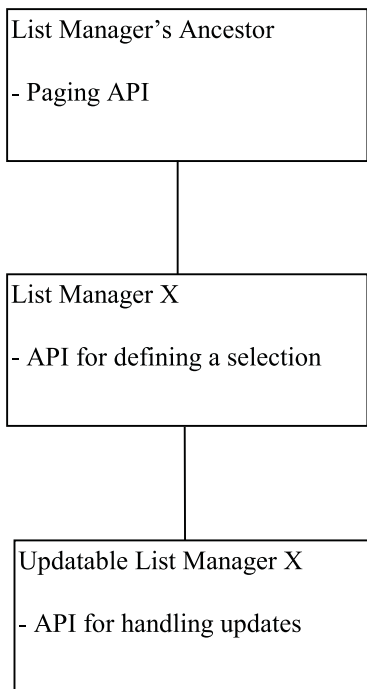
For information on the Resource Object bean/part, refer to “Resource Object bean/part” on page 301.

Note: You can use this part. It should not be modified or deleted.

Inheritance:

Note: Following information relevant with Smalltalk only. Java-related information will be available in a future edition.

Thanks to inheritance, processing that can be factorized (generic processing) is defined at the ancestor level.



Abstract List Manager:

Note: Following information relevant with Smalltalk only. Java-related information will be available in a future edition.

Genericity

The actions defined by the abstract List Manager are the generic ones: These actions are the ones that do not need to manipulate the List manager's criteria. It is also possible to define as virtual the whole setting of these criteria, implemented at the Business Object's level.

Collections

The List Manager holds the Business Object's instances in two collections : currentPage and selection.

Selection

contains the full list of Business Objects. It is reset when the (SQL) selection has changed. The pages are gathered in this collection when a new page is read. When one page is refreshed, then the following ones are removed to protect the integrity of the selection.

Current page

is built to be displayed in the lists that display only the current page. It is a subset of the selection collection.

Method description - Actions_API

	Selection (before)	Current page (before)	Selection (after)	Current page (after)
Top	N pages	Page X	1 page	Selection
Extract	N pages	Page X	1 page	Selection
Next (on last page)	N pages	Last page	N+1 pages	Last page
Next (on any other page) (a)	N pages	Page X	N pages	Page X+1
Previous (b)	N pages	Page X	N pages	Page X-1
Refresh	N pages	Page X	X pages	Page X

The rules to fill current page from selection appears clearly from this array. Therefore, we will now focus on the selection collection below and not focus on (a) and (b).

Two actions define a selection : top and extract. Top uses a default selection. Extract uses the current one.

extract

extract defines a new selection for the List Manager's data. Therefore, it resets the sort criteria data (they define which page is going to be read) to ensure that the first page will be read. It reads a page on the server. Then the selection collection is reset with the page contents. The sort criteria are set with the next row's key data.

top

top initializes the extraction criteria to their default values and calls extract.

next

next uses the current extraction and sort criteria to read a page and adds this page to selection.

refresh

Note: refresh is implemented in the List Manager but is described here, for consistency.

refresh uses the current sort criteria and retrieves the sort criteria from the data of the first row of the current page. It reads a page on the server. Then the selection collection is updated with this page and all the following rows are removed: This is because there could be a gap between this page and the next one in memory.

The List Manager bean/part also provides events to be used to indicate the result of the actions listed above:

- refresh succeeded
- extract succeeded
- next succeeded
- previous succeeded
- top succeeded
- refresh failed
- extract failed
- next failed
- previous failed
- top failed

Updatable List Manager bean/part:

Traceability Category: API

The Updatable List Manager bean/part manipulates collections of data. It manages paging and elementary actions and memorizes the read pages. It calls the Resource Object bean/part to retrieve the data

The Updatable List Manager bean/part calls the Resource Object bean/part to trigger the actions.

The Updatable List Manager bean/part inherits from a class generated at the Workspace level: <>**ListManagerAbstract**.

Note: You can use this part. It should not be modified or deleted.

For information on the List Manager ancestor, refer to topic “<>ListManagerAbstract Class” on page 320. For information on the Resource Object bean/part, refer to topic “Resource Object bean/part”.

Resource Object bean/part:

Traceability Category: API

There is one Resource Object generated for all the instances of a Business Object in the application. The Resource Object bean/part encapsulates the server accesses.

When the Business Object is used as a detail, the Resource Object’s API verbs take the Business Object instance as their parameters. The Business Object’s data is used during the server call and updated back from the server.

When the Business Object is used as a list, the Resource Object’s API verbs take a List Manager instance as their parameters. The Resource Object creates and initializes a collection of Business Objects with an access to the list server. The Resource Object API allows the reading of a data page according to extract criteria and according to the location of the page in a selection. This information is provided by the List Manager.

For information on the various servers, refer to topic “*Server Programs*” on page 289.

Note: You can use this part. It should not be modified or deleted.

TUI Component: Additional Server Program

Traceability Category: API

There is one particular server Program generated from a Business Object for TUI applications only: the Data Control Program, suffixed with F⁵, which controls the fields for which a check has been defined and provides an API layer.

This part can implement additional checks on fields by calling the CHK-HOOK Function.

Note: You can use this part. It should not be modified or deleted.

Components Generated from a Relational Table

The components generated from a Relational Table's specifications are used to access data in help lists. These components are the following:

- Functions
- Records
- Server Programs
- *GUI* Java/Smalltalk Classes

These parts are the same as those generated for a list Business Object, but they are generated from the Relational Table whose primary key is linked to the foreign key of the Business Object's primary table.

Atomic servers are generated from the Relational Table to manage actions anytime it is possible (see "*Three-tier Architecture*" on page 290, and "*Three-tier Architecture*" on page 304)

Note: These parts are prefixed with the target name of the Relational Table.

GUI and TUI components

Functions:

VG: INTERNAL

As many functions are generated as there are available actions on help lists, structured according to their roles. Each of these Functions calls:

- I/O Functions for performing the corresponding action;
- Functions for performing global checks.

Caution: These Functions are generated for internal purpose only. They should not be used, modified or deleted.

Records:

Traceability Category: API

The Records generated from the Relational Table are Data Records. They allow access to the primary table's foreign keys. The generated Records are the following:

WINSTANCE Record

This Record is responsible for storing the current data selected in the help list. It comprises the following Data-Items:

- INSTANCE-FIELDS, describing the structure in data fields of the foreign key:
 - Data-Items representing the foreign key’s fields;
- INSTANCE-NULL-FLAGS, describing the fields managing the NULL value in the foreign key:
 - N-prefixed Data-Items indicating whether the field is set to NULL (Y) or not (N).
- INSTANCE-ACTION-CODE, grouping the codes of the action to be performed on the instance: S (Select), I (Insert), U (Update), D (Delete).

WPAGE Record

This Record is responsible for storing the data in the read page of the help list. This Record comprises the following Data-Items:

- INSTANCE-FIELDS, describing the fields of the foreign key;
- INSTANCE-NULL-FLAGS, describing the fields managing the NULL value in the foreign key:
 - N-prefixed fields indicating whether the field is set to NULL (Y) or not (N);
- INSTANCE-ACTION-CODES, grouping the codes of the actions available on the help list:
 - INSTANCE-ACTION-CODE, indicating the code of each action;
- CONTROL-RETURNED-ROWS-NUMBER, describing the fields from the foreign key returned in the help list:

WBEGIN-KEY Record

This Record is responsible for storing the data selected from the help list. This Record comprises the following Data-Items:

- EXTRACT-CRITERIA, representing the structure of the extract criteria;
 - Data-Items representing the foreign key’s fields used for extracting data.
- CONTROL-SELECTED-ROW-INDEX, indicating the index of the selected item in the page.

Note: You are allowed to use these Records and modify their contents. They should not be deleted; their structures should not be modified.

Server Programs:

Traceability Category: API

Two-tier Architecture: If you specified a 2-tier server architecture for your application (*Server layers*, Workspace editor), that is one client layer and one

server layer, there is one server Program generated from a Relational Table. This multi-instance server Program, suffixed with N⁵, allows access to the help list's data.

Three-tier Architecture: If you specified a 3-tier server architecture for your application (*Server layers*, *Workspace editor*), that is one client layer and two server layers, additional servers are generated.

These servers are called atomic servers. They are responsible for performing the database accesses requested by the Business Object umbrella server. There are as many servers generated as actions available for the Business Object (create, read, update, delete, read keys, etc.).

For more information on the 3-tier architecture, refer to "*Three-tier Architecture*" on page 290.

Note: You are allowed to use this component. It should not be modified or deleted.

GUI components: Non-Visual components

Traceability Category: API

The Java/ Smalltalk Classes generated from the Relational Table are key reading components. They offer services for read-only multiple accesses on the table's foreign keys. These services implement input aid on foreign keys in detail Business Objects.

These Classes are the following:

- <>**BusinessObject**
- <>**ListManager**
- <>**ResourceObject**

Note: We will call these components respectively the Business Object bean and Business Object bean/part, the List Manager bean/part, and the Resource Object bean/part respectively in VAGTemplates on Java and in VAGTemplates on Smalltalk.

Business Object Bean/Part:

Traceability Category: API

This Business Object bean/part is generated from the Relational Table whose mono-field primary key is linked to the foreign keys of the Business Object's primary table. There is one Business Object bean/part generated per Relational Table (except the primary table).

This component encapsulates the foreign key data and the actions available for these data. It calls the Resource Object bean/part to trigger the actions and retrieve data. It also performs checks on the foreign key's fields *via* the Error Handling component.

This Business Object bean/part inherits from a class generated at the Workspace level: <>**BusinessObjectAbstract**. This class is documented below.

Note: You can use this component. It should not be modified or deleted.

For information on the Business Object ancestor, refer to “**GUI Components**” on page 317. For information on the Resource Object bean/part, refer to topic “*Resource Object bean/part*” on page 301. For information on the Error Handling component, refer to topic “*ErrorHandling Part*” on page 320.

List Manager bean/part:

Traceability Category: API

The List Manager bean/part manipulates collections of data. It manages the paging actions and memorizes the read pages in the help list. It calls the Resource Object bean/part to trigger actions and retrieve data.

It is generated from the Relational Table whose mono-field primary key is linked to the foreign keys of the Business Object's primary table.

The List Manager bean/part inherits from a class generated at the Workspace level: <>**ListManagerAbstract**.

Note: You can use this component. It should not be modified or deleted.

For Information on the List Manager ancestor, refer to “<>*ListManagerAbstract Class*” on page 320. For information on the Resource Object bean/part, refer to topic “*Resource Object bean/part*”.

Resource Object bean/part:

Traceability Category: API

There is one Resource Object generated for each Relational Table in the application. The Resource Object bean/part encapsulates the server accesses that fill the help list. It creates and initializes a collection of data with an access to the help list server. The Resource Object API allows the reading of a data page according to extract criteria and to the location of the page in a selection. This information is provided by the List Manager.

It is generated from the Relational Table whose mono-field primary key is linked to the foreign keys of the Business Object's primary table.

Note: You can use this part. It should not be modified or deleted.

For information on the List Manager, refer to topic "*List Manager bean/part*" on page 297. For information on the help list server, refer to topic "*Server Programs*" on page 303.

Components Generated from an Interface Unit

The components generated from an Interface Unit's specifications are used to present data and manage navigation. These components are the following:

- *WEB UI Records and Programs*
- *GUI* Visual beans/parts
- *GUI* Non-visual beans/parts
- *TUI* Server Programs
- *TUI* Maps
- *TUI* MapGroups
- *TUI* Tables

WEB Components

The Program: A 'web transaction' program is the backbone of a web application. It is generated from the Interface Unit. Its suffix is E.

The program contains the following three attributes:

- Type: 'Web Transaction',
- Message table prefix: xMSG (ENU table),
- First UI Record (based on the Interface Unit content).

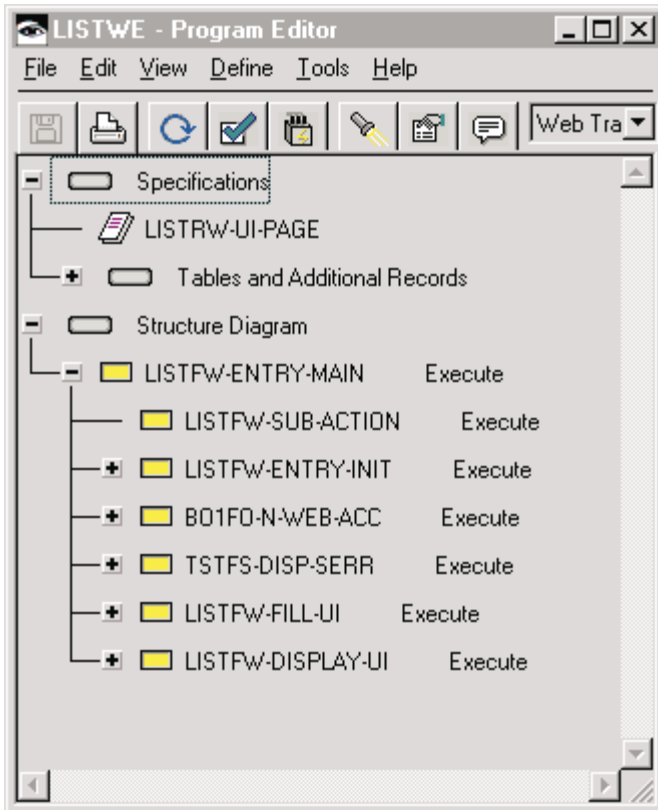
It contains:

- Records & Tables used by the program,
- Functions (according to the architecture).

Its main function (ENTRY-MAIN) calls sub-functions performs the following actions:

- Analyze the submitted action (SUB-ACTION),
- Initialize internal data (according to the submitted action) (ENTRY-INIT),
- Server access(es) (WEB-ACC),
- System Error management (DISP-SERR),
- Fill the UI Record from server data (FILL-UI),
- Display the UI Record (DISPLAY-UI).

Example:



The way retrieved data is managed at the web server level varies with the two provided transfers (Xfer and Converse).

In the case of a Converse, since all the retrieved data is kept (in memory), they can easily be manipulated. On the contrary, the Xfer mode does not keep much data. As a result, special items need to be used to make copies of what the application needs to continue.

How Xfer works

The function (EXECUTE) displaying the UI Record in the web browser is defined as follows:

```
EZEAPP = ' ';  
XFER EZEAPP , xxxxRW-UI-PAGE;
```

Where xxxxRW-UI-PAGE is the first UI Record.

The UI Record UI-PAGE: This record is displayed in the browser by the application.

Definition

- Record Type: 'User Interface',
- Title (displayed in the browser),
- Submit value item (contains the submitted action).

Content

- Data displayed in the interface,
- Non-displayed data (selected list index),
- Xfer: Hidden data (next key data, copy of the help list; etcà),
- Submit items ('none' UI Type).

Use of the 'First UI Record'

When a program has data that may be manipulated (updated or just displayed), we define a first UI Record in order to be able to prefill useful data during the call of the program.

GUI Components

Visual bean/part: View bean/part:

Traceability Category: API

The View bean/part generated from an Interface Unit contains one or more Business Object Subview beans/parts and the beans/parts of possible child Interface Units. It manages data navigation among these Subviews and the intrawindow zoom in function.

It also includes the root of the embedded parts' menus, the *Help* and *Edit* menus, the *Windows* menu. The *Navigation* menu is always generated except if the root Interface Unit does not call child Interface Units.

The View part is prefixed with the target name of the Interface Unit.

For information in the intrawindow zoom in function, refer to topic "**Automatic Zoom in (GUI)**" on page 152.

Note: You can use, modify or delete this bean/part.

Non-Visual Components: There is one Java/Smalltalk class generated from the specifications of the root Interface Unit: <>**SharedComponents**.

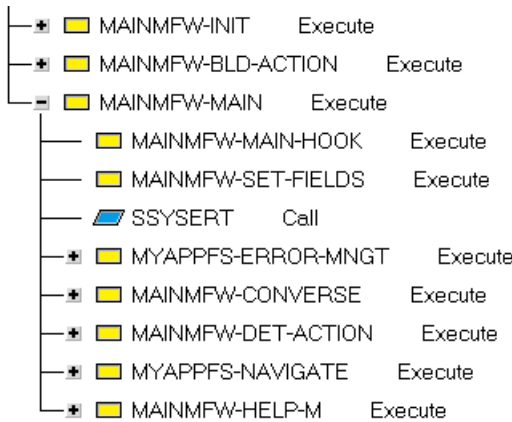
This class contains the error windows and the data that are shared by the windows in the application, like help lists. It also manages the list of open windows in the application (see “*WindowsMenu Bean/Part*” on page 318).

Note: The name of the class is prefixed with the name of the root Interface Unit.

TUI Parts

Programs: There is one Program generated per Interface Unit whatever the number of maps required to display the different Business Object calls. This Program manages the whole application.

The generated Program’s structure is the following:



Functions:

Traceability Category: INTERNAL

The generation of an Interface Unit instance produces the standard VisualAge Generator Functions used to manage map actions, data accesses, help accesses, etc.

Caution: These parts should not be used, modified or deleted.

MapGroup:

Traceability Category: INTERNAL

There is one MapGroup generated per Interface Unit as the MapGroup must contain all the Maps used by a Program.

Note: You cannot create or delete a MapGroup as it is automatically created when you define your maps, and deleted when you delete the Maps.

Map:

Traceability Category: INTERNAL

There are at least three Maps generated from an Interface Unit:

- one Header Map
- one Trailer Map
- one "data" Map.

If the *Messages display* parameter - Workspace editor - is set to *specific map* one Application Error Map is generated.

If the *Display popup* parameter - Workspace Definition editor - is unchecked, one Help Map is generated.

If the application includes a help list, one Help List Map is generated.

For information on these Maps' layout, refer to "**Standard Functions**" on page 146.

Application Error Map:

Traceability Category: INTERNAL

The Application Error Map is generated from the Interface Unit's specifications. It is filled in with the data from the WERROR-LIST Record (see "*Error Data Records*" on page 315).

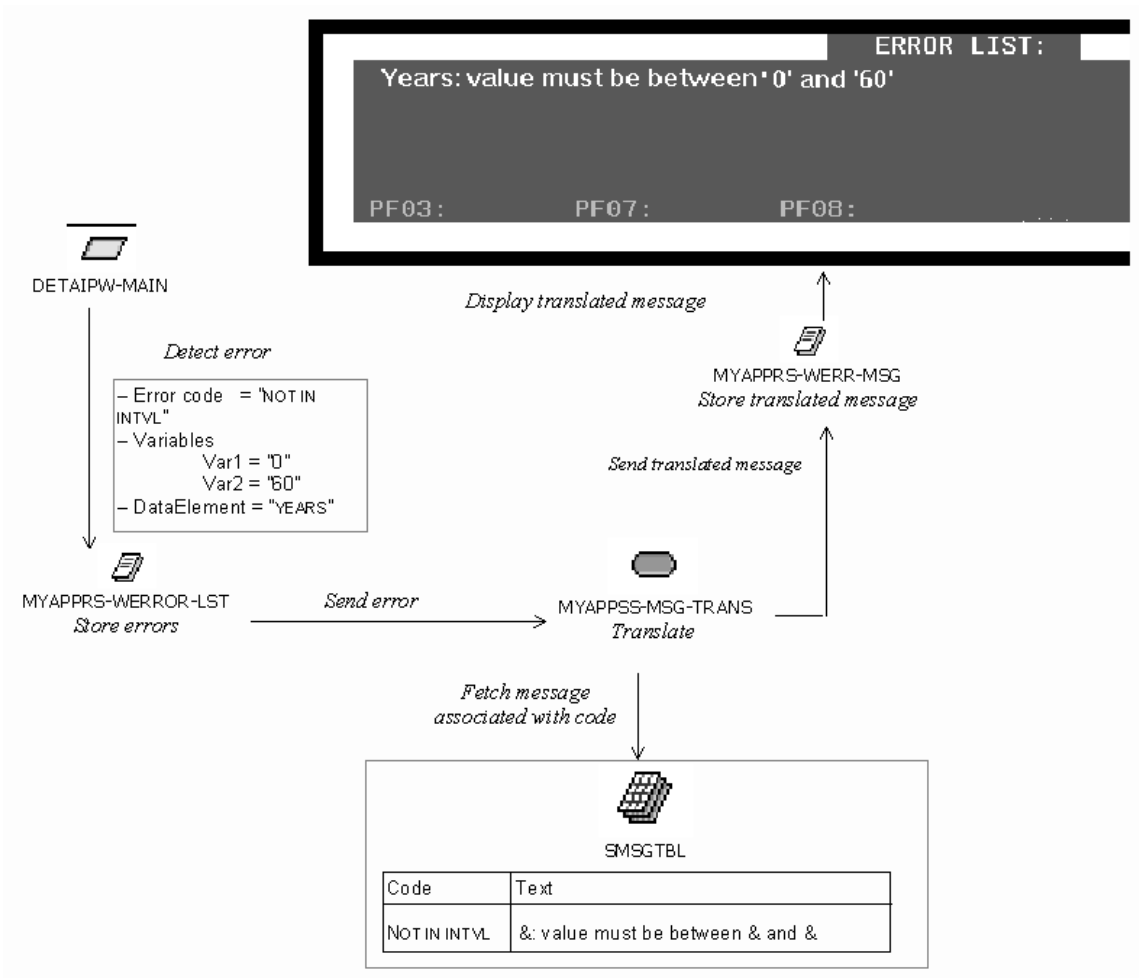


Table: The generation of the Interface Unit produces one Table that contains the action codes. The generation of Interface Unit with the Interface Unit Help TUI generator produces a Table that contains the help text.

Action Code Table:

Traceability Category: Hook

The generated action code Table is suffixed with TA. This Table stores the action keys available for each map. It comprises the following Data-Items:

- ACTION-DESCRIPTION stores the action code which can be a function key or a character;

- ACTION-ON-PFKEY stores the character that indicates whether the action code is a function key (F) or a character (blank character)
 - PFKEY-NUMBER stores the action code. It stores a 2-digit number if the action code is a function key or one character if the action code is a letter.
 - ACTION-TYPE stores one character that indicates whether the action is a navigation action (N) or an application action (A), i.e. an application that manages data in the application;
- INTERFACE-UNIT-ACTION stores the label of the function key or action code.

Note: The function key numbers and labels are Workspace parameters. The other action codes are constants defined in the `MdlInsert_Mnemo`, `MdlSave_Mnemo`, `MdlUpdate_Mnemo`, and `MdlDelete_Mnemo` poolDictionaries.

Help Text Table:

Traceability Category: RAD

The generated help text Table is suffixed with TH. This Table stores the help text related to each entity used in the generated Interface Unit. It comprises the following Data-Items:

- HELP-TYPE stores the type of the Information Model entity to which each help paragraph is related (Interface Unit, Data Element, etc.). These types are the entity mnemonics specified in the Workspace;
- HELP-IDENTIFIER stores the identifier of the entity to which each help paragraph is related;
- HELP-TEXT stores the help text you entered in the *On-line help description* field when specifying your instances in VAGTemplates.

Note: You are allowed to use these Tables and modify their contents. They should not be deleted; their structures should not be modified.

For information on GUI on-line help, refer to “**GUI On-Line Help**” on page 160, and on TUI on-line help, refer to “**TUI On-Line Help**” on page 163.

For information on the available actions, refer to topic “**Standard Functions**” on page 146.

Components Generated From a Workspace: Predefined Beans/Parts

The components generated from a Workspace’s specifications can be used by all the client and server components in the same application. We call these components *Predefined beans/parts*.

Predefined beans/parts need to be generated once for one Workspace, with the first generation of your application. If you re-generate your application within the Workspace, you will not have to re-generate them. They focus on error handling.

These parts are the following:

- Tables
- Functions
- Records
- **GUI** Visual beans/parts
- **GUI** Java/Smalltalk Classes
- **GUI** Non-visual beans/parts
- **TUI** Maps
- Web records

GUI and TUI Components

Table:

Traceability Category: Hook

There are two application error message Tables (SMSGTBL and SUSRTBL), and four system error code tables generated (SERRCRE, SERRSEL, SERRUPD, and SERRDEL).

Error Message Tables:

- SMSGTBL Table

This Table stores the error message associated with the SQL error message stored in the error code tables. This Table comprises the following Data-Items:

- ERR-MSG-CODE, describing the code of the error;
- ERR-MSG-TEXT, describing the error message associated with the detected error.

- SUSRTBL

Table This Table is a user Table. You can use it to customize or add labels, associated with error codes, if you need error messages in a language other than English, for example. This user Table has the same structure as the standard SMSGTBL Table.

Note: You can use the SMSGTBL Table and modify the error messages directly in this Table, but you should not modify its structure or delete it. You can use and fill the SUSRTBL Table, but you should not delete it.

Error Codes Tables: The error code tables store the SQL error message associated with the SQL error raised.

The SERRCRE Table stores the SQL codes emitted when a system error has been detected during a creation action.

The SERRSEL Table stores the SQL codes emitted when a system error has been detected during a read action.

The SERRUPD Table stores the SQL codes emitted when a system error has been detected during an update action.

The SERRDEL Table stores the SQL codes emitted when a system error has been detected during a delete action.

Note: The error code Tables have all the same structure. We only describe the SERRCRE Table here.

- SERRCRE Table

This Table comprises the following Data-Items:

- SQL-CODE stores the SQL error code returned by the system;
- ERR-MSG-CODE stores the code of the error message associated with the SQL error code.
- WITH-LOCATION indicates whether or not the error is related to a key field and therefore whether or not the focus must be put back on this field.

Note: The WITH-LOCATION Data-Item is not used for TUI applications.

Functions:

Traceability Category: RAD

For an application, there are six Functions related to six possible server actions generated from a Workspace.

The generated error Functions search for the SQL error code returned by the server in the error code Tables, and raise the corresponding error if the error code exists in the table, otherwise they return an unknown error.

There is one Function per available action that fetches the error code in the associated error table:

- error during the read action: SEL-ERR associated with the SERRSEL Table;
- error during the create action: ADD-ERR associated with the SERRCRE Table;

- error during the update action: UPD-ERR associated with the SERRUPD Table;
- error during the delete action: DEL-ERR associated with the SERRDEL Table;
- error during a SETINQ: SETINQ-ERR associated with the SERRSEL Table;
- error during a SCAN: SCAN-ERR associated with the SERRSEL Table.

Records: The Records generated from the Workspace are Data Records. One of them is dedicated to communication between client and server components (WCOMM), the other two are dedicated to storing errors (WERROR-LIST and WSYS-ERROR).

Communication Record:

Traceability Category: Hook

The WCOMM Record is shared among all the server components. It is generated with one Data-Item: CONTROL-COMMUNICATION. This Record is empty when generated so that you can customize it.

This Record is generated if the *Common area* parameter is set to *record and table* or *record* (Workspace editor). If the *Common area* parameter is set to *record and table* a table is also generated that can be used by the Record (see “*Communication Table*” on page 321).

Note: You can use this Record, you can modify and enrich it with data that must be shared and transferred to the server. This part should not be deleted.

Error Data Records: These Records are used by all the Business Objects in the application. There is one Record per data type:

Application Error Data WERROR-LIST Record

This Record is responsible for storing application errors. It comprises the following Data-Items:

- APPLICATIVE-ERRORS describes application errors;
 - ERRORS stores the error structure;
 - LOCATION-DATA-ELEMENT stores the name of the Data-Element that is the source of the error;
 - LOCATION-BUSINESS-OBJECT stores the name of the Business Object in which the error has been detected;
 - LOCATION-PART-TYPE stores the type of the erroneous Business Object’s layout: D, for detail, L for list;

- LOCATION-ROW-INDEX stores the index of the erroneous row in a list;
- ERROR stores the error message structure;
 - ERROR-CODE stores the error code;
 - ERROR-GRAVITY stores the error gravity which indicates whether the associated error message will be an error message, a warning message or an information message (this Data-Item is not used in the standard generated applications);
 - ERROR-VARIABLE1, 2, etc. stores the variables involved in the error (this number is a Workspace parameter: *Max number of variables*, Workspace editor);
- ERRORS-NUMBER stores the number of identified errors stored (this number is a Workspace parameter: *Max number of messages*, Workspace editor);
- CONTROL-ERROR indicates whether an application error (Y) or a system error (S), or no error (N) has been detected;
- CONTROL-BUSINESS-OBJECT stores the identifier of the current Business Object;
- CONTROL-PART-TYPE stores the identifier of the current Business Object layout: D, for detail, L, for list;
- CONTROL-ROW-INDEX stores the index of the current row in the current list;
- CONTROL-SERVER-VERSION, used for checking the consistency of the client's version and that of the accessed server (it is empty in the standard generated application);

Note: The contents of this Record are used to fill in the Application Error Window (GUI) and Error Map (TUI).

System Error Data WSYS-ERROR Record

This Record is responsible for storing the system error codes. It comprises the following Data-Items:

- SYSTEM-ERROR describes the system error structure;
 - APPLICATION-NAME stores the name of the erroneous application;
 - FUNCTION-NAME stores the name of the processing in progress when the error occurred;
 - ERR-SQLCODE stores the SQL code returned by the server.

Note: The contents of this Record are used to fill in the System Error Window (GUI) and Error Map (TUI).

GUI Components

Visual Components:

Traceability Category: API

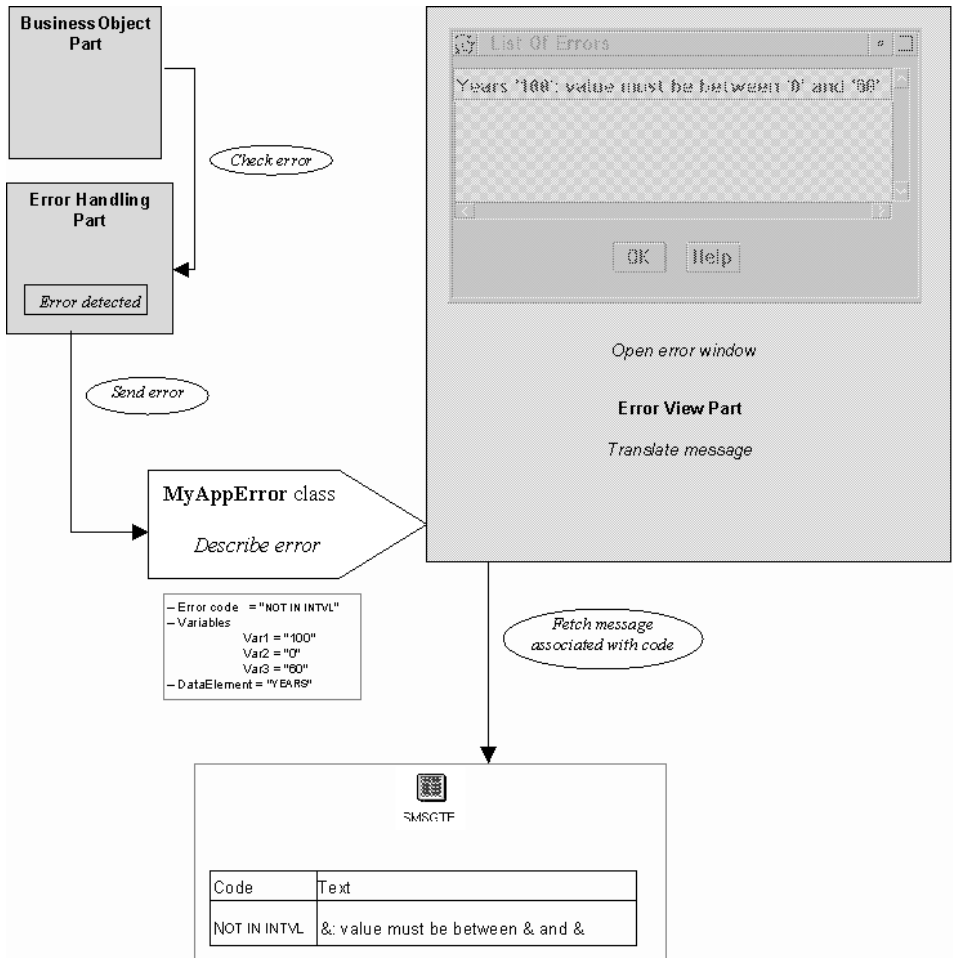
There are three visual components generated: two Views representing the Error Windows and one WindowsMenu component representing the list of open windows.

Note: These components are prefixed with the target name of the Workspace.

Error Views: There are two Views generated for displaying the application error messages and the system error messages.

- the ErrorView bean/part represents the Application Error Window that displays the list of the detected application errors and their explanatory labels. It implements the translation of errors into user error messages;
- the SystemErrorView bean/part represent the System Error Window that displays the SQL code of the error, the function during which the error occurred, and the application name.

Example: The following figure roughly illustrates the processing of an error in a Data Element for which a check by interval has been specified: The end user entered 100 in the "Years" field whereas the interval of authorized values ranges from 0 to 60.



When the end user clicks on the error message, in the error window, the window containing the corresponding erroneous field opens and the focus is set back to the erroneous field.

WindowsMenu Bean/Part: The WindowsMenu bean/part is managed by the `<>SharedComponents` class (see *"Non-Visual Components"* on page 308). It represents the **Windows** menu in the final application.

This menu displays the list of the windows that are open in the final application, in the order in which they were opened, as menu items.

If there are more than ten open windows, the **More Windows** choice is added to the **Windows** menu. Selecting this choice opens a window displaying the complete list of open windows.

Java/Smalltalk Classes:

Traceability Category: API

The generated Java/Smalltalk Classes offer error handling services.

Note: You can use these components. They should not be modified or deleted.

<>Common Services: *<>CommonServices*

This class is responsible for memorizing the errors built from the information included in the WERROR-LIST Record, it tests whether errors are occurring and opens and closes the error windows.

For information on the error View beans/parts, refer to topic “*Error Views*” on page 317. For information on the WERROR-LIST Record, refer to “*Error Data Records*” on page 315.

<>Error: *<>Error*

This class describes the errors manipulated by the application. The information it holds are used to fill the Application Error Window.

For information on the Application Error Window bean/part, refer to “*Error Views*” on page 317.

<>SystemError: *<>SystemError*

This class describes the system errors manipulated by the application. The information it holds are used to fill the System Error Window.

For information on the System Error Window bean/part, refer to “*Error Views*” on page 317.

<>Field: *<>Field*

This class represents the encapsulation of a field into the detail Business Object. It provides information about the status of the field (required, key, etc.) and stores the name of the corresponding check method in the Business Object bean/part. It stores the position of the field when it is presented in a notebook to enable the feedback of a possible error to the field.

Non-Visual Components: The generated Non-Visual components are:

- the ancestors of the Business Object and the List Manager bean/part:
<>BusinessObjectAbstract and *<>ListManagerAbstract*

- a component that offers error handling services: <>**ErrorHandling**.

Note: We call <>**ErrorHandling** the ErrorHandling bean/part.

<>*BusinessObjectAbstract Class:* This class factors out the generic methods that are common to all Business Object instances. For example, it implements the actions available for the detail Business Object.

This class is the parent class of the Business Object bean/part. This is why it is documented in “**Abstract Business Object**” on page 294.

also, for complete information on the Business Object bean/part, refer to “*Business Object Bean/Part*” on page 293.

<>*ListManagerAbstract Class:* This class factors out the generic methods that are common to all List Manager instances. For example, it implements the actions available for the list Business Object.

This class is the parent class of the List Manager bean/part. This is why it is documented in “Abstract List Manager” on page 298.

Also, for complete information on the List Manager bean/part, refer to topic “*List Manager bean/part*” on page 297.

ErrorHandling Part:

Traceability Category: INTERNAL

<>ErrorHandling

The ErrorHandling bean/part is generated along with the detail Business Object. It implements the activation of controls and the feedback to the detail.

The ErrorHandling bean/part is initialized when a detail is being created with the information about the Business Object’s fields.

It is dedicated to checking the input in the Business Object’s fields. It implements unitary error checks on each field of the detail Business Object, and global checks through sequential calls of the unitary checks. It implements the Check action of the Business Object. It is responsible for setting the focus back on the erroneous field, if the end user double-clicks on an item in the Application Error Window.

Caution: You can use this component. It should not be modified or deleted.

For more information on checks, refer to topic “**Error Handling in GUI Client applications**” on page 153.

Communication Table:

Traceability Category: API

The Communication Table is generated if the *Common area* parameter is set to *record and table* or *table* (Workspace editor). When the *Common area* parameter is set to *record and table*, a communication Record is generated to receive data from the Table (see “*Communication Record*” on page 315).

The SCOMMTBL Table is shared among all the client components. It is generated with one Data-Item: CONTROL-COMMUNICATION. This Table is empty when generated so that you can customize it.

Note: You can use this Table, modify and enrich it with data that must be shared and transferred to the server. This component should not be deleted.

TUI Components

Fastpath Table:

Traceability Category: RAD

The generated fastpath Table - SAPPLT1 Table - stores the fastpaths and the access parameters that allow the end user to access a Map directly and display extracted data in this map in the same transaction.

This Table comprises the following Data-Items:

- FASTPATH stores the fastpath of the Interface Unit (this fastpath is retrieved from the *Fastpath* parameter in the Workspace editor);
- APPLICATION-ID stores the name of the Program generated from the Interface Unit;
- APPLICATION-DESC stores a description of the Interface Unit, which corresponds to the title of the Interface Unit;
- ACCESS-PARAMETERS stores the access parameters that are available for the Interface Unit.

For information on the use of fastpaths and access parameters, refer to “**Navigating Throughout a TUI Application**” on page 158.

-GLOBAL Record: This Record is used as a common area and can be accessed by all the applications’ components to transfer data from one

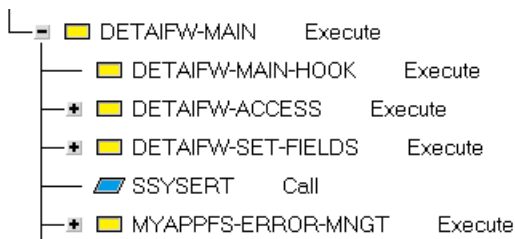
application to the other. The standard data transfer function is made possible thanks to the -GLOBAL Record (see “**Data Transfer between TUI Maps**” on page 152).

This Record comprises the following Data-Items:

- WS-NAVG-STACK stores information on the navigation stack to retrieve the location of the current Program in the navigation;
 - WS-APPLICATION-ID stores the identifier of the previous Program in the navigation stack;
 - WS-ACCESS-PARAMETERS stores the access parameters of the Program in the navigation stack;
 - WS-APPLICATION-TYPE stores the type of each Program in the navigation stack (menu, detail, or list);
- Note:** There are as many occurrences of the three Data-Item above as there are stored path, according to the *Navigation stack max number* parameter (Workspace editor).
- WS-STACK-LEVEL stores the location of each Program in the navigation stack;
- WS-CONTROL describes the structure of the system controls;
 - WS-FASPATH stores the fastpath of the current Program;
 - WS-CURR-APPL-ID stores the identifier of the current Program;
 - WS-CURR-APPL-TYPE stores the type of the current Program (menu, detail, or list);
 - WS-USER-ID stores the user identifier;
 - WS-INTERFACE-UNIT-ACTION stores the action requested in the current Program;
 - WS-NAV-FLAG stores the type of navigation requested: (N, navigation, H, help request, L, help list request);
 - WS-PFKAID stores the activated function key;
 - WS-MAP-NUMBER stores the number of the current map;
 - WS-MAP-MAX stores the total number of maps;
 - WS-MOD-PARAMETERS indicates whether the access parameter has been modified (Y) or not (N) - if yes, no new access is performed;
 - WS-TRANSACTION-TYPE stores the type of data transfer requested: D, DXFR, i.e. a transfer among Programs within the same transaction, or X XFER, I.E. a transfer from one transaction to another;
 - WS-MULTI-SELECT describes the structure of multi-selections;
 - WS-SELECT-PARAMETERS stores the selected records;

- WS-MULTI-LEVEL stores the number of selected records; the maximum number is 20 (size of the *Rows to fetch* parameter, Business Object editor);
- WS-HELP describes the structure of the on-line help;
 - WS-HELP-TYPE stores the type of the on-line help (*, General help, O, Help for Business Objects, E, help for Data Elements)
 - WS-HELP-INDEX-FIRST stores the first line of the help text;
 - WS-HELP-INDEX-LAST stores the last line of the help text;
 - WS-HELP-INDEX-CURRENT stores the current line of the help text;
 - WS-HELP-N-DISP-LINES stores the number of lines of the help text;
- WS-SOURCE describes the source of the requested action;
 - WS-SOURCE-IDENTIFIER stores the identifier of the source application;
 - WS-SOURCE-FIELD stores the identifier of the source Data-Item;
 - WS-SOURCE-TYPE stores the type of the source (E, Data Element, O, Business Object, W, Interface Unit);
 - WS-SOURCE-INDEX-N stores the location of the source field in the map;
 - WS-LKP-INDEX-N stores the location of the source field of a lookup in the map;
- WS-MSG-NUMBERS stores the number of messages;
 - WS-MSG-CURRENT-NUM stores the number of the current message;
 - WS-MSG-NUM-SEPARATOR stores the separator in the number of messages;
 - WS-MSG-TOTAL-NUM stores the total number of messages;
- WS-USER-SPECIFIC is an empty Data-Item that you can use to define user profiles and associate a user to a printer, for example.

System Error Program: There is one Program generated to manage system errors, the SSSYERT Program. This Program is called by the Main Program after a server access has been performed. If a system error is detected, it opens the System Error Map, S SYSERR.



System Error Map: The System Error Map - S SYSERR - displays the error messages that are issued following the detection of a system error.

It displays the SQL code of the error, the identifier of the application where the error has been detected, the identifier of the function in progress when the error was detected.

This Map is filled in with the data from the WSYS-ERROR Record (see “Error Data Records” on page 315).

Application Enhancement: Public Interface of GUI Generated Components

The API described in this subchapter is the API of the Resource Object, of the Business Object and of the List Manager bean/parts. This API offers services for manipulating Business Object’s data.

Note: To perform the actions on the Business Object’s data, the API introduces two Business Object states: *persistent* and *volatile*. The Business Object instance is *persistent* when it is read from the database. It is *volatile* when it is not yet stored in the database (when it is created via the *New* action, for example).

The Business Object instance is unique, it is defined by its key: once it has been read in the database, this instance is always returned if the Business Object appears several times in the application. This is visible for example when you read an instance in a detail, its key becomes read-only. When the instance is used in the application, it is removed from an internal dictionary (weak dictionary mechanism). It is not memorized in a cache.

This API is an instance API.

Resource Object Bean/Part Interface

The ResourceObject API is an API for accessing data.

API for Managing Detail Data

read: *aBusinessObject*

- **Requirements:** The Business Object instance must be *volatile*.
- **Responsibility:** Trigger a server access from the data held by *aBusinessObject*. If the access succeeds, the *aBusinessObject* data is updated from the data returned by the server; *aBusinessObject* becomes *persistent*. If the access fails, an error is raised; *aBusinessObject* is not modified.

create: *aBusinessObject*

- **Requirements:** The Business Object instance must be *volatile*.
- **Responsibility:** Trigger a server access from the data held by *aBusinessObject*. If the access succeeds, the *aBusinessObject* data is updated

from the data returned by the server; *aBusinessObject* becomes *persistent*. If the access fails, an error is raised; *aBusinessObject* is not modified.

update: *aBusinessObject*

- **Requirements:** The Business Object instance must be *persistent*.
- **Responsibility:** Trigger a server access from the data held by *aBusinessObject*. If the access succeeds, the *aBusinessObject* data is updated from the data returned by the server. If the access fails, an error is raised; *aBusinessObject* is not modified.

delete: *aBusinessObject*

- **Requirements:** The Business Object instance can be either *persistent* or *volatile*.
- **Responsibility:** Trigger a server access from the key of *aBusinessObject*. If the access succeeds, the *aBusinessObject* data is re-initialized and *aBusinessObject* becomes *volatile*. If the access fails, an error is raised; *aBusinessObject* is not modified.

save: *aBusinessObject*

- **Requirements:** The Business Object instance can be either *persistent* or *volatile*.
- **Responsibility:** Trigger a server access. If the access succeeds, the *aBusinessObject* data are updated from the data returned by the server and *aBusinessObject* becomes *persistent*. If the access fails, an error is raised; *aBusinessObject* is not modified.

API for Managing List Data

The Resource Object list API positions the extract criteria and the data that characterize the key to be read. An action retrieves a Business Object collection that corresponds to a data page read by a server access.

readBusinessObjects: *aListManager*

- **Responsibility:** Read a data page from the criteria in *aListManager*, and updates *aListManager*.

API for Managing Updatable List Data

The API accepts one or more collections holding the Business Object instances on which the elementary actions are performed. An action triggers the server call on these collections.

commitInsertionsIn: *insertionsOC* **updatesIn:** *updatesOC* **deletionsIn:** *deletionsOC*

- **Responsibility:** Activate the server call on the collection of movements to commit.

Business Object Bean/Part Interface

The Business Object API allows manipulating the Business Object *via* connections.

The API provides getters and setters on the updatable Business Object fields and getters on the read-only fields. It also provides actions.

check

- **Responsibility:** Check every layout field in the Business Object. Otherwise, call the events: *createFailed*, *createSucceeded* or *actionSucceeded*.

read

- **Requirements:** Self is *volatile*.
- **Responsibility:** Search if an instance with the same key exists in the instance dictionary and return it. Otherwise, call the **read:** method from the Resource Object. If the method succeeds, store self in the instance dictionary. The events returned are: *readFailed*, *readSucceeded* or *actionSucceeded*.

create

- **Requirements:** Self is *volatile*.
- **Responsibility:** Search if an instance with the same key exists in the instance dictionary. If such an instance exists, an error is raised. Otherwise, call the **create:** method from the Resource Object. If the method succeeds, store self in the instance dictionary. The events returned are: *createFailed*, *createSucceeded* or *actionSucceeded*.

update

- **Requirements:** Self is *persistent*.
- **Responsibility:** Call the **update:** method from the Resource Object. The events returned are: *updateFailed*, *updateSucceeded* or *actionSucceeded*.

save

- **Requirements:** Self is either *persistent* or *volatile*.
- **Responsibility:** If self is *volatile*, call the **create:** method from the Resource Object. If the action succeeds, self is stored in the instance dictionary. If self is *persistent*, call the **update:** method from the Resource Object.

delete

- **Requirements:** Self is either *persistent* or *volatile*.
- **Responsibility:** Call the **delete:** method from the Resource Object and remove from the instance dictionary the inputted data corresponding to the Business Object key to be deleted. The events returned are: *deleteFailed*, *deleteSucceeded* or *actionSucceeded*.

newInstance

- **Responsibility:** Return a new *volatile* Business Object instance initialized with the default Business Object's values. The event returned is *actionSucceeded*.

refresh

- **Requirements:** Self is either *persistent*.

- **Responsibility:** Call the **read:** method from the Resource Object. If the instance is being used in a list, it is also refreshed. The events returned are: *refreshFailed*, *refreshSucceeded* or *actionSucceeded*.

undo

- **Requirements:** Self is either *persistent*.
- **Responsibility:** Re-initialize the data from self with the data that were memorized during the last read.

List Manager Bean/Part Interface

The API provides getters and setters on the extract fields and the sort fields. It also provides actions. It provides an ordered collection of the Business Objects in the current data page and an ordered collection of the Business Objects in all the read pages.

It also provides actions. When an instance has been deleted from a detail for example, the instance is automatically removed from the ordered collection.

API for Managing List Data

top

- **Responsibility:** Call the **readBusinessObjects:** method from the ResourceObject to read the first data page. Update the ordered collection. The events returned are: *topSucceeded*, *topFailed*.

next

- **Responsibility:** Call the **readBusinessObjects:** method from the ResourceObject to read the next data page. Update the ordered collection. The events returned are: *nextSucceeded*, *nextFailed*.

previous

- **Responsibility:** Call the **readBusinessObjects:** method from the ResourceObject to read the previous data. Update the ordered collection. The events returned are: *previousSucceeded*, *previousFailed*.

extract

- **Responsibility:** Call the **readBusinessObjects:** method from the ResourceObject to read the data page that corresponds to the extract criteria. Update the ordered collection. The events returned are: *extractSucceeded*, *extractFailed*.

refresh

- **Responsibility:** Call the **readBusinessObjects:** method from the ResourceObject to refresh the current data page. Update the ordered collection. The events returned are: *refreshSucceeded*, *refreshFailed*.

Additional API for Managing Updatable List Data

addRow: *anIndex*

- **Responsibility:** Add an empty row at the index *anIndex* +1

deleteRow: *anObject*

- **Responsibility:** Remove the object designated by *anObject* from the list.

updateRow: *anObject*

- **Responsibility:** Set the modifications made to the object designated by *anObject*.

submit

- **Responsibility:** Send the addition, deletion and modification movements to the server by calling the **commitInsertionsIn:updatesIn:DeletionsIn:** method from the Resource Object bean/part. If the deletion movements fail, the corresponding objects are added at the end of the collection.

Part 4. Appendixes

Glossary

Application. A set of related Members' definitions that VisualAge Generator can generate into executable form. (Source: *Introducing VisualGen*, GH23-6570-02)

Business Object. An object *via* which persistent data can be consulted and updated. Business Objects group a set of fields (Data Element calls) mapping to the columns of one or more Relational Tables, to match the needs for a specific application to access, present, and manipulate persistent data.

Data Element. An information element stored in a Relational Table column or manipulated as a Business Object field. Its description includes a type, a length, labels, On-Line Help text for end-users, value checks ...

Data-Item. A unit of information defined by length, data type, and other characteristics. (Source: *VisualGen Library Guide and Glossary*, GH23-6554-02)

Record, Table

Graphical page. In a read-only list, the number of data displayed simultaneously on screen.

Memory page, Selection

GUI client application. A definition of a window or a set of windows that represent the graphical user interface of an application.

Hook. An empty VisualAge Generator Process or StatementGroup generated by VAGTemplates to allow addition of specific code.

Information Model. The structure of the information required to describe the applications at a logical level.

Interface Unit. The interface presenting the Business Objects and specifying whether the Business Object is used as a *detail* or as a *list*. The

Interface Unit also defines the navigation towards other Interface Units (graphic Windows or Maps).

List of movements. The set of movements passed to the server simultaneously, in an updatable list.

Map. A definition of all or part of the layouts and characteristics of the information presented on a character-based screen.

Map group

Map group. The set of all the maps used in an application. The Map group groups all the layouts and characteristics of the information presented on a character-based screen.

Map

Memory page. A selection available without a new database access, in a read-only list.

Graphical page, Selection

Movement. A creation, an update or a deletion, in an updatable list.

Process. A block of logic consisting of a set of processing statements surrounding a central input or output (I/O) operation. (Source: *Introducing VisualGen*, GH23-6570-02)

StatementGroup

Program Specification Block. A formal DL/I description of the hierarchical database structures that an application can access. (Source: *VisualGen Library Guide and Glossary*, GH23-6554-02)

PSB.

Program Specification Block

Record. A collection of Data-Items structured to describe the layout of information in memory, in a database table, or in a file.

Data-Item

Relational Table. A relational table holding the characteristics of either a *Table*, comprising a list of columns, or a *View*, comprising several columns extracted from one or several Tables.

Selection. The set of data verifying an extract criteria, in a read-only list.

Memory page, Graphical page

StatementGroup. A set of processing statements that perform processing only (no I/O operations are performed). (Source: *VisualGen Library Guide and Glossary*, GH23-6554-02)

Process

Table. A collection of related Data-Items that can be used to edit data, store messages that an application issues, and store information for reference by an application. (Source: *Introducing VisualGen*, GH23-6570-02)

Data-Item

Value Style. A set of characteristics defining a presentation style for a numeric, date or time value. It holds all the display and input characteristics a numeric, date or time value will have in the generated applications.

Workspace. A development context that corresponds to the import of one and only one relational database.

Index

A

Access parameters 159
action bar 13
application error 155, 157

C

Classes
 Targeted classes
 MdiVgApplication 211
column 183
container 191

D

display name 162, 164, 165

E

Editors
 Business Object Definition editor 67
 Business Object editor 13
 Data Element Definition editor 79
 Data Element editor 13
 Interface Unit editor 13, 89
 Relational Table editor 13, 96
 Value Style Definition editor 102
 Value Style editor 13
 Workspace editor 10
Entities area 12
Entity menu 14, 36
 Default Generation Parameters choice 36, 57, 75, 76, 84, 86, 93, 94, 101
Extension 159
extract criteria 148, 159

F

foreign key 190

G

graphical page 149

H

Help menu
 General Help choice 49
 Help Index choice 49

I

Information Model 52
Information Model entities
 Business Object 52, 56, 85

Information Model entities

(continued)

 Data Element 52, 56, 76, 102
 Interface Unit 52
 Relational Table 52, 94
 Value Style 52, 102
Instance menu 14
 Copy Generation Parameters from... choice 39
 Definition choice 38
 Generate choice
 cascaded generation radio button 41
 cascaded generation with predefined members radio button 41
 instance generation radio button 41
 Override existing parts check box 41
 Save as default check box 41
New ... choice 37
Save As... choice
 Application combo box 40
 Instance names 40
 New instance name 40
 Open now check box 40

Instances area 12

interval 172

L

line 183
list of movements 150

M

memory page 149
menu bar 12
Methods
 Instance methods
 arePFKeysEquated: 211
movement 150

N

name 11

O

Open windows area 13

P

Parameters
 BusinessObject parameters
 Alignment 59, 186

Parameters (continued)

 BusinessObject parameters (continued)

 Backward label 66, 195
 Check label 61, 190
 Copy label 165
 Create label 61, 190
 CRUD Activation Control 61
 Cut label 165
 Delete label 62, 190
 Display 58, 60, 63, 65, 68, 181, 183, 189, 193, 195, 217
 Edit menu title 165
 Extract label 66, 195
 Extraction criteria displayed 195
 Foreign key help list 220
 Forward label 66, 195
 Help list display 170, 191, 212
 Help list for all foreign keys 66, 190
 Help list page display 66, 191
 Help list prefilled 67, 191, 221
 Help list size 67, 189
 Help panel ID 75
 Label 58, 63, 184, 194
 Label and value display 60
 Layout 59, 187, 219
 Layout Suffix 57
 List pages stack number 64
 List prefilled 63, 194, 223
 List size 174
 Menu label 66
 New label 61, 190
 Notebook field location 59
 Number of columns 58, 185, 187, 194, 222
 Number of fields per page 59, 185
 Number of lines 58, 64, 185, 187, 194, 222
 Number of rows to fetch 62, 149, 150
 Page display 149
 Paging policy 149, 150
 Paste label 165

Parameters (continued)

- Position 61
- Read label 61, 190
- Refresh label 66, 195
- Save label 62, 190
- Service Level 67
- Show message label 62
- Sizing 60, 65, 188
- Submit label 66, 195
- Top label 195
- Update label 62, 190
- Zoom on double click 57
- DataElement parameters
 - Case control 177
 - Check type 169, 172, 173, 212
 - Column 176
 - Column label 213
 - Comment display 78, 171, 172
 - Default 176
 - Default label 213
 - Help panel ID 85
 - Horizontal Orientation 79
 - Max display size 79, 176
 - Pattern 79, 174
 - Size 177
 - SQL type 77
 - Target name 75, 85
 - Value display 78, 170, 171, 173, 188
- Interface Unit parameters
 - Menu label 88
- InterfaceUnit parameters
 - Allow maximize 198
 - Allow minimize 198
 - Allow resize 198
 - Edit menu title 198
 - Fastpath 207
 - Help panel ID 93
 - Interface unit display 197
 - Layout type 191, 221
 - Navigation menu title 158, 198
 - Target name 93
 - Title 197, 206
 - Type label 199
 - Windows menu title 158, 198
- RelationalTable parameters
 - Backward label 96, 191
 - Concurrency management 94
 - Forward label 96, 191
 - SQL Qualified 95
 - Table qualifier 95
 - Target name 101
 - Top label 65, 96, 191

Parameters (continued)

- Workspace parameters
 - Action display policy 207
 - Business Object
 - mnemonic 34
 - Cancel 209
 - Column 176
 - Column label 213
 - Common area 32
 - Control location 32, 153, 154, 155
 - Create 209
 - Create function key and label 32
 - Data Element mnemonic 32
 - Dataltem mnemonic 32
 - Date Format 180
 - Delete 209
 - Delete function key and label 32
 - Display actions 209
 - Display popup 211
 - Entry default value 32
 - Error color 22, 153, 155, 156, 205
 - Error message color 32, 206
 - Exit 209
 - Exit function key and label 32
 - Field selected on focus 22
 - Function key display policy 23
 - Function key label color 32, 205
 - Function keys display policy 207, 208
 - General Help File 23
 - GUI Application
 - mnemonic 32
 - Help 32, 160
 - Help function key and label 23
 - Help key 163
 - Help text color 28, 206
 - Horizontal margin 22
 - Horizontal Orientation 22
 - identifier first 29
 - Information message color 28
 - Interface Unit mnemonic 34
 - Label to value horizontal gap 22
 - Label to value vertical gap 22

Parameters (continued)

- Workspace parameters (continued)
 - Left function key and label 32
 - Letter width 22
 - Lookup 210
 - Lookup function key and label 32
 - LUW mode 32
 - Map Group mnemonic 32
 - Map mnemonic 34
 - Max navigation stack number 24
 - Max number of messages 24
 - Max number of variables 24
 - Max size of variables 24
 - Messages display 24, 157, 207
 - Messages per page 24, 207
 - Next message 210
 - Next page 210
 - Normal color 22, 205
 - Null managed 29
 - Popup policy 164
 - Previous message 210
 - Previous page 210
 - Process mnemonic 34
 - Program mnemonic 34
 - PSB mnemonic 34
 - Read 210
 - Read only color 22
 - Read-only color 205
 - Record mnemonic 34
 - Refresh 210
 - Relational Table
 - mnemonic 34
 - Right 211
 - Screen resolution 22
 - Server layer 32
 - Show message label 155
 - Standard device 199
 - Statement Group
 - mnemonic 34
 - Table mnemonic 34
 - Title color 28, 205
 - Top 211
 - type first 24
 - Update 211
 - Update GUI policy 148
 - Update policy 23
 - Update TUI policy 148
 - Vertical margin 22
 - Warning message color 206
 - Workspace mnemonic 34

- Pop-up menu 14
 - Add choice 56, 91
 - Add volatile choice 56
 - Copy Parameters from... choice 14
 - Default Generation Parameters choice 14, 36
 - Delete choice 14
 - Editions... choice 14
 - Generate choice
 - instance generation radio button 14
 - Load... choice
 - Another Edition 14
 - Previous Edition 14
 - New... choice 14
- presentation label 176, 213
- Push buttons
 - Cancel push button 55
 - Customize Value Style push button 102
 - Delete push button 14
 - Help push button 55
 - OK push button 55
 - Reset push button 55

S

- selection 149
- Settings
 - BusinessObject settings
 - Access Level 72
 - Application 68
 - Data Element 76
 - Default use name 68
 - Display name 68, 161, 162, 164, 165
 - Extract Field 71
 - Extraction Criteria 71
 - Field Name 71, 72
 - Fields 70, 72
 - Join condition type 73
 - Laid Out 73
 - Logical key 71
 - On line help description 69
 - On-line help description 160, 162
 - Required 72
 - Retrieve Policy 71
 - Sort Criteria 71
 - Sort Field 71
 - Source field 74
 - Source Table 73
 - Table 72
 - Target field 74
 - Textual description 68

- Settings (*continued*)
 - BusinessObject settings (*continued*)
 - Updatable 73
 - CRUD Properties 100
 - DataElement settings
 - Column label 78
 - Default label 77
 - Default use name 80
 - Default value mode 82
 - Display name 80, 161, 162, 164, 165
 - Maximum 84
 - Minimum 84
 - On line help description 82
 - On-line help description 160, 162
 - Textual description 81
 - Type 84
 - Value style 81
 - Value type 80
 - InterfaceUnit settings
 - Application 89
 - Business Object 91
 - Display name 161, 162, 164, 165
 - Fastpath 86
 - Interface Unit 91
 - Layout type 91
 - On line help description 90
 - On-line help description 160, 162
 - Textual description 90
 - Title 86
 - Type 90
 - Use name 91
 - RelationalTable settings
 - Application 96
 - Columns 98
 - Concurrency management column 98
 - Data Element 98
 - Default use name 96
 - Delete rule 98
 - Display name 96
 - Identifier 98
 - Join condition 100
 - Keys 98
 - Label 98
 - Links 98
 - Mapped column 100
 - Mapped table 100
 - Mapper 100
 - Mapping 100
 - Source Columns 73

- Settings (*continued*)
 - RelationalTable settings (*continued*)
 - Source key 98
 - Table name 97
 - Table type 97
 - Target Columns 74
 - Target table 98
 - Textual description 96
 - Update rule 98
 - ValueStyle settings
 - Am string 103, 180
 - Application 103
 - Capacity 214
 - Cycle 103, 180
 - Date style 181
 - Decimal separator 103, 177, 214
 - Default use name 103
 - Display name 103
 - Mask 103, 179, 180, 216
 - Name 103
 - Negative sign 103, 177, 214
 - Pm string 103, 180
 - Positive sign 103, 177, 214
 - Separator 103, 179, 181, 216
 - Sign position 103
 - Textual description 103
 - Thousand separator 103
 - Thousands separator 177, 214
 - Time style 181
 - Type 103
 - Unit 103, 177, 214
 - Unit and sign alignment 103, 173, 177, 214
 - Unit position 103, 177, 214
 - Year style 103, 179, 216
 - Workspace settings
 - Application 20
 - Date Internal Format 30
 - Decimal separator 30
 - SQL High value 20, 30
 - SQL Low value 20, 30
 - Target application 20
 - Target Name 20
- system error 156, 157
- System menu 10
 - Maximize choice 13
 - Move choice 13
 - Reduce choice 13
 - Resize choice 13

T

Tabs

- Definition editor 52
- Extensions panels 56
- Generation Parameters editor 53

title bar 12, 13

Tools menu

- Import from Database choice
 - Application 46
 - DBMS 46
 - Identifier 46
 - Password 46
 - Qualifier 46
 - Re-use data element 46
 - Tables 46
 - UserId 46

V

value table 169, 212

View 94

VisualAge Generator Parts

Map

- Data Map 211
- Header Map 200, 201, 202, 207
- Help List Map 204, 211
- Help Map 165, 199, 204, 205, 207, 211
- Root Map 158, 199, 200, 201
- Simple Map 199, 201, 202, 203
- Trailer Map 157, 199, 201, 203, 207

W

window dialog 157

Window menu 13

Workspace menu 15

- Delete choice 16
- Generate choice 16
 - Override existing parts check box 20
 - Save as default check box 20
- Open... choice 14

Workstation windows

- Add Business Objects window 91
- Copy Instance Parameters from Workspace window 39
- Delete Workspace window 16
- Entity Default Generation Parameters editor 36
- Generate (instance) window 41
- Generate window 36

Workstation windows (*continued*)

- Import from Database window 46
- New VAGT Instance window 37
- Open Workspace window 15

Readers' Comments — We'd Like to Hear from You

VisualAge Generator Templates Standard Functions
User's Guide
Version 4.5

Publication No. SH23-0269-01

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Complete	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to find	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to understand	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Well organized	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Applicable to your tasks	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? Yes No

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

Name

Address

Company or Organization

Phone No.



Fold and Tape

Please do not staple

Fold and Tape



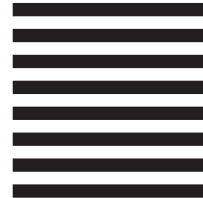
NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
IBM SWS - Paris Laboratory
1, place Jean-Baptiste Clément
93881 Noisy-le-Grand CEDEX
France.



Fold and Tape

Please do not staple

Fold and Tape



Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SH23-0269-01

