VisualAge Generator

Getting Started

Version 4.5

GH23-0258-01

Note

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

Second Edition (April 2001)

This edition applies to the following licensed programs:
- IBM VisualAge Generator Developer for OS/2 and Windows NT Version 4.5
- IBM VisualAge Generator Server for OS/2, AIX, Windows NT, HP-UX, and Solaris Version 4.5
- IBM VisualAge Generator Server for AS/400 Version 4 Release 4
- IBM VisualAge Generator Server for MVS, VSE, and VM Version 1.2

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) 445-9269. Faxes should be sent Attn: Publications, 3rd floor.

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.

IBM welcomes your comments. You can send your comments in any one of the following methods:

Electronically, using the online reader comment form at the address listed below. Be sure to include your entire network address if you wish a reply.
- http://www.ibm.com/software/ad/visgen

By mail to the following address:

IBM Corporation, Attn: Information Development, Department G7IA Building 503, P.O. Box 12195, Research Triangle Park, NC 27709-2195.

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.

Contents

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, 500 Columbus Avenue, Thornwood NY 10594, 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:

ACF/VTAM
AD/Cycle
AIX
AS/400
CICS
CICS OS/2
CICS/ESA
CICS/MVS
CICS/VSE
CICS/6000
COBOL/370
COBOL/400
DataJoiner
DB2
DB2/2
DB2/400
DB2/6000
Distributed Relational Database Architecture
DRDA
IBM
IBMLink
IMS
IMS/ESA
Language Environment
Operating System/2
OS/2
OS/400
Presentation Manager
SAA
SQL/400
SQL/DS
TeamConnection
Virtual Machine/Enterprise Systems Architecture
VisualAge
VisualGen
VSE/ESA
VM/ESA

The following terms are trademarks of other companies:

Micro Focus	Micro Focus Limited
Micro Focus COBOL	Micro Focus Limited
Oracle	Oracle Corporation
PowerBuilder	PowerSoft Corporation
Sybase	Sybase Corporation
ENVY	Object Technology, Inc.
HP-UX	Hewlett-Packard

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

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

Solaris, Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Terminology used in this document

Unless otherwise noted in this publication, the following references apply:

- MVS CICS applies to Customer Information Control System/Enterprise Systems Architecture (CICS/ESA) systems.
- CICS applies to CICS for VSE/ESA, CICS/ESA, CICS for OS/2, CICS for AIX, CICS for Windows NT, and CICS for Solaris.
- CICS for Windows NT refers to IBM TXSeries for Windows NT Version 4.2.
- CICS for AIX refers to IBM TXSeries for AIX Version 4.2.
- CICS for Solaris refers to IBM WebSphere Enterprise Edition Version 3.0.
- IMS/VS applies to Information Management System/Enterprise System Architecture (IMS/ESA) and IMS/ESA Transaction Manager systems.
- IMS applies to IMS/ESA and IMS/ESA Transaction Manager, and to message processing program (MPP), IMS Fast Path (IFP), and batch message processing (BMP) regions. IMS/VS is used to distinguish MPP and IFP regions from the IMS BMP target environment.
- LE applies to the IBM Language Environment for MVS and VM.
- COBOL applies to any of the following types of COBOL:
  - IBM VisualAge for COBOL for OS/2
  - ILE COBOL/400
  - IBM COBOL for VSE
  - IBM COBOL for MVS and VM
- "Region" and "CICS region" correspond to the following:

  - CICS for MVS/ESA region
  - IMS region
  - CICS for VSE/ESA partition
  - CICS for OS/2 system
  - CICS for AIX system
  - CICS for Windows NT system
  - CICS for Solaris system
- DB2/VSE refers to SQL/DS Version 3 Release 4 or later. Any references to SQL/DS refer to DB2/VSE and SQL/DS on VM. In addition, any references to SQL/400 refer to DB2/400.
- OS/2 CICS applies to CICS Operating System/2 (CICS for OS/2).
- Workstation applies to a personal computer, not an AIX workstation.
- The make process applies to the generic process not to specific make commands, such as make, nmake, pmake, polymake.
- Unless otherwise noted, references to VM apply to Virtual Machine/Enterprise Systems Architecture (VM/ESA) environments.
- References to VM batch apply to any batch facility running on VM.
- DB2/2 applies to DB2/2 Version 2.1 or later, and DB2 Universal Database (UDB) for OS/2 Version 5.
- DB2/6000 applies to DB2/6000 Version 2.1 or later, and DB2 Universal Database (UDB) for AIX Version 5.
- Windows applies to Windows 95, Windows 98, Windows NT, and Windows 2000.
- Unless a specific version of Windows NT is referenced, statements regarding Windows NT also apply to Windows 2000.

Terminology differences between Java and Smalltalk

VisualAge Generator Developer can be installed as a feature of VisualAge for Java or VisualAge Smalltalk. Where appropriate, the documentation uses terminology that is specific to Java or Smalltalk. But where the information is specific to VisualAge Generator and virtually the same for both environments, the Java/Smalltalk term is used.

Table 1. Terminology differences between Java and Smalltalk

Java term	Combined Java/Smalltalk term	Smalltalk term
Project	Project/Configuration map	Configuration map
Package	Package/Application	Application
Workspace	Workspace/Image	Image
Beans palette	Beans/Parts palette	Parts palette
Bean	Visual part or bean	Visual part

Table 1. Terminology differences between Java and Smalltalk (continued)

Java term	Combined Java/Smalltalk term	Smalltalk term
Repository	Repository/ENVY library	ENVY library manager
Options	Options/Preferences	Preferences

Welcome

Welcome to VisualAge Generator—a revolutionary development environment that offers you a quick and easy way to create client/server and standalone applications.

VisualAge Generator gives you one tool with the power to create both client and server applications that run in both workstation and host environments. It also provides access to VisualAge Generator's patented VisualAge/PowerServer technology. This technology enables developers to use VisualAge Generator or a tool of their choice (for example, PowerBuilder, VisualBasic, VisualAge for Smalltalk, or VisualAge for Java) to build client applications and then use VisualAge Generator to quickly build powerful distributed server applications that scale to support thousands of concurrent users.

See your application development capabilities expand with VisualAge Generator. For more information about VisualAge Generator, visit our web site at: `http://www.ibm.com/software/ad/visgen`.

Documentation provided with VisualAge Generator

VisualAge Generator documents are provided in one or more of the following formats:

- Printed and separately ordered using the individual form number.
- Online book files (.pdf) on the product CD-ROM. Adobe Acrobat Reader is used to view the manuals online and to print desired pages.
- HTML files (.htm) on the product CD-ROM and from the VisualAge Generator web page (**http://www.ibm.com/software/ad/visgen**).

The following books are shipped with the VisualAge Generator Developer CD. Updates are available from the VisualAge Generator Web page.

- *VisualAge Generator Getting Started* (GH23-0258-01) [1,2]
- *VisualAge Generator Installation Guide* (GH23-0257-01) [1,2]
- *Introducing VisualAge Generator Templates* (GH23-0272-01) [2,3]

The following books are shipped in PDF and HTML formats on the VisualAge Generator CD. Updates are available from the VisualAge Generator Web page. Selected books are available in print as indicated.

- *VisualAge Generator Client/Server Communications Guide* (SH23-0261-01)[1, 2]
- *VisualAge Generator Design Guide* (SH23-0264-00) [1]
- *VisualAge Generator Generation Guide* (SH23-0263-01) [1]
- *VisualAge Generator Messages and Problem Determination Guide* (GH23-0260-01) [1]
- *VisualAge Generator Programmer's Reference* (SH23-0262-01) [1]
- *VisualAge Generator Migration Guide* (SH23-0267-00) [1]
- *VisualAge Generator Server Guide for Workstation Platforms* (SH23-0266-01) [1,4]
- *VisualAge Generator System Development Guide* (SG24-5467-00) [2]
- *VisualAge Generator User's Guide* (SH23-0268-01) [1, 2]
- *VisualAge Generator Web Transaction Development Guide* (SH23-0281-00) [1]

The following documents are available in printed form for VisualAge Generator Server for AS/400 and VisualAge Generator Server for MVS, VSE, and VM:

- *VisualAge Generator Server Guide for AS/400* (SH23-0280-00) [2]
- *VisualAge Generator Server Guide for MVS, VSE, and VM* (SH23-0256-00) [2]

---

1. These documents are available as HTML files and PDF files on the product CD.
2. These documents are available in hardcopy format.
3. These documents are available as PDF files on the product CD.
4. This document is included when you order the VisualAge Generator Server product CD.

The following information is also available for VisualAge Generator:

- *VisualAge Generator External Source Format Reference* (SH23-0265-01)
- *Migrating Cross System Product Applications to VisualAge Generator* (SH23-0244-01)
- *VisualAge Generator Templates V4.5 Standard Functions—User's Guide* (SH23-0269-01)[2, 3]

Part 1. VisualAge Generator overview

Chapter 1. What's new in VisualAge Generator

New function for VisualAge Generator 4.5

VisualAge Generator V4.5 introduces functional enhancements.

Java Server Generation

VisualAge Generator 4.5 enables you to run Java programs under VisualAge Generator Server. With Java Server support, users can generate main batch, called batch, and Web transaction program parts into Java source code. The Java code is deployed to the target machine and compiled and executed in an e-business environment where it can interact with other Java program parts.

When you generate a main batch or a called batch program as a Java Server, you can also generate a session Enterprise Java Bean (EJB). The session bean enables the program to participate in Enterprise Java Server (EJS) transactions.

Integrated MQSeries support

The integration of MQ Support into VisualAge Generator allows message queues to be accessed as files by using the ADD and SCAN I/O options instead of MQSeries API calls. Support includes:

- Automatic connection to the queue manager
- Automatic opening and closing of queues and disconnection at end of main program
- Optional termination on hard errors
- Automatic return code checking
- Transaction control using EZECOMIT and EZEROLLB functions
- Automatic data conversion of messages based on the message record structure
- Optional access to MQ control blocks for specifying MQ options that are not selectable using the record level interface

Web transaction enhancements

Web transaction development has been enhanced in the following ways:

- Web transaction generation and the GatewayServlet now enable Web transactions for multiple languages. A resource bundle can be generated for a UI record that contains all titles, labels, and help text required by the UI record. Additional resource bundles can be created for other languages that may be used. The GatewayServlet will load the resource bundle for the language of the client if that resource bundle is installed at the gateway.
- The ability to run a mixture of completed (deployed) Web transactions and Web transactions under development is now supported. The completed

Web transactions are run in an application server environment while the Web transactions being developed are run in ITF.

- Web transaction JSP generation is enhanced to, as an option, add default values for fields displayed for a UI record. JSP designers can display modifications made to the generated JSPs without running the Web transaction that would normally provide data formatted by the JSP.
- UI records are enhanced to allow developers to specify a general UI Record edit function that is run whenever any UI record field is changed in response to the display of the UI record. This makes it easier to perform cross-field edits.

Remote DL/I access during test

When you are testing a program that includes a call to DL/I, a new, built-in feature provides remote DL/I access on MVS, where VisualAge brings up an IMS batch environment. You can access DL/I from either Windows NT or OS/2, even if your organization lacks an IMS Transaction Manager on MVS.

Although DL/I access is on MVS, the test facility simulates the runtime behavior of any of several target environments.

OS/400 Support

VisualAge Generator now provides:

- Generation of Web transactions for the OS/400 platform
- Ability to call COBOL server programs from generated Java programs.

Solaris Client/Server Support

Solaris is now supported as the 2nd-tier in a 3-tier configuration with TCP/IP, CICS Client, IPC, and Direct as supported protocols. Solaris can also now act as a client using C++ TUIs or Java programs.

New function for VisualAge Generator 4.0

VisualAge Generator V4.0 introduces both functional enhancements and new development and runtime platforms.

VisualAge for Java interoperability

VisualAge Generator is fully integrated into the VisualAge for Java Enterprise development environment. In the Developer on Java platform:

- VisualAge Generator 4GL parts can be created and stored in a VisualAge for Java development environment.
- VisualAge Generator 4GL parts and logic as well as Java methods can be used in a VisualAge for Java component.
- VisualAge Generator 4GL parts used in a VisualAge for Java component are generated as native Java for use at runtime.

WebSphere Rapid Application Development Support

This new programming model provides traditionally skilled programmers a fast path to creating scalable e-business systems using VisualAge Generator, WebSphere Studio, and WebSphere Application Server. This new function enables programmers to develop main transaction programs, using 4GL programming techniques, that can interact with end users through a web browser.

Functions

In VisualAge Generator V4.0, functions replace statement groups and processes to handle the logic for a program. During migration to V4.0, the Statement Group parts and Process parts are converted to Function parts.

A Function part is used the same way as a Process or Statement Group part with the following support:

- Parameters
- Local storage
- Development of true reusable 4GL code

Object Scripting

In VisualAge Generator V4.0, you can access non-VisualAge Generator objects from within the 4GL through the synchronous execution of a Smalltalk or Java method (Object Script). A new EZE word, EZESCRPT, allows the you to make a call to a Smalltalk or Java method or an Enterprise JavaBean from the 4GL logic.

A new set of wizards and help guide you through the task of building basic method logic (object scripts) in the target language (Java or Smalltalk).

VisualAge Generator Templates Enhancements

VisualAge Generator Templates now provides:

- Generation of Java-based systems
- Improved integration with VisualAge Generator
- Implementation of a VisualAge Generator Templates on Java version of the information model and generators
- Redesigned user interface to improve both entity definition and information model source management.

Solaris Runtime Support

VisualAge Generator programs can be generated as C++ programs that support:

- Both the native operating system and CICS-based execution
- DB2, Oracle, and ODBC database access

Enterprise JavaBeans (EJBs) support

VisualAge Generator support for Enterprise Java Beans (EJBs) will allow Java clients to call VisualAge Generator server programs through an intermediary session bean.

Support for calling server programs through an EJB session bean is only provided in VisualAge Generator Developer on Java development platform.

Automated IC Packaging support

VisualAge Generator support for IC packaging feature allows you to package your GUI system in a modular way. Each ENVY application containing GUIs can be packaged into a separate file (typically 100K-1Mb in size) called an IC (Image Component) file which can be repackaged and distributed separately. Also, the runtime system loads these ICs dynamically as needed.

Native Oracle support

Generated programs can now access Oracle databases directly on CICS for AIX and CICS for Windows NT platforms.

Enhanced Container Details part

The Container Details part has been enhanced for Smalltalk GUI clients to support proportional column widths, tabbing between cells, multiline headings, and common controls at the cell level.

Accessing remote VSAM files

Now when you test programs with the Interactive Test Facility, you can access VSAM files that reside on a remote OS/390 system.

Chapter 2. What is VisualAge Generator?

VisualAge Generator is a high-end system development environment for building and deploying e-business and multitier client/server applications. This award winning OO/4GL application development solution exploits the use of existing system resources and provides developers with a highly productive visual programming environment. VisualAge Generator also provides:

- An integrated, interpretive test environment which allows developers to test their entire N-tier solution (one, two, three or more tiers) on a single development workstation (Windows NT or OS/2) before deployment and use intelligent dynamic application partitioning to recommend the best server deployment options
- Optimized source code generated in either C++ or COBOL depending on the target platform which is then compiled to high-performance object code
- Generation of server programs into JavaBeans and EJBs
- Automatic generation of complete application systems using templates
- Support for legacy systems as your enterprise transitions to higher performance solutions

Programs developed with VisualAge Generator can be written independent of the underlying database, operating system, or transaction processing monitor. You can mix and match their network client/server applications in an open system environment. No special coding is required – VisualAge Generator hides all the complexities of the databases, operating systems, TP monitors, and communications protocols.

This open enterprise capability is further extended by VisualAge Generator's patented VisualAge/PowerServer technology. VisualAge/PowerServer enables companies to use VisualAge Generator or a tool of their choice (for example, PowerBuilder, VisualBasic, VisualAge Smalltalk, or VisualAge for C++) to build the client portions of their client/server systems. Then they can use VisualAge Generator to build powerful distributed server applications that scale to support thousands of concurrent users with high levels of reliability, performance, and portability while accessing their critical enterprise data.

With VisualAge Generator's Rapid Application Development (RAD), you can quickly develop, test, and deploy *Web transactions*:programs for the Internet or a corporate intranet. This technology has several benefits:

- You can develop web programs more quickly than with other technologies.

- You can use existing skill in writing procedural code, using the language with which all VisualAge Generator programs are developed.
- You can develop simpler, more easily maintained code.
- You can convert existing VisualAge Generator programs to provide service on the web.
- Your organization can use personnel more effectively, assigning one team to work on the business logic while another team works on the user interface.

VisualAge Generator products

VisualAge Generator is a workstation-based set of products for application development that provides state-of-the-art integrated facilities to fully define and interactively test business applications. It includes a powerful workbench that allows the creation of various parts of an application (data specifications, user interface definitions and business logic) in a full multitasking and graphical environment.

The following are the VisualAge Generator products:

- **VisualAge Generator Developer** Version 4.5

  With VisualAge Generator Developer you can define, test, and generate programs and various types of user interfaces. VisualAge Generator Developer is fully integrated with both VisualAge for Java and VisualAge Smalltalk and takes full advantage of their object-oriented source code management system and state-of-the-art team development environment. Using the common visual builder, programmers visually define GUI layouts, user actions, and how the occurrence of events trigger execution of business logic, data access or UI changes. A layout editor allows the programmer to quickly build sophisticated GUI windows by simply choosing parts from a rich set of GUI controls, such as entry fields complete with formatting and validation rules, list boxes that automatically implement scrolling, push buttons, and radio buttons. Then these application systems can be generated for various run-time environments.

  VisualAge Generator Developer generates:

  – COBOL programs for the CICS for OS/2, OS/400, VM CMS, VM batch, MVS CICS, MVS/TSO, MVS batch, IMS/VS, IMS BMP, VSE CICS, and VSE batch environments.

  – C++ programs for the OS/2, Windows NT, CICS for Windows NT, AIX, CICS for AIX, HP-UX, Solaris, and CICS for Solaris environments.

- **VisualAge Generator Server for OS/2, AIX, Windows NT, HP-UX, and Solaris** Version 4.5

  VisualAge Generator Server for OS/2 enables you to install and run generated COBOL programs in the CICS for OS/2 environment and C++ programs in the OS/2 environment.

VisualAge Generator Server for AIX enables you to install and run C++ programs in the AIX and CICS for AIX environments.

With VisualAge Generator Server for Windows NT, you can prepare, install, and run C++ programs in the CICS for Windows NT, Windows NT, and Windows 2000 environments.

With VisualAge Generator Server for HP-UX, you can prepare, install, and run C++ programs in the HP-UX environment.

With VisualAge Generator Server for Solaris, you can prepare, install, and run C++ programs in the CICS for Solaris and Solaris environments.

- **VisualAge Generator Server for AS/400** Version 4.4

  VisualAge Generator Server for AS/400 enables you to install and run COBOL programs in the OS/400 environment.

- **VisualAge Generator Server for MVS, VSE, and VM** Version 1.2

  VisualAge Generator Server for MVS, VSE, and VM enables you to prepare, install, and run generated COBOL programs in the following environments:
  - MVS CICS
  - MVS/TSO
  - MVS BATCH
  - IMS/VS
  - IMS BMP
  - VM CMS
  - VM BATCH
  - VSE CICS
  - VSE BATCH

  These programs can be standalone host programs or server programs that communicate with client programs through the provided communications support.

Chapter 3. VisualAge Generator Developer Overview

VisualAge Generator Developer is a complete application development environment that enables you to create, manage, test, and generate programs for use on various platforms. You can build parts and store them in the source code repository where they can be shared by multiple developers.

As you develop, VisualAge Generator Developer provides you with a program diagram that shows the hierarchical structure of the program's components. This diagram is especially helpful to developers working on more than one program. With the diagram, you can easily determine what you have done and what you have left to do. VisualAge Generator Developer maintains database and program test preferences for you in profiles you set up when you start. Both features can help you respond quickly to changing requirements.

VisualAge Generator Developer also provides a test facility for you to interactively test your programs and remove errors as you develop. After testing, you can generate a program for COBOL or C++, depending on the intended run-time environment.

Packaged with VisualAge Generator Developer is a fully functioning object-oriented language (Java or Smalltalk) that you can use for certain tasks or use to support a phased, enterprise-wide move to an component-based development model.

VisualAge Smalltalk Repository management

VisualAge Generator code is managed by the same built-in library system used by VisualAge Smalltalk. This system, called ENVY, facilitates team-development and code reuse by operating in concert with object-oriented principles and providing programmers with an integrated environment (Figure 1 on page 13) in which they can develop, share, and manage source code. Code is managed as methods (or parts), classes, applications, and configuration maps. Images enable developers to customize their environments and changes to parts in the repository are managed as application editions and versions.

Managing VisualAge Smalltalk parts

In this team-development environment VisualAge Generator and VisualAge Smalltalk source code is managed at four levels:

Methods
: are pieces of executable code that implement the logic of a particular message for a class. Methods are the smallest unit of source code that the repository maintains. Each time a method is changed and saved, an edition of its source code is saved in the repository, so that individual changes can be tracked and managed as classes are developed.

Classes
: are specifications that determine the attributes and behavior of software objects. Classes, which typically contain several methods, are also tracked in the repository. Developers can share versions of classes and use them as templates to standardize objects across code libraries. All VisualAge parts are stored as classes.

ENVY applications
: are functionally related sets of defined and extended classes that provide developers with reusable pieces of functionality. Developers can also create and share versions of applications. Loaded applications are listed in the **Applications** pane of the VisualAge Organizer window.

Configuration maps
: are named groups of application editions that are usually associated with a product or a major component of a product. Configuration maps provide an easy way for developers to import and export sets of applications and share them with other development teams.

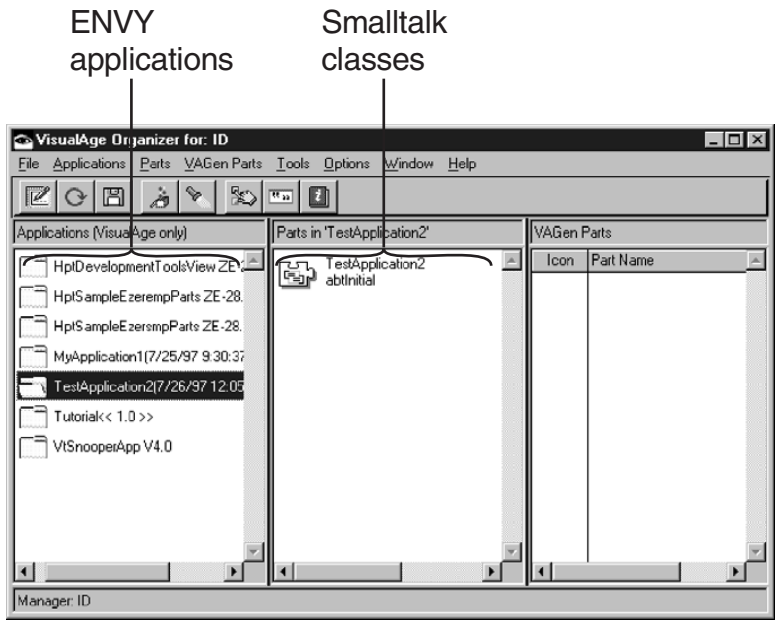


Figure 1. VisualAge Organizer window

For more information on VisualAge for Smalltalk repository management, refer to the *IBM Smalltalk User's Guide.*

VisualAge Generator on Smalltalk repository management

The VisualAge Generator team-development environment (Figure 2 on page 14) uses the same library management system as VisualAge Smalltalk, but VisualAge Generator adds some enhancements to the environment, so there are a few terms that apply to VisualAge Generator development only:

VAGen parts
: are units of VisualAge Generator 4GL code that are associated with a Smalltalk method in an extension of a VAGen part class. VAGen parts are listed by part type in the VAGen parts pane (available only if you have installed VisualAge Generator) in the VisualAge Organizer window. You can also look at an alphabetical list of VAGen parts associated with a loaded application by opening the VAGen Parts Browser. You can open a VAGen part by double-clicking on its name in either window.

VAGen part classes
: are extensions of a VisualAge part class. Like VisualAge part classes, they are used as templates for creating parts to standardize object specifications, attributes, and behavior for different part types. VAGen part classes listed in the **Parts** pane in the VisualAge Organizer window have as the first five letters of their class names "VAGen."

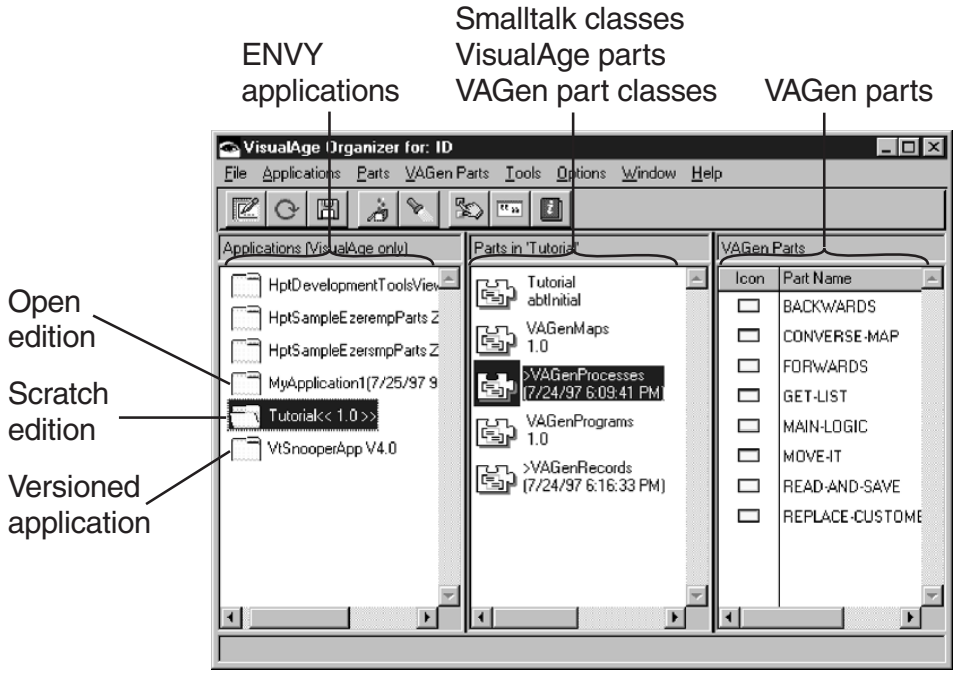


Figure 2. VisualAge Organizer Window: VAGen Part Classes and Parts.

Version control

Unlike traditional source code management systems, where source files are checked out and checked in, VisualAge's access control authorizes groups of developers to work on specific part classes, applications, and configuration maps. Developers customize their workspaces by loading copies of the applications (editions) containing the parts they need to work on into their image. During the development cycle, programmers create open editions and scratch editions of applications to make changes to code and run tests. But each application, and each part class has a single owner, or manager, who is responsible for releasing stable versions at appropriate times.

Familiarity with the following terms is essential to understanding how to use ENVY to manage your code libraries:

Image In the simplest terms, an image is a customized view of items in the repository, limited to the applications that contain the parts you need to change or test. All developers have an image file stored on their workstations. The default image file name is abt.icx. When you save your image, this file is updated to track which editions and versions of applications you have loaded. The next time you start VisualAge

Generator, all the applications that were loaded into your image when you saved it last will be listed in the VisualAge Organizer window and in the VAGen Parts Browser.

Edition

In general, editions are applications that are subject to change. The two kinds of editions you will encounter most often are:

- Open editions—applications that have not been frozen. You can save parts to loaded open editions, and you can recognize them by the date and time stamp next to their names in the **Applications** pane of the VisualAge Organizer window. Open application editions can have new classes added to them, classes deleted from them, and different versions of existing classes released into them.
- Scratch editions—copies of applications that are created automatically when you save changes to an edition that is not open. Scratch editions allow you to make changes for testing existing classes in applications owned by other developers, but you cannot add new classes to a scratch edition. Unlike open editions, scratch editions have a version number in double angle brackets (<<>>) instead of a date and time stamp beside the application name in the **Applications** pane of the VisualAge Organizer window. Changes made to scratch editions only exist in the image of the developer who made them and they cannot be released into an application that does not have an open edition.

Version

Versions are editions that have been frozen by the application manager to prevent further changes to that level of code. They have version numbers (without angle brackets) instead of date and time stamps next to their names in the **Applications** pane of the VisualAge Organizer window.

For hands-on practice loading applications, creating new editions, and other repository management tasks, complete the steps outlined in "Chapter 6. VisualAge Generator Developer on Smalltalk: a tutorial" on page 53.

VisualAge for Java Repository management

VisualAge Generator code is managed by the same built-in library system used by VisualAge for Java. This system facilitates team development and code reuse by operating in concert with object-oriented principles and providing programmers with an integrated environment in which they can develop, share, and manage source code. Code is managed as methods, classes and interfaces, projects, and packages. Workspaces enable developers to customize their environments. Changes to parts in the repository are managed as editions and versions.

VisualAge for Java repository management

In this team-development environment, VisualAge Generator and VisualAge for Java source code is managed at four levels:

Methods

are pieces of executable code that implement the logic of a particular behavior for a class. Methods are the smallest unit of source code that the repository maintains. Each time a method is changed and saved, an edition of its source code is saved in the repository so that individual changes can be tracked and managed as classes are developed.

Classes and Interfaces

are specifications that determine the attributes and behavior of software objects. Classes typically contain several attributes and methods. Interfaces specify a set of behaviors that unrelated objects can use to interact with each other without either object having to know the full specification of the other. Both classes and interfaces are tracked in the repository. Developers can share versions of classes and interfaces and use them as templates to standardize objects across code repositories. All VisualAge for Java beans are stored as classes or interfaces.

Packages

are functionally related sets of classes and interfaces that provide developers with reusable pieces of functionality. Developers can also create and share versions of packages. Loaded packages are listed under the **Packages** tab in the Workbench. They are also listed under their individual projects under the **Projects** tab.

Projects

are named groups of packages that are usually associated with a product or a major component of a product. Projects provide an easy way for developers to import and export sets of packages and share them with other development teams. The hierarchical relationship between projects, packages, and classes and interfaces is shown under the **Projects** tab in the Workbench.

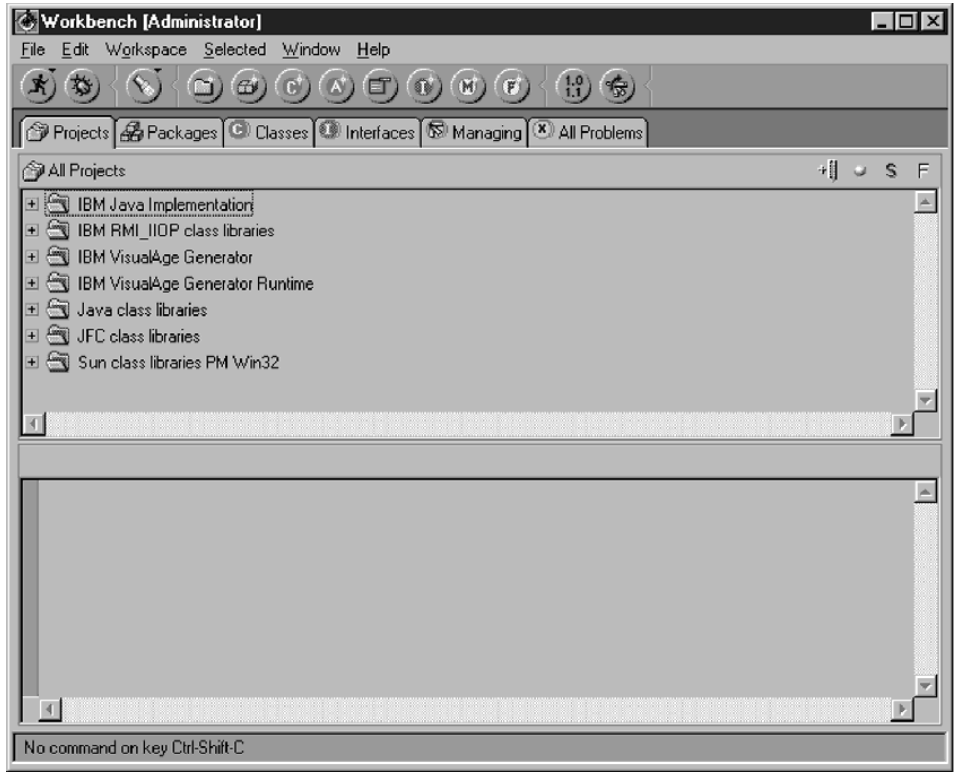


Figure 3. VisualAge for Java Workbench

For more information on VisualAge for Java repository management, refer to the online help.

VisualAge Generator on Java repository management

The VisualAge Generator team-development environment uses the same repository management system as VisualAge for Java, but VisualAge Generator adds some enhancements to the environment so there are a few terms that apply to VisualAge Generator development only:

VAGen parts
: are units of VisualAge Generator 4GL code that are associated with a Java method in a VAGen part class. VAGen parts are listed in the VisualAge Generator Parts Browser and are displayed in the VAGen Parts pane depending on your selections in the Types pane. To display the VAGen Parts Browser, select the Workspace menu, then VAGen Parts Browser. You can open a VAGen part by double-clicking on its name in the Parts Browser.

VAGen part classes
: are Java classes that are used as templates for creating parts to

standardize object specifications, attributes, and behavior for different part types. VAGen part class names begin with "VAGen."

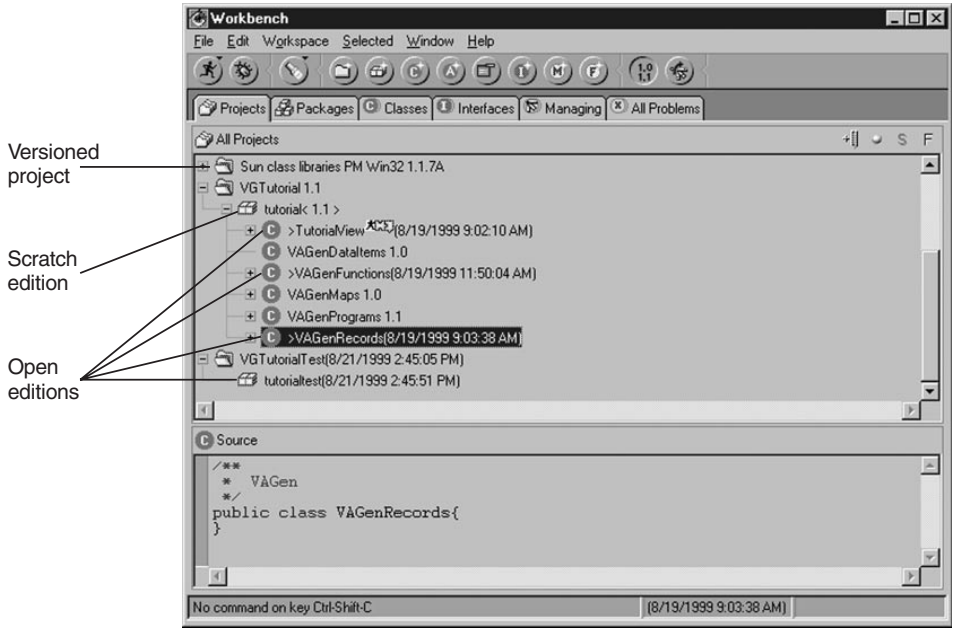


Figure 4. VisualAge for Java Workbench: VAGen Part Classes and Parts

Version control

Unlike traditional source code management systems, where source files are checked out and checked in, VisualAge for Java's access control authorizes groups of developers to work on specific classes, packages, and projects. Developers customize their environment by adding to their workspace copies of packages (editions) that contain the classes they need to work on. During the development cycle, programmers create open editions and scratch editions of packages to make changes to code and run tests. But each package and each class has a single owner, or manager, who is responsible for releasing stable versions at appropriate times.

Familiarity with the following terms is essential to understanding how to manage your code repository:

Workspace

In the simplest terms, a workspace is a customized view of items in the repository, limited to the packages that contain the parts you need to change or test. All developers have a file stored on their workstations that defines their workspace. The default image file name is ide.icx. When you save your workspace, this file is updated

to track which editions and versions of projects and packages you have added. The next time you start VisualAge Generator, all the projects and packages displayed in your workspace when you saved it last will be listed on the **Projects** and **Packages** tab of the Workbench.

Edition

In general, editions are packages that are subject to change. The two kinds of editions you will encounter most often are:

- Open editions—packages that have not been frozen. You can save classes to open editions in your workspace, and you can recognize them by the date and time stamp enclosed in parentheses next to their names in the Workbench. Open package editions can have new classes added to them, classes deleted from them, and different versions of existing classes released into them.

  Note: Select the **Show Edition Names** button on the toolbar to include edition information beside package and project names.

- Scratch editions—copies of packages that are created automatically when you save changes to an edition that is not open. Scratch editions allow you to make changes for testing existing classes in packages owned by other developers, but you cannot add new classes to a scratch edition. Unlike open editions, scratch editions are displayed in the Workbench with a version number in single angle brackets (<>) beside the edition name. Changes made to scratch editions only exist in the image of the developer who made them and they cannot be released into a package that does not have an open edition.

Version

Versions are editions that have been frozen by the application manager to prevent further changes to that level of code. They have version numbers (without angle brackets) instead of date and time stamps next to their names in the **Packages** pane of the VisualAge for Java Workbench.

For hands-on practice loading packages, creating new editions, and other library management tasks, complete the steps outlined in "Chapter 8. VisualAge Generator Developer on Java: a tutorial" on page 65.

Part types

Parts used with VisualAge Generator are described in the following table.

Part type	Part class	Part description
Program 	VAGenPrograms	A program is a set of related parts that VisualAge Generator can generate into executable form.
Function 	VAGenFunctions	A function is a block of logic consisting of a set of statements. Functions can be used as subroutines or to perform input or output (I/O) operations. Processing operations or I/O options are provided as high-level verbs. Some examples are: • **Converse** – display a screen and wait for input • **Add** – add information to a file or database • **Scan** – read information from a file or database
Map 	VAGenMaps	A map defines the layout and characteristics for all or part of the information presented on a character-based screen or printout when users run associated programs.
Map Group 	VAGenMapGroups	Each map is part of a named collection called a map group. All maps used in an application must be in the same map group, except for help maps, which can be in a separate map group.
Record 	VAGenRecords	A record is a collection of data items (a data structure) organized to serve a specific purpose. Records are analogous to rows in a database table. Records can be used to describe the layout of information in memory, in a database table, or in a file.
Table 	VAGenTables	A table is a collection of related data items that can be used to edit data, store messages that a program issues, and store information for reference by an application system.
PSB 	VAGenPSBs	A program specification block (PSB) describes the hierarchical database structures that an application system uses to access DL/I databases.
Data Item 	VAGenDataItems	A data item describes the characteristics of a single unit of information in a record or table.

Part type	Part class	Part description
Generation Options 	VAGenOptions	Generation options are parts managed in the library that list specifications used to customize generation for different environments.
Linkage Table 	VAGenLinkages	A linkage table lists specifications required for calls to non-VisualAge Generator programs or calls to VisualAge Generator servers on remote systems.
Resource Associations 	VAGenResources	Resource associations specify default overrides used during generation of COBOL or CICS/6000 programs that use printer maps or serial, indexed, or relative files.
Bind Controls 	VAGenBindControls	Bind controls list commands that control DB2 applications. They are used in the COBOL program generation process.
Link Edit 	VAGenLinkEdits	Link edits contain program-specific control statements used in generation of programs that run in MVS, VSE, and VM environments and call or are called by other programs using static COBOL calls.

VAGen Parts Browser

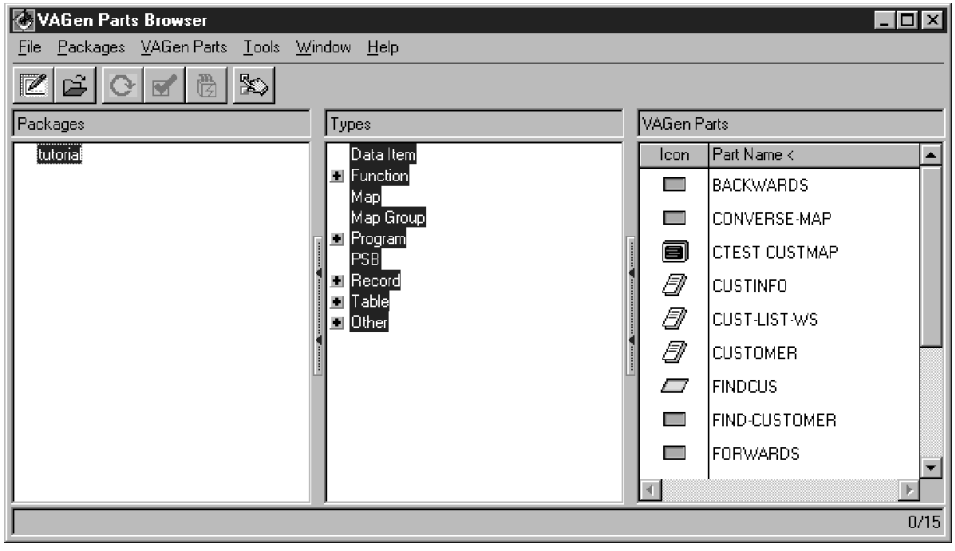


Figure 5. VAGen Parts Browser

The VAGen Parts Browser provides you with a complete listing of all of the VAGen parts loaded in your workspace/image. You can also control how the list is displayed. Using the Parts Browser, you can display VAGen parts by package/application or part type. You can also sort them by name.

If you are using VisualAge Generator Developer on Java, you can open the VAGen Parts Browser from the Workbench by selecting **Workspace→Open VAGen Parts Browser**.

If you are using VisualAge Generator Developer on Smalltalk, you can open the VAGen Parts Browser from the VisualAge Organizer window by selecting **VAGen Parts→Parts Browser**.

To begin building new VAGen parts, you'll use the New VAGen Part window, shown in Figure 6 on page 23.

In this window, you'll name your new part, choose the part type, and select a package/application to contain it.

If you are using VisualAge Generator Developer on Java, you can display the New VAGen Part window from the VAGen Parts Browser by selecting **VAGen Parts→Add→New Part**.

If you are using VisualAge Generator Developer on Smalltalk, you can display the New VAGen Part window from the VAGen Parts Browser by selecting **VAGen Parts→New**.

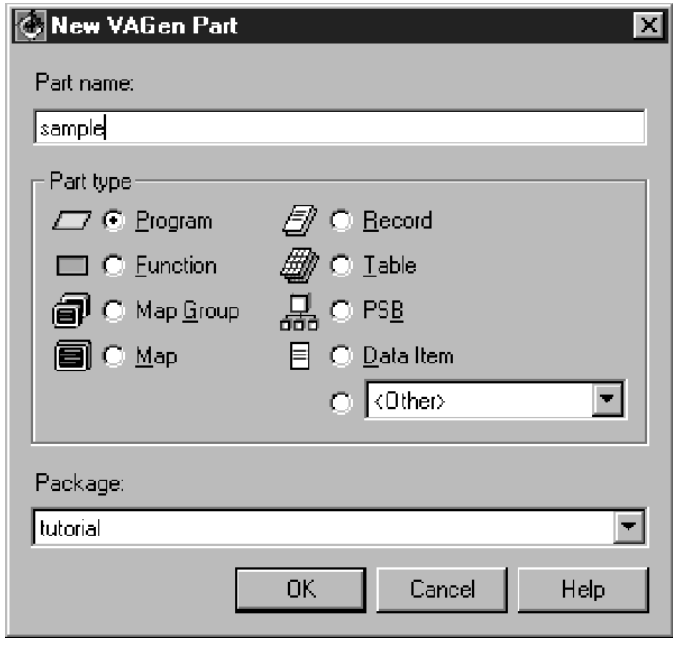


Figure 6. New VAGen Part

For hands-on practice developing VAGen parts, complete the steps outlined in "Part 2. VisualAge Generator Tutorial" on page 51.

VisualAge Composition Editor

The VisualAge Composition Editor, shown in Figure 7 on page 24, is used to build VisualAge visual and nonvisual parts. A visual part has a visual representation at run time. It can include other visual parts, such as windows, labels, text parts (text entry fields), and push buttons.

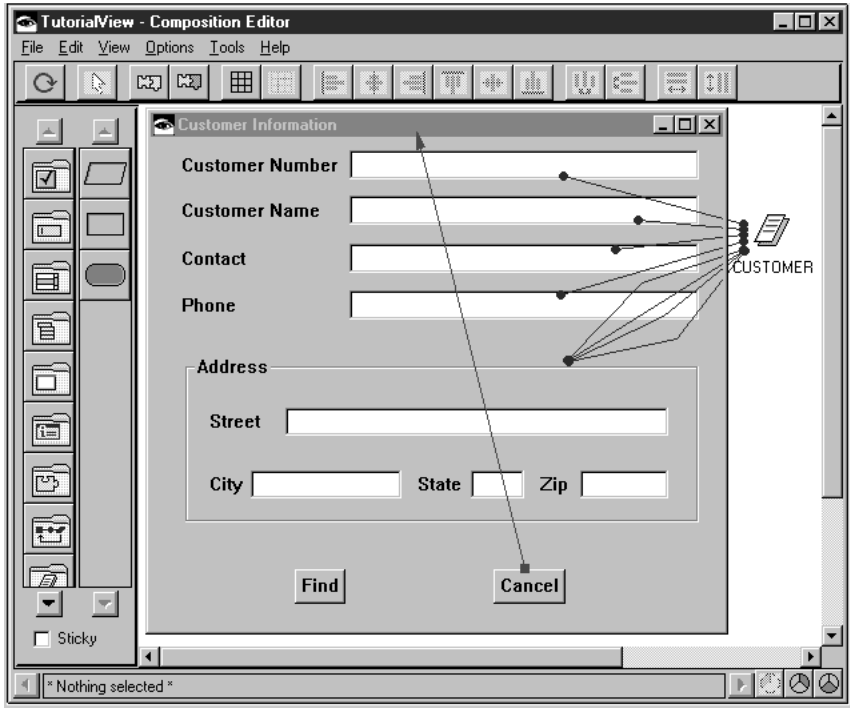


Figure 7. Composition Editor on Smalltalk

It can also include nonvisual parts—parts that don't have visual representations at run time. A set of VAGen nonvisual parts are available to be used with the Composition Editor. As you build your user interface, the Composition Editor enables you to visually define connections that control how that interface communicates with other parts.

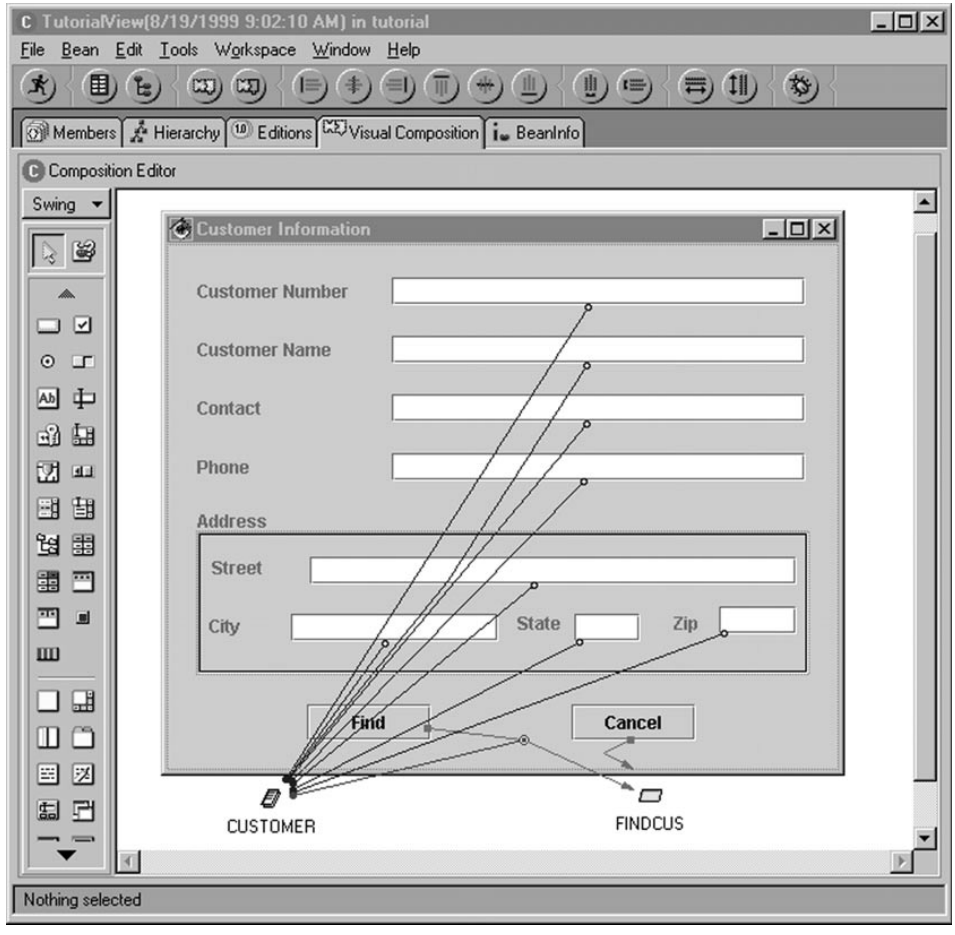


Figure 8. Composition Editor on Java

The Composition Editor provides you with the following:

Tool bar

The horizontal row of icons that appear underneath the menu bar. The tool bar provides convenient access to common functions, such as aligning and sizing your visual parts.

Beans/Parts palette

The two vertical columns of icons that appear on the left side of the screen contain the parts you use to construct your graphical user interface. The left column is the set of part categories, and the right column is the set of parts within the selected part category.

Status area
The scrollable static text field at the bottom of the screen. This field displays information about your last definition operation or your current selection.

Free-form surface
The large open area in the middle of the screen where you place parts to visually define a GUI.

You can specify the user interface characteristics and logic flow visually by doing the following:

- Selecting visual parts from the Beans/Parts palette and placing them on the free-form surface. Visual parts, such as push buttons, entry fields, and windows, are the basic components of user interfaces.
- Defining connections between visual parts and VisualAge Generator parts. For example, the GET-NEXT VAGen function is connected to the Find button. When users click the Find button, the GET-NEXT function will be performed.
- Specifying settings options to further refine the appearance and function of the part.
- Building VisualAge Generator parts, such as records and tables, to be used as your data structures and functions and programs to be used as the procedural logic for your application systems.

Refer to the online help for VisualAge for Java or the *VisualAge for Smalltalk User's Guide* for more information on designing visual parts.

For hands-on practice building visual parts, complete the steps outlined in "Chapter 17. Building a visual part using VisualAge for Java" on page 137 or in "Chapter 18. Building a visual part using VisualAge Generator Developer on Smalltalk" on page 159.

VAGen Script Wizard

The VAGen Script Wizard (shown in Figure 9 on page 28) is an extension provided by VisualAge Generator to augment the capabilities of the Composition Editor. It enables you to quickly and easily compose simple scripts that get or set property values, or invoke methods in your parts. These scripts use the syntax of an object-oriented language (shipped as part of your development environment) without requiring you to master a new programming language. These scripts can be invoked from VAGen functions and give you access to non-VisualAge Generator parts, like graphical user interface objects or Enterprise JavaBeans (EJBs), during execution of your 4GL logic. Using the wizard can help reduce the number of static connections you make between objects on the free-form surface. It simplifies maintenance and can improve the performance of your visually composed clients.

Before you compose a script, lay out your parts visually and save the part or bean. To display the wizard, if you are using VisualAge Generator on Java, select the Members tab, followed by the Members menu, and then VAGen Script Wizard. If you are using VisualAge Generator on Smalltalk, select the Script Editor icon, followed by the VAGen Script Wizard icon.

Methods developed with the VAGen Script Wizard are called VAGen scripts or object scripts and are associated with the instance of the class (the GUI client) in which you develop them. You can access these scripts by including the reserved word EZESCRPT in a VAGen function and specifying the literal script name or a data item that contains the script name. Scripts invoked this way cannot require arguments. EZESCRPT cannot be used to pass arguments to your scripts, but you can share information between your GUI client and your 4GL parts by storing data in a record placed on the free-form surface of the GUI client where you are developing the script.

The VAGen Script Wizard steps you through the process of naming a script, declaring local variables, choosing patterns (including get properties/attributes, set properties/attributes, invoke method/send message, and perform action), and selecting objects to get data from and pass data to. For example, if a condition specified in your VAGen function is not met by input received from a GUI client, you can indicate which field is in error by using a VAGen Script to change the color of that field.

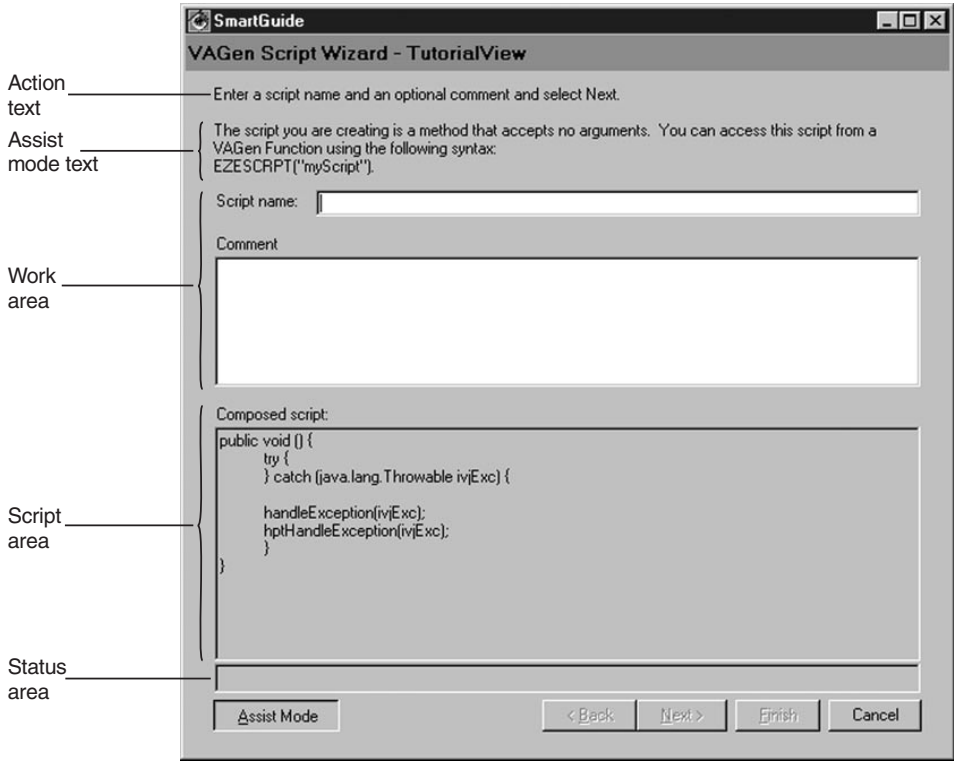


Figure 9. VAGen Script Wizard

Areas

The VAGen Script Wizard has five areas on each window that provide you with information or allow you to enter data and make selections:

Action text

The action text is displayed at the top of each panel and directs you to make a selection or type text for the current step.

Assist mode text

The Assist mode text provides you with additional information about what to do in each step and, where appropriate, gives examples. When you start the VAGen Script Wizard, the Assist mode text is displayed. You can turn it off and on by selecting the **Assist Mode** button.

Work area

The work area displays a set of fields where you specify logic and data to be used in your script. Depending on the step you are working on, different fields are displayed and the fields change as

you develop your script. See the section on the panel you are working with for more specific information about this area.

Script area

The script area displays the current state of your script. This area is for display only, but if you know the syntax for the statement you want to add, you can select **Free-form statement** from the work area and type statements directly into an edit window.

You can also use the **Back** button to sequentially remove changes that you added, change a step, and then use the **Next** button to add your selections back in.

Status area

The status area located near the bottom of the window provides you with valuable information about your selections as you make them. For example, if the **Next** button is still not available after you make a selection, the status bar prompts you for another action.

VAGen part editors

The editors enable you to create the various parts of a program, such as business logic, interface definitions, and data specifications. The editors detect and prevent errors before you test them or generate applications by providing statement templates and validation support.

Each VAGen part belongs to one of the following categories:

- Logic
- Maps
- Data

Logic

Use the logic editors to build VAGen parts that can access records, display user interfaces, manipulate data, and perform decision-making processes. VAGen logic parts are programs and functions that use the data and user interface parts you define.

Program Editor

A VAGen program contains logic, data, and elements that define the program's structure.

Using VisualAge Generator Developer, you can view and work with the hierarchical structure of a program in a graphical form called a program diagram. Figure 10 on page 30 shows an example of a program diagram.

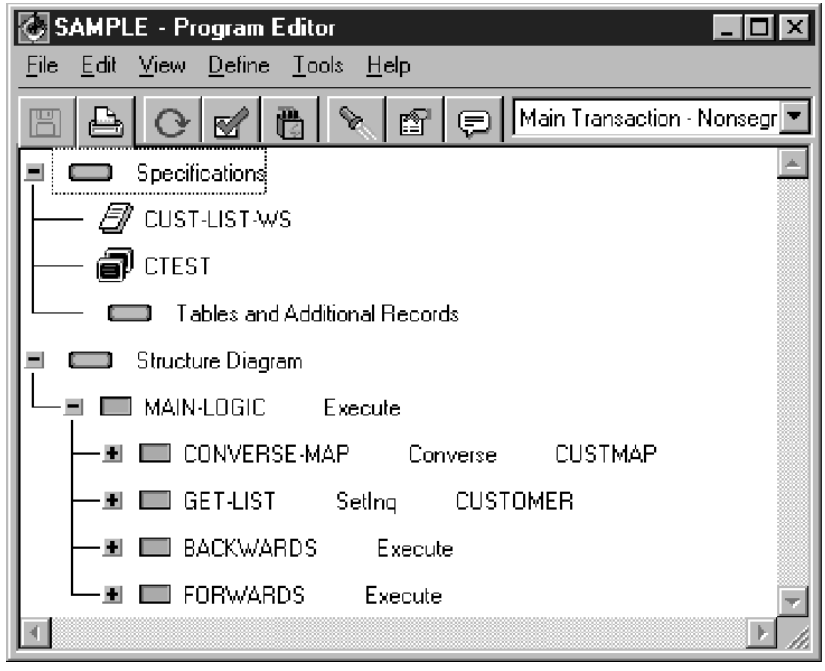


Figure 10. Program Diagram

From the program diagram you can get to all the related VAGen parts. Double-click on any VAGen part to open it in the appropriate editor. You can also add various new parts to the Program Editor using one of many context menus, or create new parts by selecting **File→New** to display the New VAGen Parts window.

For hands-on practice defining a program, complete the steps outlined in "Part 2. VisualAge Generator Tutorial" on page 51.

Function Editor

Logic is defined in groups of statements called functions. A VAGen program consists of functions and their associated data and user interface parts. Each function contains statements that handle the data for a specific task.

Processing logic in VisualAge Generator does the following:

- Accesses files and databases
- Moves data among records, user interface parts, files, and databases
- Displays character-based screens and prints reports
- Performs arithmetic calculations and other types of operations

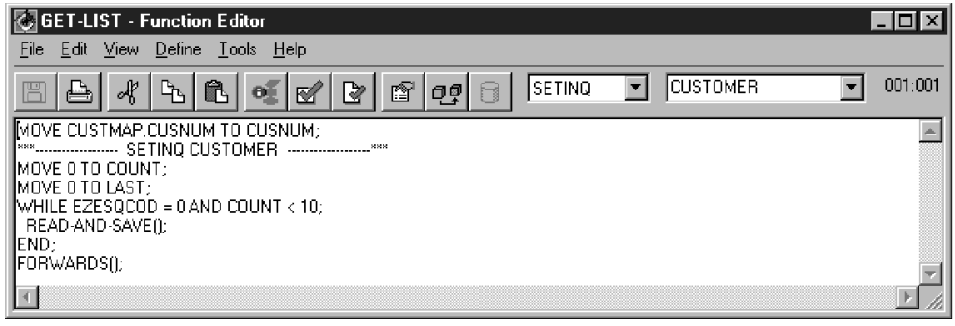


Figure 11. Function Editor

You can also customize SQL statements and add them to some functions using the SQL Statement Editor. Depending on the SQL statement you want to edit, a window like the one shown in Figure 12 on page 32 will be displayed for you. For more information on which clauses you can modify in each SQL statement, refer to "SQL Statement Editor" in the VisualAge Generator Developer help facility index.

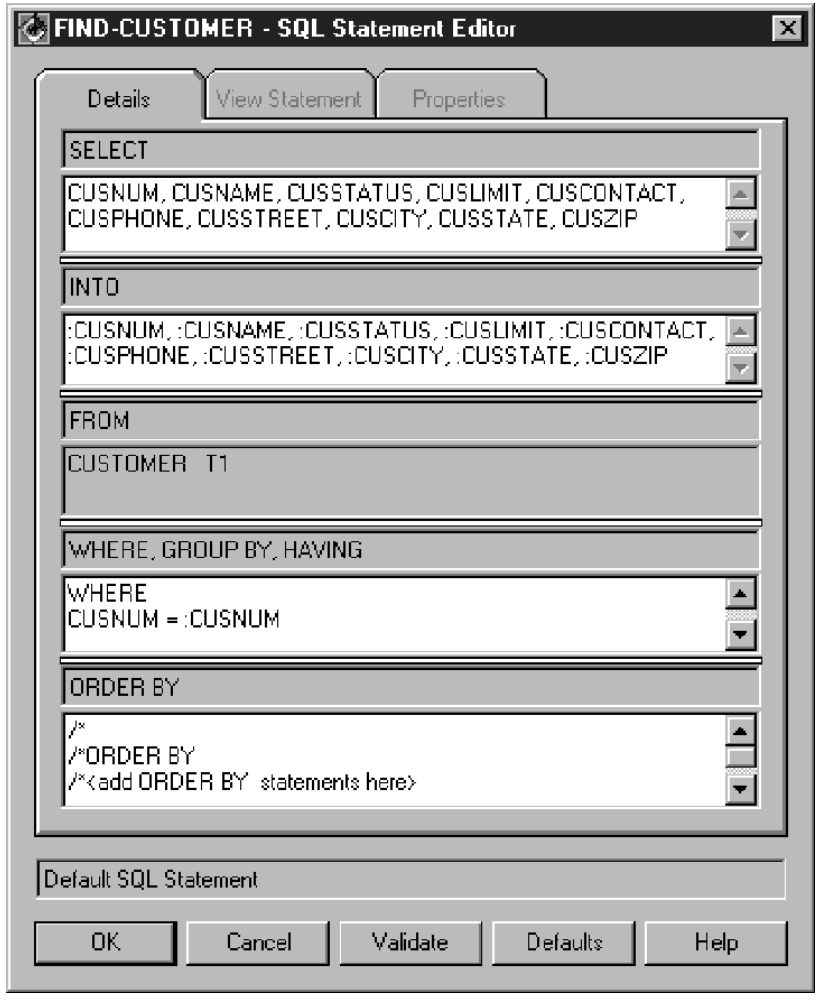


Figure 12. SQL Statement Editor

Templates are provided for all VisualAge Generator statements to help you create your statements as well as to help you avoid introducing syntax errors. Figure 13 on page 33 shows an IF statement template.

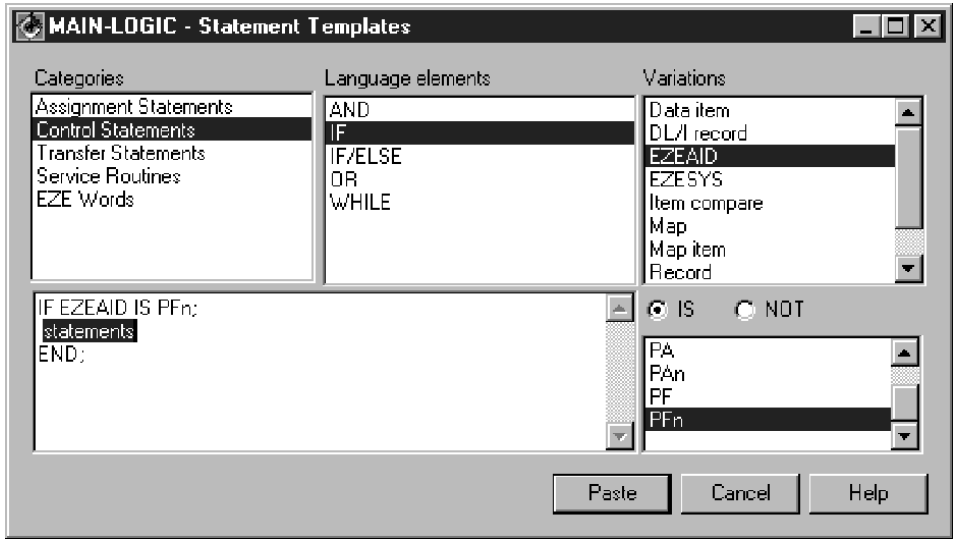


Figure 13. IF Statement Template

The Paste Part Name window provides you with a list of VAGen parts that are loaded into your workspace/image. Use this window to add the names of parts you want to use in functions. Figure 14 on page 34 shows a list of functions.

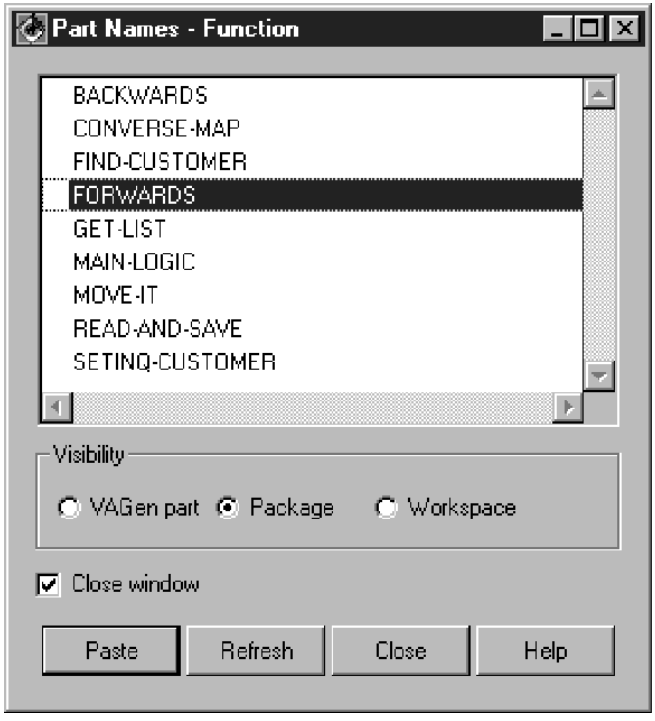


Figure 14. Paste Part Name window

For hands-on practice defining functions, complete the steps outlined in "Part 2. VisualAge Generator Tutorial" on page 51.

Maps

Use the map editor to define the layout and characteristics of character-based user interfaces and printed reports.

Map Editor

Map Editor provides you with a wide range of editing capabilities needed to define character-based user interfaces. Like the Composition Editor, the Map Editor's parts palette gives you predefined components like variable fields, constants, and arrays for designing SBCS, DBCS, and mixed format screens.

Figure 15 on page 35 shows a sample map.

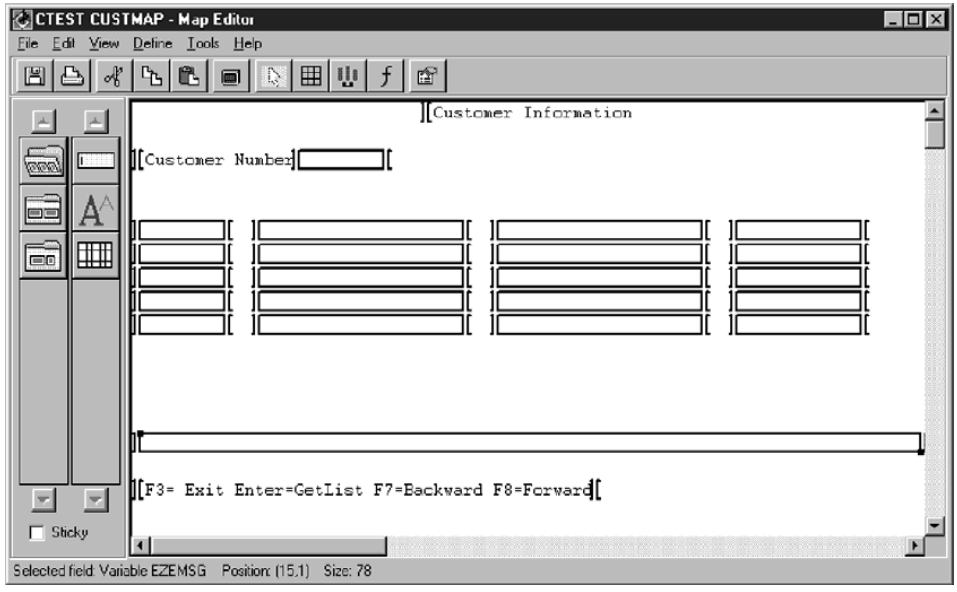


Figure 15. Map Editor

Map Group Editor

Map Group Editor enables you to view a list of maps. Using the Map Group Editor, you can define floating area characteristics for each device supported by one or more maps in the map group.

Figure 16 on page 36 shows a sample map group.

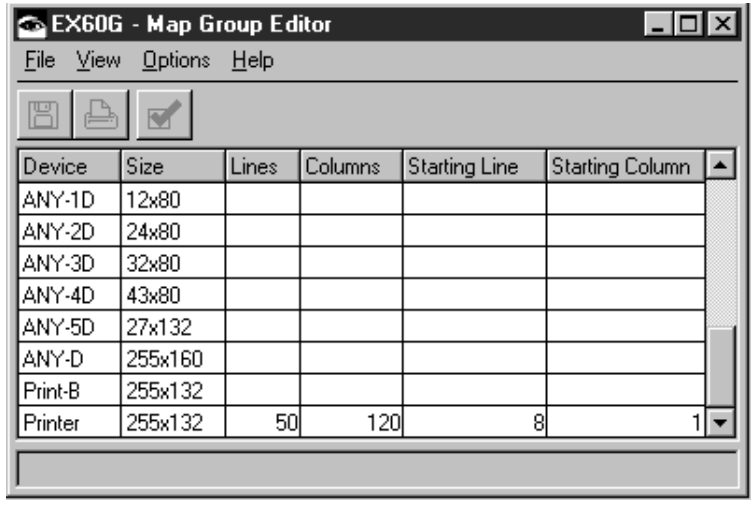


Figure 16. Map Group Editor

For hands-on practice building a map, complete the steps outlined in "Part 2. VisualAge Generator Tutorial" on page 51.

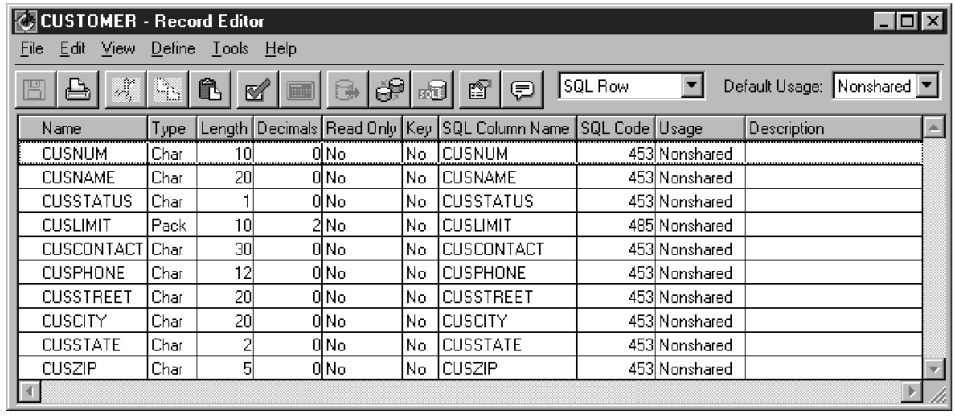
Data

Use the data editors to define the database, file information or user interface description data that a program can access. Data is stored as records, tables, program specification blocks (PSBs), and data items.

Record Editor

Record Editor enables you to build and format data structures intended for specific purposes. Using Record Editor, you can easily reformat database information for use in other programs.

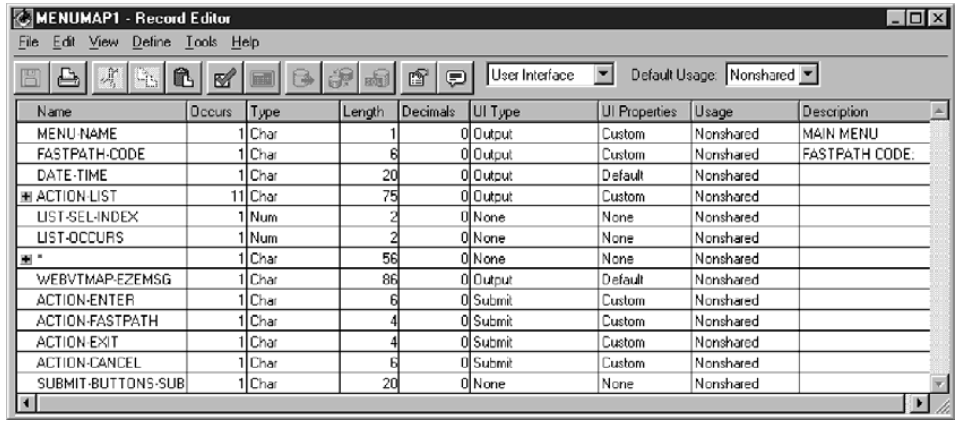
Figure 17 on page 37 shows a record that uses SQL Row format. This record can be defined manually or VisualAge Generator can fill the record with data from an existing SQL table.



Name	Type	Length	Decimals	Read Only	Key	SQL Column Name	SQL Code	Usage	Description
CUSNUM	Char	10	0	No	No	CUSNUM	453	Nonshared	
CUSNAME	Char	20	0	No	No	CUSNAME	453	Nonshared	
CUSSTATUS	Char	1	0	No	No	CUSSTATUS	453	Nonshared	
CUSLIMIT	Pack	10	2	No	No	CUSLIMIT	485	Nonshared	
CUSCONTACT	Char	30	0	No	No	CUSCONTACT	453	Nonshared	
CUSPHONE	Char	12	0	No	No	CUSPHONE	453	Nonshared	
CUSSTREET	Char	20	0	No	No	CUSSTREET	453	Nonshared	
CUSCITY	Char	20	0	No	No	CUSCITY	453	Nonshared	
CUSSTATE	Char	2	0	No	No	CUSSTATE	453	Nonshared	
CUSZIP	Char	5	0	No	No	CUSZIP	453	Nonshared	

Figure 17. Record Editor

Figure 18 shows a record that describes a user interface. Generating a UI Record yields a bean (a Java part) and a default HTML user interface.



Name	Occurs	Type	Length	Decimals	UI Type	UI Properties	Usage	Description
MENU-NAME	1	Char	1	0	Output	Custom	Nonshared	MAIN MENU
FASTPATH-CODE	1	Char	8	0	Output	Custom	Nonshared	FASTPATH CODE.
DATE-TIME	1	Char	20	0	Output	Default	Nonshared	
⊞ ACTION-LIST	11	Char	75	0	Output	Custom	Nonshared	
LIST-SELINDEX	1	Num	2	0	None	None	Nonshared	
LIST-OCCURS	1	Num	2	0	None	None	Nonshared	
⊞ *	1	Char	56	0	None	None	Nonshared	
WEBVTMAP-EZEMSG	1	Char	86	0	Output	Default	Nonshared	
ACTION-ENTER	1	Char	6	0	Submit	Custom	Nonshared	
ACTION-FASTPATH	1	Char	4	0	Submit	Custom	Nonshared	
ACTION-EXIT	1	Char	4	0	Submit	Custom	Nonshared	
ACTION-CANCEL	1	Char	6	0	Submit	Custom	Nonshared	
SUBMIT-BUTTONS-SUB	1	Char	20	0	None	None	Nonshared	

Figure 18. Record Editor

For hands-on practice defining records, complete the steps outlined in "Part 2. VisualAge Generator Tutorial" on page 51.

Table Editor

Use Table Editor to define a collection of related data items that can be used for the following:

- Editing data that a user enters on a map
- Storing messages that your program issues
- Storing information a program references when it runs

Figure 19 shows a sample table.

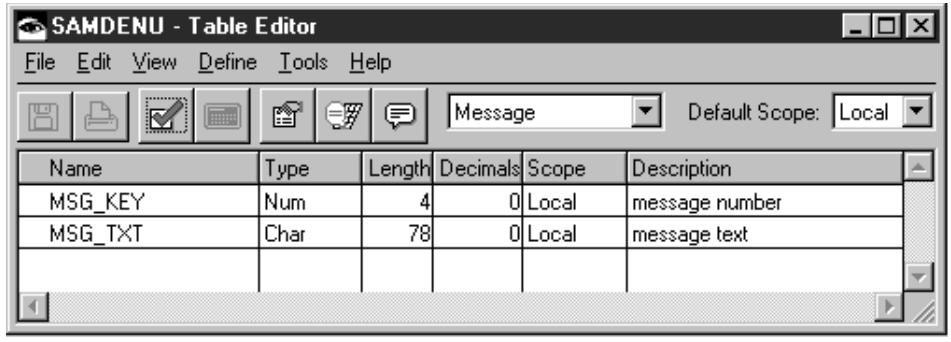


Figure 19. Table Editor

PSB Editor

Use the PSB Editor to add program communication block (PCB) entries to the program specification block (PSB). PSBs describe the hierarchical database structures of your DL/I databases. VisualAge Generator uses PSBs to build and validate DL/I calls for I/O functions that access records in DL/I databases.

Figure 20 shows a sample PSB.

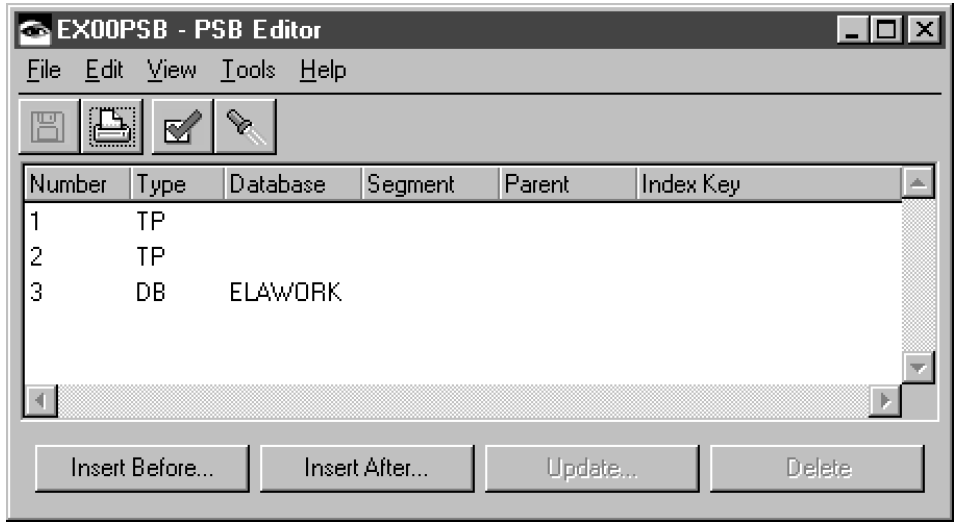


Figure 20. PSB Editor

Data Item Editor

Use the Data Item Editor to create global data items. Data items, both global and local, are units of information defined by data type, length, and other characteristics. You can define global data items that are independent of any record structure or table structure, but changes made to global data items in the Data Item Editor affect all records and tables that use them. Local data items are defined in records and tables.

Figure 21 shows a sample data item displayed in the Data Item Editor.

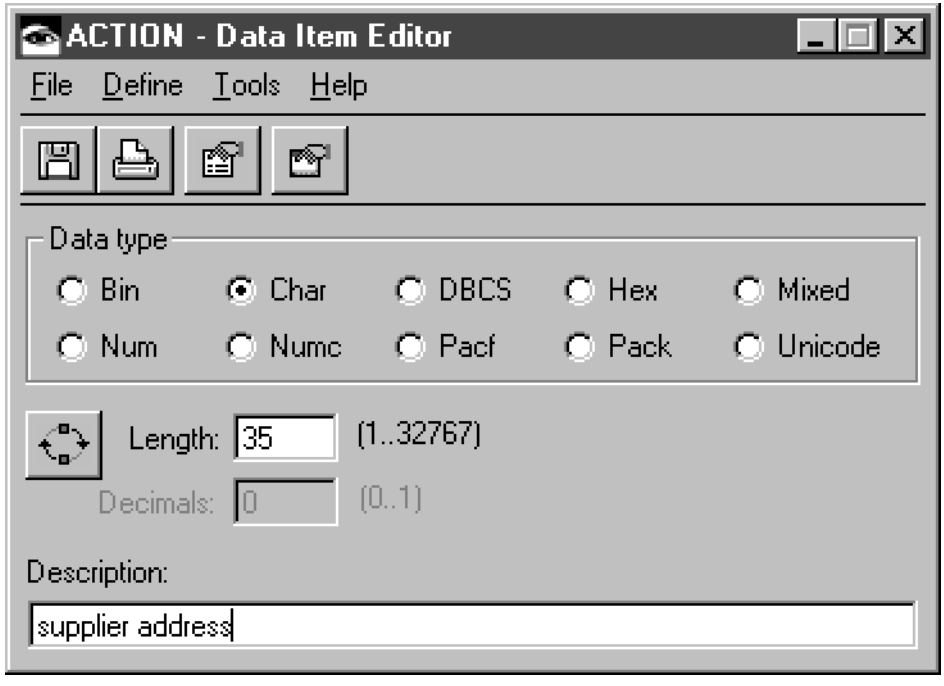


Figure 21. Data Item Editor

Test facility

The VisualAge Generator Developer test facility is a unique feature that dramatically streamlines the programming process. This facility enables you to view development and test functions as tightly integrated activities rather than distinct phases of development. This approach to prototyping and debugging programs is faster and more interactive.

Figure 22 shows the Test Monitor window.

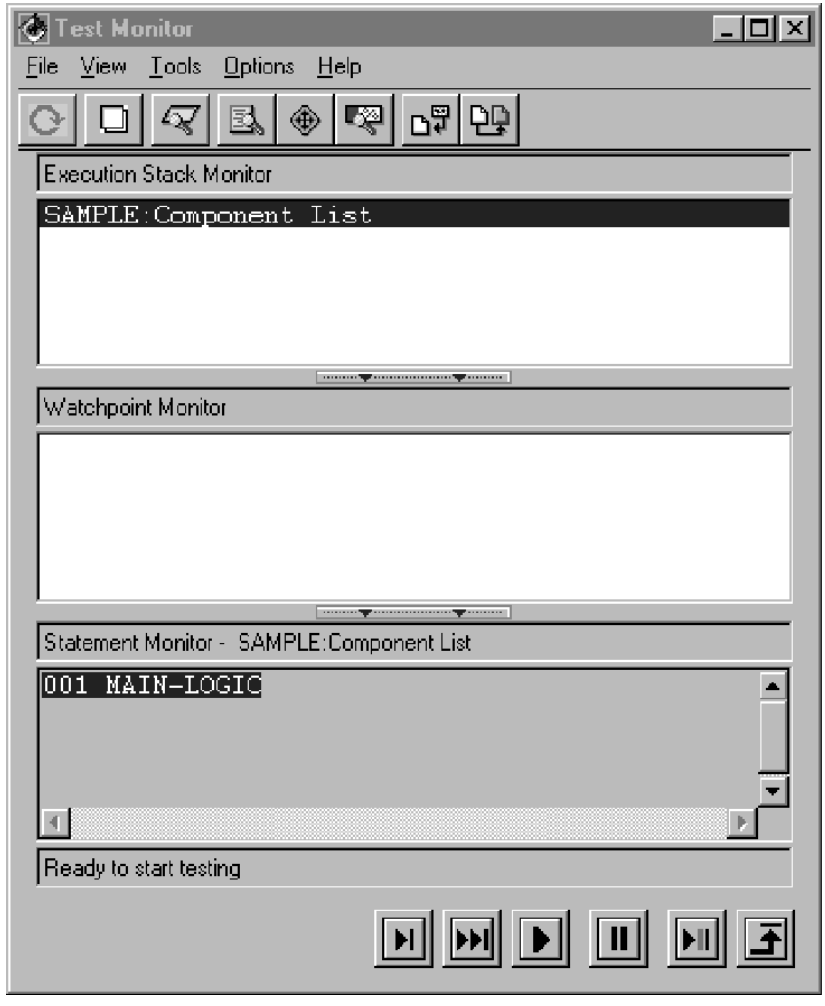


Figure 22. Test Monitor

The tasks supported by the test facility span a wide range of testing activities, from low-level debugging of new and unfamiliar programs to simple regression testing of existing programs. You are free to move between building parts and testing activities without jeopardizing any information that has already been set.

To aid in debugging programs, the test facility includes the following:

- Breakpoints to enable you to suspend the test
- Watchpoints to enable you to observe changes and, if necessary, dynamically make extensive revisions to data
- Tracepoints to enable you to trace specific program components

The test facility gives you a historical trace log that enables you to set filters on certain categories to be tracked. Using filters, you can get more detailed traces. You can also reset these filters after a test session and rebuild the trace log without running the test again.

Figure 23 shows the **VAGen Test Trace** page.

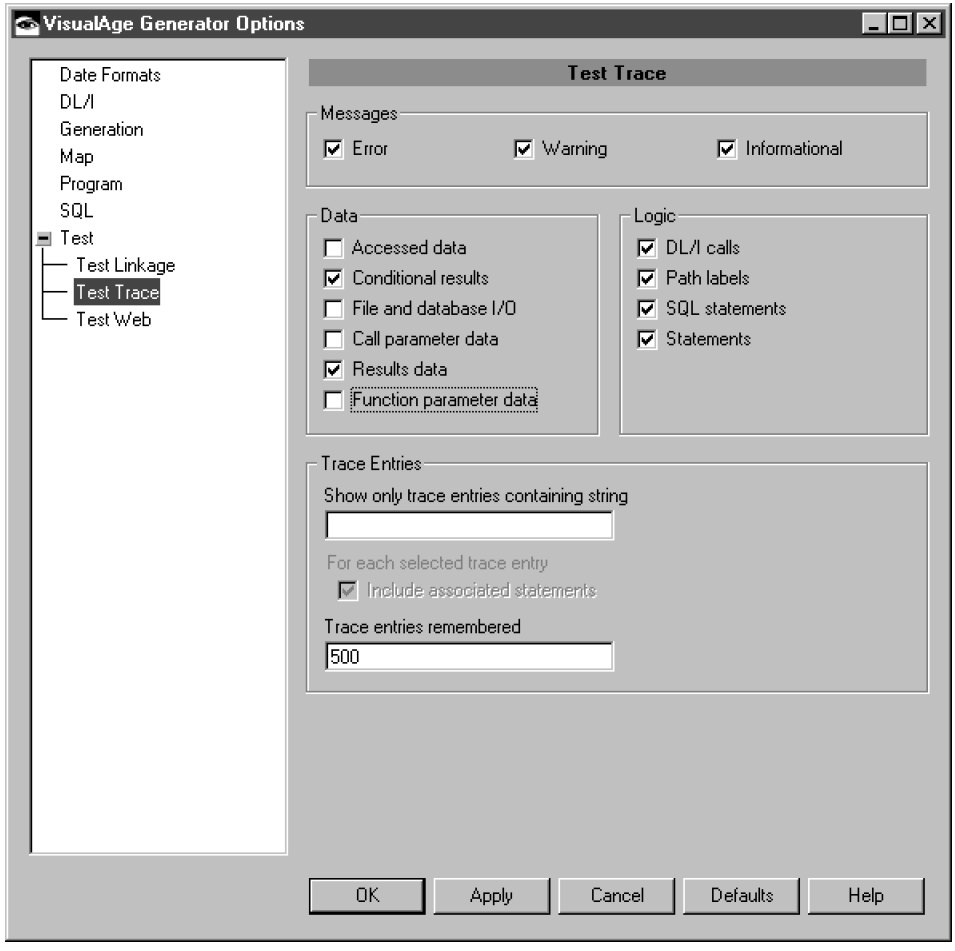


Figure 23. VisualAge Preferences— VAGen Test Trace

The test facility also enables you to test maps as well as logic parts. Map Monitor, a feature of the test facility, enables you to view your map as it will appear when the program is running.

During testing, a VisualAge Generator program on an OS/2 or Windows NT system can access relational data stored in DB2 directly or other databases through ODBC.

Also, during testing, your program can access generated VisualAge Generator programs and non-VisualAge Generator programs on local or remote systems.

VisualAge Generator program generation

The generation process uses VisualAge Generator (VAGen) part definitions, which are created using VisualAge Generator Developer and are stored in the repository. The part definitions and control files are used to generate COBOL or C++ programs, depending upon the target environment.

You can generate programs, tables, and maps from the user interface or from the command line using the HPTCMD command.

Figure 24 shows the Generate window where you can select the target system.

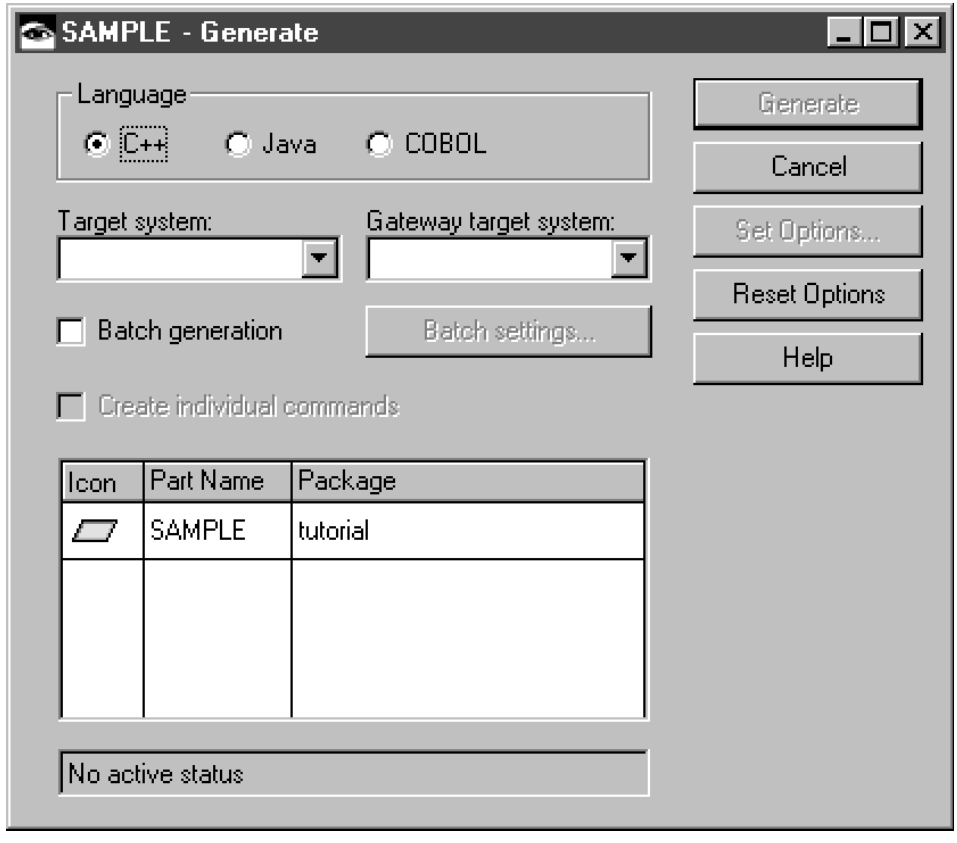


Figure 24. Generate package/application

The generation options part enables you to customize the generation process for specific target environments. You can specify generation options using the following methods:

- By entering values using the VisualAge Generator Developer user interface
- As options on the command interface when you are using the GENERATE subcommand
- By setting values for these options in generation options parts

Figure 25 shows the Generation Options window where you set generation options such as validation parameters.

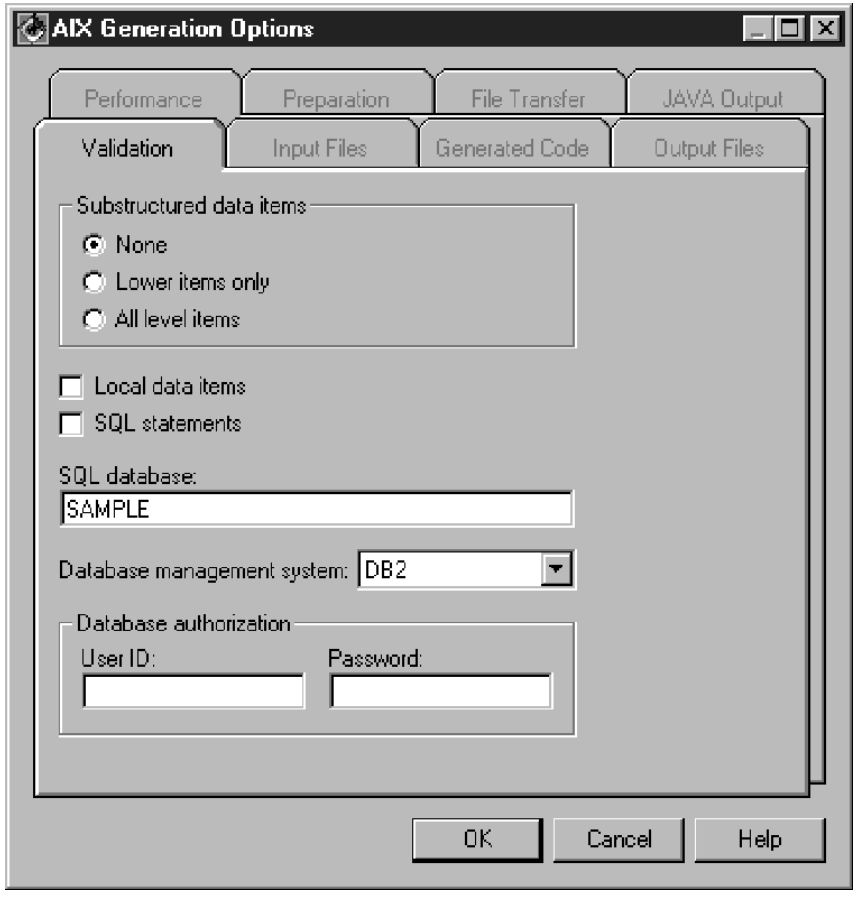


Figure 25. Generation Options

Generation options are stored in generation options parts, which enable developers working on the same project to share generation options. You can specify a default generation options part using the /OPTIONS option on the

GENERATE subcommand. A generation options part can also specify the /OPTIONS option, creating a succession of generation options parts.

For C++, Java, and COBOL programs, the generation process consists of the following processes:

Generation

Generation includes the following steps:

Validation

Collects definition information from parts for the object being generated. Validation also includes cross-checking the parts to ensure that the definitions are complete and correct. Information and error messages are issued when problems are detected. The generation process stops if there are errors. You can validate the parts without generating the program.

Production

Builds the source code and related program objects from the part definitions during this phase.

Preparation

Preparation includes the following steps:

- Transferring parts to the target environment
- Running precompilers, compilers, and linkers on the target system

The generation and preparation processes work together. On the development system, generation produces source code and related program objects. Then, on the target system, the preparation process prepares the generated source to run.

The VisualAge Generator Developer command interface syntax is specified by the command HPTCMD, followed by a subcommand. The VALIDATE subcommand specifies that you want validation only, and the GENERATE subcommand specifies that you want both validation and production. Using the VisualAge Generator Developer command interface, you can issue HPTCMD commands one at a time by specifying them on the command line, or you can create a command file to issue multiple HPTCMD commands.

VisualAge Generator Developer is enabled to the TeamConnection environment to provide TeamConnection functions, such as versioning, change control, problem tracking, and configuration management for VisualAge Generator members. The VisualAge Generator COBOL and C++ generators can be invoked by the TeamConnection build function.

The TeamConnection build environment for generation and preparation consists of the following:

- A TeamConnection server that provides access to the TeamConnection database for all TeamConnection clients.
- A TeamConnection client from which build requests are issued. Application development tools like VisualAge Generator Developer run using TeamConnection client code.
- One or more TeamConnection build servers that consist of the following:
  - A build processor that runs a build step. The build processor can reside on a workstation or on a host machine. The actual build steps (generation, compilation, and linking) are run by the build processor.
  - A build agent is connected to the TeamConnection database and to a build processor. Each build agent does the following:
    1. Responds to build requests of a particular type that have been stored in the TeamConnection database
    2. Sends the necessary input parts to its connected build processor
    3. Invokes the requested build operation on that processor
    4. Returns the outputs to the TeamConnection database

Because a build processor can be CPU intensive, it is recommended that you generate applications on a workstation other than one on which the TeamConnection server is installed and other than a workstation being used to develop and test applications with VisualAge Generator Developer.

Chapter 4. VisualAge Generator Server

After VisualAge Generator applications have been generated, prepared, and stored on the target system, the applications can be run with the support of VisualAge Generator Server for OS/2, AIX, Windows NT, HP-UX, and Solaris and VisualAge Generator Server for MVS, VSE, and VM.

VisualAge Generator Server for OS/2, AIX, Windows NT, HP-UX, and Solaris and VisualAge Generator Server for MVS, VSE, and VM provide the following support:

- Error management
- Message services
- Display services
- File and database services
- CICS services
- Segmentation storage services
- Application support services and functions common to every application
- Miscellaneous services

Running an application

The VisualAge Generator Server products provide client/server communications support, a runtime library, and procedures and utilities for preparing generated applications for run-time. Client/server communications support enables GUI applications to communicate with COBOL and C++ applications. The runtime library implements installation-specified error handling, transaction control, and other functions shared among generated applications.

VisualAge Generator Server for OS/2, AIX, Windows NT, HP-UX, and Solaris

VisualAge Generator Server provides support for running generated COBOL applications targeted for CICS for OS/2, and generated C++ applications targeted for OS/2. For more information, refer to the *VisualAge Generator Server Guide for Workstation Platforms*.

VisualAge Generator Server for AS/400

VisualAge Generator Server for AS/400 provides support for running generated COBOL applications targeted for the OS/400 environment. For more information, refer to the *VisualAge Generator Server Guide for AS/400* document.

VisualAge Generator Server for MVS, VSE, and VM

VisualAge Generator Server for MVS, VSE, and VM provides support for running generated COBOL applications targeted for the MVS, VSE, and VM environments. For more information, refer to the *VisualAge Generator Server Guide for MVS, VM, and VSE*.

Chapter 5. Summary

VisualAge Generator provides powerful solutions for creating applications. The VisualAge Generator products enable you to develop, test, and generate applications on your workstation and run applications on workstations or host systems.

For step-by-step introductions on defining and testing application systems:

- If you are using VisualAge Generator Developer on Smalltalk, see "Chapter 6. VisualAge Generator Developer on Smalltalk: a tutorial" on page 53.
- If you are using VisualAge Generator Developer on Java, see "Chapter 8. VisualAge Generator Developer on Java: a tutorial" on page 65.

Part 2. VisualAge Generator Tutorial

Chapter 6. VisualAge Generator Developer on Smalltalk: a tutorial

This tutorial gives you a broad look at what you can do with VisualAge Generator. After completing this tutorial, you will have built and tested a small sample program that enables a user to search a customer database using the customer number.

Before you can perform the steps in this tutorial, you must have VisualAge Generator installed and the appropriate features loaded. You must also have access to an installed IBM relational database like DB2 UDB. Some prerequisite knowledge of the IBM relational databases is required.

Before you start, do the following:
- Validate the VisualAge Generator installation.
- Verify the installation of the sample database, SAMPLE, and the CUSTOMER table.

  See "Appendix A. Installing samples for VisualAge Generator Developer on Smalltalk" on page 187 for instructions on installing the SAMPLE database.

This tutorial is intended to help you get started quickly with VisualAge Generator, and therefore covers only a small set of the tasks you can do. After you complete this tutorial, experiment and refer to the product documentation to learn more about what VisualAge Generator can do for you.

Note: If you have installed VisualAge Generator Developer on Java, refer to "Creating a project and a package" on page 65.

Creating an ENVY application

When you start VisualAge Generator, two windows will be displayed on your desktop: the System Transcript window and the VisualAge Organizer window. The System Transcript window has numerous functions, but for this tutorial, you will only need to look at it for system information. You'll start building your sample application system in the VisualAge Organizer window, shown in Figure 26 on page 54.

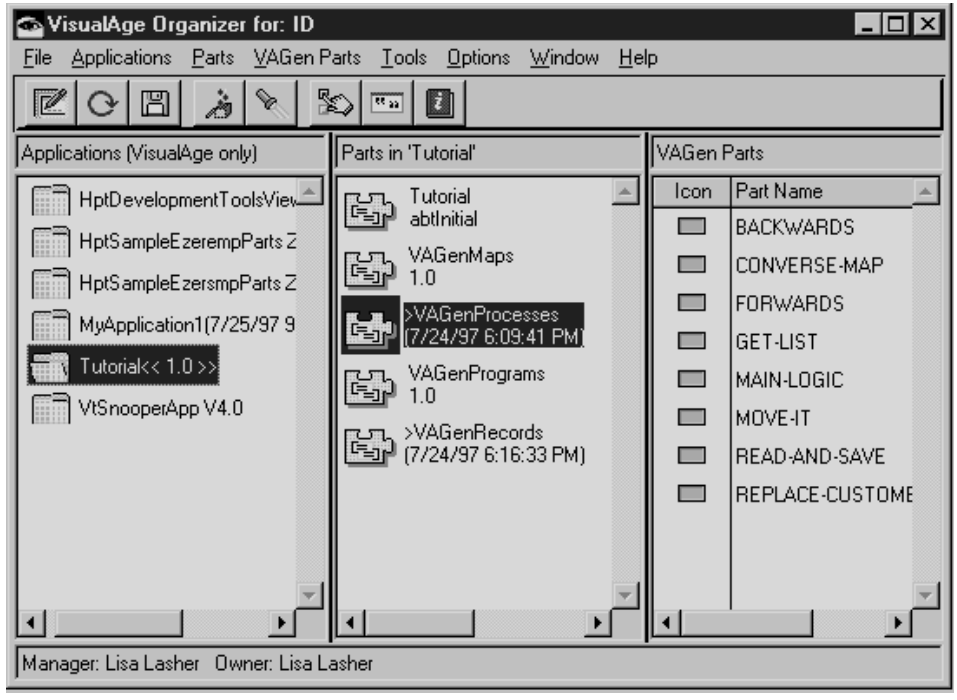


Figure 26. VisualAge Organizer Window

VisualAge Generator stores VAGen parts in ENVY applications. Before you build a part, you must create a new application or load an edition of an application to contain the parts you build.

Note: For a description of the 14 types of VisualAge Generator parts, see "Part types" on page 20. For more information about working with applications, see "VisualAge Smalltalk Repository management" on page 11 or the *IBM Smalltalk User's Guide*.

To create a new ENVY application called **Tutorial**, do the following:

1. On the VisualAge Organizer window, select **Applications→New**.
   The New Application window is displayed.

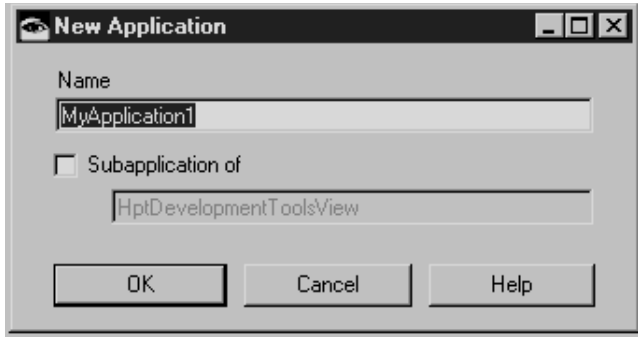


Figure 27. New Application Window

2. In the **Name** field, type `Tutorial`.
3. Select **OK**.

   The application called **Tutorial** is displayed in the **Applications** pane.

   Note: By Smalltalk convention, application names are capitalized. By Java convention, package names are lowercase. For the tutorial which follows, Smalltalk users should replace any references to the package/application named `tutorial` with `Tutorial`.

Importing and loading sample applications

As you develop your program, you might want to look at some VAGen sample parts. Applications that contain sample VAGen parts were stored on your system in .DAT files when you installed VisualAge Generator. To look at these sample parts, you'll need to import the .DAT file and load the sample applications it contains. For this tutorial you'll import from EZEREMPS.DAT and EZERSMPS.DAT and load the sample applications HptSampleEzerempParts and HptSampleEzersmpParts.

Importing sample applications

If your installation gives you access to a common repository (that is, you are working from a client, not from a standalone install), you will need two pieces of information to do the following tasks:

- The host name or IP address of the system where your repository (server) is located
- A directory to which you have access that is defined to the server

You'll then need to copy EZEREMPS.DAT and EZERSMPS.DAT to that directory and specify it at the appropriate point in the following task.

To import an application, perform the following steps:

1. From the **Applications** menu, select **Import/Export** → **Import Applications**.

A prompt is displayed asking for the IP address or host name of the machine where the .DAT files are located.

Note: If you are using a standalone system, a system file selection window is displayed, so you can skip the following step.

2. Enter the host name for the machine where EZEREMPS.DAT resides and select **OK**.

   A file selection window is displayed.
3. In the **File name** field, type the complete path name to `EZEREMPS.DAT`.
4. Select **Open**.

   A selection window is displayed, prompting you for the name and version of the application you want to import.
5. From the **Names** pane, select **HptSampleEzerempParts**.
6. In the **Versions** pane, select the version name and click on the top arrow button.
7. Select **OK**.

   The selection window closes. A statement indicating that the import is complete is displayed in the System Transcript window when the import is finished.
8. Repeat these steps but select EZERSMPS.DAT and **HptSampleEzersmpParts**.
9. Select **OK**.

   The selection window closes. A statement indicating that the import is complete is displayed in the System Transcript window when the import is finished.

Loading sample applications

Now that you have imported the sample applications, you must load them into your image to work with them.

To load the sample applications:

1. From the VisualAge Organizer window, select **Applications→Load→Available Applications**.

   Tip: You can use the context menus in the VisualAge Organizer window to make many tasks faster and easier. To display a context menu, click with mouse button 2 in the appropriate pane of the VisualAge Organizer window.

   The selection window, shown in Figure 28 on page 57, displays a list of all the available applications in the library that are not loaded into your image.

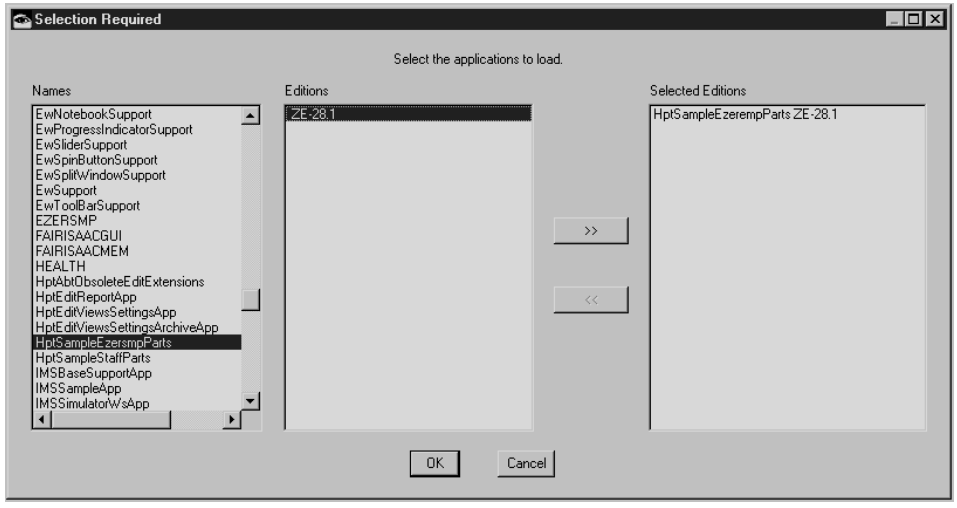


Figure 28. List of Applications

2. From the **Names** pane, select **HptSampleEzerempParts**.
3. In the **Editions** pane, select the edition name and click on the top arrow button.
4. Repeat the previous two steps to add **HptSampleEzersmpParts**.
5. Select **OK**.

   The applications you selected are displayed in the **Applications** pane of the VisualAge Organizer window. If you don't see them in the **Applications** pane, from the **Applications** menu, select **View→Show All Applications**.

Note: You cannot load all of the sample applications simultaneously because some of them contain VAGen parts with the same name. This way you cannot accidently overwrite parts you already have in your image with parts of the same name in another application. If you are working with samples, the easiest solution is to unload the sample application that contains the conflicting part names and load the other sample application. Under other circumstances, if the parts contained by two applications are identical, it is better to create a new application to contain the common parts, delete those parts from the two applications, and make the new application a prerequisite for the other two.

For more information on managing applications, refer to the *IBM Smalltalk User's Guide*.

Chapter 7. Tailoring preferences on Smalltalk

VisualAge Generator lets you set preferences that can save you time as you develop programs and build parts. These profiles are used as defaults and by setting them, you can tailor the environment to your needs. Preferences you set on the VAGen Preferences window are stored in `hpt.ini` when you save your Image.

Note: If you have installed VisualAge Generator Developer on Java, refer to "Chapter 9. Tailoring options on Java" on page 73.

In this tutorial, you will set the following preferences:
- Program preferences
- SQL Database preferences
- Test preferences

To open the VisualAge Generator Preferences window:

1. From the VisualAge Organizer window, select **Options→VAGen Preferences**.

   The Preferences window is displayed.

Program preferences

As in many programs, this sample program uses implicit data items. The system default is not to allow implicit data items. To change this default:

1. Select **Program**.
2. Select **Allow implicits**.

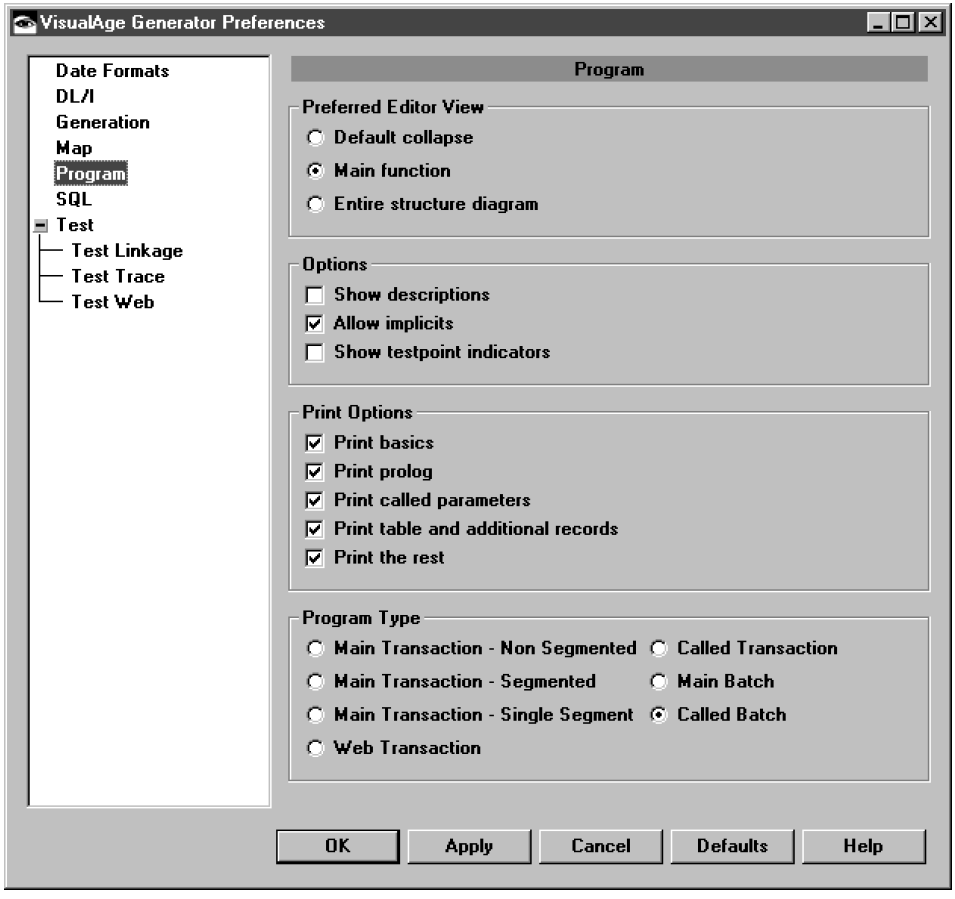


Figure 29. VAGen Program Preferences Window

Database preferences

The sample program you are creating uses an SQL database called **SAMPLE**. By selecting a default database in your preferences, you can ensure that you are connecting to the correct database. To select database option:

1. Select **SQL**.
2. Ensure that `SAMPLE` is entered in the **Database name** field.

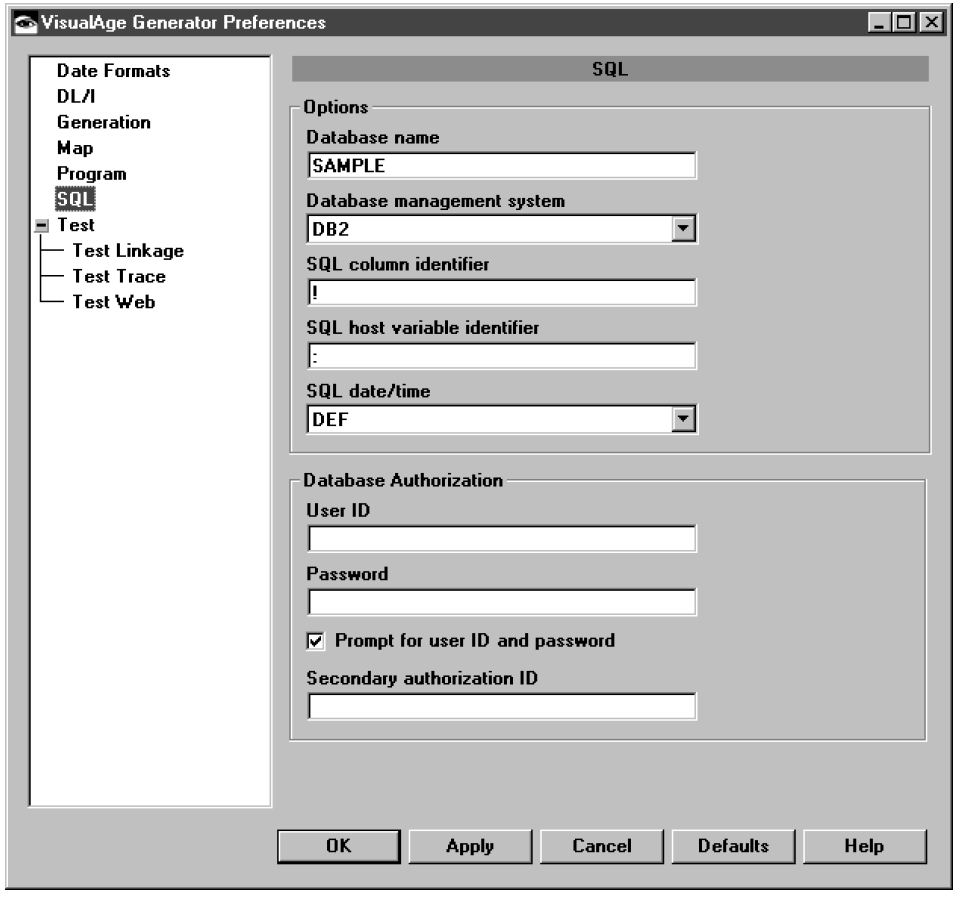


Figure 30. VAGen SQL Preferences Window

Test preferences

To set preferences for the VisualAge Generator test facility:

1. Select **Test** .
2. In the **Relative execution speed** field, use the arrow buttons to set the value to 3.

   The test facility runs relatively fast. You might want to reduce the test speed to give you more time to stop it if you need to.

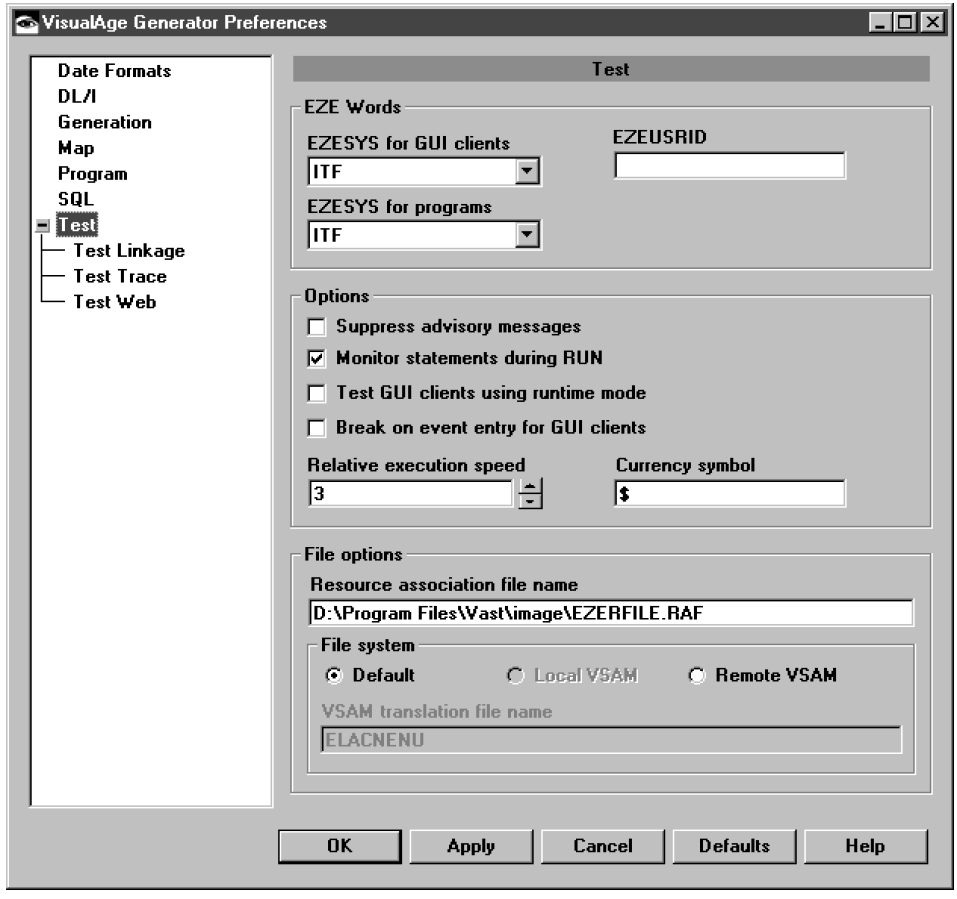


Figure 31. VAGen Test Preferences Window

You can use the test facility to collect trace data for your programs. Use the trace filters to control the types of trace data you see. You can change the filter settings in the test facility if you want to see other types of trace data. To set trace filters:

1. Select **Test Trace**.

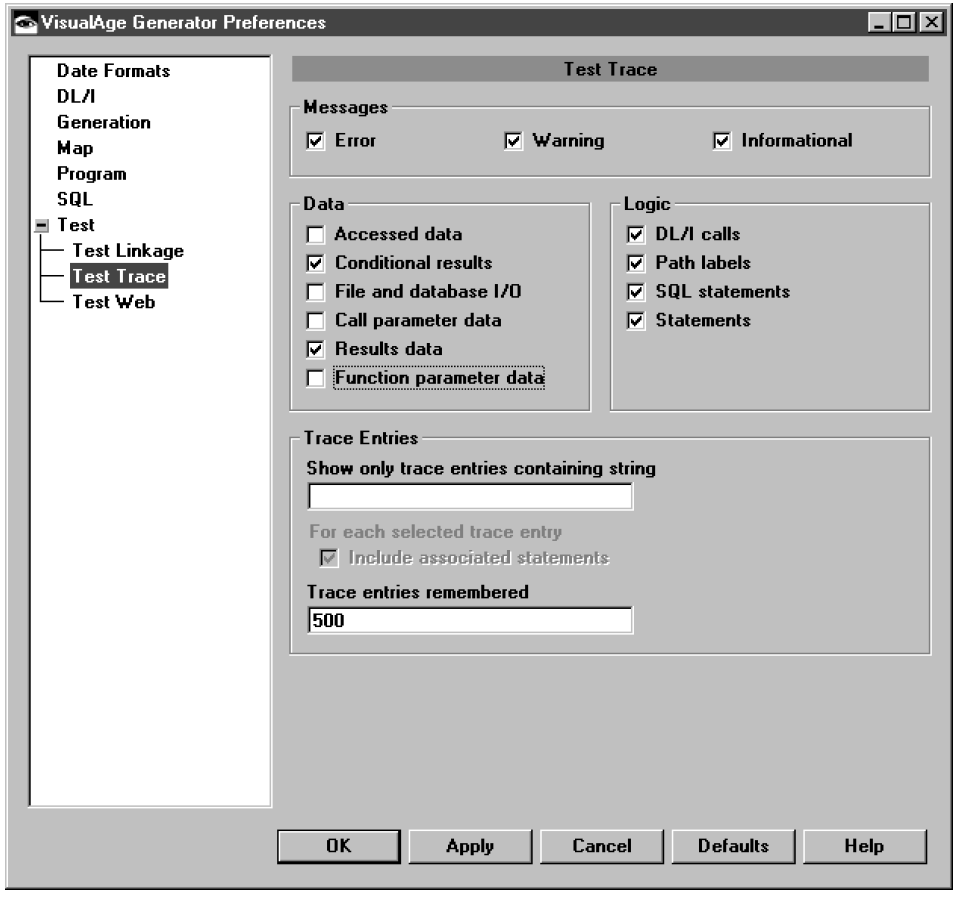


Figure 32. VAGen Test Trace Preferences Window

2. Deselect everything under the heading **Data** except **Conditional results** and **Results data**.
3. Select **OK**.

You can specify trace filters to view information, such as:
- The statement where the expression was evaluated or the assignment was made
- Results of expression evaluations and assignments

Selecting a trace entry filter causes all trace entries that match that filter's characteristics to appear in the Trace Log window. Setting a filter does not alter the information that is collected by the test facility.

Note: To continue with this tutorial, skip to "Chapter 10. Defining a program" on page 79.

Chapter 8. VisualAge Generator Developer on Java: a tutorial

This tutorial gives you a broad look at what you can do with VisualAge Generator. After completing this tutorial you will have built and tested a small sample program that enables a user to search a customer database using the customer number.

Before you can perform the steps in this tutorial, you must have VisualAge Generator installed and the appropriate features loaded. You must also have access to an installed IBM relational database like DB2 UDB. Some prerequisite knowledge of the IBM relational databases is required.

Before you start, do the following:
- Validate the VisualAge Generator installation.
- Verify the installation of the sample database, SAMPLE, and the CUSTOMER table.

  Refer to the "Appendix B. Installing samples for VisualAge Generator Developer on Java" on page 191 for instructions on installing the SAMPLE database.

This tutorial is intended to help you get started quickly with VisualAge Generator, and therefore covers only a small set of the tasks you can do. After you complete this tutorial, experiment and refer to the product documentation to learn more about what VisualAge Generator can do for you.

Note: If you have installed VisualAge Generator Developer on Smalltalk, refer to "Creating an ENVY application" on page 53.

Creating a project and a package

When you start VisualAge Generator, the VisualAge for Java Workbench is displayed. The Workbench has numerous functions, but for this tutorial, you will only need to do two tasks here. You'll start building your sample application system in the Workbench, shown in Figure 33 on page 66.

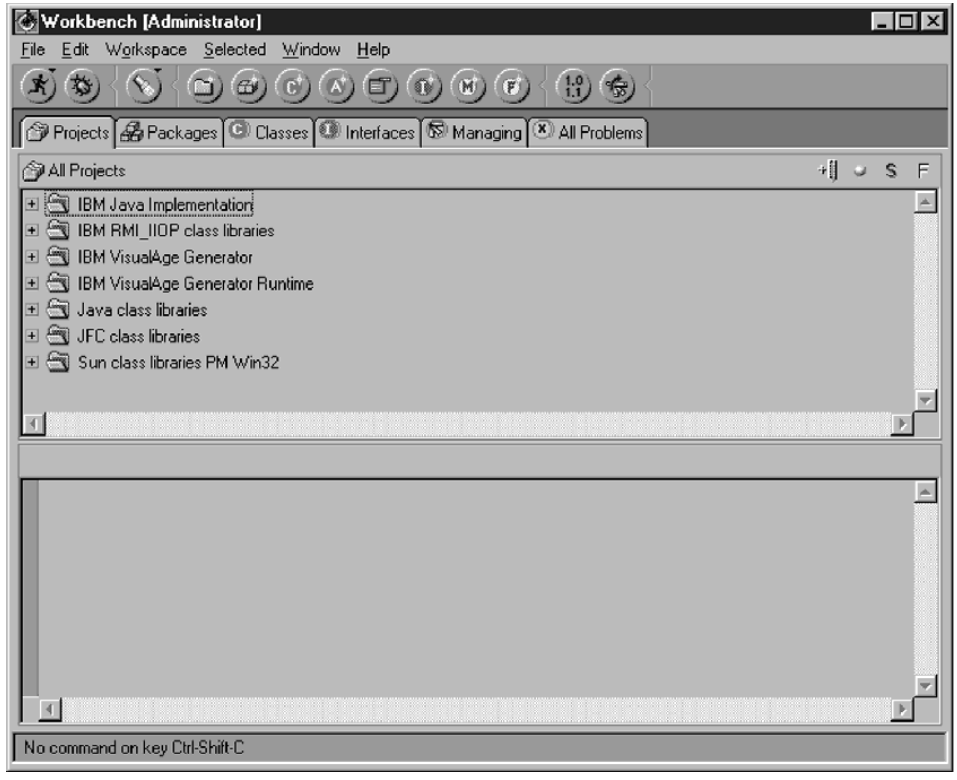


Figure 33. VisualAge for Java Workbench

VisualAge Generator stores VAGen parts in packages. Packages are stored in projects. Before you build a VAGen part, you must add a project and a package to your workspace to contain the parts you build. You can add an existing project and package from the repository or create a new one.

Note: For a description of the 14 types of VisualAge Generator parts you can create, see "Part types" on page 20. For more information about working with projects and packages, see the online help for VisualAge for Java.

For this tutorial, we'll create a new project called **VGTutorial** and a package called **Tutorial**. Do the following:

1. From the Workbench **Selected** menu, select **Add→ Project**.

   The Add Project window is displayed.

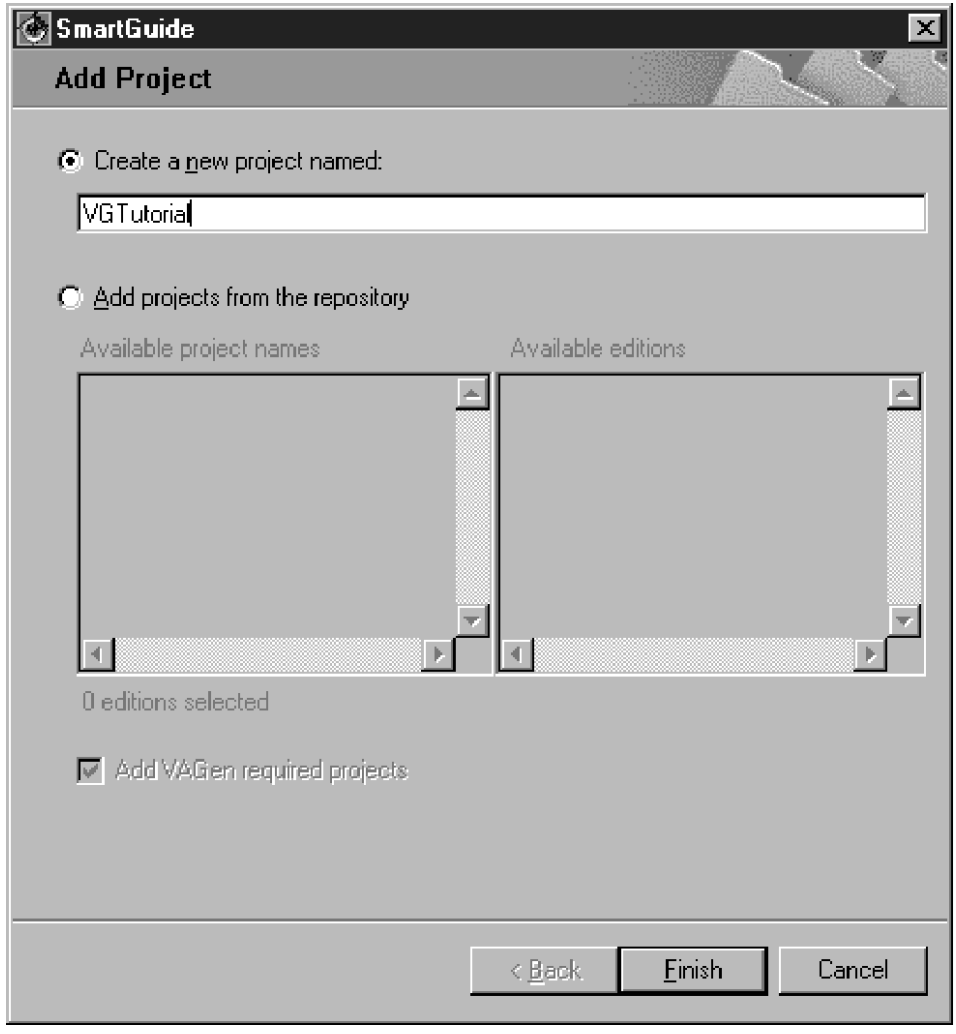


Figure 34. Add Project Window

2. In the **Create a new project named** field, type `VGTutorial`.
3. Select **Finish**.
   The project called **VGTutorial** is displayed on the **Projects** tab.
4. On the **Projects** tab, select **VGTutorial**.
5. From the **Selected** menu, select **Add→ Package**.
   The Add Package window is displayed.

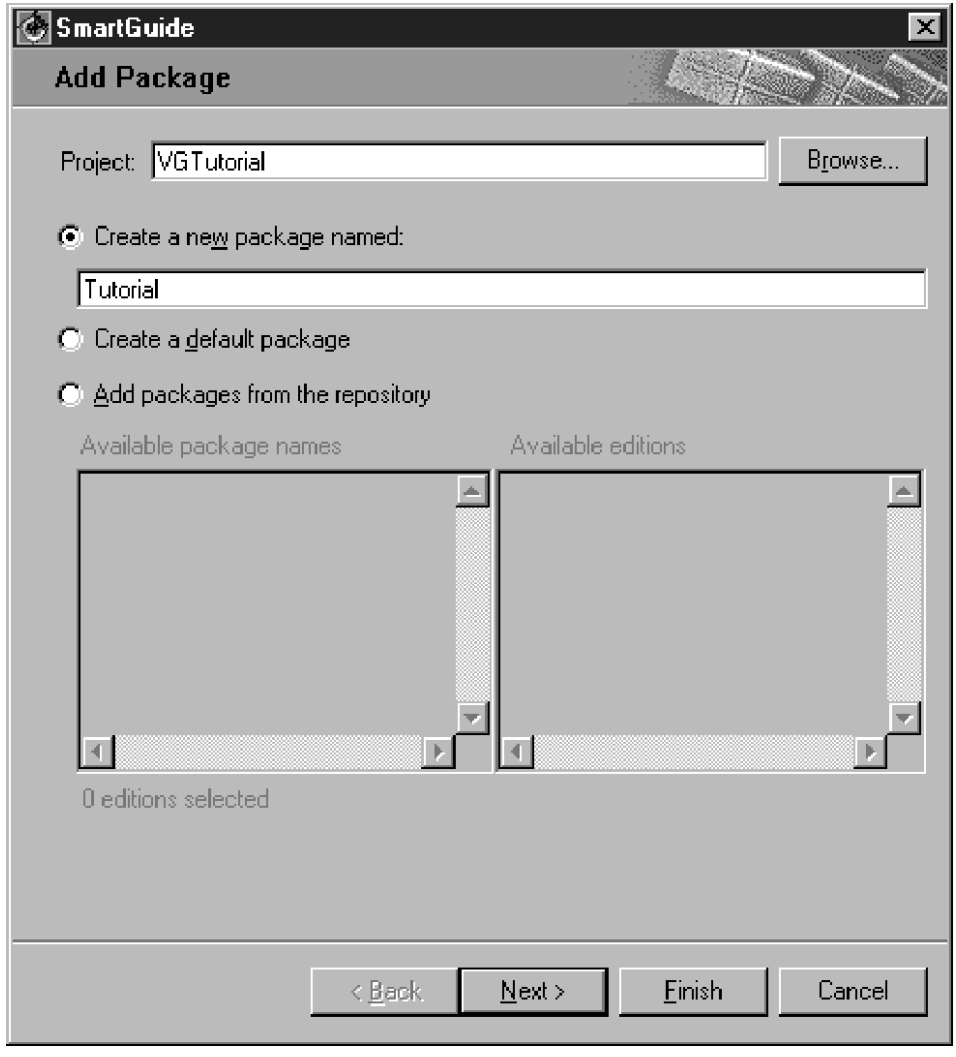
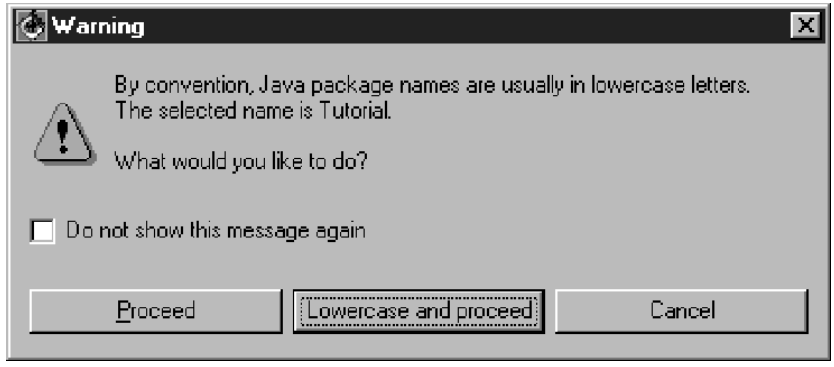


Figure 35. Add Package Window

6. In the **Create a new package named** field, type `Tutorial`.
7. Select **Finish**.
8. The following message is displayed:



Select **Lowercase and proceed**.

The package called **tutorial** is displayed under the **VGTutorial** project on the **Projects** tab.

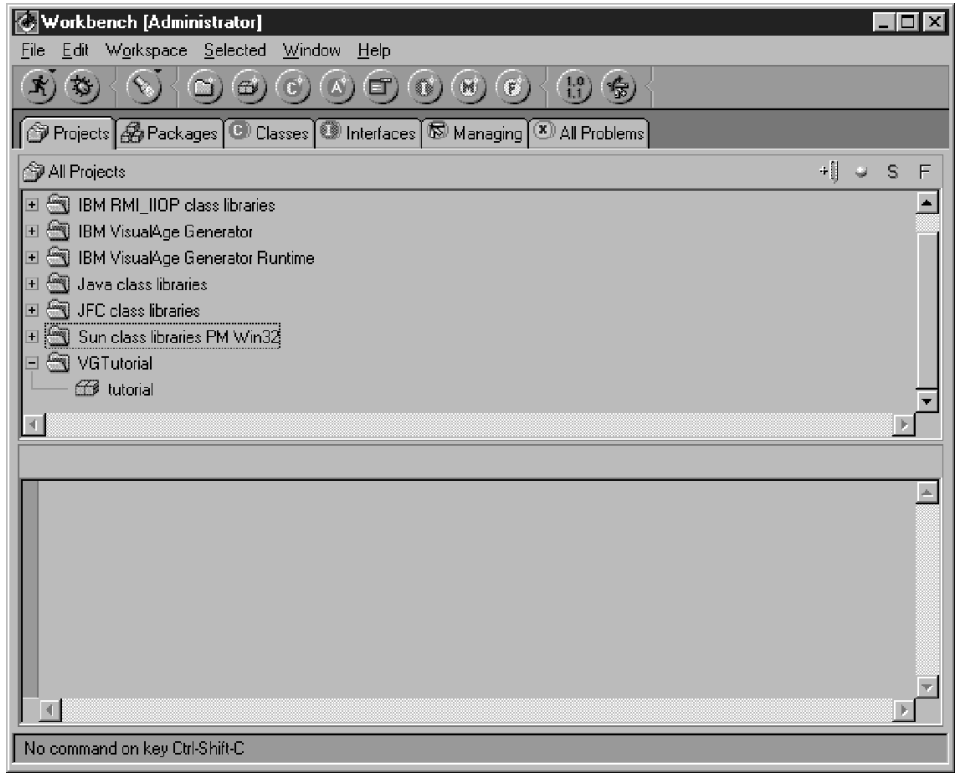


Figure 36. VisualAge for Java Workbench with new project and package

Tip: To show additional information about the parts displayed on the Workbench tabs, select the **Show Edition Names** icon,

Importing and loading samples

As you develop your program, you might want to look at some VAGen sample parts. Packages that contain sample VAGen parts were stored on your system in .DAT files when you installed VisualAge Generator. To look at these sample parts, you'll need to import the .DAT files and load the sample projects it contains. For this tutorial you'll import EZEREMPJ.DAT and EZERSMPJ.DAT and load the samples projects they contain.

For a brief description of the samples, see "Appendix B. Installing samples for VisualAge Generator Developer on Java" on page 191.

Importing sample projects

To import the sample projects:

1. From the **Selected** menu, select **Import**.

   The Import SmartGuide is displayed.
2. Select **Repository**, then select **Next**.

   A file selection window is displayed.
3. Select Local repository.
4. In the **Repository name** field, type the complete path name to `EZEREMPJ.DAT`.

   The files will be in c:*install_dir*\IDE\program\samples, where *install_dir* is the directory to which you installed VisualAge Generator.
5. Select **Projects**.
6. Select the **Details** button.

   The Project import window is displayed.
7. From the **Projects** list, select **VAGen Employee Sample**.

   Note: Ensure that a check mark is shown in the check box. If the project is highlighted but no check mark is displayed, it is not selected. To select it, click directly on the check box.
8. Select **OK**.
9. Select **Finish**.

Repeat these steps but select EZERSMPJ.DAT to import the **VAGen Ezersmp Sample** project.

Note: You can use this same procedure to import a similar project, **VAGen Staff Sample**, from STAFFJ.DAT. However, because it includes parts with the same name as those included in the samples you will load in the following section, you should not load all three projects into your workspace at the same time.

Loading the sample projects

To load the sample projects:

1. In the Workbench, **All Projects** list, click with mouse button 2 to display the **Selected** context menu.

   This context menu displays the same options as the **Selected** menu above the tool bar.

2. From the context menu select **Add→Project**.

   The Add Project SmartGuide is displayed.

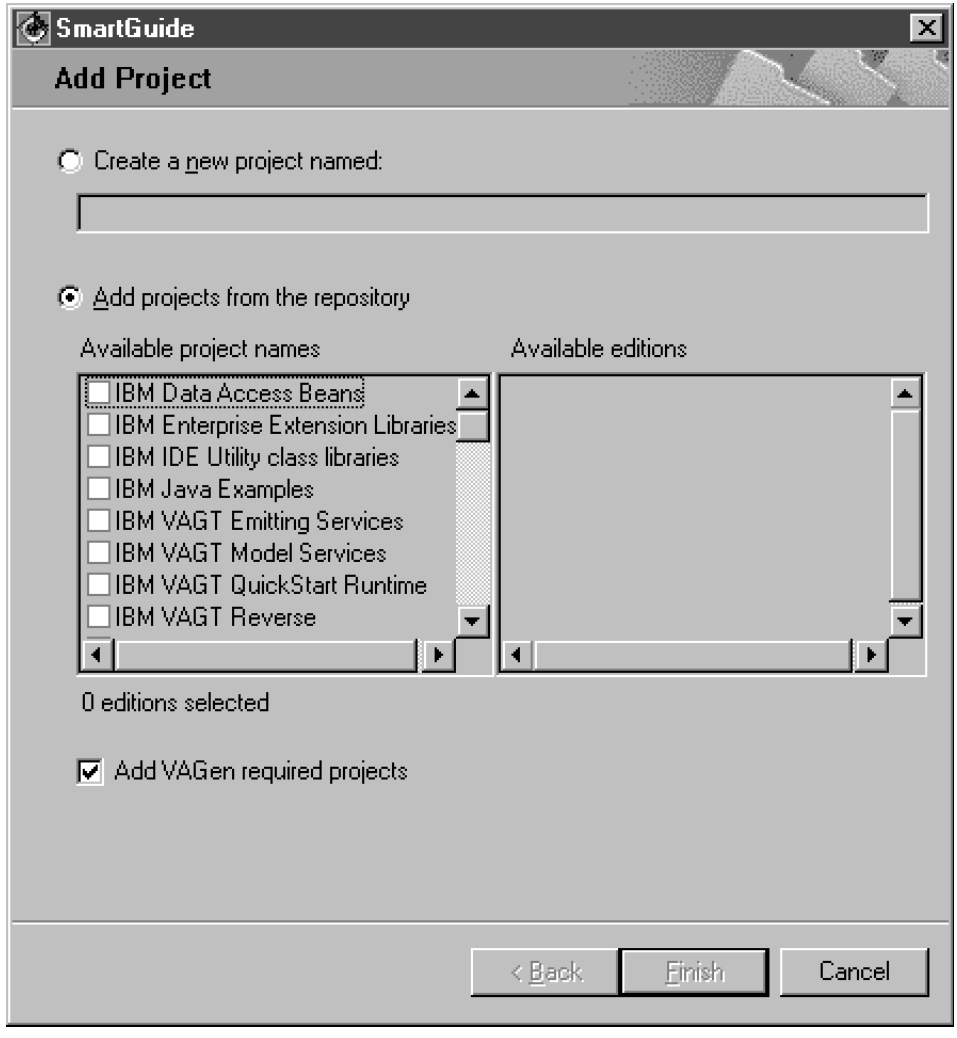


Figure 37. Add Project SmartGuide

3. Select **Add projects from the repository**.
4. From the **Available Project Names** pane, select **VAGen Employee Sample**.
5. In the **Available Editions** pane, select the latest edition name.
6. From the **Available Project Names** pane, select **VAGen Ezersmp Sample** .
7. In the **Available Editions** pane, select the latest edition name and select **Finish**.

   The projects you selected are displayed in the **All Projects** list.

For more information on managing projects and packages, refer to the online help for VisualAge for Java.

Chapter 9. Tailoring options on Java

VisualAge Generator lets you set options that can save you time as you develop programs and build parts. These profiles are used as defaults and by setting them, you can tailor the environment to your needs. Options you set on the VAGen Options window are stored in `hpt.ini` when you save your Workspace.

Note: If you have installed VisualAge Generator Developer on Smalltalk, refer to "Chapter 7. Tailoring preferences on Smalltalk" on page 59.

In this tutorial, you will set the following options:
- Program options
- SQL Database options
- Test options

To open the VisualAge Generator Options window:

1. From the VisualAge Workbench, select **Workspace→Open VAGen Parts Browser**. The **VAGen Parts Browser** is displayed.
2. From the VAGen Parts Browser, select the **Window→Options** menu.

   The Options window is displayed.

Program options

As in many programs, this sample program uses implicit data items. The system default is not to allow implicit data items. To change this default:

1. Select **Program**.
2. Select **Allow implicits**.

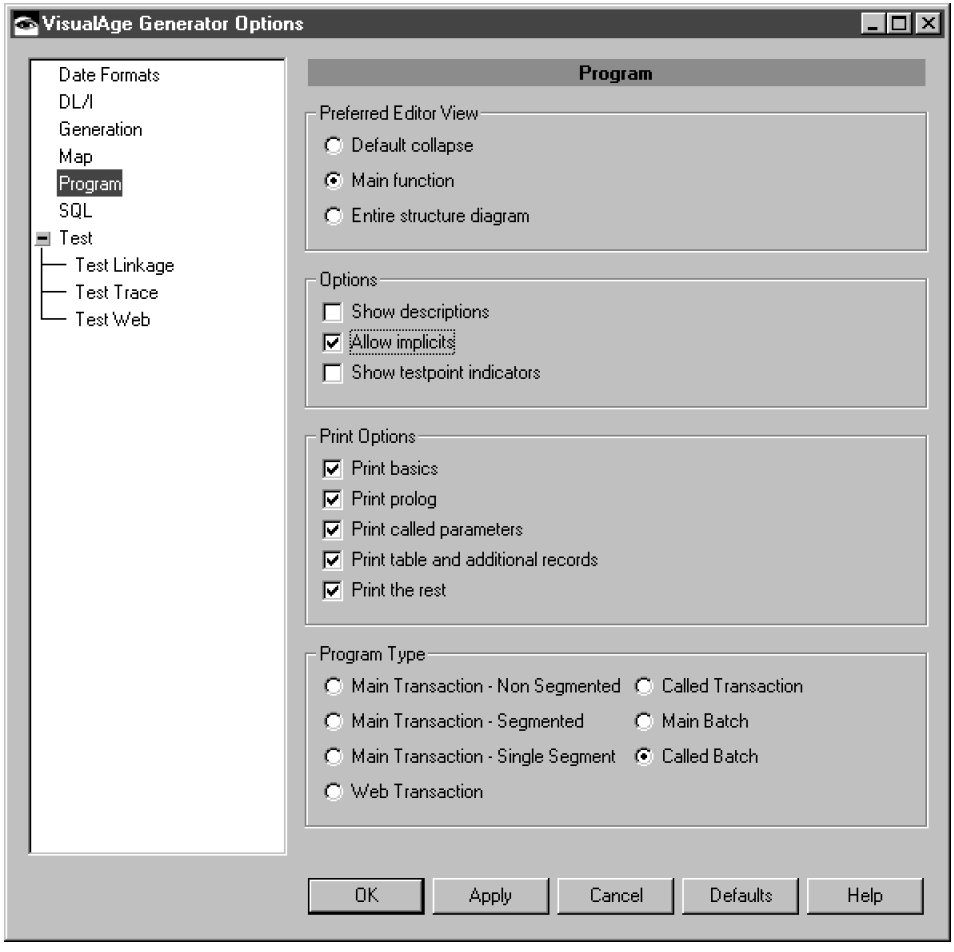


Figure 38. VAGen Program Options Window

Database options

The sample program you are creating uses an SQL database called **SAMPLE**. By choosing a default database in your options, you can ensure that you are connecting to the correct database. To choose database option:

1. Select **SQL**.
2. Ensure that `SAMPLE` is entered in the **Database name** field.

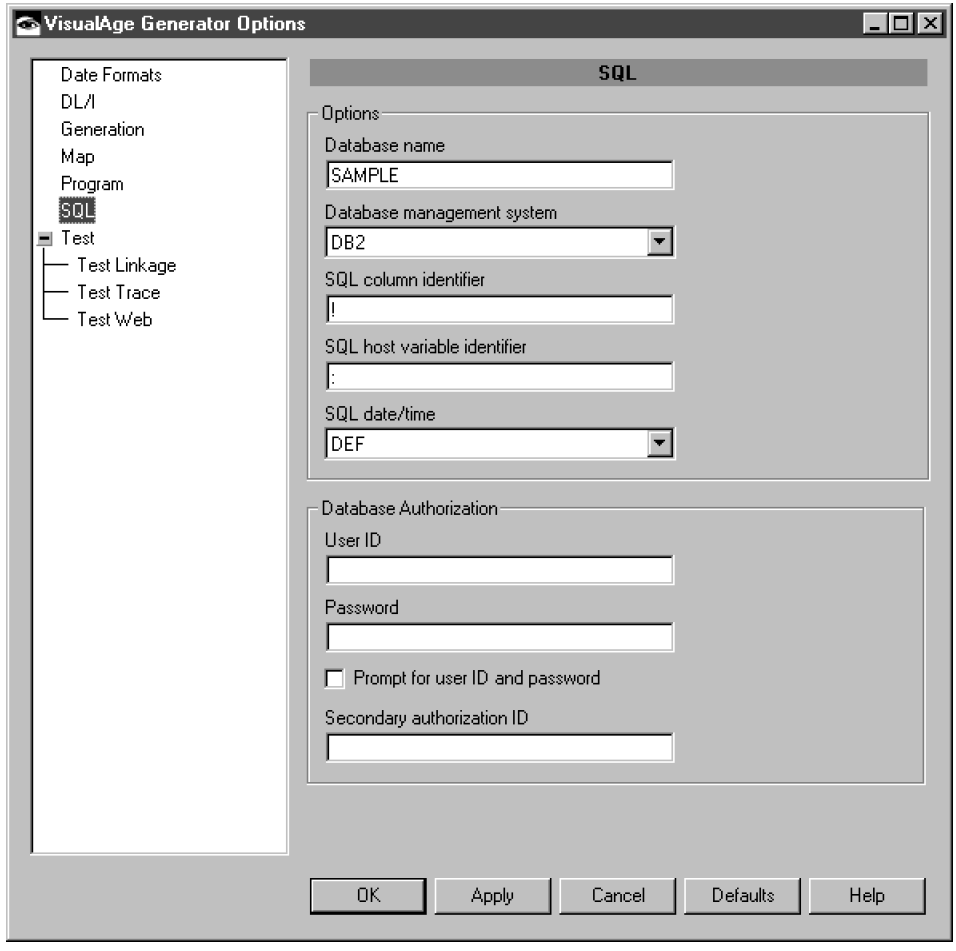


Figure 39. VAGen SQL Options Window

Test options

To set options for the VisualAge Generator test facility:

1. Select **Test** .
2. In the **Relative execution speed** field, use the arrow buttons to set the value to 3.

   The test facility runs relatively fast. You might want to reduce the test speed to give you more time to stop it if you need to.

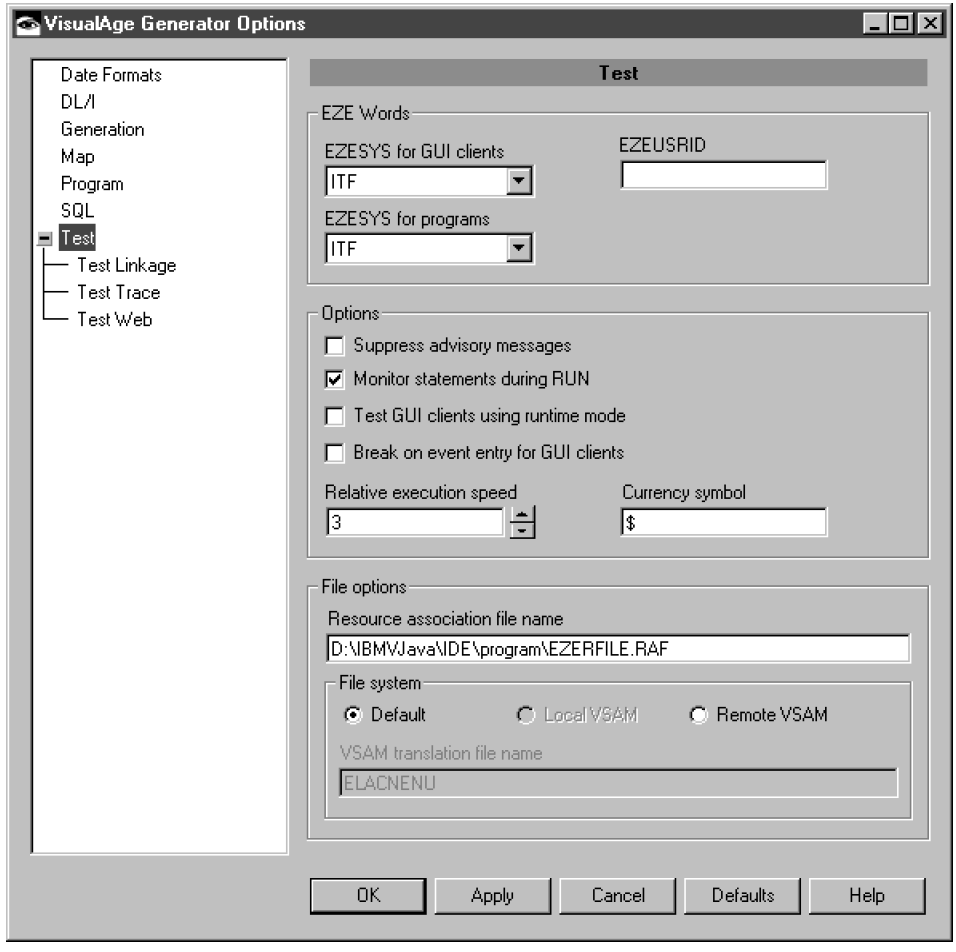


Figure 40. VAGen Test Options Window

You can use the test facility to collect trace data for your programs. Use the trace filters to control the types of trace data you see. You can change the filter settings in the test facility if you want to see other types of trace data. To set trace filters:

1. Select the **Test Trace**.

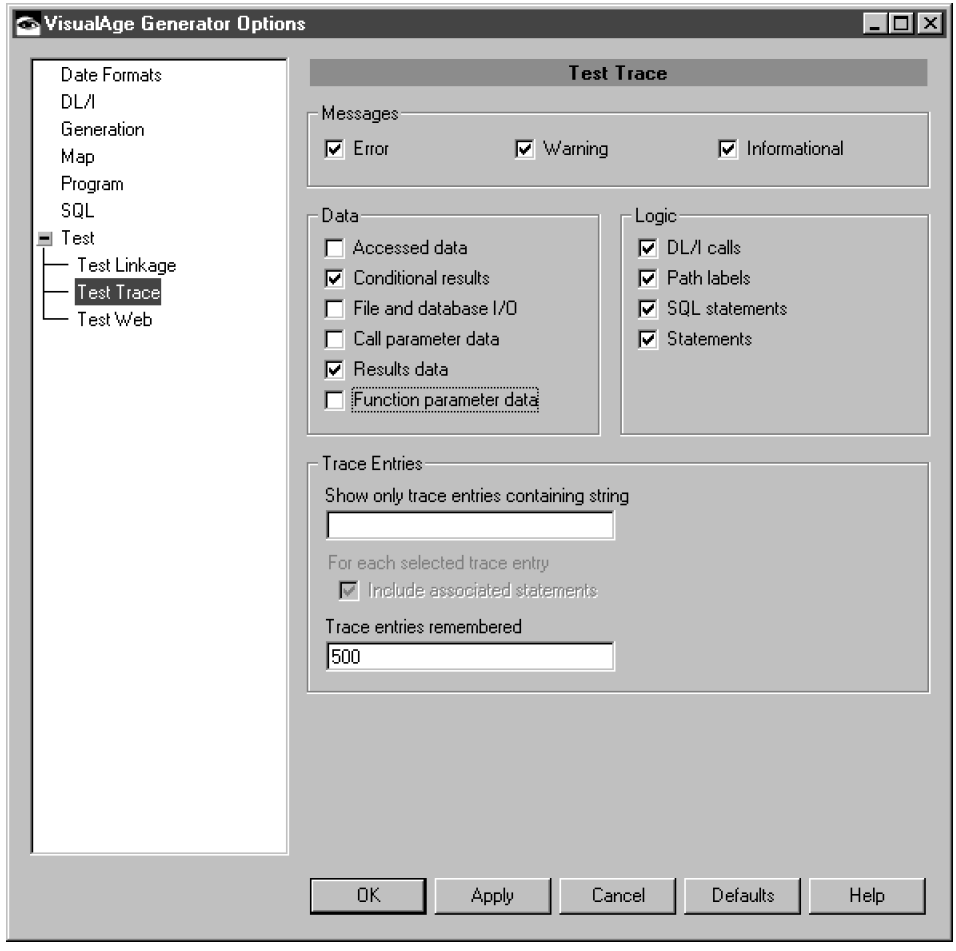


Figure 41. VAGen Test Trace Options Window

2. Deselect everything under the heading **Data** except **Conditional results** and **Results data**.
3. Select **OK**.

You can specify trace filters to view information, such as:
- The statement where the expression was evaluated or the assignment was made
- Results of expression evaluations and assignments

Selecting a trace entry filter causes all trace entries that match that filter's characteristics to appear in the Trace Log window. Setting a filter does not alter the information that is collected by the test facility.

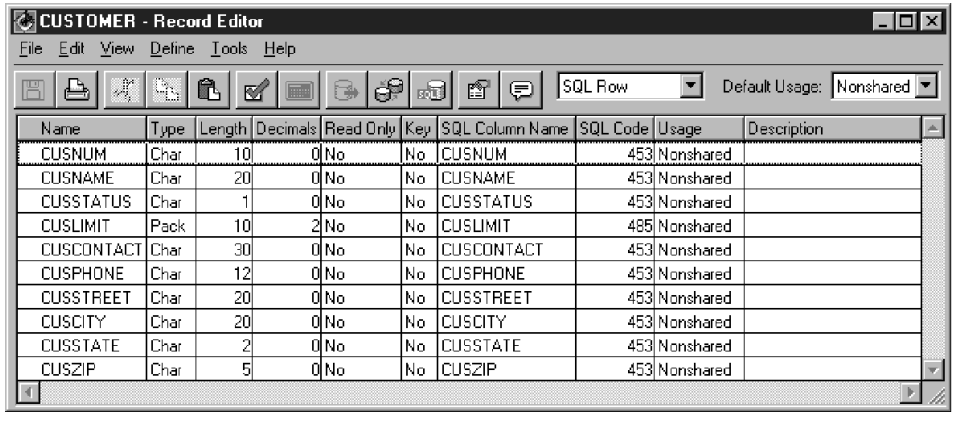
Chapter 10. Defining a program

After you have created a new container for your parts and set your preferences, you can begin developing a program. This tutorial takes you through a top-down approach to developing a small sample program.

The sample program you develop in this tutorial lists customers from a table that is shipped with VisualAge Generator. With your sample program, users can perform the following tasks:

- Enter a starting customer identifier number
- Scroll forward and backward through the list of customers
- Close the program

The relational table shipped with VisualAge Generator contains the SQL column information shown in Figure 42.



Name	Type	Length	Decimals	Read Only	Key	SQL Column Name	SQL Code	Usage	Description
CUSNUM	Char	10	0	No	No	CUSNUM	453	Nonshared	
CUSNAME	Char	20	0	No	No	CUSNAME	453	Nonshared	
CUSSTATUS	Char	1	0	No	No	CUSSTATUS	453	Nonshared	
CUSLIMIT	Pack	10	2	No	No	CUSLIMIT	485	Nonshared	
CUSCONTACT	Char	30	0	No	No	CUSCONTACT	453	Nonshared	
CUSPHONE	Char	12	0	No	No	CUSPHONE	453	Nonshared	
CUSSTREET	Char	20	0	No	No	CUSSTREET	453	Nonshared	
CUSCITY	Char	20	0	No	No	CUSCITY	453	Nonshared	
CUSSTATE	Char	2	0	No	No	CUSSTATE	453	Nonshared	
CUSZIP	Char	5	0	No	No	CUSZIP	453	Nonshared	

Figure 42. Record Editor

The columns of the table can be displayed using arrays in a single map. The map can also contain the starting staff identifier number, the current date, a message line, and a line of prompts for the function keys that are available.

As you develop, you can look at a diagram of your program using the Program Editor shown in Figure 43 on page 80.

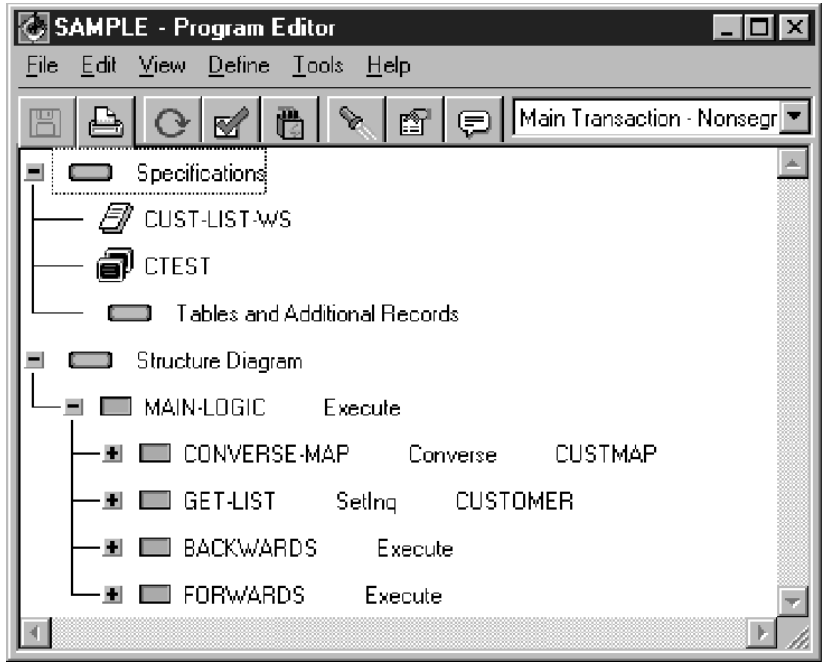


Figure 43. Program Diagram

VisualAge Generator shows the elements of the program symbolically. The program diagram is divided into two parts: **Specifications** and **Structure Diagram**. The program diagram shows the logical hierarchy of the components of a program. The following icons represent elements used in this tutorial:

A function—For functions, the connecting lines from the function symbol show the functions that are logically beneath the function.

I/O options and I/O object names appear to the right of the function symbol. Record or map symbols displayed immediately under functions, in the expanded view, are I/O objects.

A record

A map

You can expand or collapse sections of the program diagram to control what levels of the hierarchy you see in the Program Editor. You can click on the following symbols to expand or collapse a section:

 Indicates that the section is expanded. Click on this symbol to collapse the section next to it.

 Indicates that the section is collapsed. Click on this symbol to expand the section next to it.

Parts in the program diagram that do not show one of these symbols have no child symbols and cannot be expanded.

A  next to a symbol indicates that the part has not yet been defined.

In this tutorial, you are developing from the top down, starting with the program, then defining the functions, and then the I/O objects. As you define the program, you will perform an intermediate test to ensure that the program is defined correctly.

Creating a new program

To create a new program, do the following :

1. From the VAGen Parts Browser, select **VAGen Parts→Add/New**.
   If you selected **Add**, select **New Part**.
2. On the New VAGen Part window, in the **Part name** field, type `sample`.
3. Select the **Part type Program**
4. From the **Package/Application** drop-down list box, select **tutorial**.

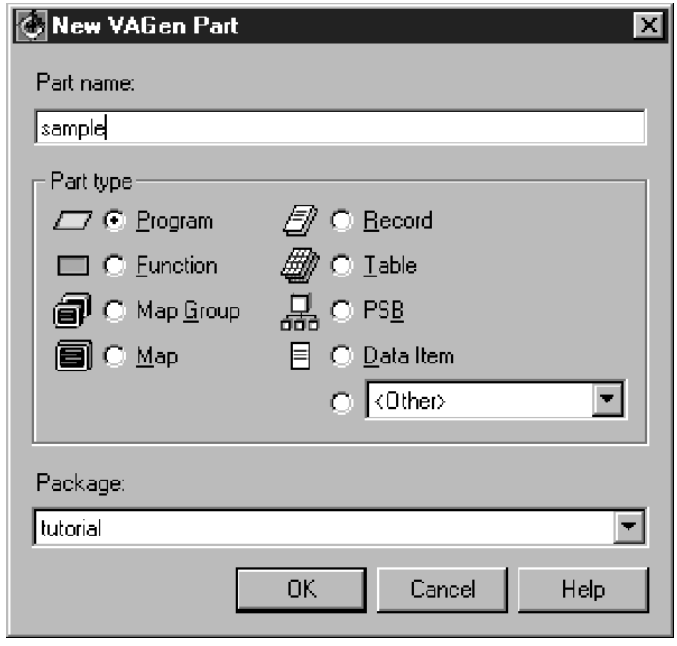


Figure 44. New VAGen Part

Program specifications and main functions are the elements that make up the basic structure of your program. Use the Program Editor to set global characteristics for the program and define the main functions.

For this tutorial, you will define a main transaction program with a working storage record and a map group. To define properties for the program you just created:

1. From the **Define** menu, select **Properties**.
2. Select **Allow implicit data items** and select **OK**.
3. From the drop-down list box on the Program Editor, select **Main Transaction-Nonsegmented**.
4. In the Add Map Group window, type `ctest`.
5. Select **OK**.

   You've defined the program **SAMPLE** as a nonsegmented transaction and specified association with the map group **CTEST**.

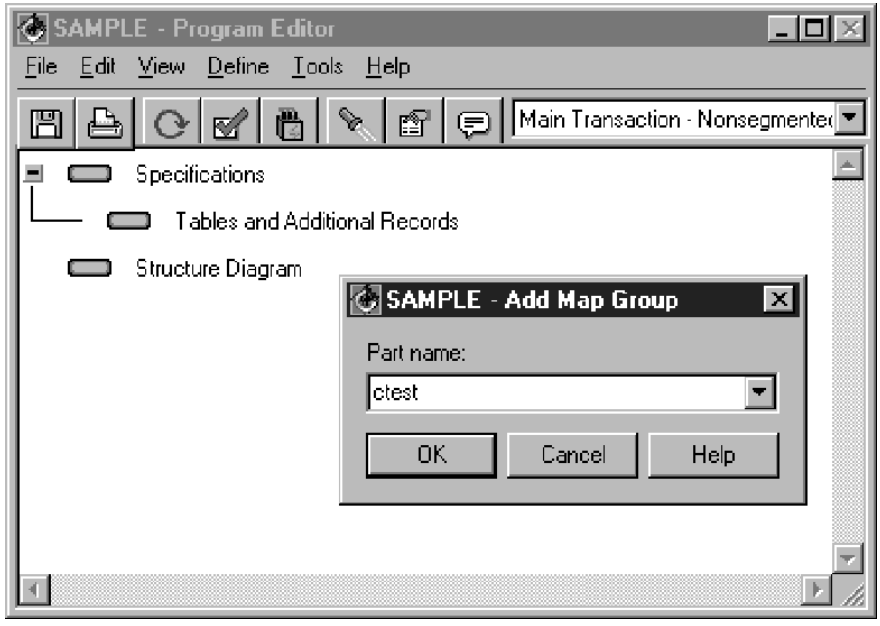


Figure 45. Add Map Group

Now define a working storage record for this sample program. In the Program Editor:

1. From the **Define** menu, select **Specifications→Add Working Storage**.
2. In the **Part name** field, type `cust-list-ws`.
3. Select **OK**.

   The name of the working storage record is displayed under **Specifications** in the Program Editor. The question mark beside it indicates that the record has not yet been defined.

Tip: You can also perform many tasks in the Program Editor using context menus. For example, you could have added this working storage record by clicking mouse button 2 on **Specifications** and selecting **Add Working Storage** to display the Add Working Storage window. You get different context menus in the Program Editor, depending on what you click on with mouse button 2. Feel free to experiment with the context menus in the tasks that follow.

Creating a main function

Main functions provide the basic structure of a VisualAge Generator program. You can define up to 254 main functions. When the program runs, control usually transfers from one main function to the next.

To insert a main function into your the structure diagram of your sample program:

1. In the Program Editor, select **Define→Main Functions**.
2. In the **Part name** field, type `main-logic`. Select **OK**.

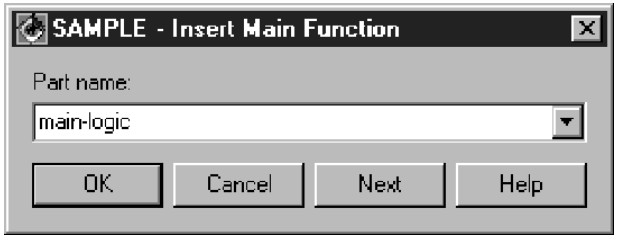


Figure 46. Insert Main Function window

The name of the main function is displayed in the structure diagram in the Program Editor. The question mark beside it indicates that the function has not yet been defined.

Defining a main function

A function provides the basic unit of the logic structure for a program. For example, input and output operations that take place are done in functions.

A single function can contain only one I/O operation. However, a function is not required to contain an I/O operation. In this tutorial, your sample program includes some functions that include an I/O operation and some that don't.

You need to build a simple logic structure within the main function you have defined. This allows you to do a quick, basic definition of the overall structure of the program.

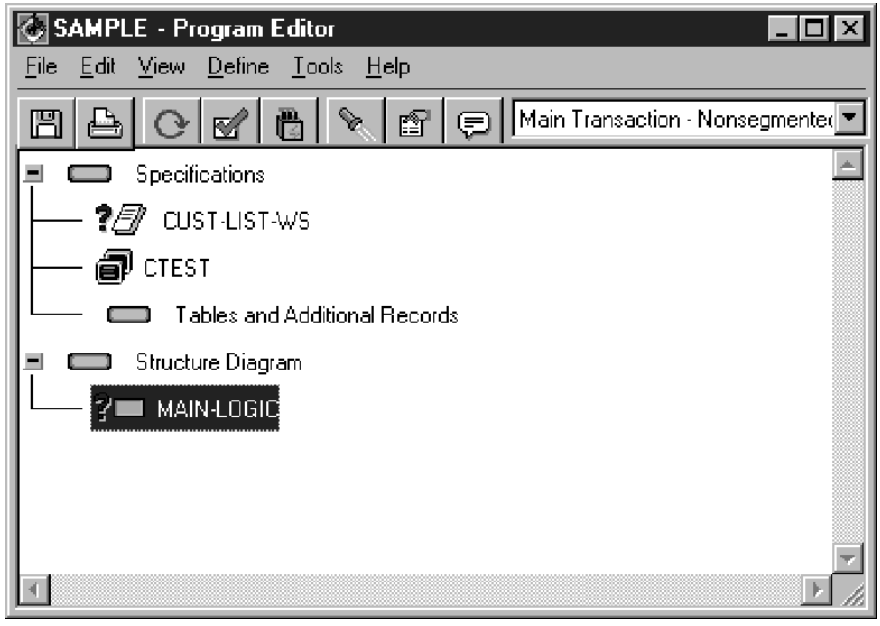


Figure 47. Program Editor

To begin building the program structure:

1. In the Program Editor double-click on **MAIN-LOGIC**.
   The New Part Package/Application window is displayed.

Figure 48. New Part Package/Application

2. Ensure that **tutorial** is selected, and select **OK**.

3. Type the statements as shown in Figure 49.

   Note: You can type in lowercase. The editor will convert your code to uppercase when you select **Save**, **Validate**, or **Validate and Format**.

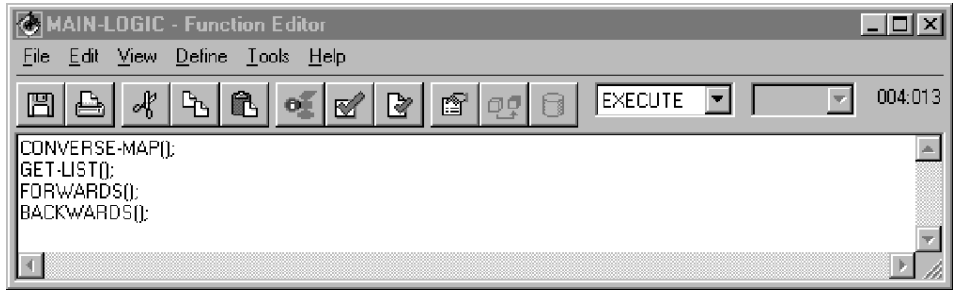


Figure 49. Function Editor

   Each line in this function invokes the function named.

4. Ensure that **EXECUTE** is displayed in the I/O option drop-down list box.

   The list box on the left displays I/O options, which define the type of operation a function will carry out. The list box on the right displays I/O objects, the data parts on which I/O options are performed. The EXECUTE option does not have an I/O object. Here it is used to control the flow between functions.

5. From the **Tools** menu, select **Validate and Format**.
6. From the **File** menu, select **Save**.
7. Close the Function Editor.

Now you can partially define each of the performed functions. By partially defining these functions, you can test the program structure without completely defining the program. You can supply the logic of the program later. Providing partial definitions of the program and its components is one way to use VisualAge Generator Developer for rapid prototyping.

1. In Program Editor, click on the + beside **MAIN-LOGIC**.

   That branch of the structure diagram is expanded.

2. Double-click on **CONVERSE-MAP**.

   The New Part Package/Application window is displayed.

3. Ensure that **tutorial** is selected, and select **OK**.

   The new part is created and the **Function Editor** is displayed.

4. From the **I/O options** drop-down list box, select **CONVERSE**.
5. In the **I/O object** drop-down list box, type `custmap`.
6. From the **File** menu, select **Save**.

7. Close the Function Editor.

To define an I/O option and an I/O object for GET-LIST:

1. In the Program Editor double-click on **GET-LIST**.
   The New Part Package/Application window is displayed.
2. Select **OK**.
3. From the **I/O options** drop-down list box, select **SETINQ**.
   The SETINQ option selects a set of rows from an SQL row record.
4. In the **I/O object** drop-down list box, type `customer`.
   Customer is the name you will use when you define the SQL row record that SETINQ will select from.
5. From the **File** menu, select **Save**.
6. Close the Function Editor.

Your program structure should look like the one shown in Figure 50.

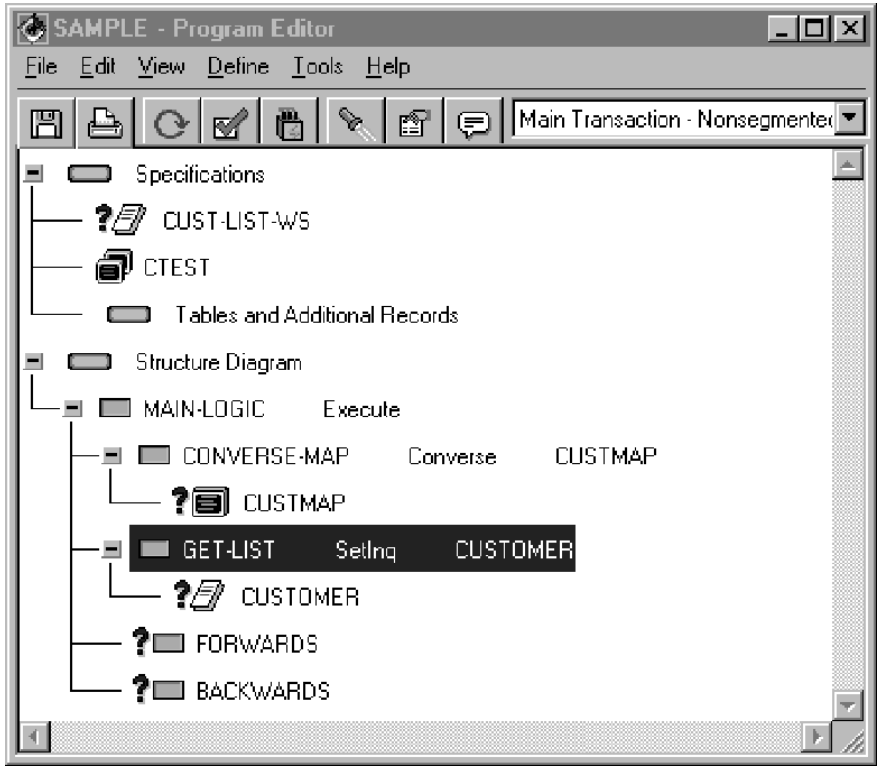


Figure 50. Program Diagram

Summary of creating a function

All functions for a program are created using the steps you just completed. The following is a summary of those steps:

1. In the **Program Editor**, double-click on the function name in the structure diagram.
2. On the New Part Package/Application window, select **OK** to add the part to the selected package/application.
3. In the **Function Editor**, select an I/O option and an I/O object, and enter statements.
4. Save the statements, then close the Function Editor.

Summary of defining a program

You have defined enough of your program structure to run a test, but before you do, you need to further define the I/O options, an SQL record and a map used by the CONVERSE-MAP function and the GET-LIST function.

Chapter 11. Defining an SQL record

VisualAge Generator supports various file access methods and databases, including IBM's implementation of relational databases. In this section of the tutorial, you define the SQL record for the CUSTOMER sample table.

You define all records in the Record Editor, which you have easy access to from the Program Editor.

To begin defining the SQL record:

1. In the Program Editor, click on the + beside **GET-LIST**.

   This section of the structure diagram is expanded.
2. Double-click on the record **CUSTOMER**.

   The New Part Package/Application window is displayed.
3. Ensure that **tutorial** is selected and select **OK**.

   The Record Editor is displayed.
4. From the record type drop-down list box, select **SQL Row**.

   Note: You'll find the record type drop-down list box between the tool bar buttons and the **Default Usage** drop-down list box. The default record type is **Working Storage**.
5. If there is an item in the record, delete it. To delete an item, select it. Then, select **Edit→Delete**.
6. Select **Define→Properties**.

   The SQL Row Properties window is displayed.
7. On the SQL Row Properties window, select **Insert**.
8. A row is added and the cursor appears in the **Name** pane.

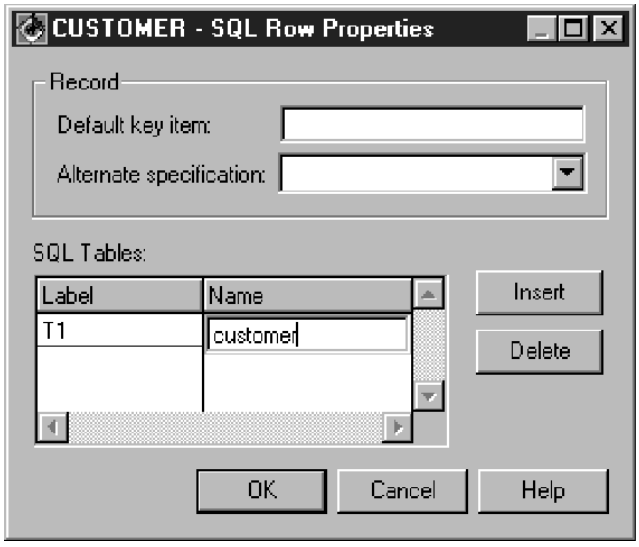


Figure 51. SQL Row Properties

When you define an SQL row record, you specify the SQL table or table joins for the record. In this tutorial, VisualAge Generator Developer uses the CUSTOMER table from the SAMPLE database you specified in VAGen **Options/Preferences**.

Once the SQL table or table join has been specified, VisualAge Generator Developer can automatically build data item names and characteristics by retrieving the column information for the table or join. This function is only available if you are on a system that has access to the database catalog.

To finish defining an SQL table:

1. In the **Name** pane, type `customer`.
2. Select **OK**.

   The SQL Row Properties window closes.
3. From the **Tools** menu, select **Retrieve SQL**. If you are prompted to run the SQLBIND command, select **Yes**.

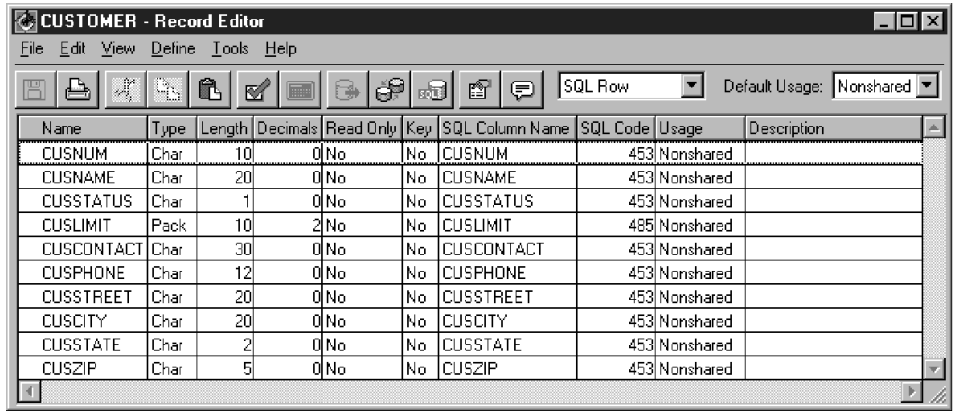
   Data items based on the SQL table column characteristics are created and inserted into the SQL row record.

   Note: If **Retrieve SQL** is not available, delete any data items that appear in the record.
4. Select the **Key** field of the CUSNUM data item. Select the **Key** checkbox.
5. Press **Tab**.

Tip: Before you can save a record, you must press the tab key to move the cursor out of the changed field.

6. From the **File** menu, select **Save**.
7. Close the Record Editor.

Note: If you have problems with the previous task, refer to the section on installing sample applications in the *VisualAge Generator Installation Guide*.



Name	Type	Length	Decimals	Read Only	Key	SQL Column Name	SQL Code	Usage	Description
CUSNUM	Char	10	0	No	No	CUSNUM	453	Nonshared	
CUSNAME	Char	20	0	No	No	CUSNAME	453	Nonshared	
CUSSTATUS	Char	1	0	No	No	CUSSTATUS	453	Nonshared	
CUSLIMIT	Pack	10	2	No	No	CUSLIMIT	485	Nonshared	
CUSCONTACT	Char	30	0	No	No	CUSCONTACT	453	Nonshared	
CUSPHONE	Char	12	0	No	No	CUSPHONE	453	Nonshared	
CUSSTREET	Char	20	0	No	No	CUSSTREET	453	Nonshared	
CUSCITY	Char	20	0	No	No	CUSCITY	453	Nonshared	
CUSSTATE	Char	2	0	No	No	CUSSTATE	453	Nonshared	
CUSZIP	Char	5	0	No	No	CUSZIP	453	Nonshared	

Figure 52. Record Editor

In the SQL row record, VisualAge Generator Developer creates data items that have the characteristics of the columns in the table.

Because you can retrieve SQL column information directly from any supported relational database, you do not have to supply data item characteristics. This saves you time and minimizes errors.

Chapter 12. Defining a map

Maps are used to display information to users and for printing information from a program. Your sample program uses one map. In this section of the tutorial, you define the constant fields, variable fields, and arrays on the map.

Every position on the map is part of a field. If there are no fields defined on the map, then there is one implicit field spanning the whole map.

Any map other than a help map can contain both constant and variable information. Help maps can contain only constant information. Constants typically consist of field labels and help fields that end users cannot change. You define constant fields by dropping constant parts on the map presentation area and defining properties for them.

Variable information can be changed either by users or other functions or programs. You define variable fields by dropping variable parts on the map presentation area. You can specify editing characteristics for the variable fields, and VisualAge Generator automatically formats and edits the fields.

Assume that the design of your program allows users to enter a beginning customer identifier when scrolling forward and backward through a list. You only need five customers listed at once.

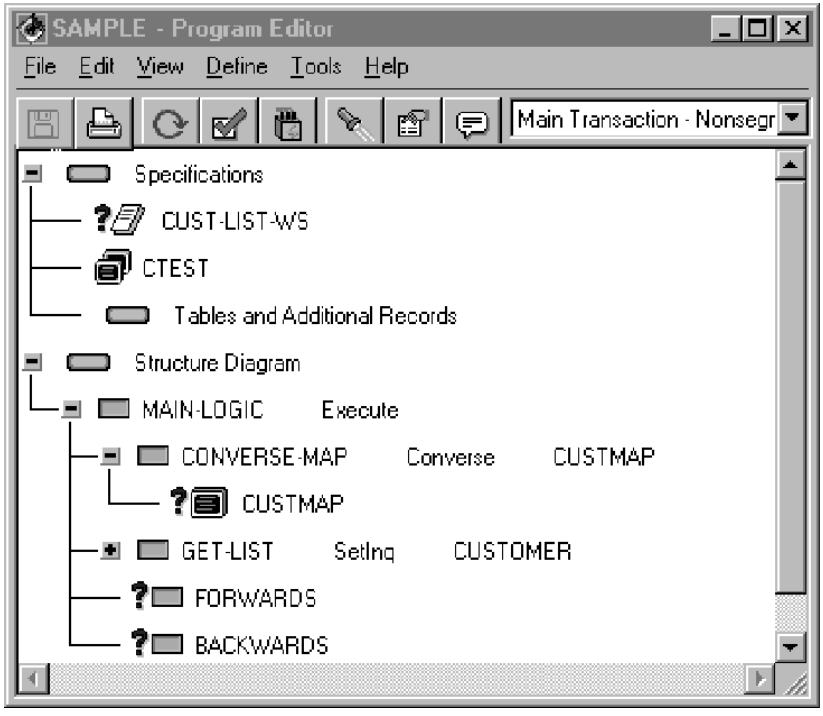


Figure 53. Program Editor

You have already specified the I/O object named CUSTMAP. The question mark beside it, shown in Figure 53, means that you need to further define it.

To begin defining the map:

1. In Program Editor, click on the + beside **CONVERSE-MAP**.

   That part of the structure diagram is expanded and a map symbol is displayed.
2. Double-click on CUSTMAP.

   The New Part Package/Application window is displayed.
3. Ensure that **tutorial** is selected, and select **OK**.

   The new part is created and the **Map Editor** is displayed.

Specifying the map title

The title of a map often consists of a single constant field centered on the map. VisualAge Generator Developer provides a centering function to enable you to center fields quickly.

The first line of the map is to contain the centered title **Customer Information**. This title is a constant area on the map.

Tip: To help you align fields in the map presentation area, turn on the grid. To turn on the grid, from the **View** menu, select **Grid**.

A grid pattern is displayed in the map presentation area.

To define the panel title:

1. On the **Palette**, select the **Constant** part  .
2. The mouse pointer becomes a crosshair. Category, part, and mouse position information is displayed in the status area at the bottom of the Map Editor.
3. Click on the first line in the map presentation area.
   The Constant Field Properties window is displayed.

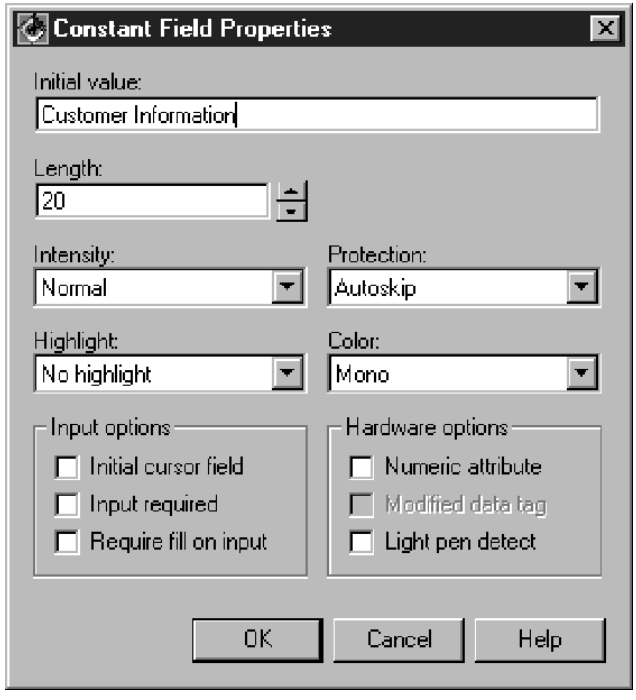


Figure 54. Constant Field Properties

1. In the **Initial value** field, type `Customer Information`.
   The field length is automatically resized to fit the text you type.

2. Select **OK**.

   The text you type is entered in a constant field on the map.
3. To center the title click on the text and, from the tool bar, select **Center**.

   
   .

   The text you typed is centered on the map.

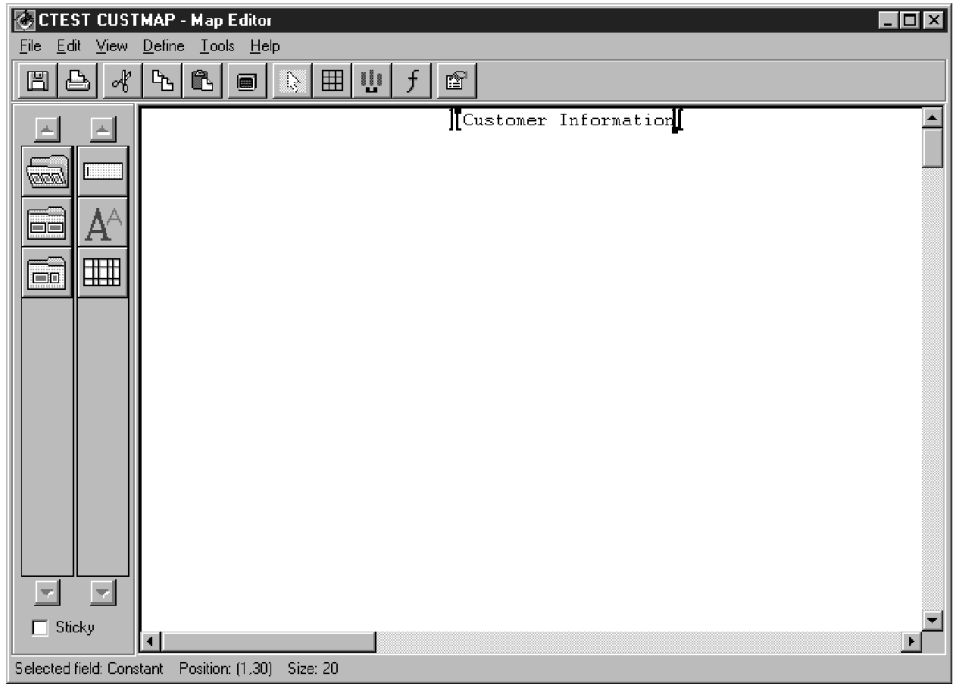


Figure 55. Map Editor

If you want to view the map as it will look to users, from the **View** menu, select **Preview**. To continue editing, select **View→Preview** again.

Note: You cannot edit in **Preview** mode.

If you want to see the colors and highlighting as they will be displayed to users while you are editing the map, select **View→Runtime Color Mode**.

Specifying a prompted entry field

Prompted entry fields require constant text for the field prompt and a variable field for entering data into the program.

To define the prompted entry field, perform the following steps:

1. On the **Palette**, select the **Constant** part  .
   The mouse pointer becomes a crosshair.
2. Click in row 3, column 1 (3,1) on the map presentation area.
   The Constant Field Properties window is displayed.

   Tip: Using choices on the map context menu can make editing tasks faster. To display the context menu, place the mouse pointer in the map presentation area and click mouse button 2. For more information on editing maps, see the online help.
3. In the **Initial value** field, type `Customer Number` and select **OK**.
   The Constant Field Properties window closes and the text you typed is displayed in a constant field on the map presentation area.
4. On the **Palette**, select the **Variable** part  .
5. Click between the left and right brackets][at the right side of the constant field.

On the Variable Field Properties window:

1. On the **General** tab, in the **Name** field, type `cusnum`.
2. In the **Length** field, type 8.

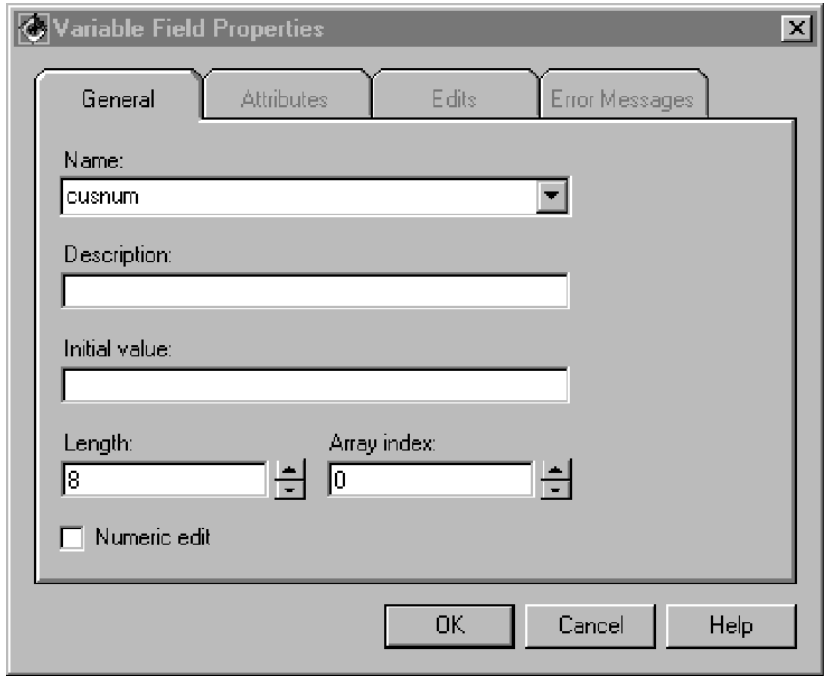


Figure 56. Variable Field Properties

3. Select **OK**.

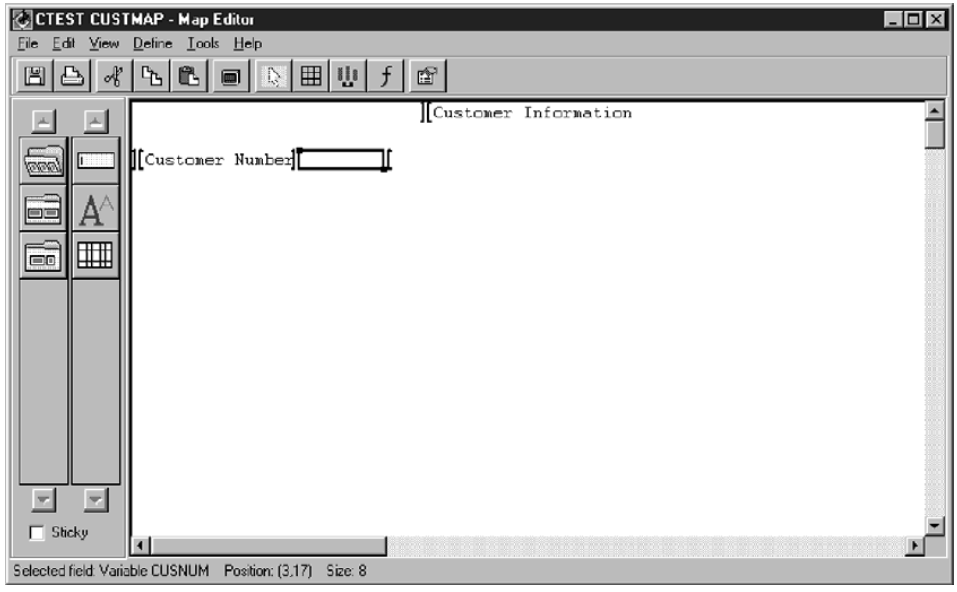


Figure 57. Map Editor

Specifying a variable message field

Programs frequently require messages to provide feedback to users concerning errors or other special conditions.

You can use the EZEMSG special function word and a map variable field for presenting messages to users. When you use EZEMSG, you should provide a variable field long enough to contain the text of the message.

To specify a variable message field:

1. From the **Palette**, select the **Variable** part  .
2. Move the mouse pointer to line 15 and click in column 1 on the map presentation area.
3. On the **General** tab, in the **Name** field, type `ezemsg`.
4. In the **Length** field, type 78.
5. Select **OK**.

While you are defining a map, you can change any of your previous definitions. For example, you can change the length of a variable field or the text and position of a constant field.

To change the specifications for a constant field or a variable field, double-click on the field, make the necessary changes in the properties window and select **OK**.

To change the content of a constant field, you must enter edit mode. To enter edit mode press and hold the **Alt** key and, click mouse button 1 in the field where you want to make a change. You can type over the text or cut and paste text using selections from the **Edit** menu. To select text to cut, copy, paste, or delete, with your cursor in edit mode, click to the left of the text you want to select. Drag the mouse pointer to the right until the text you want to change is selected. Release the mouse button. From the **Edit** menu select the appropriate menu choice.

Adding a constant field

You add constant fields the same way you specified the title.

To add a constant field showing the function key definitions, perform the following steps:

1. On the **Palette**, select the **Constant** part  .
2. Click in row 17, column 1 (17,1) on the map presentation area.
3. In the **Initial value** field, type

   ```
   F3=Exit  Enter=GetList  F7=Backward  F8=Forward
   ```

   and select **OK**.

Your map should look something like Figure 58 on page 101.

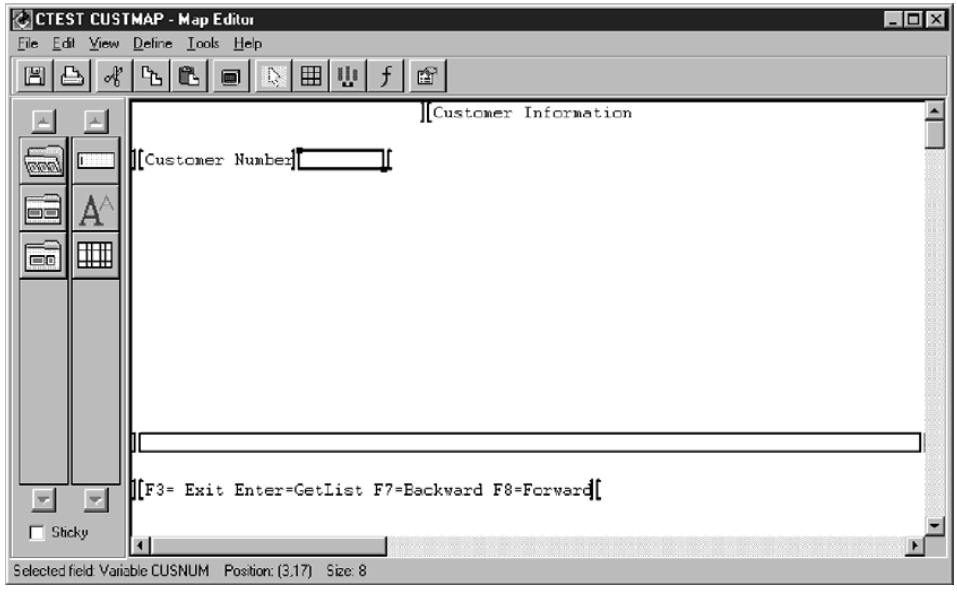


Figure 58. Map Editor

Defining a map array

You can define arrays of variable fields in a map to make it easier to work with several related fields at the same time.

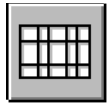
To define an array, you first specify the array's physical arrangement on the map and give the array a name.

You need to define four arrays for your sample program. These four map arrays are arranged in four columns named CUSNUM-A, CUSNAME-A, CUSCONTACT-A, and CUSPHONE-A.

Turn on the grid, select **View→Grid**, to help you locate lines and columns referenced in the following tasks. If you want to look at array indices, select **Define→Field Edit Order→Show Tags**. Both of these features can be turned off by repeating the menu selections you used to turn them on.

To define the first map array:

1. In the Map Editor, from the **Palette**, select the **Array** part



.

2. Click in row 6, column 1 (6,1) of the map presentation area.
3. On the **Array** tab, in the **Fields down** field, type 5.

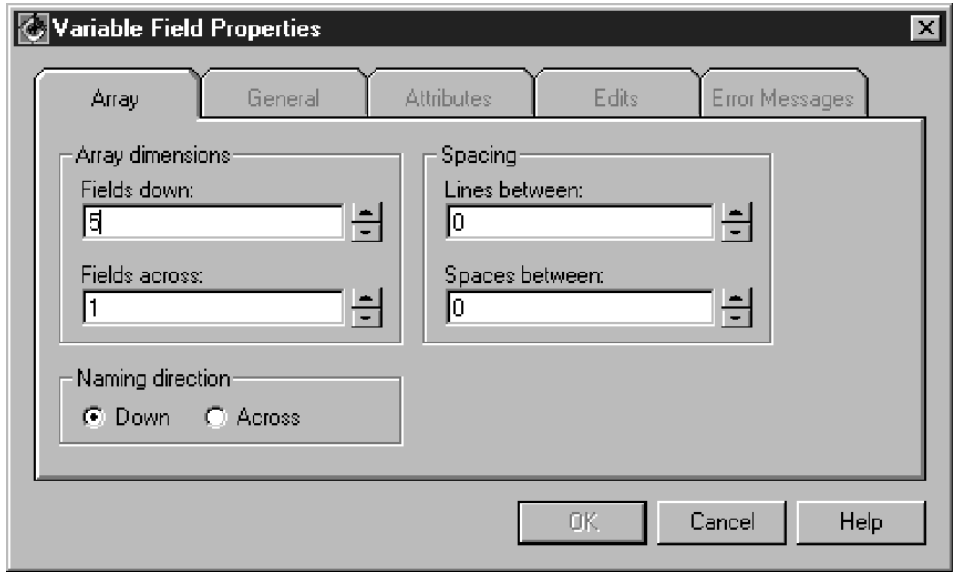


Figure 59. Array Tab on the Variable Field Properties Window

4. Select the **General** tab.

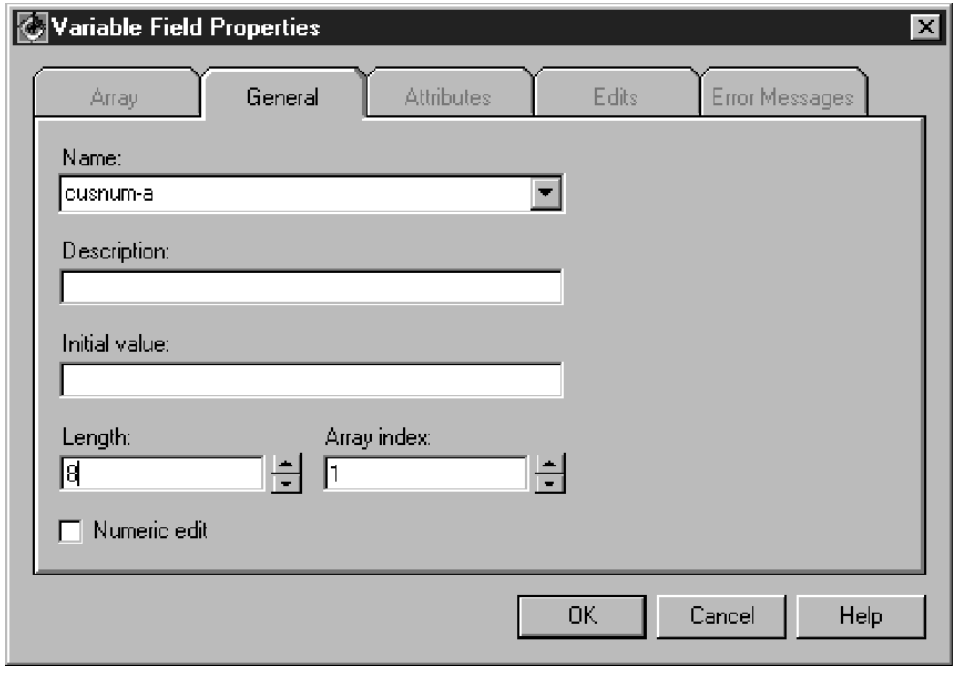


Figure 60. General Tab on the Variable Field Properties Window

On the **General** tab, you specify the name of the array variable for the map array. You can also specify an array index. Each variable field in the map array is assigned a subscripted name, starting with this index. For instance, if you specify SSN as the name and 1 as the starting index, the variable names assigned are SSN(1), SSN(2), SSN(3), and so on.

If the map array you are defining has multiple rows and columns, you can use the Naming Direction selections on the **Array** tab to choose the direction used first in assigning names to the variable fields. Because you are defining an array with only one column in this exercise (1 is the default value in the **Fields across** field), the naming direction is not important.

1. In the **Name** field, type `cusnum-a`.
2. In the **Length** field, type 8.
3. Select **OK**.

Now define three more arrays using these same steps and the names and values in the following table.

Start column	Fields down	Array name	Length
13	5	cusname-a	20
37	5	cuscontact-a	20
61	5	cusphone-a	12

Your map should look like the one shown in Figure 61.

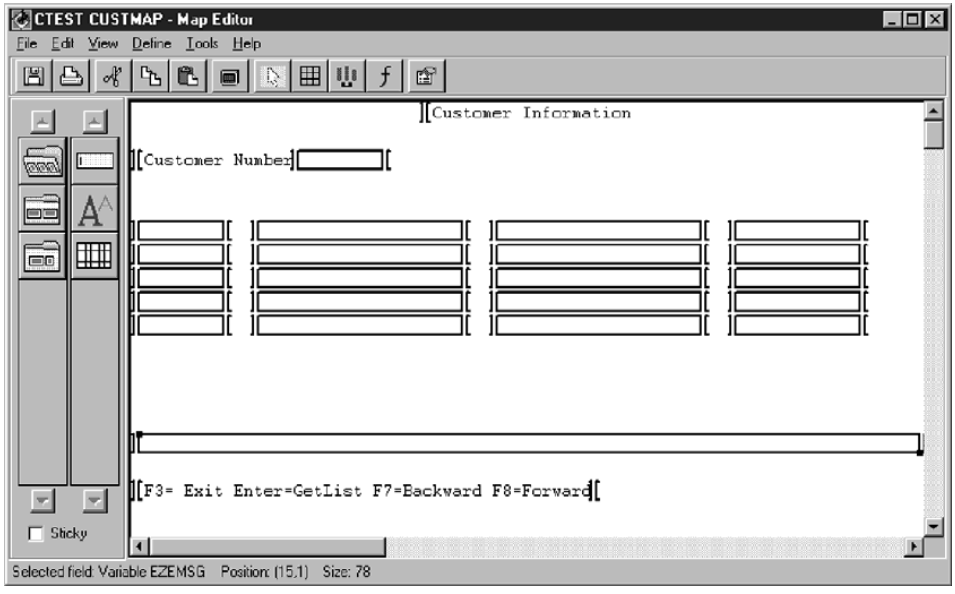


Figure 61. Map Editor

Previewing and saving the map

At any time while defining a map, you can preview it. When you preview a map, graphics indicating field positions are removed, and the color and highlighting attributes are displayed as the user will see them.

Perform the following steps:

1. From the **View** menu, select **Preview**.
2. The map preview is displayed.
3. To leave the preview mode, select **View→Preview**.
4. The Map Editor is displayed.

   For now, you are finished defining the map for your sample program.
5. In the Map Editor, select **File→Save** and close the Map Editor.
6. In the Program Editor, from the **File** menu, select **Save**.

Chapter 13. Running an intermediate test

After you define the basic structure of a program, you can run an intermediate test to check your progress.

You can begin testing a program from the Program Editor by selecting the

Test button on the tool bar  . You can also start tests from the **VAGen Parts Browser** or the VisualAge Organizer window. To start a test from either of these windows, select the program you want to test, then select the **Test** button.

Before you begin testing a program, you often need to set one or more breakpoints to suspend the test at specific points.

When the test is suspended, you can inspect the state of the program data and the current location of the test to verify that the program is working as designed.

Setting a breakpoint

One way to set a breakpoint is from Tools menu on the Program Editor.

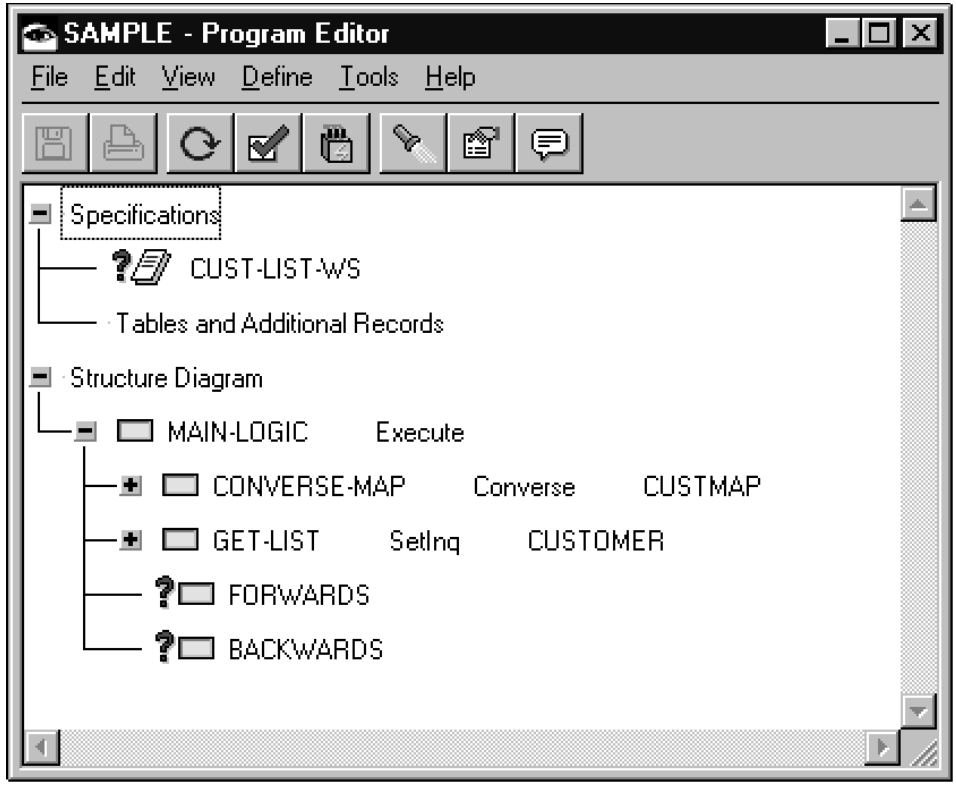


Figure 62. Program Editor

Assume that you want the test to be suspended when it reaches the GET-LIST function. To set a breakpoint on that function:

1. In the Parts Browser, double-click on the SAMPLE program icon to open it in the Program Editor.

   Tip: If you don't see the program in the **VAGen parts** pane, select **tutorial** in the Packages/Applications pane

2. From the **Tools** menu, select **Set Testpoints→Part**.

   The Set Part Testpoint window is displayed.

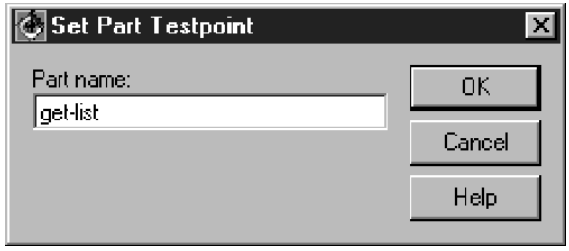


Figure 63. Set Part Testpoints

To set a breakpoint at the GET-LIST function, perform the following steps:

1. On the Set Part Testpoint window, type `get-list` and select **OK**.
   The Set Testpoints window is displayed.

On the Set Testpoints window, you define testpoints for the components of your program. For a function, you can set the following types of testpoints:

- A tracepoint that takes effect while the function has control
- Breakpoints on each statement in the function, including the I/O option
- A breakpoint that takes effect when control transfers to the function, before any statements in the function run

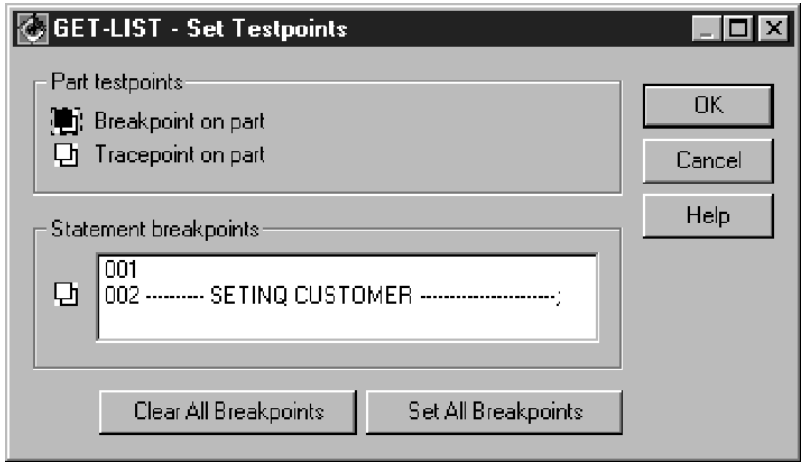


Figure 64. Set Testpoints

Qualifications can be set on any or all of these testpoints. You can set certain conditions which must be true for the testpoint to take effect.

To set a breakpoint on the function GET-LIST, perform the following steps:

1. On the Set Testpoints window, select the  icon beside **Breakpoint on part**.

   The symbol changes to  .
2. Select **OK**.

   Note: If you want to see indicators for testpoints you have set, from the Program Editor **View** menu, select **Show Testpoint Indicators**.

Starting the test

When the testpoints are set you can start the test and specify how tracing takes place from the Test Monitor window.

To start the VisualAge Generator test facility:

1. In the Program Editor, select the **Test** button on the tool bar.

   The **Test Monitor** is displayed.

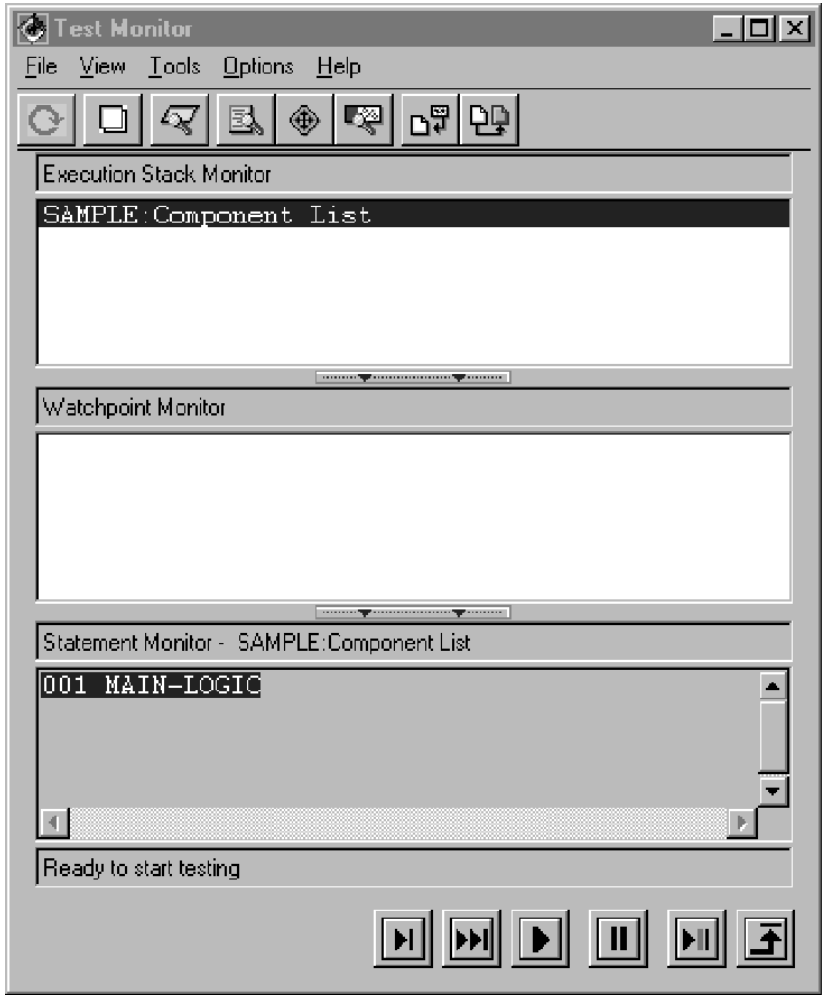


Figure 65. Test Monitor

Use the row of push buttons along the bottom of the Test Monitor window to monitor and control the test.

Step runs the highlighted statements in the **Statement Monitor** list and steps into any performed functions.

Leap runs the highlighted statements in the **Statement Monitor** list to completion without stepping into any performed functions.

Run starts or resumes the test.

Stop suspends the test.

Bypass skips over the highlighted statement in the **Statement Monitor** list.

Return runs up to the return point of the current function, main function flow, or the program's components list entry.

To trace every action:

1. From the **Tools** menu, select **Trace→Trace All**.

   The test facility traces all statements.

The basic structure of your program is defined, so the map is displayed when you run a test. To allow the program to run up to that point:

1. Select **Run** 

   The CONVERSE I/O option displays your map in the **Map Monitor**.

You open the map using the Map Editor, without exiting the test facility. When you save your change, you can reposition the test pointer and run the CONVERSE I/O option again, using the updated map definition. Refer to the VisualAge Generator help facility, for more information on test facility features.

To continue the test, on the Map Monitor window:

1. On the Map Monitor window, type `10` and press **Enter**.

   The Map Monitor window closes and the test continues. It stops at the breakpoint you set on GET-LIST.

Making changes or fixing a problem while testing

If you have a problem or see that something is not fully defined, you can open the part and make changes without closing the test facility. The test facility will pick up the changes, discarding data or repositioning the statement pointer as necessary to allow you to continue the test.

Viewing the trace entries

While you are testing a program or after you have finished a test, you can open a Trace Log window to view the collected trace entries. You can use these trace entries to view a historical record of your test.

Note: No trace entries are collected unless **Trace All** or **Tracepoints Only** is selected from the **Trace** menu.

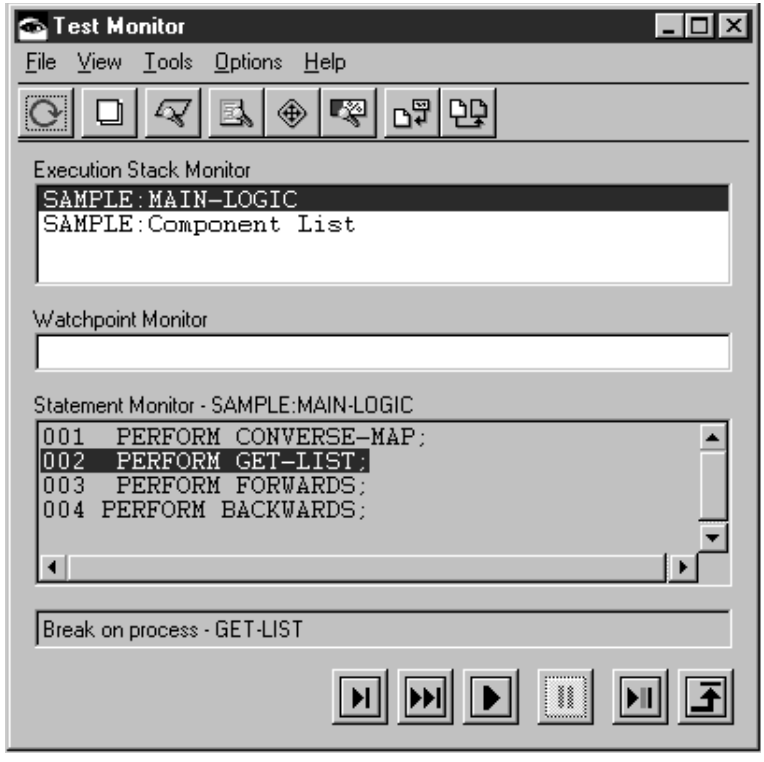


Figure 66. Test Monitor

Assume you want to display only certain types of trace entries that have been collected. On the Test Monitor window:

1. From the **Tools** menu, select **Trace→View Trace**

   The Trace Log window is displayed.

Even though the portion of the program that you tested was small, the Trace Log window displays several trace entries. You can reduce the number of trace entries by setting trace filters.

There are several different trace filters. Each trace filter allows only one type of trace entry to pass through.
Assume that you want to see only those trace entries where transfer of control

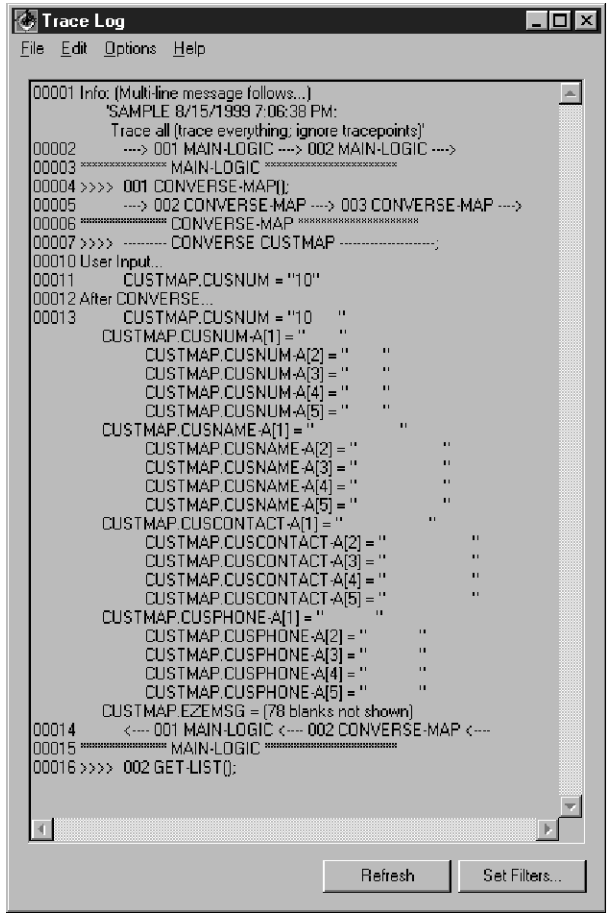


Figure 67. Trace Log

has taken place. Also, you want to see the statements that caused the transfers of control. On the Trace Log window:

1. Select **Options**, then **Set Filters**

   The Trace Entry Filters window is displayed.

On the Trace Entry Filters window, you select the trace filters you want to apply to the Trace Log.

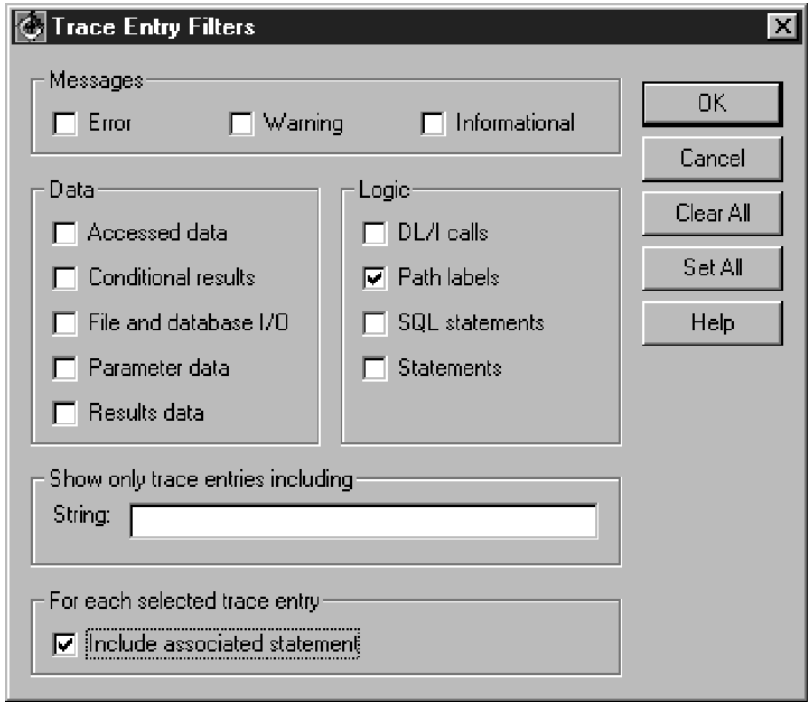


Figure 68. Trace Entry Filters

On the Trace Entry Filters window:

1. Select **Clear All**.
2. Select **Path labels** and **Include associated statement**.
3. Select **OK**.

   After you change the trace filters, you need to refresh the contents of the Trace Log window to display only the trace entries that apply to the selected trace filters.
4. On the Trace Log window, select **Refresh**.

The refreshed Trace Log window shows only the trace entries that passed through the selected trace filters. This amount of information is easier to view and understand than all the trace entries displayed at once.

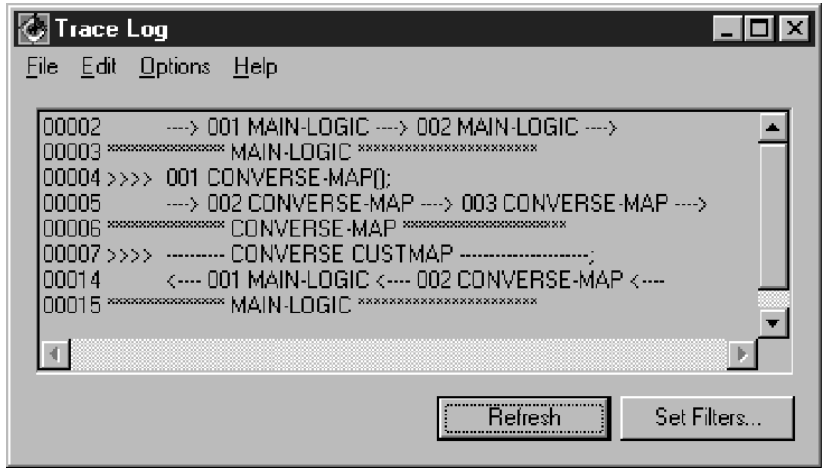


Figure 69. Trace Log

You can select trace entries in the Trace Log window and use the selections in the **Edit** menu to copy selected information onto the clipboard.

The complete trace log is still available and can be viewed by selecting other filters. You do not have to rerun the test to recreate the trace log.

When you are finished viewing the Trace Log window:

1. Close the Trace Log window.
2. Close the Test Monitor window.

Chapter 14. Defining the working storage record

After you run an intermediate test and are satisfied with the results, you can begin defining the working storage record and supplying the logic for your program.

A working storage record holds temporary and intermediate data values that a program uses while it runs. Your sample program requires a working storage record containing four nonshared data items. Specifying that data items are nonshared means that they are not part of other record parts and cannot be used outside the context of the record they are used in.

You named the working storage record for this program when you defined the program specifications in the Program Editor. The working storage record is represented on the structure diagram by the record symbol and the name CUST-LIST-WS to the right of the symbol. The question mark (?) symbol is removed after you define the record.

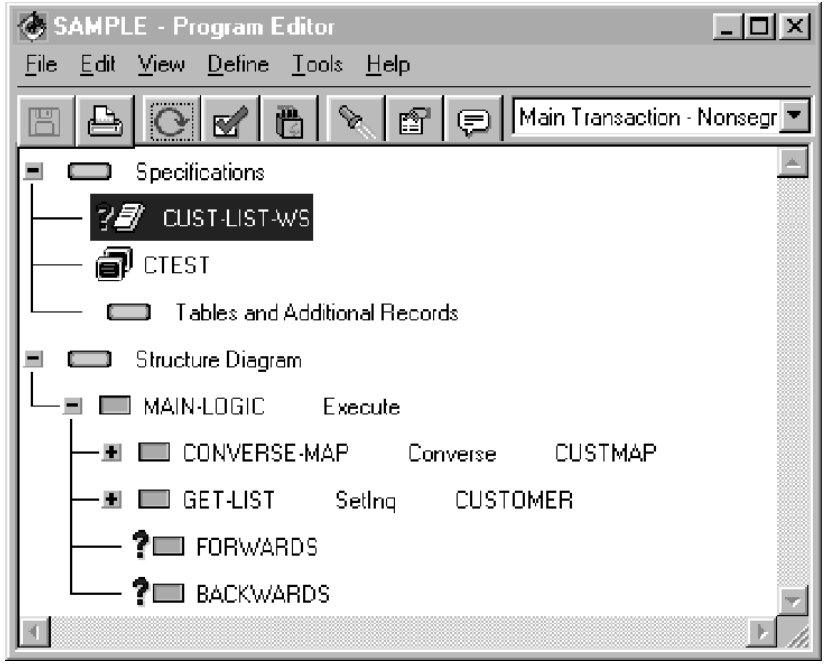


Figure 70. Program Editor

To begin defining the working storage record, do the following:

1. In the Program Editor, double-click on **CUST-LIST-WS**.

   The **New Part Package/Application** is displayed.
2. Ensure that **tutorial** is selected, and select **OK**.

   The new part is created and the **Record Editor** is displayed.

You have already named CUST-LIST-WS as a working storage record. Now, you need to define the nonshared data items for CUST-LIST-WS.

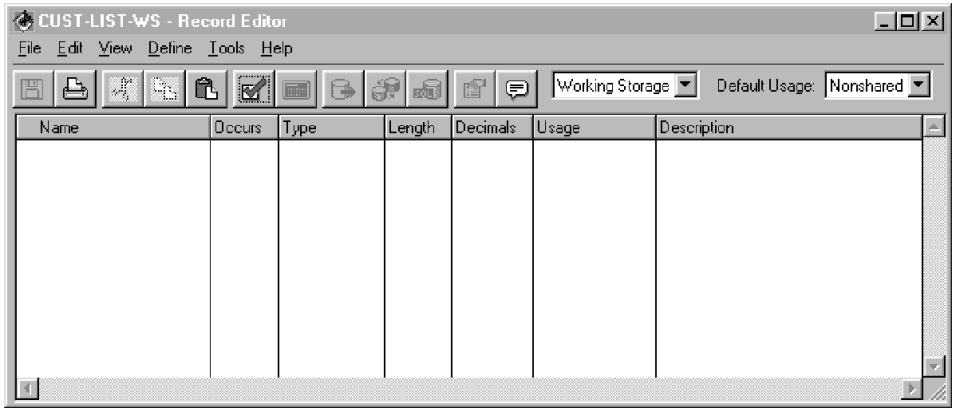


Figure 71. Record Editor

To define the nonshared data items for CUST-LIST-WS, perform the following steps:

1. Ensure that **Working Storage** is displayed in the **Record Type** drop-down list box and **Nonshared** is displayed in the **Default Scope** drop-down list box.
2. From the **Edit** menu, select **Insert After**.

   A new row is inserted and the cursor is displayed in the **Name** field.

   Tip: Choices on the context menus can make defining records faster. To display the context menu, place the mouse pointer in the data items list and click mouse button 2. For more information on defining records, see the online help.
3. In the Name field, type `cusnum-ws` and press **Tab**.

   The cursor moves to the **Occurs** field.
4. In the **Occurs** field, type `10` and press **Tab**.

   The cursor moves to the **Type** field.
5. In the **Type** field, press **Tab**.

   The default type of **Char** remains in this field and the cursor is displayed in the **Length** field.

6. In the **Length** field, type 8.

   Now add three more data items by selecting **Edit→Insert After** three times. Then, enter the values from the following table into the three new data items.

Name	Occurs	Length
cusname-ws	10	20
cuscontact-ws	10	20
cusphone-ws	10	12

   When you've entered all the data items for your working storage record, it should look like the record shown in Figure 72.

7. From the **File** menu, select **Save**.

   The record is saved.

8. Close the Record Editor.

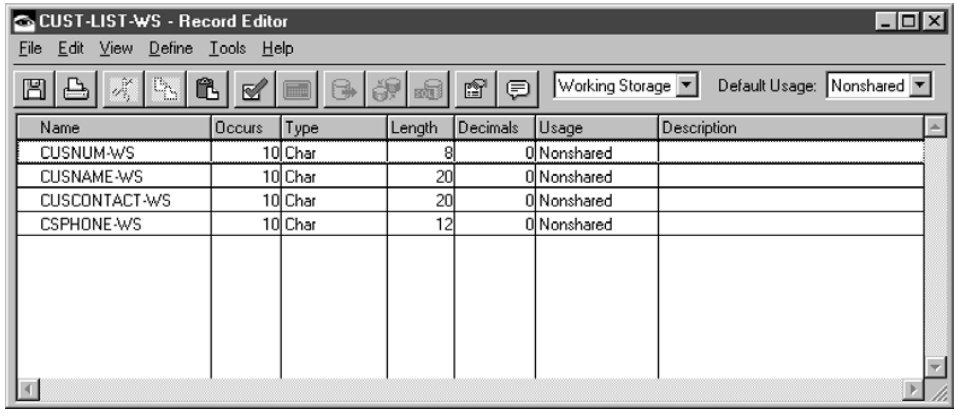


Figure 72. Record Editor

Chapter 15. Defining the processing logic

After you define the working storage record, you supply the logic for the program.

First, you create the main function for your program. Next, you define statements within the main function that start other functions.

In VisualAge Generator Developer, there are two ways to supply processing statements:
- Using the Statement Template window to construct the statement
- Typing the statement directly into the Function Editor window

Completing the main function

The steps in this section will help you construct the logic so that the program does the following:
- Displays the Customer Information map
- Enables the user to press F3, Enter, F7, and F8
- Displays a message if the user presses a key that is not valid

To begin defining the processing logic:

1. In the **Program Editor**, double-click on **MAIN-LOGIC**.

   The **Function Editor** is displayed.

Earlier, you prototyped the program by supplying a few of the statements for the MAIN-LOGIC function.

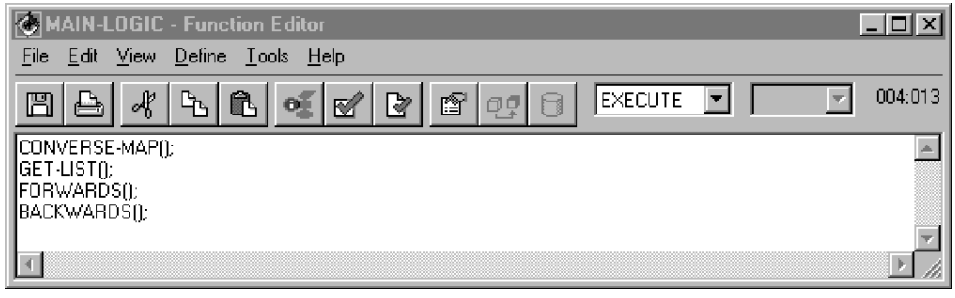


Figure 73. Function Editor

In the Function Editor, you can finish supplying processing statements for MAIN-LOGIC. Using **Statement Templates** makes this task faster and easier.

To open the Statement Templates window, from the **Tools** menu, select **Statement Templates**. The Statement Templates window, shown in Figure 74, is displayed.

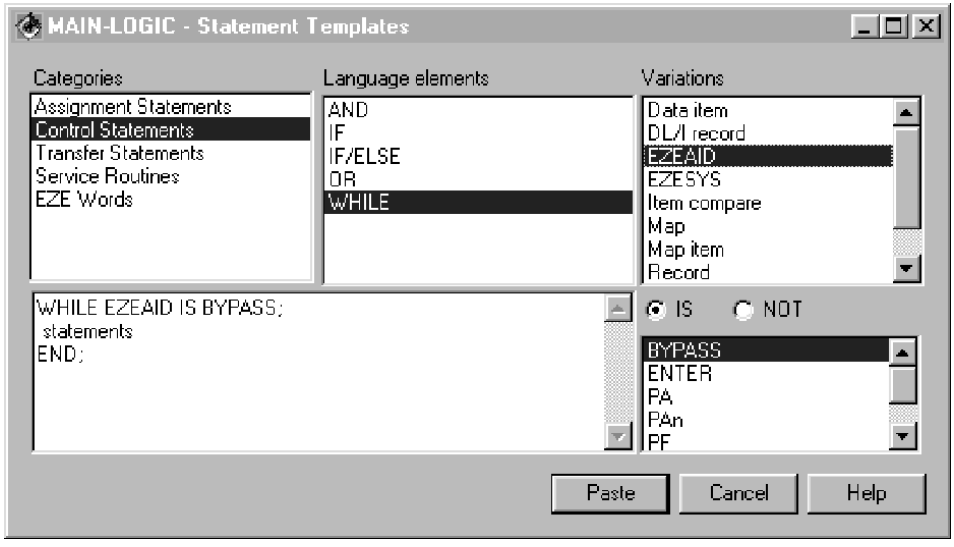


Figure 74. Statement Templates

To look at the templates, select a category in the **Categories** pane and click on an item in the **Language elements** pane. A generic form of the element is displayed in the statement area at the bottom of the window, and other forms are displayed in the **Variations** pane. You can add this generic form by placing the cursor where you want to add it in your function and double-clicking on the element in the **Language elements** pane. You can look at the variations by clicking on them in the **Variations** pane. As you click on them, the statement variations are displayed with correct syntax and appropriate place holders in the statement area. Double-click on a variation to add it to your function. Then, if you double-click on a place holder like *statements* in the Function Editor, you can easily replace it by double-clicking on a statement in the Statement Templates window.

Try using the Statement Templates window to help you enter these statements in MAIN-LOGIC:

1. In the **Function Editor**, enter the following statements:

```
while ezeaid not pf3;
converse-map();
if ezeaid is enter;
get-list();
else;
if ezeaid is pf7;
backwards();
```

```
else;
if ezeaid is pf8;
forwards();
else;
move "Key not valid, use Enter, PF7, or PF8" to ezemsg;
set ezemsg red;
end;
end;
end;
end;
backwards();
```

2. From the **Tools** menu, select **Validate and Format**.

   VisualAge Generator will validate and format the statements to look like those shown in Figure 75. You must correct any errors before you save the definition.

3. From the **File** menu, select **Save**.
4. Close the **Function Editor**.

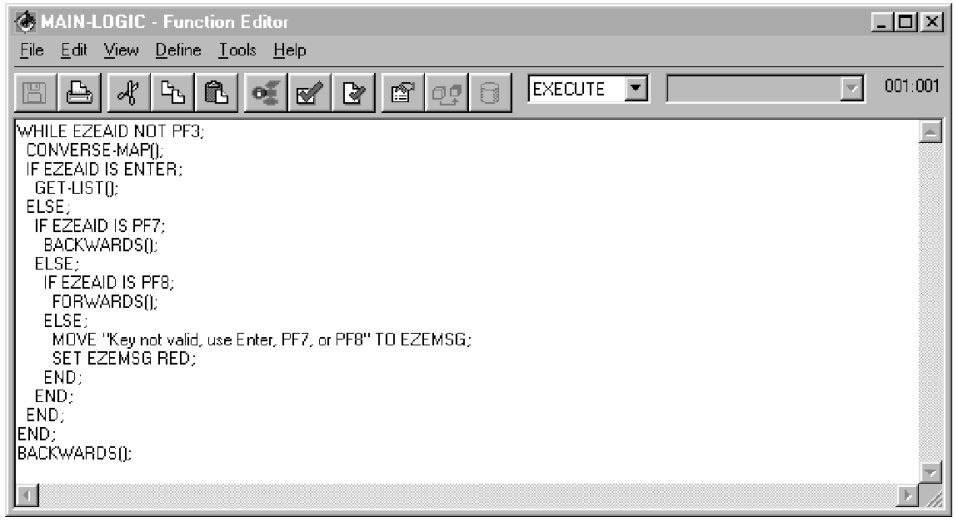


Figure 75. Function Editor

Note: As you can see, in Figure 70 on page 115 and Figure 76 on page 122, the functions FORWARDS and BACKWARDS have reversed positions in the program diagram. When you added BACKWARDS to MAIN-LOGIC between GET-LIST and FORWARDS, you changed the order of the program diagram display. The original use of BACKWARDS is still at the end of MAIN-LOGIC, but the program diagram only displays one instance of each function, even if it is used more than once in the main function.

Completing the functions

The following are functions in the sample program:

- CONVERSE-MAP
- GET-LIST
- BACKWARDS
- FORWARDS

You already defined minimum forms of some of these functions before you ran an intermediate test of the program. Now you need to finish defining them.

Complete the GET-LIST function first.

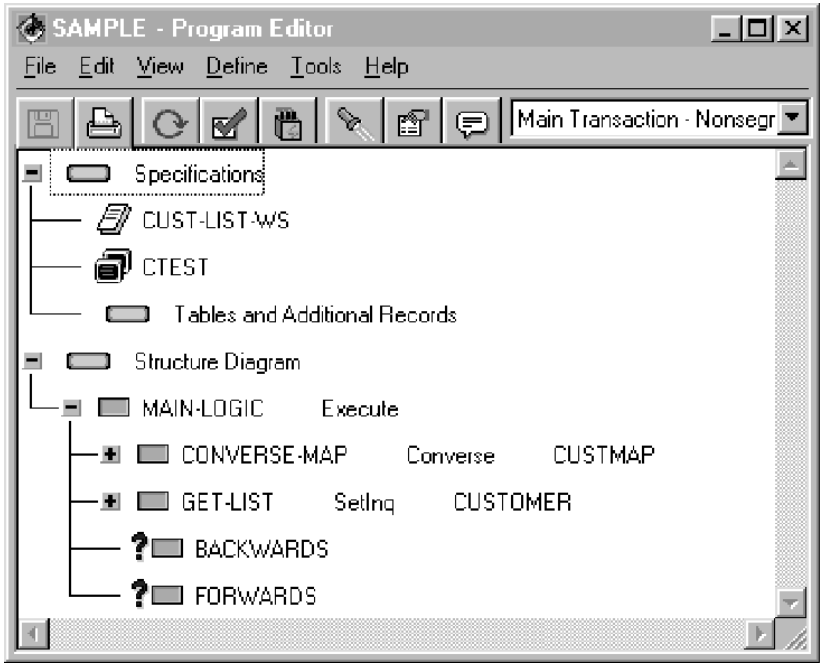


Figure 76. Program Editor

To define GET-LIST:

1. In the **Program Editor**, double-click on **GET-LIST**.
2. In the **Function Editor**, enter the following statements:

```
move custmap.cusnum to cusnum;
***---- SETINQ CUSTOMER ---------
move 0 to count;
move 0 to last;
```

```
while ezesqcod = 0 and count < 10;
read-and-save();
end;
forwards();
```

Note: You already defined the I/O option and I/O object. Add the text around the line containing the name of the I/O option and I/O object as shown here. The first line of the example should be entered on line one. The second line of the example should be entered on line three. Line numbers are displayed in the top right corner of the editor beside the **I/O Object** drop-down list box. The first number is the line your cursor is on. The second number is the number of the column your cursor is in.

3. From the **Tools** menu, select **Validate and Format**.
4. VisualAge Generator will validate and format the statements. You must correct any errors before you save the part.
5. From the **File** menu, select **Save**.
6. Your function should look like the one shown in Figure 77.
7. Close the **Function Editor**.

   As shown in Figure 78 on page 124, READ-AND-SAVE and FORWARDS have been added to the program diagram under GET-LIST.

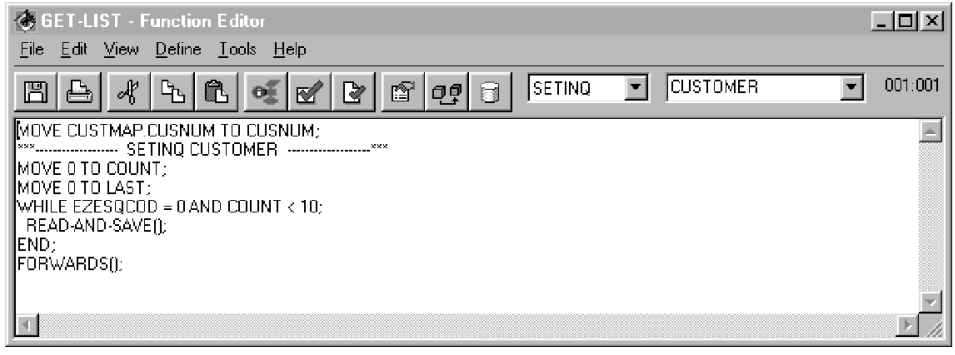


Figure 77. Function Editor

Now, you can finish defining the BACKWARDS function.

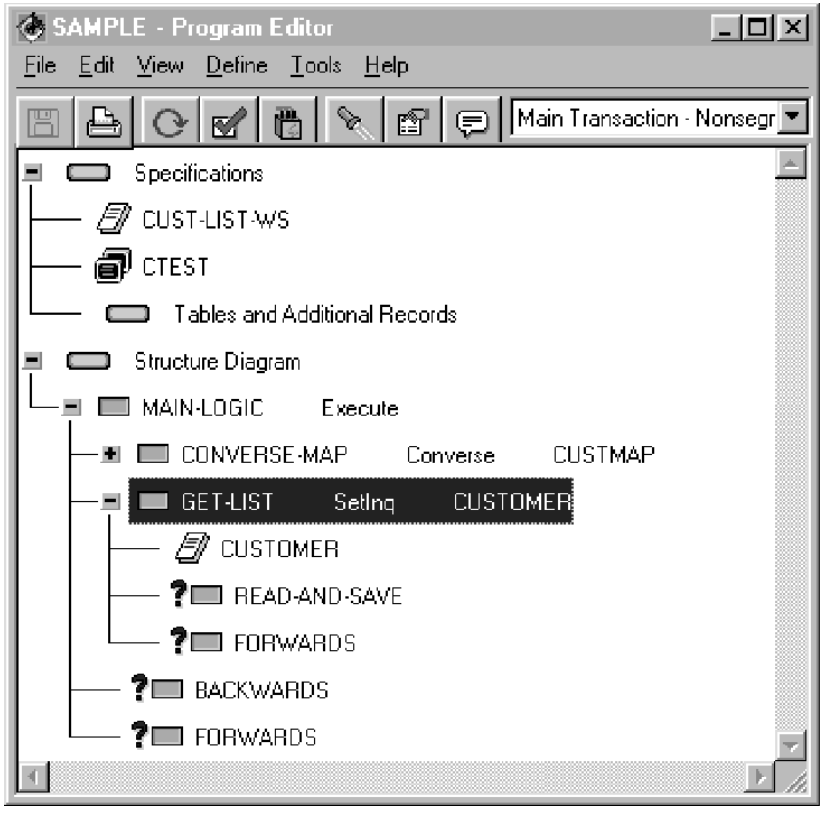


Figure 78. Program Editor

To define BACKWARDS:

1. In the **Program Editor**, double-click on **BACKWARDS**.
   The New Part Package/Application window is displayed.
2. Ensure that **tutorial** is selected, and select **OK**.
   The new part is created and the **Function Editor** is displayed.
3. Enter the following statements:

```
sub2 = last - 5;
if sub2 < 1;
move "Top of Customer file" to ezemsg;
else;
last = sub2 - 5;
if last < 0;
move 0 to last;
end;
move-it();
end;
```

4. From the **Tools** menu, select **Validate and Format**.

VisualAge Generator will validate and format the statements. You must correct any errors before you save the part.

5. From the **File** menu, select **Save**.

   Your function should look like the one shown in Figure 79.

6. Close the **Function Editor**.

   As shown in Figure 80 on page 126, MOVE-IT has been added to the program diagram under BACKWARDS.

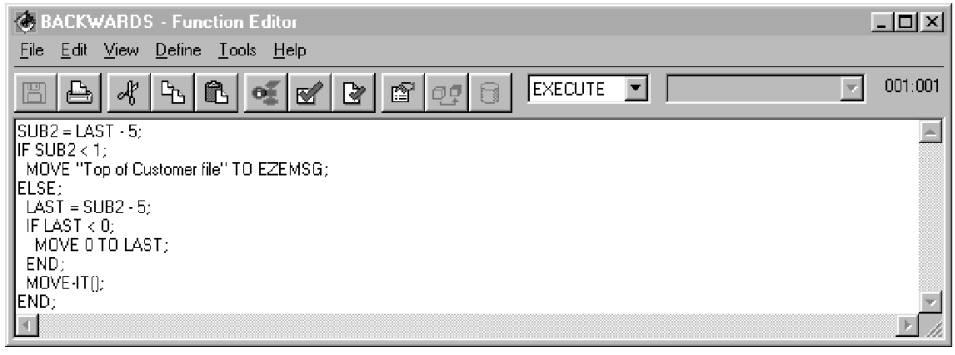


Figure 79. Function Editor

Now, you can finish defining the FORWARDS function.

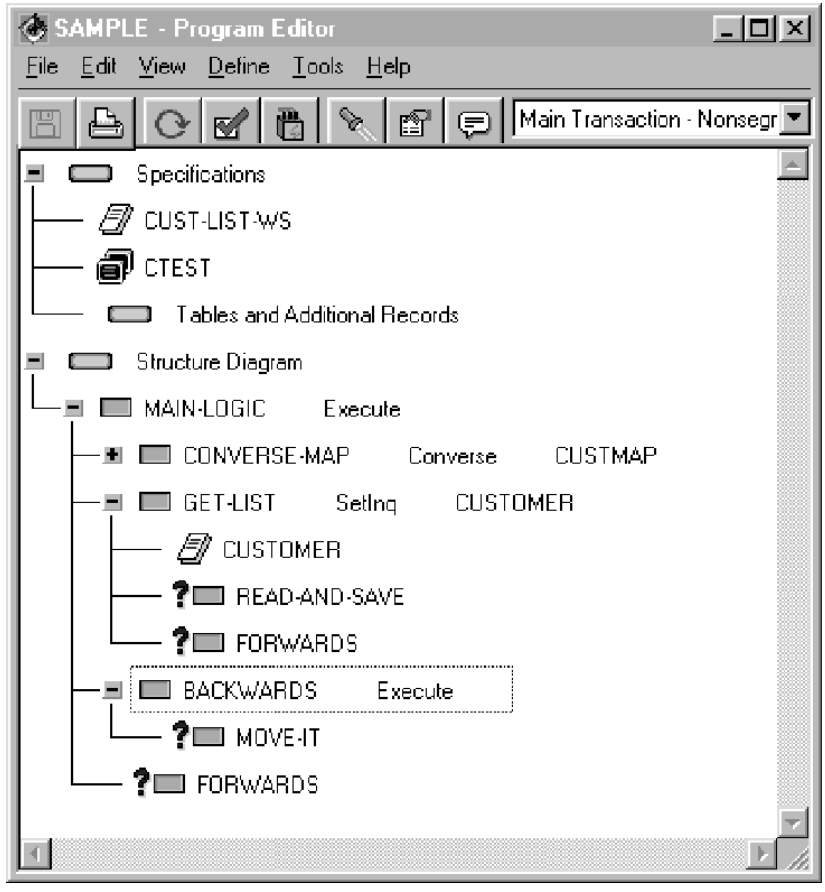


Figure 80. Program Editor

To define processing statements for FORWARDS:

1. In the **Program Editor**, double-click on **FORWARDS**.
   The New Part Package/Application window is displayed.
2. Ensure that **tutorial** is selected, and select **OK**.
   The new part is created and the **Function Editor** is displayed.
3. In the **Function Editor**, enter the following statements:

```
if last = count;
move "No more Customer Records" to ezemsg;
else;
move-it();
end;
```

4. From the **Tools** menu, select **Validate and Format**.
   VisualAge Generator will validate and format the statements. You must correct any errors before you save the part.

5. From the **File** menu, select **Save**.

   Your function should look like the one shown in Figure 81.
6. Close the **Function Editor**.
7. In the **Program Editor**, MOVE-IT has been inserted under FORWARDS in two places.

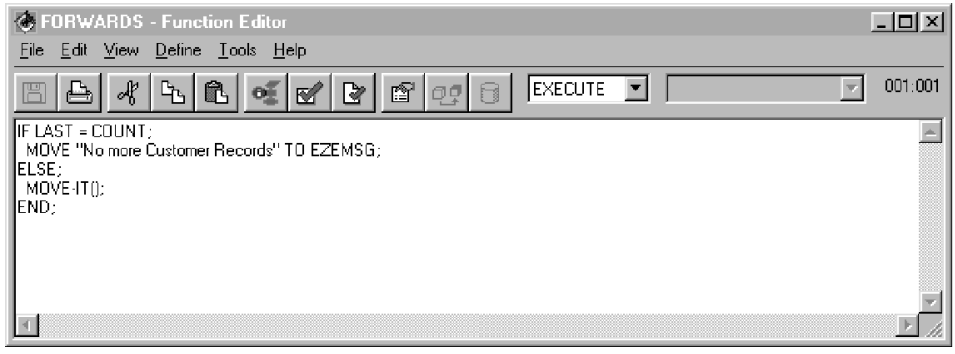


Figure 81. Function Editor

Completing the remaining functions

In the functions you just completed, you named some implicit data items, for example LAST and COUNT. VisualAge Generator will create these implicit data items dynamically as needed in the program. You can specify in program properties whether or not VisualAge Generator creates implicit data items.

Two of the performed functions you named, READ-AND-SAVE and MOVE-IT are third or fourth-level performed functions. You must define these functions.

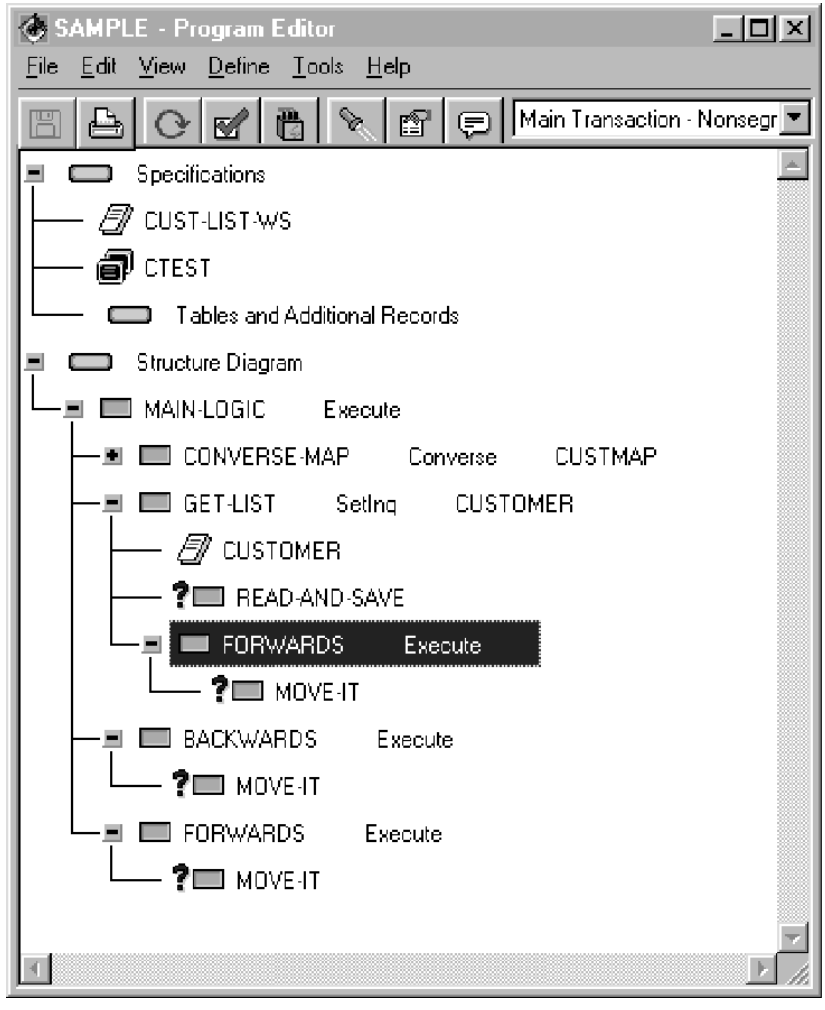


Figure 82. Program Editor

Define a I/O option that will read the next record from the database selected by the SETINQ I/O option in the function GET-LIST. Also, specify an error routine that causes an immediate return from the READ-N-SAVE function if an error condition occurs when the record is read.

To define READ-AND-SAVE:

1. In the **Program Editor**, double-click on **READ-AND-SAVE**.
   The New Part Package/Application window is displayed.
2. Ensure that **tutorial** is selected, and select **OK**.
   The new part is created and the **Function Editor** is displayed.
3. In the **I/O Option** drop-down list box, select SCAN.

4. In the **I/O Object** drop-down list box, select CUSTOMER.
5. From the **Define** menu, select **Properties**.

   The Function Properties window (Figure 83) is displayed.
6. From the **Error routine** drop-down list box, select **EZERTN**.
7. Select **OK**.

To supply processing statements for READ-AND-SAVE:

Figure 83. Function Properties

1. In the **Function Editor**, enter the following statements after the line containing the I/O option and I/O object:

```
if ezesqcod = 0;
count = count + 1;
sub = sub + 1;
move cusnum to cusnum-ws[sub];
move cusname to cusname-ws[sub];
move cuscontact to cuscontact-ws[sub];
move cusphone to cusphone-ws[sub];
end;
```

2. From the **Tools** menu, select **Validate and Format**.

   VisualAge Generator will validate and format the statements. You must correct any errors before you save the part.
3. From the **File** menu, select **Save**.

   Your function should look like the one shown in Figure 84 on page 130.
4. Close the **Function Editor**.
5. In the **Program Editor**, the CUSTOMER record and EZERTN are displayed under READ-AND-SAVE.

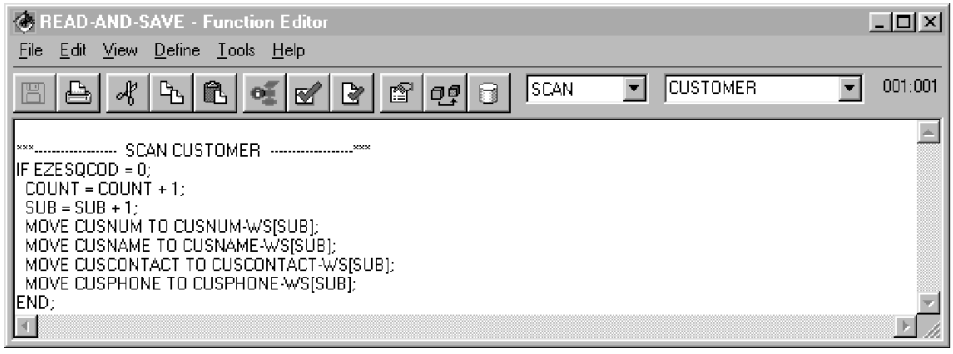


Figure 84. Function Editor

Now, you can define the logic behind MOVE-IT.

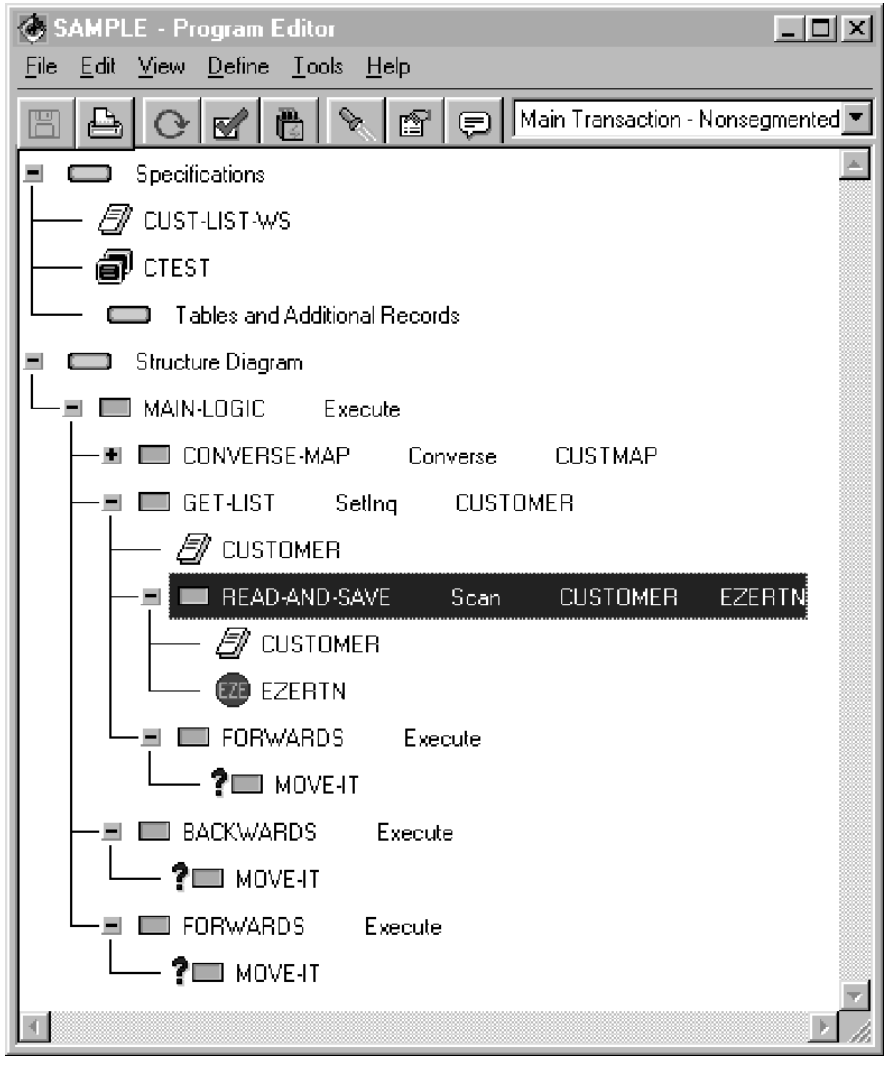


Figure 85. Program Editor

To define MOVE-IT:

1. In the **Program Editor**, double-click on **MOVE-IT**.

   The new part is created and the New Part Package/Application window is displayed.
2. Ensure that **tutorial** is selected, and select **OK**.

   The new part is created and the **Function Editor** is displayed.
3. Enter the following statements:

```
move 0 to sub;
move last to sub2;
set custmap empty;
while sub < 5 and sub2 < count;
sub = sub + 1;
sub2 = sub2 + 1;
move cusnum-ws[sub2] to cusnum-a[sub];
move cusname-ws[sub2] to cusname-a[sub];
move cuscontact-ws[sub2] to cuscontact-a[sub];
move cusphone-ws[sub2] to cusphone-a[sub];
end;
move sub2 to last;
```

4. From the **Tools** menu, select **Validate and Format**.

   VisualAge Generator will validate and format the statements. You must correct any errors before you save the part.

5. From the **File** menu, select **Save**.

6. Close the **Function Editor**.

   In the **Program Editor**, MOVE-IT is displayed as defined under FORWARDS and BACKWARDS. Your program should look like the one shown in Figure 86 on page 133.

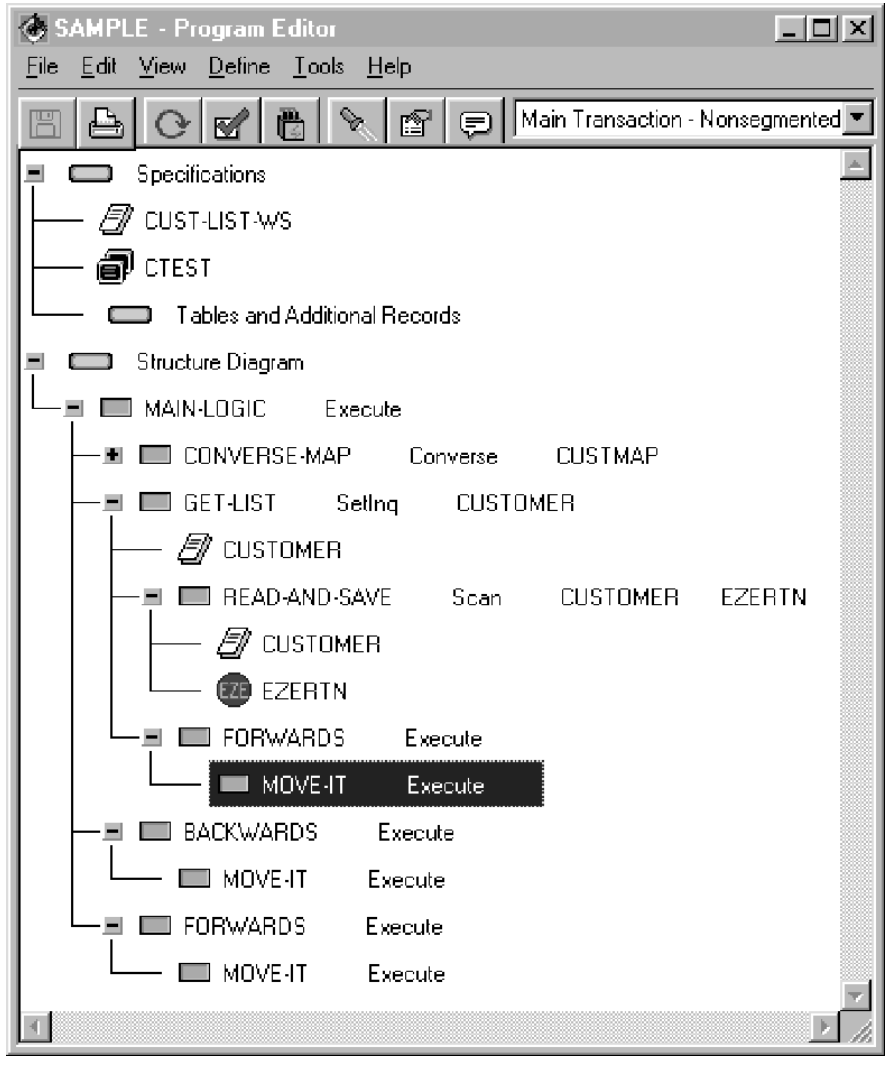


Figure 86. Function Editor

Chapter 16. Preparing for generation

Now that you have defined your program fully, you can test it thoroughly. You can begin to look for the correct handling of boundary conditions and errors in the logic.

Remember that, while testing, you can immediately move into the appropriate editor and correct any errors you find, without closing the test facility. After you have corrected the error and saved the part, reposition the test run to a location just before the error, and continue testing. When you are satisfied with the way the program works under the test facility, you are ready for generation.

To generate your program:

1. In the **Program Editor**, from the **Tools** menu, select **Generate**
   The Generate window is displayed.

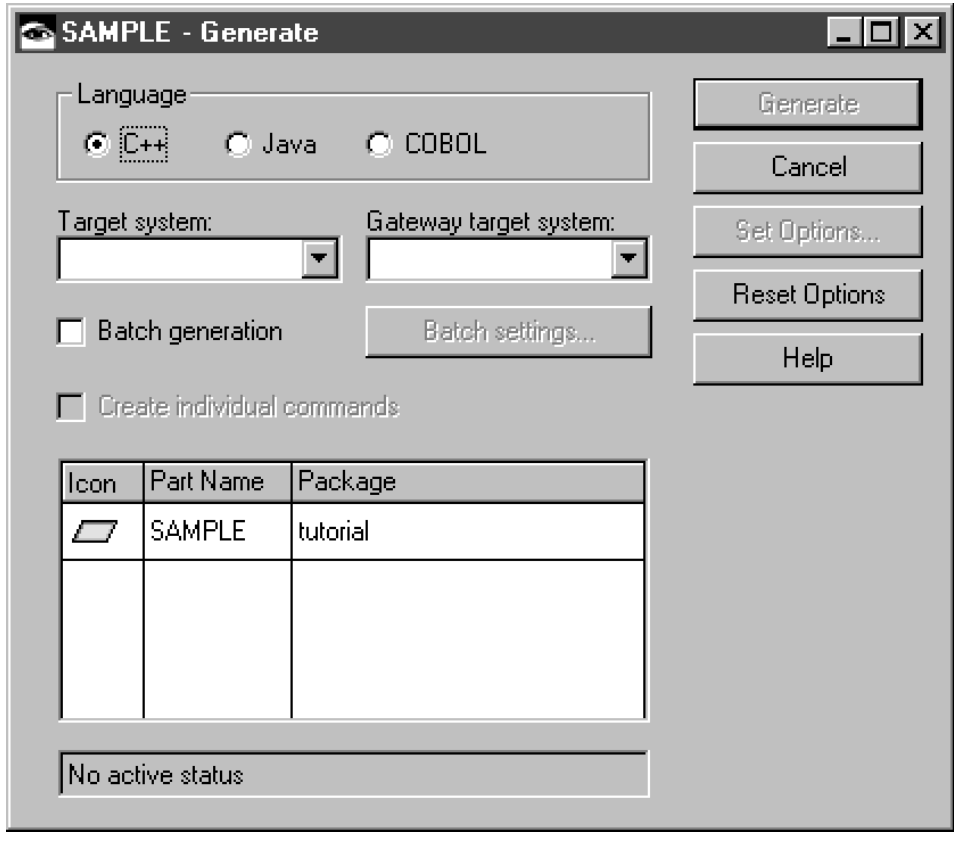


Figure 87. Generate Window

From this point you could start the generation process for your program. However, this tutorial does not take you through the steps to generate a program. For information on generating and preparing programs, refer to the *VisualAge Generator Design Guide*.

2. Select **Cancel** to close the Generate window.

For step-by-step instructions on building a client/server application with a graphical user interface, proceed to "Chapter 17. Building a visual part using VisualAge for Java" on page 137 or "Chapter 18. Building a visual part using VisualAge Generator Developer on Smalltalk" on page 159.

Chapter 17. Building a visual part using VisualAge for Java

This section of the tutorial provides you with experience building visual parts. The objective of this section is to help you become familiar with the Composition Editor and the tools it provides you for building Java visual parts called beans. Using the Beans palette, a graphical list of parts you can use to design a user interface, you will build the Customer Information window and size and align labels, fields and buttons within the window. You'll also use this palette to place VisualAge Generator data and logic parts on the free-form surface where you'll connect them to your visual part.

Before you start

If you are using VisualAge Generator on Smalltalk, refer to "Chapter 18. Building a visual part using VisualAge Generator Developer on Smalltalk" on page 159.

If you have not completed the steps in "Chapter 8. VisualAge Generator Developer on Java: a tutorial" on page 65, please go back and complete all the steps in that section before you create a new visual part.

Creating a new bean

The window or bean you create in this tutorial (shown in Figure 88 on page 138) displays detailed customer information.

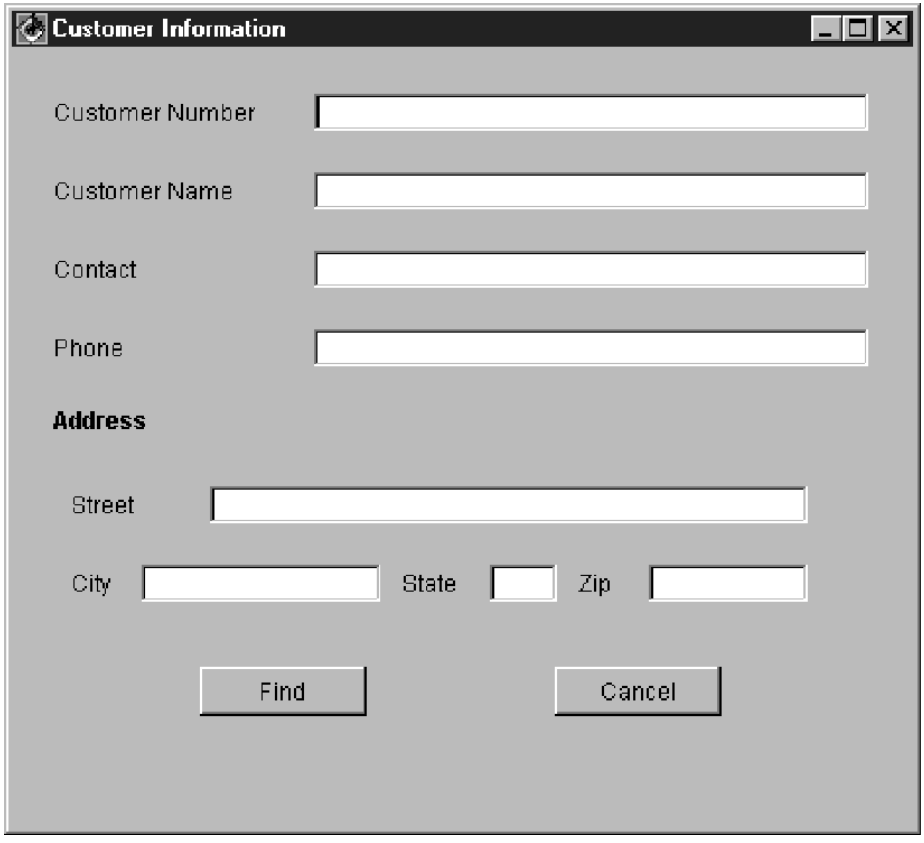


Figure 88. Customer Information Window

To create a new visual part:

1. On the Workbench **Projects** tab, select the **tutorial** package.
2. From the **Selected** menu, select **Add→Application**.

   The Create Application SmartGuide is displayed. The **Project** and **Package** fields are filled for you.
3. In the **Class name** field, type the name `TutorialView`.

   Create Swing based application is selected for you.
4. Select **Next**.

   The Application Details page is displayed.
5. In the **Title bar text** field, type `Customer Information`.
6. Deselect all options except **Center and pack frame on screen**.
7. Select **Finish**.

The window is displayed in the Composition Editor. The default size of the window is width: 460 and height: 300. The area outside of the window is called the free-form surface.

Adding a VAGen Record to the visual part

Now you are going to build a data part for your visual part. The fields (columns) in a single row of a relational database table are described in a VAGen record.

1. From the list above the Beans palette, change **Swing** to VAGen Parts.

   The Beans palette is refreshed to show all the part types of the VAGen Parts category.

2. Select the **VAGen Record** part.  .
3. Drop the part by clicking on the free-form surface to the right of the Customer Information window. The mouse pointer becomes a cross-hair when your cursor is over an area where you can drop the part.

   The Add Part window, shown in Figure 89, is displayed. You can enter a part name (in this case, a record name) or use an existing VisualAge Generator record by selecting from the drop-down list.

Figure 89. Add Part Window

4. Type `customer` as the record name, and select **OK**.

   The CUSTOMER record is now on your free-form surface and ready to be defined. This name can be the same or different from the table name in the relational database.

5. Click with mouse button 2 on the CUSTOMER record.

   A context menu is displayed.

6. Select **Open**.

   The CUSTOMER record is displayed in the Record Editor. This is the same record defined in "Chapter 11. Defining an SQL record" on page 89. If you completed that section of the tutorial, this record is already defined, except for the data item descriptions.

Note: If you have not already defined the CUSTOMER record, the New Part Package/Application window is displayed. Ensure that **tutorial** is selected and select **OK**. The record editor is displayed.

Defining the CUSTOMER VAGen Record Part

Here you define the VAGen Record part as an SQL row record and use the CUSTOMER table definition in the relational database catalog to define the contents of the record.

Note: If you have already completed the steps in "Chapter 11. Defining an SQL record" on page 89, skip to the last step in this section to customize the data item descriptions.

1. In the Record Editor, select **SQL row** from the Record Type drop-down list box.

   VisualAge Generator supports accessing a variety of file types and databases. In this case you are going to access a relational database table, so you selected SQL row as the record type.
2. From the **Define** menu, select **Properties**.

   The SQL Row Properties window is displayed.
3. Select **Insert**.

   A blinking cursor is displayed in the **Name** field.
4. In the **Name** field, type `customer` and select **OK**.

   The SQL Row Properties window closes.
5. From the **Tools** menu, select **Retrieve SQL**.

   Note: If a message window is displayed, indicating that the system cannot locate a database access plan, select **Yes** to have a plan created.

   If you have not already connected to the database, you will be prompted for an ID and Password. Enter the ID and Password you used in setting up your database.

   The table definition information is retrieved from the relational database catalog and used to create data item definitions for the record.
6. Select the **Key** field of the CUSNUM data item. Select the **Key** check box displayed in that field.
7. From the **File** menu, select **Save**.
8. Close the Record Editor.

For more information about defining records, refer to the VisualAge Generator help facility.

Now that you've defined an SQL Row record, you're ready to add fields to the Customer Information window and define connections to the record. Data items in the CUSTOMER record will be visually connected to parts of the Window with a blue line, indicating a property-to-property connection. With this type of connection:

- When data is entered in the fields, it is automatically moved to the corresponding data items in CUSTOMER.
- When data items in CUSTOMER are changed, the new values are also automatically moved to the fields (when the window is displayed).

Customer Information window

In this section of the tutorial, you'll begin building the parts of the Customer Information Window. You'll add labels and fields for the Customer Number, Customer Name, Contact, and Phone and connect those fields to data items in the CUSTOMER record. You'll add a label for the Address panel and Find and Cancel buttons.

Adding and connecting labels and fields

1. From the list above the Beans palette, select the **Swing** category.
2. Select **JLabel**  and drop this bean inside the Customer Information Window.
3. Select **JTextField**  and drop this bean inside the Customer Information Window.

   See Figure 96 on page 160 for the placement of these beans. These beans are for the Customer Number label and its corresponding text area. You will need to resize the JTextField because it is small by default.
4. Double-click **JLabel1** in the Customer Information window.

   The Properties window is displayed with properties listed in alphabetical order.
5. In the **text** field, type the name `Customer Number`.

   The JLabel now says Customer Number. It may be necessary to resize the JLabel so that all of the text is displayed.
6. Close the Properties window to commit the changes.
7. Select the Customer Number JTextField with mouse button 2.

   A context menu is displayed.
8. Select **Connect→Connectable Features** from the context menu.

   The Start connection from window is displayed.
9. Ensure that the Property radio button is selected and select **text**.

10. Select **OK**

    The mouse pointer is displayed as connected to a dashed line. This indicates that you are in the process of making a connection.

11. Click the CUSTOMER record part with mouse button 1.

    A context menu is displayed.

12. Select **Connectable Features**.

    The End connection to window is displayed.

13. Select **CUSNUM data**, then select **OK**.

    A blue line is displayed connecting the Customer Number field and the *CUSNUM data* property in the CUSTOMER record. The description of the connection is displayed in the status area at the bottom of the Composition Editor.

14. Double-click on the connection you just created.

    The property-to-property connection window is displayed.

15. From the **Source event** drop-down list, select **keyReleased**, then select **OK**.

    Specifying the **keyReleased** event for this connection causes the data item CUSNUM in the CUSTOMER record to be updated when the Customer Number field is changed, and causes the Customer Number field (when displayed) to be updated when the data item CUSNUM is changed.

Use the same procedures to create labels and fields for the CUSNAME, CUSCONTACT, and CUSPHONE data items in the CUSTOMER record.

The basic steps you should follow add and connect these parts are:

1. Add a label.
2. Change the label to reflect data item name.
3. Add the field.
4. Make a property-to-property connection between the field and the appropriate data item in the CUSTOMER record and set the source event.

Adding the Find and Cancel buttons

Complete the following steps to add push buttons to the bottom of your Window.

1. Press **Ctrl** and from the **Beans palette**, select **JButton**  .

   Tip: Pressing **Ctrl** while you select a bean from the **Beans palette** enables the **Sticky** feature. Use **Sticky** to drop more than one of the same type of bean. To turn **Sticky** off, select another bean or the selection tool.

2. Click twice at the bottom of the Window.

   Two push buttons are displayed at the bottom of the Window.

3. Click the selection tool  .
4. To change the push button label, double-click on the button on the left side of the Customer Information window to display the Properties window.
5. Click in the field beside **text** and type `Find`.

   The push button is now labeled Find.
6. Close the Properties window to commit the changes.
7. Double-click the second button
8. Click in the field beside **text** and type `Cancel`.
9. Close the Properties window to commit the changes.

Now that you have most of the visual parts of your user interface on the Window, you can use the tool bar to align those parts in the window.

Arranging visual parts

To arrange visual parts you can use the tool bar buttons or selections from the Tools menu. Each tool bar button has a corresponding menu choice with the same name that performs the same function. The menu choices or buttons you use to arrange visual parts are as follows:

Align Left 

Align Center 

Align Right 

Align Top 

Align Middle 

Align Bottom 

Distribute Horizontally 

Distribute Vertically

Match Width

Match Height

In this tutorial, you'll be directed to the menu choices, but feel free to use the tool bar buttons instead.

Note: Like the choices on the **Tools** menu, many of the buttons on the tool bar are unavailable unless you have more than one visual part selected. To select two or more parts at the same time,

1. Place your mouse pointer over the part you want to select.
2. While holding down the **Ctrl** key, move the mouse pointer to another part and click and release mouse button 1.

   Repeat step 2 as many times as necessary, until you have selected all the parts you need to size or align in relation to each other.

   Notice that the part you select last has solid handles, while the other selected parts have hollow handles. The part with solid handles is used as a reference point. So, when you want to align several parts, select the parts you want to move first. Those parts will be aligned with the part you select last.

In "Defining a reusable visual part" on page 172, you will add an Address panel to the Customer Information window. So arrange the labels and fields at the top of the Window and the push buttons at the bottom, leaving enough room for the Address panel between them.

For a complete description of the tool bar and the tools available, refer to *Composition Editor* in the VisualAge help facility or the *VisualAge for Java User's Guide*.

To arrange the visual parts on the Customer Information window, perform the following steps:

1. Select the Find button and then the Cancel button.

   The Find button has hollow handles and the Cancel button has solid handles.
2. From the **Tools** menu, select **Match Width**.

The Find button is resized to match the Cancel button in width.

3. From the **Tools** menu, select **Match Height**.

   The Find button is resized to match the Cancel Push Button in height.

4. From the **Tools** menu, select **Align Middle**.

   The Find button's vertical center is aligned to the center of the Cancel button.

5. From the **Tools** menu, select **Distribute Horizontally**.

   The two buttons are placed at regular intervals across the Window.

Using the bounding box

Next, align the labels and fields in the top portion of the Window, leaving space in the middle for the Address panel.

The *bounding box* enables you to align selected parts within a portion of your Window, rather than the entire Window. The selected portion is defined by the topmost and bottommost, or leftmost and rightmost sides of the parts selected.

1. Select the Phone label and the associated field.

   Both parts display selection handles.

2. Move these parts to where you want them in the Window using mouse button 1.

   The Phone label and the associated field define the bottom of the bounding box, while Customer Number defines the top of the bounding box.

3. Select all the fields (do not select the labels), making sure the Customer Number field is the last one selected.

   The Customer Number Text part should have primary selection handles (solid).

4. Click mouse button 2 on one of the selected parts. Select **Layout→Distribute→Vertically In Bounding Box**.

   Now the fields are all evenly distributed in the top portion of the Window.

5. From the **Tools** menu, select **Align Left**  to align the parts on the left.

   All the fields are aligned on the left.

6. Select all the labels, making sure that the Customer Number label is the last one selected.

7. Click mouse button 2 on one of the selected parts. Select **Layout→Distribute→Vertically In Bounding Box**.

   Now the labels are all evenly distributed in the top portion of the Window. If you are not satisfied with your results, from the **Edit** menu, select **Undo** and try again.

8. From the **Tools** menu, select **Align Left**.

   All labels are aligned on the left.

In the next chapter, you will add the Address panel to this Window.

Adding an action to the Cancel button

When users select the Cancel button, you want the Customer Information window to close. This relationship is called an event-to-method connection.

1. To make a connection from the Cancel push button, click mouse button 2 on the Cancel push button.

   The context menu is displayed.

2. From the context menu, select **Connect→actionPerformed**.

   The mouse pointer is displayed as ✳ connected to a dashed line. This indicates that you are in the process of making a connection.

3. Click mouse button 1 on the title bar of your Window (the area that says Customer Information).

   A context menu is displayed.

4. Select **dispose()**.

   A green line with an arrow pointing from the Cancel push button to the Window edge is displayed. This connection means that when the Cancel push button is clicked, the **dispose()** method is triggered and the window is closed.

5. From the **Bean** menu, select **Save Bean**.

Testing the Customer Information window

The next step is to test the window (the bean) to ensure that the user interface is displayed and the Cancel button works the way you want it to.

1. From the tool bar, select the **Run**  button.

   The Customer Information window is displayed. Verify your layout of the window.

   Note: The window will need to be resized to display the entire layout.

2. Select Cancel on your Customer Information window.

   The window is closed.

3. Close the Composition Editor.

For more information on testing visual parts, refer to the VisualAge for Java online help.

After you have added some VAGen logic parts, you can use VisualAge Generator test facility to test the connections from this user interface to your VAGen logic and data parts. This powerful feature enables you to step through a test of your user interface connections and logic and data parts, modify code, and set break, watch, and test points.

Defining the address panel

The objective of this section is to familiarize you with creating reusable visual parts and using them with VAGen parts. This exercise focuses on the following tasks:

- Building a reusable bean called *ReusableAddressBoxView*, consisting of a panel that includes labels and fields for address data
- Promoting the features of the bean to the public interface so that they can be accessed by other beans

Creating a reusable bean

You will build a panel and add labels and fields for address information. Figure 90 shows the panel you will define.

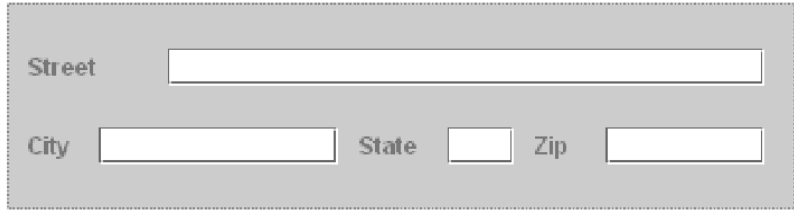


Figure 90. Address Panel

1. On the Workbench **Projects** tab, select the **tutorial** package.
2. From the **Selected** menu, select **Add→Class**.
   The **Create Class** SmartGuide is displayed.

   Tip: Choices on the **Selected** menu are the same as those on the context menu displayed when you select a package and click mouse button 2.
3. In the **Class name** field, type the name `ReusableAddressBoxView`.
4. In the **Superclass** field, type the class `javax.swing.JPanel`.
5. Select **Compose the class visually**.
6. Select **Finish**.
   The new visual part is displayed in the Composition Editor.

Adding parts to the address panel

In this section of the tutorial, you will add labels and fields for street, city, state, and zipcode.

Complete the following steps to add parts to the Address panel:

1. Resize the address panel by selecting it and dragging the selection handles on the right or left side.

   Alternatively, you can resize the panel by double-clicking on it to display the Properties window where you can expand the **constraints** field and set the values in the **width** and **height** fields to 400 and 100 respectively.

2. From the Beans palette, select **JLabel**  .
3. Drop the part on the panel.
4. Double-click the label you dropped.

   The Properties window is displayed.
5. In the **text** field type `Street`.
6. Select **JTextField**  .
7. You will need to resize the JTextField because it is small by default.
8. Repeat the previous steps to create labels and associated text entry fields for City, State, and Zip as shown in Figure 90 on page 147.

   For more detailed information about arranging visual parts see "Arranging visual parts" on page 143.

Promoting features of the reusable part

If you want to access any information in the Address panel from another visual part, you must add the appropriate features to the bean's public interface. In this case, the features needed are the properties for the text field beans. In this section, you will add properties for all four beans (Street, City, State, and Zip) to the public interface.

For more complete information on topics covered in this section, refer to the VisualAge for Java online help.

To promote bean features to the public interface, perform the following steps:

Note: Before you promote bean features, you should rename the beans so that you will recognize them when you need to access them from other beans. To rename a bean, select it with mouse button 2 to display the context menu. Select **Change Bean Name** and type a name in the **New bean name** field.

1. Select the Street entry field and rename it to `StreetField`.

2. Select the Street entry field with mouse button 2. From the context menu, select **Promote Bean Feature**.

   The Promote features from window is displayed.

3. On the Promote features from window, select the property **text** and select the >> button.

   `StreetFieldText` is displayed in the **Promoted features** list.

4. Select **OK**.
5. Repeat the above steps for the City, State, and Zip entry fields, using the names `CityField`, `StateField`, and `ZipField`, respectively.

   The *text* property feature of each of the four beans has been added to the public interface. Now these properties can be accessed by other beans.

6. Select **Bean→Save Bean**.
7. Close the Visual Composition Editor.

Now you are ready to embed the address box in your tutorial view and add VAGen logic parts.

Defining a VAGen server program

The focus of this section is to finish developing your user interface and connect it to a server application that reads data to be displayed in the Customer Information window. The objective of this section is to familiarize you with making connections between user interfaces and nonvisual parts, such as records and server applications. In addition, you will build the server application and embed the Address panel you created in the previous section. Finally, you will use the interactive test facility to test how the client and server applications work together.

Embedding a reusable visual part

You can now complete the window in Figure 91 on page 150, which displays detailed customer information.

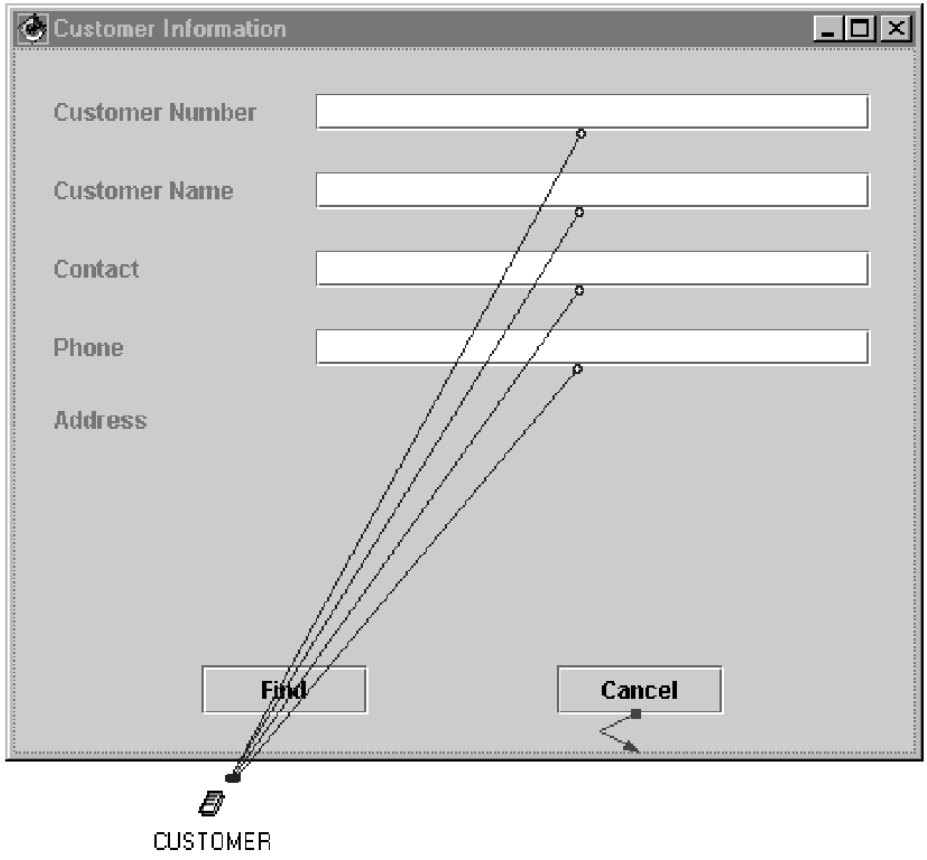


Figure 91. Customer Information Window

To complete the window, add the Address panel you created in "Defining the address panel" on page 147 to the Customer Information Window. To add the reusable visual part, perform the following steps:

1. On the Workbench **Projects** tab, double-click on **TutorialView**.

   The Composition Editor opens, displaying the Customer Information window.

2. From the Beans palette, select **Choose bean**  .

   The Choose bean window is displayed

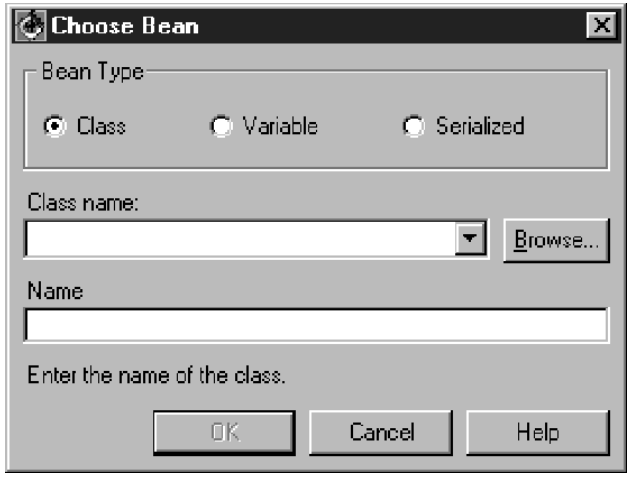


Figure 92. Choose Bean Window

1. In the **Class Name** field, type `tutorial.ReusableAddressBoxView` and select **OK**.

   The mouse pointer becomes a cross hair.

   Tip: Alternatively, you can select the **Browse** button and select the class name from the displayed list. The package name is automatically appended for you.
2. Click on the Customer Information window where you want to add the bean.

   The Address panel is added to the Window.
3. Size the Address panel (and the Customer Information window if necessary) until the panel fits inside the window.
4. Select the Address panel with mouse button 2. From the context menu, select **Connect→Connectable features**.

   The **Start connection from** window is displayed.
5. Select **Property**.
6. Select **StreetFieldText**.

   A dotted line with a ✳ at the end is displayed.
7. Select the CUSTOMER Record part.
8. From the context menu, select **Connectable features**.

   The **End connection to** window is displayed.
9. Select CUSSTREET data and select **OK**.

   This creates a property-to-property connection between the Street entry field and the data item CUSSTREET in the CUSTOMER Record part.

By selecting the *data* property, you are making a connection to only the data contents of the data item. If the connection required a type description in addition to the data contents, you would have selected CUSSTREET, which represents the *this* property of the data item.

The entry fields in windows, as in this example, typically only require the data contents. However, when you connect to a parameter in a called parameter list for a program, the data contents and the type description are required.

10. Repeat the previous steps for CityFieldText, StateFieldText, and ZipFieldText. Connect them to CUSCITY data, CUSSTATE data, and CUSZIP data, respectively.

    This creates property-to-property connections between the fields in the panel and the data items in the CUSTOMER Record part. If you want to check any of the connections, select the connection and the status area will contain a description of it.

Adding a VAGen Program to the visual part

Now that all the user interface parts are in place on the Window, you'll add a VAGen Program part to the free-form surface and connect it to the Find button. The VAGen Program part represents a server program that you will define later in this exercise. The server program will read the customer details from the database and return the data to TutorialView to populate the fields in the Customer Information window.

1. From the list above the Beans palette, select the **VAGen Parts**.

   A graphical list of VAGen parts is shown on the Beans palette.

2. Select the **VAGen Program** part  and click on the free-form surface outside of your Window.

   The Add Part window is displayed.

3. In the **VAGen part** field, type `findcus` and select **OK**.

   The FINDCUS program part is now on your free-form surface.

4. Select the Find button with mouse button 2.

   The context menu for the Find button is displayed.

5. Select **Connect→actionPerformed**.

   A line with a  on the end is displayed.

6. Select the FINDCUS program part.

   A context menu is displayed.

7. From the context menu, select **execute**.

A green line with an arrow now connects the Find button with the FINDCUS program part. This connection means that when the Find button is clicked, FINDCUS is called.

8. Select the FINDCUS program part with mouse button 2.

   A context menu is displayed

9. Select **Open**.

   The New Part Package window is displayed

10. Select the **tutorial** package and select **OK**.

    The Program Editor opens to display FINDCUS.

Defining the VAGen server application

In this section you define the VAGen server application FINDCUS, which reads the customer information from a relational database. To retrieve the correct customer information, the record key (CUSNUM) for the database must be passed to the server application from TutorialView. To display the retrieved information, the data to populate the Customer Information window must be returned from the server application. Parameters are the mechanisms by which data is passed between applications.

To define the server application, in the Program Editor, perform the following steps:

1. From the **Define** menu, select **Called Parameters→Insert Parameter**.

   The Insert Parameter window is displayed.

2. Ensure that the **Record** radio button is selected, and from the **Part Name** drop-down list box, select **CUSTOMER**.

3. Select **OK**.

   The CUSTOMER record is displayed in the program diagram under **Called Parameters**. Now you are ready to define the main functions for FINDCUS.

4. From the **Define** menu, select **Main Functions**.

   The Insert Main Function window is displayed.

5. In the Part name field, type `find-customer` and select **OK**.

   The FIND-CUSTOMER function is displayed in the structure diagram. The ? beside it means that it has not yet been defined.

6. Double-click on the function icon.

   The New Part Package window is displayed.

7. Select **OK**.

   The Function Editor is displayed.

8. From the **I/O Option** drop-down list box, select **INQUIRY**. In the **I/O Object** field, type `custinfo`.

The I/O option and I/O object are displayed on the second line in the Function Editor.

9. From the **Define** menu, select **Properties**.

   The Function Properties window is displayed.

10. From the **Error routine** drop-down list box, select **EZERTN** and select **OK**.

    The Function Properties window closes. The information retrieved from the database inquiry will be placed in the CUSTINFO record you will define later. The error routine EZERTN will return control to your application when an error occurs. This prevents the application from ending with system generated messages, which might confuse an application user.

11. In the Function Editor, type the statements as shown in Figure 93.

    The first statement executes before the actual database query. The other statements execute after the query takes place.

```
custinfo.cusnum = customer.cusnum; /*customer number to be retrieved
---------------------- INQUIRY CUSTINFO --------------------
if custinfo not err;          /*If the query did not return an error
customer = custinfo;          /*Place all data from the retrieved
end;                          /*row into the customer record
```

Figure 93. Sample Statements

1. From the **Tools** menu, select **Validate and Format** .

   The statements have been validated and formatted. If a Validation Errors window is displayed, correct the errors listed in the window, and select **Validate and Format** again.

2. From the **File** menu, select **Save**.
3. Close the FIND-CUSTOMER function.

   A + is displayed beside the FIND-CUSTOMER function in the Program Editor.

Defining the CUSTINFO record

In the previous section, you defined an INQUIRY function that uses the CUSTINFO record. Now, you need to define the record. This record will contain the detailed customer information after it is retrieved from the database.

1. In the Program Editor, click the + beside the FIND-CUSTOMER function.

   The CUSTINFO record is displayed.

2. Double-click on the CUSTINFO record. In the New Part Package window, select **OK**.

   The Record Editor is displayed.

3. From the record type drop-down list box, select **SQL Row**.
4. Ensure that there are no items in the record. If there is an item in the record, delete it. To delete an item, select it. Then, from the **Edit** menu, select **Delete**.
5. From the **Define** menu, select **Properties**.

   The SQL Row Properties window is displayed.
6. On the SQL Row Properties window, select **Insert**.

   A row is added and the cursor appears in the **Name** pane.
7. In the **Name** pane, type `customer`.
8. Select **OK**.

   The SQL Row Properties window closes.
9. From the **Tools** menu, select **Retrieve SQL**.

   The SQL table is displayed in the Record Editor.
10. Click in the **Key** column of the CUSNUM data item and select the check box.

    The check box is checked. The CUSNUM item is defined as the key data item.
11. From the **File** menu, select **Save**.
12. Close the Record Editor.

    The fully-defined CUSTINFO record is displayed in the Program Editor.
13. In the Program Editor, from the **File** menu, select **Save**.
14. Close the Program Editor.

Building parameters for the program

In this section you build the parameters that will be passed between the client view and the FINDCUS program. You also visually connect the parameters to complete the definition.

1. In TutorialView, select the FINDCUS program part with mouse button 2. From the context menu, select **Build parameters from definition**.

   Based on the parameter list of FINDCUS, this action creates the parameters needed for the CALL.

   Selecting **Build parameters from definition** causes defined parameters to be added to the *execute* method of the FINDCUS program part. The solid green line changes to a dashed green line and a statement displayed in the status line confirms that default parameters were built.
2. Select the CUSTOMER part with mouse button 2. From the context menu, select **Connect→this**.

   You now have a line with a ✳ on the end.
3. Place the pointer on the *JButton1,actionPerformed → FINDCUS,execute* connection and click mouse button 1.

A context menu is displayed.

4. Select **CUSTOMER**.

   A purple line is displayed connecting the CUSTOMER record to the FINDCUS program. The dashed green becomes solid

A property-to-property connection connects the CUSTOMER parameter of FINDCUS to the CUSTOMER record. You selected the *this* property to pass the data contents and the type description of the record to the server application.

Your completed server application should look like Figure 103 on page 182

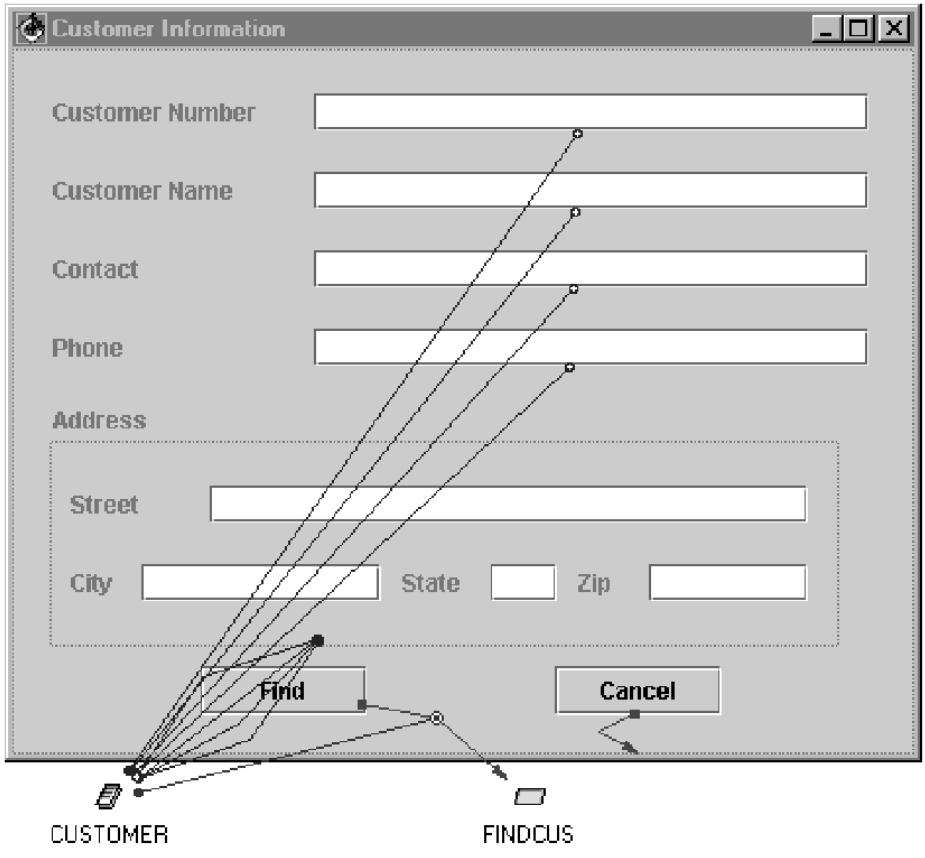


Figure 94. TutorialView

For more information on building visual parts, refer to the VisualAge for Java online help.

Testing the bean

Now test your bean using the Interactive Test Facility.

1. From the tool bar, select the **Run**  button.

   The Customer Information window is displayed.

2. Place the cursor in the Customer Number field. Type a customer number from the list below; then select Find. Use `111-0101` through `111-0106` and `122-0001` through `122-0003`. If you type an invalid number, nothing changes.

   If you want **Test Monitor** to have focus while it tests your VAGen parts, VAGen Parts Browser, select **Window→Options**. On the VAGen Test window, under options, select **Break on event entry for GUI clients**.

   Start the Test Monitor. The Test Monitor gets focus if you have an error in a VAGen part. If you would like to see the Test Monitor window throughout your test run, move it to the right side of your screen.

   If errors are encountered during the test, you can change your code without leaving the test. After you make changes and save the part, the test facility automatically repositions the test for you to test your new changes.

   Otherwise, the program runs and the Customer Information window is updated with data from the database. Your application ran successfully if it looks like the one shown in Figure 95 on page 158.

3. Continue entering other customer numbers. After you finish testing, in the Customer Information window, select Cancel.

   The Customer Information window closes.

4. Close the Test Monitor window.
5. Close the Composition Editor.

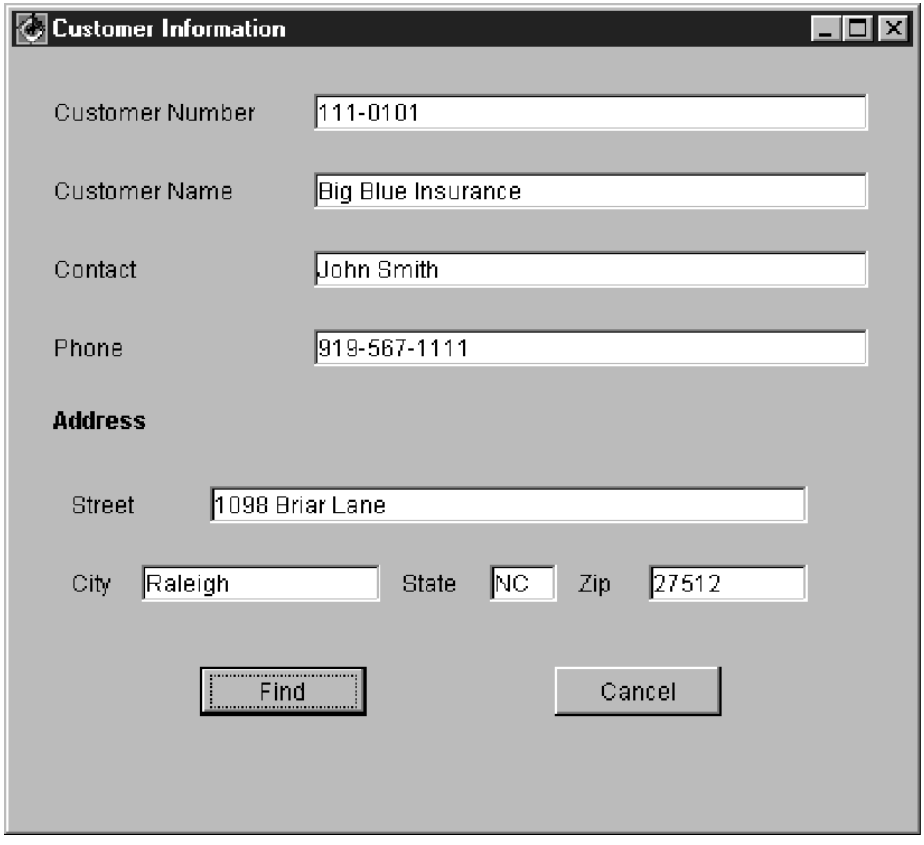


Figure 95. Customer Information Window

Summary

Congratulations! You have completed all the steps in this tutorial, and you have experienced firsthand how easy it is to define and test applications using VisualAge Generator. As you begin working with VisualAge Generator, you can refer back to this tutorial, all the other VisualAge Generator documentation, and VisualAge Generator's extensive online help system.

Chapter 18. Building a visual part using VisualAge Generator Developer on Smalltalk

This section is the first of three sections that provide you with experience building visual parts. The objective of this section is to help you become familiar with the Composition Editor and the tools it provides you for building visual parts. Using the parts palette, you will design a user interface window and size and align labels and fields within the window. You'll also use the parts palette to connect VisualAge Generator data and logic parts to your visual part.

Before you start

If you have not completed the steps in "Chapter 6. VisualAge Generator Developer on Smalltalk: a tutorial" on page 53, please go back and complete all the steps in that section before you create a new visual part.

Creating a new visual part

The window you create in this tutorial displays detailed customer information.

Note: The Address part shown in this picture is created in "Defining a reusable visual part" on page 172. Leave space for it, so it will fit as shown in Figure 96 on page 160.

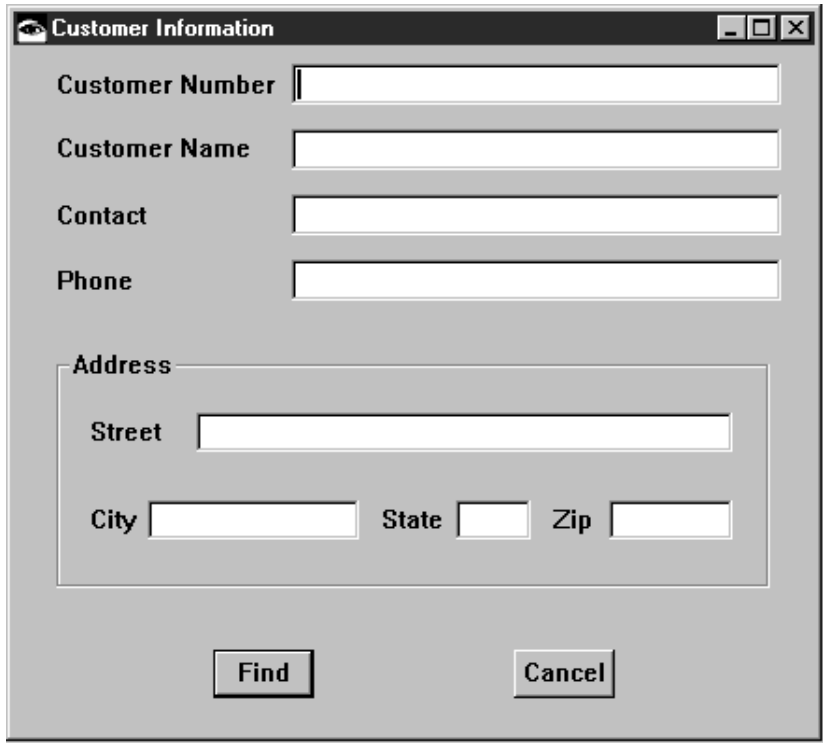


Figure 96. Customer Information Window

To create a new visual part, on the VisualAge Organizer window:

1. Ensure that the Tutorial application is selected in Applications pane.
2. From the **Parts** menu, select **New**.

   The VisualAge New Part window is displayed.
3. In the **Part class** field, type the name `TutorialView` for your new part class.

   Note: Names of VisualAge parts are usually several words written together without spaces. Visual part class names are case sensitive and the first letter of each word is usually capitalized to make them easier to read. The names of visual parts usually end with the word *View* to help you distinguish them from nonvisual VisualAge parts. The class shown in the **Inherits from** drop-down list box is one example.
4. Select **OK**.

The Composition Editor is displayed. Because the *AbtAppBldrView* class was selected in the **Inherits from** drop-down list box, the `TutorialView` part already contains a Window part. The area outside of the Window part is called the free-form surface.

5. Maximize the Composition Editor to get a larger work area.
6. Select the window on the free-form surface.

   Small solid boxes appear on the Window's corners. These are called *selection handles*.
7. Click and drag one of the selection handles to resize the Window.

   For this sample, make the Window as large as possible in the viewable area of the Composition Editor. Leave at least one inch of the free-form surface visible to the right of the Window part for future use.
8. Give the Window a new title to replace the generic name Window. To do this, move the mouse pointer to the Window title and press **ALT+mouse button 1**.

   The cursor is now at the end of the default name, which is blocked for deletion in text edit mode.
9. Type the title, `Customer Information`. Click anywhere except on the title to end text edit mode.

   The Window now has the new title.

Note: Refer to the *VisualAge for Smalltalk User's Guide* for basic techniques used in creating visual parts.

Adding a VAGen Record to the visual part

Now you are going to build a data part for your visual part. The fields (columns) in a single row of a relational database table are described in a VAGen record.

1. Select the **VAGen Parts** category  .

   The parts palette is refreshed to show all the parts of the VAGen Parts category.

2. Select the **VAGen Record** part  . Drop the part by clicking on the free-form surface to the right of the Customer Information window. The mouse pointer becomes a cross-hair when your cursor is over an area where you can drop the part.

   The Add Part window, shown in Figure 97 on page 162, is displayed. You can enter a part name (in this case, a record name) or use an existing VisualAge Generator record by selecting from the drop-down list. For this exercise, you will enter a record name.

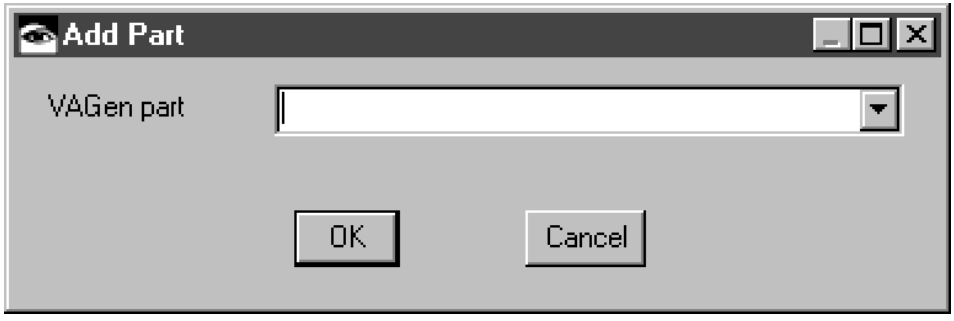


Figure 97. Add Part Window

1. Type `customer` as the record name, and select **OK**.

   The CUSTOMER record is now on your free-form surface and ready to be defined. This name can be the same or different from the table name in the relational database.
2. Click with mouse button 2 on the CUSTOMER record.

   A context menu is displayed.
3. Select **Edit Part**.

   The New Part Application window is displayed.
4. Ensure that **Tutorial** is selected, and select **OK**.

   The CUSTOMER record is displayed in the Record Editor. This is the same record defined in "Chapter 11. Defining an SQL record" on page 89. If you have already completed that section, this record is already defined, except for the data item descriptions.

Defining the CUSTOMER VAGen Record Part

Here you define the VAGen Record part as an SQL row record and use the CUSTOMER table definition in the relational database catalog to define the contents of the record.

Note: If you have already completed the steps in "Chapter 11. Defining an SQL record" on page 89, the next five steps will be redundant. (Do not repeat the steps if you completed them earlier.) Step 6 in this section is not redundant, so be sure to follow it and customize the data item descriptions.

1. In the Record Editor, select **SQL row** from the Record Type drop-down list box.

   VisualAge Generator supports accessing a variety of file types and databases. In this case, you are going to access a relational database table, so you selected SQL row as the record type.

2. From the **Define** menu, select **Properties**.

   The SQL Row Properties window is displayed.
3. Select **Insert**.

   A blinking cursor is displayed in the **Name** field.
4. In the **Name** field, type `customer` and select **OK**.

   The SQL Row Properties window closes.
5. From the **Tools** menu, select **Retrieve SQL**.

   Note: If a message window is displayed, indicating that the system cannot locate a database access plan, select **Yes** to have a plan created.

   If you have not already connected to the database, you will be prompted for an ID and Password. Enter the ID and Password you used in setting up your database.

   The table definition information is retrieved from the relational database catalog and used to create data item definitions for the record.
6. Customize the descriptions of data items in the SQL row record so that you can use them as field labels later. For each data item in the record, select the **Description** field and type descriptions for the data item names as follows:

Name	Description
CUSNUM	Customer Number
CUSNAME	Customer Name
CUSSTATUS	Customer Status
CUSLIMIT	Customer Limit
CUSCONTACT	Contact
CUSPHONE	Phone
CUSSTREET	Street
CUSCITY	City
CUSSTATE	State
CUSZIP	Zip

   Note: You may need to maximize the Record Editor in order to view the **Description** fields.
7. Select **File→Save**.
8. Close the Record Editor.

For more information about defining records, refer to the VisualAge Generator help facility.

Now that you've defined an SQL Row record and assigned data item descriptions, you're ready to add fields to the Customer Information window and define connections to the record. Data items in the CUSTOMER record will be visually connected to parts of the Window with a blue line, indicating an attribute-to-attribute connection. With this type of connection:

- When data is entered in the fields, it is automatically moved to the corresponding data items in CUSTOMER.
- When data items in CUSTOMER are changed, the new values are also automatically moved to the fields (when the window is displayed).

Customer Information window

You can add Text parts (data fields) to your Window, either manually or with the **Quick Form** function. The next section takes you through both techniques. In addition to the Text and Label parts, you will also add two Push Buttons to your Window.

1. Select the **Data Entry** category  , and drop a **Text** part  and a **Label** part  inside the top portion of the Window.

   See Figure 96 on page 160 for placement of these parts. These parts are for the Customer Number label and its corresponding text area.
2. Place the mouse pointer on the word Label1 in your Customer Information window, and press **ALT+mouse button 1**. Type the name `Customer Number`. Click anywhere except on the label to end text edit mode.

   The Label is automatically resized and now says Customer Number.
3. Click mouse button 2 on the customer number Text part.

   A context menu is displayed.
4. Select **Open Settings**.

   Note: If a Settings notebook is displayed, changing it to a Properties window will make the tutorial instructions easier to follow. To change the window, from the VisualAge Organizer window, select **Options**, then **Preferences**. On the General tab, under **Preferred Settings View**, select **Properties Table**. Select **OK**. In order to bring up the **Properties Table**, you will need to close the Settings notebook and repeat steps 3 and 4.

The Properties window is displayed. Property names are listed in alphabetical order.

5. Locate **converter** in the **Name** column and click beside it in the **Value** column.

   A push button is displayed.

6. Select the push button.

   The Converter window is displayed.

7. From the **Data Type** drop-down list box, select **String**.

8. Select **OK**.

   The Converter window closes.

9. Ensure that the **notifyChangeOnEachKeystroke** property is set to **false**. If it is set to **true**, click in the **Value** field with mouse button 1 to display the radio buttons. Select **false**.

   By setting **notifyChangeOnEachKeystroke**to false, editing and other actions will not occur until the cursor moves out of the Text part field. Otherwise, every keystroke during data entry to the field will trigger a data edit check of the field by VisualAge Generator.

10. Close the Properties window to accept the values you have changed.

    The changes have now taken effect.

11. Select the CUSTOMER record with mouse button 2.

    A context menu is displayed.

12. Select **Connect** from the context menu.

    The Start connection from (CUSTOMER) window is displayed.

13. Select CUSNUM data from the **Attribute** column; then select **OK**.

    The mouse pointer is displayed as ✳ connected to a dashed line. This indicates that you are in the process of making a connection.

14. Select the Customer Number Text part.

    A context menu is displayed.

15. Select **object**.

    A blue line now is displayed connecting the Customer Number Text part and the *CUSNUM data* attribute in the CUSTOMER record. By selecting the blue line, the description of the connection is displayed in the status area at the bottom of the Composition Editor. This attribute-to-attribute connection causes the data item CUSNUM in the CUSTOMER record to be updated when the Customer Number Text part is changed, and causes the Customer Number Text part (when displayed) to be updated when the data item CUSNUM is changed.

A faster way to add the parts and connections to the window is to use a function called **Quick Form** to complete the following steps:

1. Build the Label part.
2. Change the label to the data item description from the record.
3. Build the Text part.
4. Set the correct data type using the data item definition from the record.
5. Make an attribute-to-attribute connection between the Text part and the appropriate data item in the CUSTOMER record.

To add the parts:

1. Select the CUSTOMER record part with mouse button 2

   A context menu for the VAGen Record part is displayed.
2. Select **Quick Form**.

   A list of the data item names is displayed. These data items are defined in the VAGen record part.
3. Select CUSNAME.

   Your mouse pointer is now in the form of a cross-hair.
4. Click on the Window where you want this field positioned.

   A Text part and a Label (Customer Name) are added to the Window. An attribute-to-attribute connection is created, and the data type is set according to the CUSNAME data item definition.
5. Repeat the previous four steps for the CUSCONTACT and CUSPHONE data items.

   The Window now has four Text parts with Labels, corresponding to those shown in Figure 96 on page 160.

Complete the following steps to add Push Buttons to the bottom of your Window:

1. Select the **Buttons** category  .

   The parts palette shows a list of all parts in the Buttons category.

2. Select the **Push Button** part  . Select the **Sticky** checkbox and click once at the bottom of the Window.

   A Push Button is displayed at the bottom of the Window. The Push Button part remains selected and your mouse pointer remains loaded.
3. Click to the right of your first Push Button.

   A second Push Button is displayed on the Window. The Push Button part remains selected and your mouse pointer remains loaded.
4. Deselect the **Sticky** checkbox.

   The Push Button part is no longer selected and your mouse pointer changes from a cross hair to an arrow.

Note: You can unload the mouse pointer at any time by selecting the selection tool  from the tool bar.

5. To change the Push Button labels, select the button label by placing your mouse pointer over one of the Push Buttons and pressing **ALT+mouse button 1**.

   The cursor is now at the end of the default name of the Push Button.

6. Type `Find` on the first Push Button and click anywhere in the editor except on the part you just changed.

   The Push Button is now labeled Find and has automatically resized to fit the text.

7. Select the button label on the next Push Button and change it to `Cancel`. Click anywhere to exit the text edit mode.

   The Push Buttons have now been changed to Find and Cancel. The size of each Push Button automatically adjusts to the size of its label.

Now that you have most of the visual parts of your user interface on the Window, you can use the tool bar to align those parts in the window.

Arranging visual parts

To arrange visual parts you can use the tool bar buttons or selections from the Tools menu. Each tool bar button has a corresponding menu choice with the same name that performs the same function. The menu choices or buttons you use to arrange visual parts are as follows:

Align Left 

Align Center 

Align Right 

Align Top 

Align Middle 

Align Bottom 

Distribute Horizontally

Distribute Vertically

Match Width

Match Height

In this tutorial, you'll be directed to the menu choices, but feel free to use the tool bar buttons instead.

Note: Like the choices on the **Tools** menu, many of the buttons on the tool bar are unavailable unless you have more than one visual part selected. To select two or more parts at the same time,

1. Place your mouse pointer over the part you want to select, press and hold **Ctrl** and click and release mouse button 1.
2. While holding down the **Ctrl** key, move the mouse pointer to another part and click and release mouse button 1.

Repeat these steps as many times as necessary, until you have selected all the parts you need to size or align in relation to each other.

Notice that the part you select last has solid handles, while the other selected parts have hollow handles. The part with solid handles is used as a reference point. So, when you want to align several parts, select the parts you want to move first. Those parts will be aligned with the part you select last.

In "Defining a reusable visual part" on page 172, you will add an Address group box to the Customer Information window. So arrange the Label and Text parts at the top of the Window and the push buttons at the bottom, leaving enough room for the Address Group Box between them.

For a complete description of the tool bar and the tools available, refer to *Composition Editor* in the VisualAge help facility or the *VisualAge for Smalltalk User's Guide*.

To arrange the visual parts on the Customer Information window, perform the following steps:

1. Select both the Find Push Button and Cancel Push Button

   Both buttons are selected. The Find Push Button has hollow handles and the Cancel Push Button has solid handles.
2. From the **Tools** menu, select **Match Width**.

   The Find Push Button is resized to match the Cancel Push Button in width.
3. From the **Tools** menu, select **Match Height**.

   The Find Push Button is resized to match the Cancel Push Button in height.
4. From the **Tools** menu, select **Align Middle**.

   The Find Push Button's vertical center is aligned to the center of the Cancel Push Button.
5. From the **Tools** menu, select **Distribute Horizontally**.

   The two Push Buttons are placed at regular intervals across the Window.

Using the bounding box

Next, align the Label and Text parts in the top portion of the Window, leaving space in the middle for the Address Group Box.

The *bounding box* enables you to align selected parts within a portion of your Window, rather than the entire Window. The selected portion is defined by the topmost and bottommost, or leftmost and rightmost sides of the parts selected.

1. Select the Phone Text part and the associated Label.

   Both parts display selection handles.
2. Move these parts to where you want them in the Window using mouse button 1.

   The Phone Text part and the associated Label define the bottom of the bounding box, while Customer Number defines the top of the bounding box.
3. Select all the Text parts (not the Label parts), making sure the Customer Number Text part is the last one selected.

   All the Text parts are selected. The Customer Number Text part should have primary selection handles (solid).
4. Click mouse button 2 on one of the selected parts. Select **Layout→Distribute→Vertically In Bounding Box**.

   Now the Text parts are all evenly distributed in the top portion of Window.
5. From the **Tools** menu, select **Align Left**  to align the parts on the left.

   All Text parts are aligned on the left.

6. Select all the Label parts, making sure that the Customer Number Label part is the last one selected.

   All the Label parts are selected.
7. Click mouse button 2 on one of the selected parts. Select **Layout→Distribute→Vertically In Bounding Box**.

   Now the Label parts are all evenly distributed in the top portion of Window. If you are not satisfied with your results, from the **Edit** menu, select **Undo** and try again.
8. From the **Tools** menu, select **Align Left**.

   All Label parts are aligned on the left.

In the next chapter, you will add the Address part to this Window.

Adding an action to the cancel push button

When users select the Cancel button, you want the Customer Information window to close. This relationship is called an event-to-action connection.

1. To make a connection from the Cancel Push Button, click mouse button 2 on the Cancel Push Button.

   The context menu is displayed.
2. From the context menu, select **Connect→clicked**.

   The mouse pointer is displayed as  connected to a dashed line. This indicates that you are in the process of making a connection.
3. Click mouse button 2 on the title bar of your Window (the area that says Customer Information).

   A context menu is displayed.
4. Select **closeWidget**.

   A green line with an arrow pointing from the Cancel Push Button to the Window title is displayed. This connection means that when the Cancel push button is *clicked* (an event), *closeWidget* (an action that means close the window) is triggered. This is an event-to-action connection, as indicated by the green rather than the blue connection.
5. Select **File→Save** to save the visual part.

Testing the view

The next step is to test the view to ensure that the user interface is displayed and the Cancel Push Button works the way you want it to.

1. From the tool bar, select the **Test** icon  .

The Customer Information window is displayed. Verify your layout of the window.

2. Select Cancel on your Customer Information window.

   The window is closed.

3. If you have time to do the optional exercise below, wait to close the Composition Editor until after you complete the exercise. If you don't have time to do the optional exercises, close the Composition Editor.

For more information on testing visual parts, refer to the *VisualAge for Smalltalk User's Guide*.

After you have added some VAGen logic parts, you can use the VisualAge Generator test facility to test the connections from this user interface to your VAGen logic and data parts. This powerful feature enables you to step through a test of your user interface connections and logic and data parts, modify code, and set breakpoints, watchpoints, and testpoints.

Optional exercises

This portion of the sample is your opportunity to navigate within VisualAge Generator Developer without step-by-step instructions while adding functionality to your view.

Setting the focus on a field

If you want to have the cursor in a particular field when the window is first displayed, complete the following step. In this example, you are going to set the focus (or put the cursor) in the Customer Number Text part.

Connect the *openedWidget* event of the Customer Information Window to the *setFocus* action of the Customer Number Text part.

Making a push button the default

A default push button allows the user to select the push button by pressing **Enter** or by using the mouse. You can define a default push button by completing the following step:

Open the settings for the FIND Push Button, click in the **showAsDefault** field, and select **true**.

Testing the application

The cursor should be in the Customer Number field and the Find push button should have an outline indicating that it is the default push button.

Be sure to save the part when you are finished.

If you did the optional exercises, your view should look like the one shown in Figure 98.

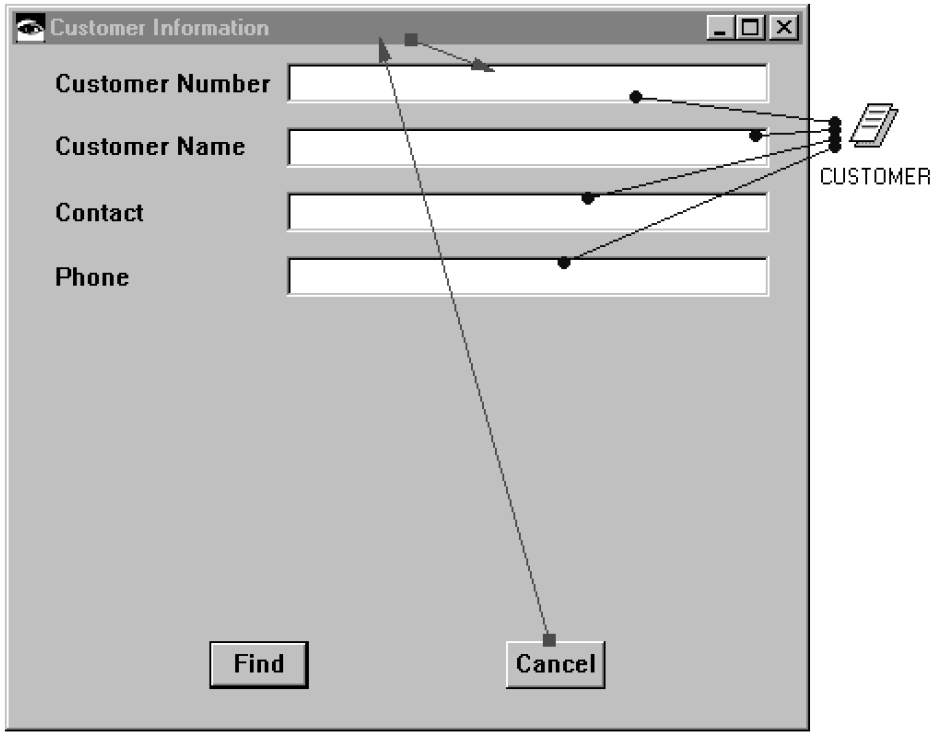


Figure 98. Customer Information Window

Defining a reusable visual part

The objective of this section is to familiarize you with creating reusable visual parts and using them with VAGen parts. This exercise focuses on the following tasks:

- Building a reusable visual part called ADDRESS, consisting of a group box that includes labels and fields for address data.
- Promoting the view to the public interface so that the visual part and its functionality can be embedded in other visual parts.

Creating an embeddable visual part

You will build a group box and use the **Quick Form** function to add labels and fields to it. Figure 99 on page 173 shows the group box you will define.

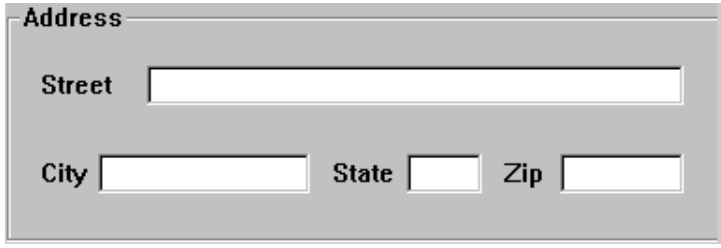


Figure 99. Address Box

First, you will create a new visual part (just as you did in "Creating a new visual part" on page 159). In the next section you will add this reusable visual part to your Customer Information window.

To build a Group Box using **Quick Form**, perform the following steps:

1. From the VisualAge Organizer window, select **Part→New**.
   The VisualAge New Part window is displayed.
2. In the **Part class** field, type the name `ReusableAddressBoxView` for your new part class.
3. Select **OK**.
   The Composition Editor is displayed.
4. Maximize the Composition Editor to get a larger work area.
   Now you can see the entire Window part.
5. Select the default Window part with mouse button 2.
   The context menu is displayed.
6. Select **Delete**.
   The Window is deleted.

Adding the Group Box

A Group Box places a rectangular box around a set of parts. It also allows you to move and use them as a single unit.

Complete the following steps to create the Address Group Box.

1. Select the **Canvas** category  .
   The parts palette is refreshed to show all parts in the Canvas category.
2. Select the **Group Box** part  and drop it on the free-form surface.
   A default size Group Box is displayed on your free-form surface.
3. Select the Group Box (if it is not already selected).

The Group Box corners have selection handles.

4. Size the Group Box to fit into the Customer Information window created in "Chapter 18. Building a visual part using VisualAge Generator Developer on Smalltalk" on page 159.

   The Group Box is resized.

5. Update the label of the Group Box. To do this, place the mouse pointer on the text Group Box and press **Alt+mouse button 1**.

   The Group Box label becomes blocked for deletion in text edit mode.

6. Type `Address` and then click anywhere except on the label.

   Address is now the label of the Group Box.

Populating the Address Group Box

To add parts to the Group Box, you will use **Quick Form** just like you did in "Chapter 18. Building a visual part using VisualAge Generator Developer on Smalltalk" on page 159.

Complete the following steps to add parts to the Address Group Box.

1. From the parts palette, select the **VAGen parts** category  . Select a **VAGen record** part  and drop it on the free-form surface.

   The Add Subpart window is displayed.

2. From the drop-down list, select **CUSTOMER**; then select **OK**.

   The CUSTOMER record part is displayed on the free-form surface.

3. Select the CUSTOMER record part with mouse button 2.

   The context menu is displayed.

4. Select **Quick Form**.

   A list of data items is displayed.

5. Select CUSSTREET and move the mouse pointer (cross-hair) to the appropriate area in the **Address** Group Box.

   The Street Label part and a Text part are added to the Group Box. An attribute-to-attribute connection is created, and the data type is set using the CUSSTREET data item definition.

6. Repeat the above step for CUSCITY, CUSSTATE, and CUSZIP.

7. Click mouse button 2 on the CUSTOMER record part. From the context menu, select **Delete**. When the warning message is displayed, select **OK**.

   The record and all the connections are now deleted. You do not need the attribute-to-attribute connections in this scenario. Later, you will promote these fields to the public interface so that you can define connections to them in any view where you embed this address box.

Laying out the Group Box

Lay out the Label and Text parts in the Group Box as shown in Figure 99 on page 173. Remember, to size a part, select it with mouse button 1, place the mouse pointer on one of the selection handles, and move the mouse while holding down mouse button 1. For more detailed information about laying out visual parts, see "Arranging visual parts" on page 167.

Promoting features of the reusable part

If you want to access any information in the ADDRESS Group Box from another view, you must add the appropriate features to the public interface. In this case, the features needed are the attributes for the Text parts. In this section, you will add attributes for all four Text parts (Street, City, State, and Zip) to the public interface.

For more complete information on topics covered in this section, refer to the *VisualAge for Smalltalk User's Guide*.

To promote part features to the public interface, perform the following steps:

1. Select the Street Text part with mouse button 2. From the context menu, select **Promote Part Feature**.

   The Promote features from window is displayed.
2. On the Promote features from window, from the **Attribute** column, select **object**.

   `textCUSSTREETObject` is displayed in the **Promote feature name** field.
3. Change the **Promote feature name** to `streetobject` and select **Promote**.

   Now other parts can reference this Text part by the attribute name. By selecting **object**, you are adding the actual value of the data that the part represents, including all customized parameters, such as currency symbols or numeric separators. Selecting **string** would only have added a reference to the input string typed into the field.
4. Select **OK** to accept your changes and close the Promote features from window.
5. Repeat the above steps for the City, State, and Zip Text parts, using the names `cityobject`, `stateobject`, and `zipobject`, respectively.

   The *object* attribute feature of each of the four Text parts has been added to the public interface. Now these attributes can be accessed by other parts.
6. Select **File→Save Part**.

You can also do this same task from the **Promote** tab in Public Interface Editor. Refer to the *VisualAge for Smalltalk User's Guide* more information on using the Public Interface Editor.

Now you are ready to embed the address box in your tutorial view and add VAGen logic parts.

Defining a VAGen server program

The focus of this section is to finish developing your user interface and connect it to a server application that reads data to be displayed in the Customer Information window. The objective of this section is to familiarize you with making connections between user interfaces and nonvisual parts, such as records and server applications. In addition, you will build the server application and embed the Address box you created in the previous section. Finally, you will use the Interactive Test Facility to test how the client and server applications work together.

Embedding a reusable visual part

You can now complete the window in Figure 100, which displays detailed customer information.

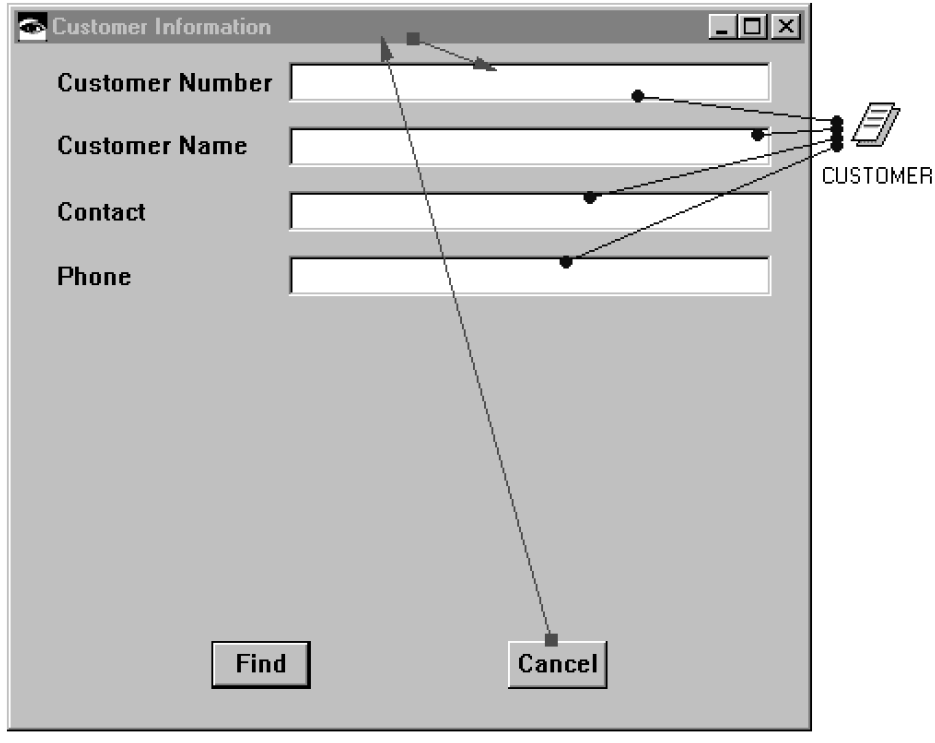


Figure 100. Customer Information Window

To complete the window, add the Address box you created in "Defining a reusable visual part" on page 172 to the Customer Information Window. To add the reusable visual part, perform the following steps:

1. In the VisualAge Organizer window, Double-click on **TutorialView**.

   The Composition Editor opens, displaying the Customer Information window.
2. From the **Options** menu, select **Add Part**.

   The Add Part window is displayed

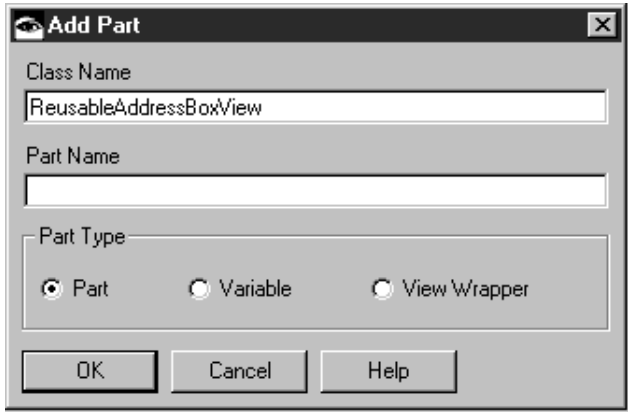


Figure 101. Add Part Window

1. In the **Class Name** field, type `ReusableAddressBoxView` and select **OK**.

   The mouse pointer becomes a cross hair.

   Note: Selecting Browse will open the Choose a Valid Class window, which lets you to search for available classes.
2. Click on the Customer Information window where you want to add the part.

   The Address box is added to the Window.
3. Size the Address box (and the Customer Information window if necessary) until the entire Group Box fits inside the window.
4. Select the ADDRESS Group Box with mouse button 2. From the context menu, select **Connect→streetobject**.

   A dotted line with a ✳ at the end is displayed.
5. Select the CUSTOMER Record part.

   The Connect attribute named: window is displayed.
6. From the **Attribute** column, select CUSSTREET data and select **OK**.

This creates an attribute-to-attribute connection between the Street Text part and the data item CUSSTREET in the CUSTOMER Record part.

By selecting the *data* attribute, you are making a connection to only the data contents of the data item. If the connection required a type description in addition to the data contents, you would have selected CUSSTREET, which represents the *self* attribute of the data item.

Populating Text parts on a Window, as in this example, only requires the data contents. However, when you connect to a parameter in a called parameter list for a program, the data contents and the type description are required.

7. Repeat the previous steps for cityobject, stateobject, and zipobject. Connect them to CUSCITY data, CUSSTATE data, and CUSZIP data, respectively.

   This creates attribute-to-attribute connections between the fields in the Group Box and the data items in the CUSTOMER Record part. If you want to check any of the connections, select the connection and the status area will contain a description of it.

Adding a VAGen program to the visual part

Now that all the user interface parts are in place on the Window, add a VAGen Program part to the free-form surface and connect it to one of the Push Buttons. The VAGen Program part represents a server application that you will define later in this exercise. The server application will read the customer details from the database and return the data to TutorialView to populate the fields in the Customer Information window.

1. Select the **VAGen Parts** category  .

   The list of parts in the VAGen Parts category is shown in the parts palette.

2. Select the **VAGen Program** part  and click on the free-form surface to the right of your Window.

   The Add Part window is displayed.

3. In the **VAGen part** field, type `findcus` and select **OK**.

   The FINDCUS program part is now on your free-form surface.

4. Select the Find Push Button with mouse button 2.

   The context menu for the Find Push Button is displayed.

5. Select **Connect→clicked**.

   A line with a ✳ on the end is displayed.

6. Select the FINDCUS program part.

A context menu is displayed.

7. From the context menu, select **execute**.

   A green arrow now connects the Find Push Button with the FINDCUS Program part. This connection means that when the Find Push Button is clicked, FINDCUS is called.

8. Select the FINDCUS Program part with mouse button 2.

   A context menu is displayed

9. Select **Edit Part**.

   The New Part Application window is displayed

10. Ensure that **Tutorial** is selected, and select **OK**.

    The Program Editor opens to display FINDCUS.

Defining the VAGen server application

In this section, you define the VAGen server application FINDCUS, which reads the customer information from a relational database. To retrieve the correct customer information, the record key (CUSNUM) for the database must be passed to the server application from TutorialView. To display the retrieved information, the data to populate the Customer Information window must be returned from the server application. Parameters are the mechanisms by which data is passed between applications.

To define the server application, in the Program Editor, perform the following steps:

1. From the **Define** menu, select **Called Parameters→Insert Parameter**.

   The Insert Parameter window is displayed.

2. Ensure that the **Record** radio button is selected, and from the **Part Name** drop-down list box, select **CUSTOMER**.

3. Select **OK**.

   The CUSTOMER record is displayed in the program diagram under **Called Parameters**. Now you are ready to define the main functions for FINDCUS.

4. From the **Define** menu, select **Main Functions**.

   The Insert Main Function window is displayed.

5. In the Part name field, type `find-customer` and select **OK**.

   The FIND-CUSTOMER function is displayed in the structure diagram. The ? beside it means that it has not yet been defined.

6. Double-click on FIND-CUSTOMER function icon.

   The New Part Application window is displayed.

7. Ensure that **Tutorial** is selected, and select **OK**.

   The Function Editor is displayed. It contains two drop-down list boxes. The list box on the left displays I/O options, which define the type of

operation a function will carry out. The list box on the right displays I/O objects, the data parts on which I/O options are performed.

8. From the I/O Option drop-down list box, select **INQUIRY**. In the I/O Object field, type `custinfo`.

   Your entries are displayed on the second line in the Function Editor.

9. From the **Define** menu, select **Properties**.

   The Function Properties window is displayed.

10. From the **Error routine** drop-down list box, select **EZERTN** and then select **OK**.

    The Function Properties window closes. The information retrieved from the database inquiry will be placed in the CUSTINFO record you will define later. The error routine EZERTN will return control to your program when an error occurs. This prevents the program from ending with system generated messages, which might confuse an application user.

11. In the Function Editor, type the statements as shown in Figure 102.

    The first statement executes before the actual database query. The other statements execute after the query takes place.

```
custinfo.cusnum = customer.cusnum; /*customer number to be retrieved
---------------------- INQUIRY CUSTINFO --------------------
if custinfo not err;          /*If the query did not return an error
customer = custinfo;          /*Place all data from the retrieved
end;                          /*row into the customer record
```

Figure 102. Sample Statements

1. From the **Tools** menu, select **Validate and Format** .

   The statements have been validated and formatted. If a Validation Errors window is displayed, correct the errors listed in the window, and select **Validate and Format** again.

2. From the **File** menu, select **Save**.

   A+ is displayed beside the FIND-CUSTOMER function in the Program Editor.

Defining the CUSTINFO record

In the previous section, you defined an INQUIRY function that uses the CUSTINFO record. Now, you need to define the record. This record will contain the detailed customer information after it is retrieved from the database.

1. In the Program Editor, click the + beside the FIND-CUSTOMER function.

   The CUSTINFO record is displayed.

2. Double-click on the CUSTINFO record.

The New Part Application window is displayed.

3. Ensure that **Tutorial** is selected, and select **OK**.

   The Record Editor is displayed.

4. From the record type drop-down list box, select **SQL Row**.
5. Ensure that there are no items in the record. If there is an item in the record, delete it. To delete an item, select it. Then, from the **Edit** menu, select **Delete**.
6. From the **Define** menu, select **Properties**.

   The SQL Row Properties window is displayed.

7. On the SQL Row Properties window, select **Insert**.

   A row is added and the cursor appears in the **Name** pane.

8. In the **Name** pane, type `customer`.
9. Select **OK**.

   The SQL Row Properties window closes.

10. From the **Tools** menu, select **Retrieve SQL**.

    The SQL table is displayed in the Record Editor.

11. Click in the **Key** column of the CUSNUM data item and select the check box.

    The check box is checked. The CUSNUM item is defined as the key data item.

12. From the **File** menu, select **Save**.
13. Close the Record Editor.

    The fully-defined CUSTINFO record is displayed in the Program Editor.

14. In the Program Editor, from the **File** menu, select **Save**.
15. Close the Program Editor.

Building parameters for the program

In this section, you build the parameters that will be passed between the client view and the FINDCUS program. You also visually connect the parameters to complete the definition.

1. In TutorialView, select the FINDCUS Program part with mouse button 2. From the context menu, select **Build parameters from definition**.

   Based on the parameter list of FINDCUS, this action creates the parameters needed for the CALL.

   Selecting **Build parameters from definition** causes one parameter to be added to the connection context menu for the FINDCUS Program part. A statement displayed in the status line confirms that default parameters were built.

2. Select the CUSTOMER part with mouse button 2. From the context menu, select **Connect**.

The Start connection from window is displayed

3. From the **Attribute** column, select self; then select **OK**.

   You now have a line with a ✳ on the end.

4. Select the *Push Button1,clicked→ FINDCUS,execute* connection and click mouse button 1.

   Select CUSTOMER from the context menu.

5. From the **Attribute** column, select **self**; then select **OK**.

   A parameter-from-attribute connection passes the CUSTOMER record to the FINDCUS program. You selected the *self* attribute to pass the data contents and the type description of the record to the server program.

Your completed server application should look like Figure 103

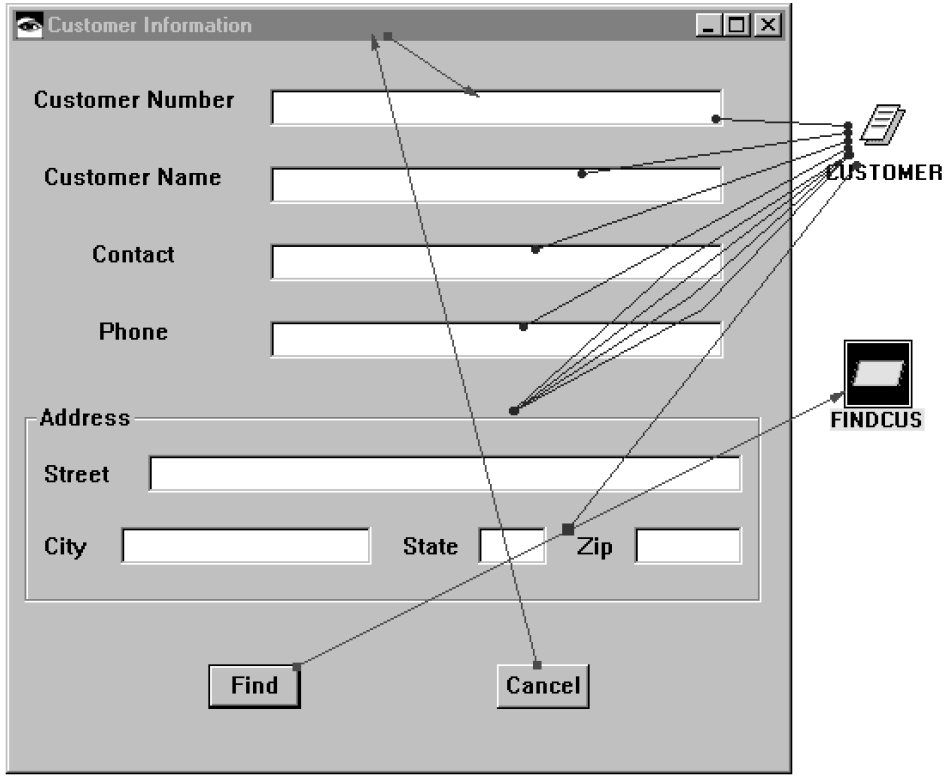


Figure 103. TutorialView

For more information on building visual parts, refer to the *VisualAge for Smalltalk User's Guide*.

Testing the view

Now test your view using the Interactive Test Facility.

1. From the tool bar, select **Test**  .

   The Customer Information window is displayed.

2. Place the cursor in the Customer Number field. Type a customer number from the list below; then select Find. Use `111-0101` through `111-0106` and `122-0001` through `122-0003`. If you type an invalid number, nothing changes.

   If you want **Test Monitor** to have focus while it tests your VAGen parts, on the VisualAge Organizer window, select **Options→VAGen Preferences**. On the VAGen Test window, under options, select **Break on event entry for GUI clients**.

   Start the Test Monitor. The Test Monitor gets focus if you have an error in a VAGen part. If you would like to see the Test Monitor window throughout your test run, move it to the right side of your screen.

   If errors are encountered during the test, you can change your code without leaving the test. After you make changes and save the part, the test facility automatically repositions the test for you to test your new changes.

   Otherwise, the program runs and the Customer Information window is updated with data from the database. Your application ran successfully if it looks like the one shown in Figure 104 on page 184.

3. Continue entering other customer numbers. After you finish testing, in the Customer Information window, select **Cancel**.

   The Customer Information window closes.

4. Close the Test Monitor window.

5. Close the Composition Editor.

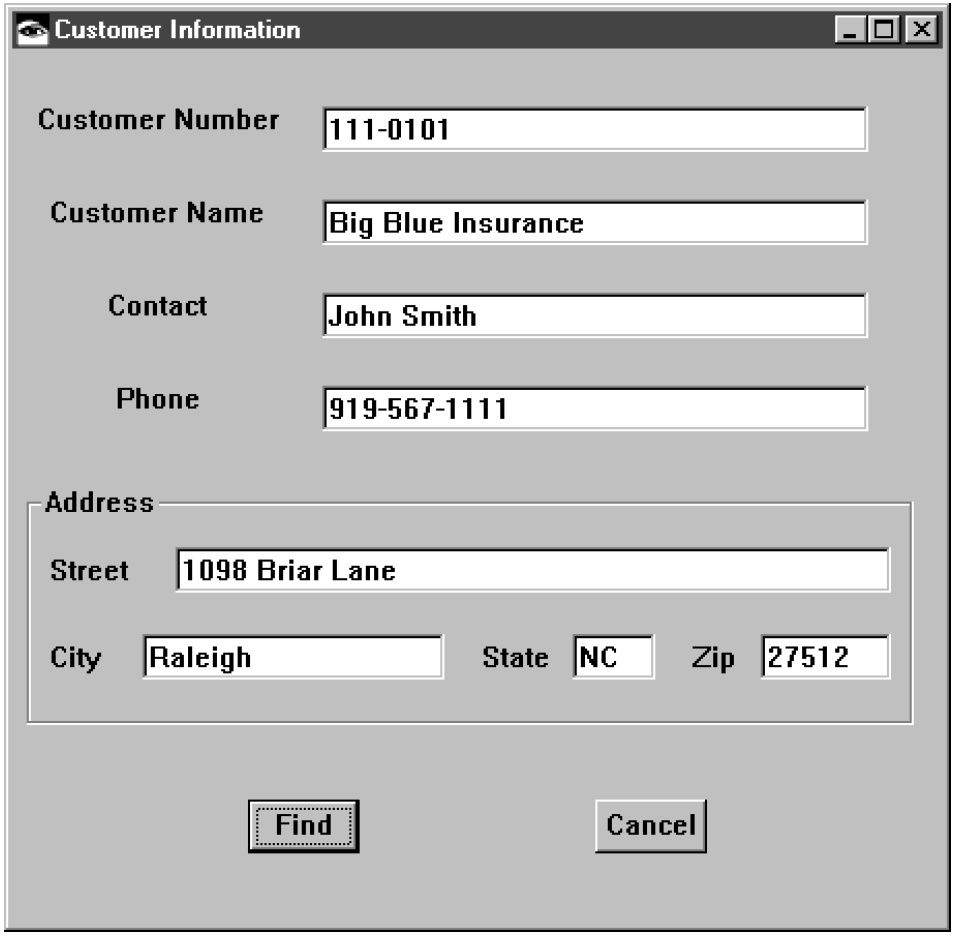


Figure 104. Customer Information Window

Summary

Congratulations! You have completed all the steps in this tutorial, and you have experienced firsthand how easy it is to define and test applications using VisualAge Generator. As you begin working with VisualAge Generator, you can refer back to this tutorial, all the other VisualAge Generator documentation, and VisualAge Generator's extensive online help system.

Part 3. Appendixes

Appendix A. Installing samples for VisualAge Generator Developer on Smalltalk

The following sample applications are shipped as .DAT files with VisualAge Generator Developer on Smalltalk. If you do a typical install, they are installed in the samples subdirectory (for Windows, C:\Program Files\VAST\samples and for OS/2, C:\VAST\samples). To use the sample applications, you must import the .DAT file and load the applications into your image. Other files needed to run and test the sample applications are installed in the same subdirectory.

See "Importing and loading applications" on page 188 for additional installation information or refer to"Importing and loading sample applications" on page 55 for step-by-step instructions.

Note: If you are using VisualAge Generator Developer on Java, see "Appendix B. Installing samples for VisualAge Generator Developer on Java" on page 191.

A description of each application is provided in "Appendix C. Samples in repositories" on page 193.

Table 2. Sample applications shipped with VisualAge Generator Developer on Smalltalk

Repository Name	Configuration Map Name	Application Name
WEBS.DAT		HptSampleWebApp / HptSample3270MapApp
MQS.DAT		HptSampleMQApp
ICPKGS.DAT	VAGen Samples IC Packaging	HptICSample4GLApp / HptICSampleApp1 / HptICSampleApp2 / HptICSampleCommonPartsApp
EZEREMPS.DAT		HptSampleEzerempParts
EZERSMPS.DAT		HptSampleEzersmpParts
STAFFS.DAT		HptSampleStaffParts
EZERMSGS.DAT		HptSampleEzermsgParts
EZERPCBS.DAT		HptSamplePcbParts
EZERADFS.DAT		HptSampleEzerwadfParts

Note: If your installation gives you access to a common repository (that is, you are working from a client, not from a standalone install), you will need two pieces of information to do the following tasks:

- The host name or IP address of the system where your repository (server) is located
- A directory to which you have access that is defined to the server

You'll then need to copy repository files to that directory and specify the directory at the appropriate point in the following task.

Importing and loading applications

Most of the samples shipped with VisualAge Generator Developer on Smalltalk are stored as applications. The IC Packaging sample, however, is stored as a configuration map. See "Importing and loading configuration maps" for instructions on accessing this sample.

To import and load an application:

1. From the VisualAge Organizer window, select the **Applications** menu, then **Import/Export→Import Applications**
2. Specify the host name or IP address for the machine (for most clients this will be the local host).
3. Specify the fully qualified path to the .DAT file you want to import and click OK.
4. To load imported applications into your image, from the VisualAge Organizer window, select the **Applications** menu, then **Load→Available Applications**.
5. Select the application you want to load and the appropriate edition. Use the >> push button to copy your selections into the **Selected Editions** list box.

   Note: You can load another application by repeating the previous two steps.
6. When you have selected all of the applications you want to load, select **OK**.

Importing and loading configuration maps

To import and load a configuration map:

1. From the System Transcript window, select **Tools→Browse Configuration Maps**.
2. From the Configuration Maps Browser, select **Name→Import**

3. Specify the host name or IP address of the server where the repository is located.
4. Specify the repository you want to import from.
5. Select the configuration map and the version you want to import, move it into the list at the right by selecting the >> button. Then select **OK**.
6. To load a configuration, from the Configuration Maps Browser, select the configuration map from the Names panel and the edition from the Editions and Versions panel.
7. From the Configuration Map Browser, select **Editions→Load**.

Importing database tables

The file SAMPTBLS.CMD is stored in the C:\Program Files\VAST\samples directory for Windows (the C:\VAST\samples directory for OS/2). To import all database tables required for the sample applications and the tutorial into a database on your workstation, run the SAMPTBLS command.

Binding to a database

If you have not used this release of VisualAge Generator Developer with the DB2 database before, VisualAge Generator prompts you to run the SQLBIND command. Click **Yes** on the prompt window to run the command and bind to a database, or select **No** and set database preferences. To set database preferences:

1. From the VisualAge Organizer window, select **Options→VAGen Preferences**.
2. Select the **SQL** option and type the appropriate preferences in the fields on that page.
3. Click **OK**.

Appendix B. Installing samples for VisualAge Generator Developer on Java

The following samples are shipped as .DAT files with VisualAge Generator Developer on Java. If you do a typical install, they are installed in the samples subdirectory (c:\IBMVJava\IDE\program\samples). To use the samples, you must import the .DAT file and load the project or package into your workspace. Other files needed to run and test the samples are installed in the same subdirectory.

See "Importing and loading projects and packages" for additional installation information or refer to the section on importing and loading samples in "Chapter 8. VisualAge Generator Developer on Java: a tutorial" on page 65 for step-by-step instructions.

Note: If you are using VisualAge Generator Developer on Smalltalk, see "Appendix A. Installing samples for VisualAge Generator Developer on Smalltalk" on page 187.

Table 3. Samples shipped with VisualAge Generator Developer on Java

Repository Name	Project Name	Package Name
WEBJ.DAT	VAGen Web Transaction Sample	com.ibm.vgj.sample.map3270 com.ibm.vgj.sample.webmap
MQJ.DAT	VAGen MQ Sample	com.ibm.vgj.sample.mq
EZEREMPJ.DAT	VAGen Employee Sample	com.ibm.vgj.sample.ezeremp
EZERSMPJ.DAT	VAGen Ezersmp Sample	com.ibm.vgj.sample.ezersmp
STAFFJ.DAT	VAGen Staff Sample	com.ibm.vgj.sample.staff
EZERMSGJ.DAT	VAGen Message Conversion Sample	com.ibm.vgj.sample.ezermsg
EZERPCBJ.DAT	VAGen PCB Sample	com.ibm.vgj.sample.ezerpcb
EZERADFJ.DAT	VAGen ADF Work Database Sample	com.ibm.vgj.sample.ezerwadf

Importing and loading projects and packages

To import and load a project or package:

1. From the Workbench, select the **File** menu, then **Import**.
2. Select **Repository**.

3. Using the **Import from another repository** SmartGuide, specify the location of the repository (.DAT file).
4. To import projects, select the **Projects** radio button, then select **Details** and select the projects you want to load.

   Note: You can also import individual packages by selecting the **Packages** radio button, the **Details** button and then the packages you want to import.
5. Select **Finish** to finish importing.
6. To load imported projects into your workspace, from the **Projects** tab Selected menu, select **Add→Project**.

   Note: You can load packages using these same instructions if you select**Add→Package**. If you want to add packages instead of projects, you must have an open edition of a project in your workspace to add packages to.
7. Select Add projects from the repository and select the project to load.
8. Select the appropriate edition.
9. When you have finished selecting all the projects you want to load, select **Finish**.

Additional files needed to generate and test the sample applications are also stored on client workstations. If you did a typical install, these files are located in the default directory C:\IBMVJava\IDE\program\samples.

Importing database tables

The file SAMPTBLS.CMD is stored in the c:\IBMVJava\IDE\program\samples directory. To import all database tables required for the samples and the tutorial into a database on your workstation, run the SAMPTBLS command.

Binding to a database

If you have not used this release of VisualAge Generator Developer with this DB2 database before, VisualAge Generator prompts you to run the SQLBIND command. Click **Yes** on the prompt window to run the command and bind to a database or, select **No** and set the database options. To set database options:

1. From the Workbench **Workspace** menu, select **Open VAGen Parts Browser**.
2. Select the **Window** menu, then **Options**
3. Select the **SQL** and type the appropriate options in the fields on that page.
4. Click **OK**.

Appendix C. Samples in repositories

Samples can be used to test your system setup. You should be able to generate and run these application systems on the workstation or on the host. The samples are listed in alphabetical order by repository name. File names in the form *J.DAT file are for Java and *S.DAT are for Smalltalk.

Where appropriate, a configuration map or project name is shown. Many of the samples are managed as applications on Smalltalk and therefore, will not have a configuration map name corresponding to a project name. Many of the samples that are available for Smalltalk show only the repository name and the application name.

Repository Name on Java/Repository Name on Smalltalk
Project/Configuration Map Name

Package/Application Name(s)
Brief explanation of what the sample does.

EZEREMPJ.DAT / EZEREMPS.DAT
VAGen Employee Sample / No configuration map

com.ibm.vgj.sample.ezeremp / HptSampleEzerempParts
This sample is a simple employee database system. You can use this example to verify that your system can properly access DB2 data. The file EZEREMP.IXF contains a DB2 table and data for use with this application.

EZERMSGJ.DAT / EZERMSGS.DAT
VAGen Message Conversion Sample / No configuration map

com.ibm.vgj.sample.ezermsg / HptSampleEzermsgParts
This sample is used on the host to convert user message files to message table parts in external source format.

Note: Because this file includes some VAGen parts that have the same names as parts in EZEREMP*.DAT, you should unload the applications/packages you imported from this file, before you load the one imported from EZERMSG*.DAT.

EZERPCBJ.DAT / EZERPCBS.DAT
VAGen PCB Sample / No configuration map

com.ibm.vgj.sample.ezerpcb / HptSamplePcbParts
This sample contains examples of record definitions corresponding to all PCB types.

EZERSMPJ.DAT / EZERSMPS.DAT

VAGen Ezersmp Sample / No configuration map

com.ibm.vgj.sample.ezersmp / HptSampleEzersmpParts

This sample contains examples showing how to develop applications for basic business requirements. Applications for accessing SQL databases, DL/I databases and data files, and versions of these applications customized for the IMS environment are included.

The following files contain DB2 tables and data for the EX00A program. To use any of the tables, you must import them using:

File	Table name
INVENTRY.IXF	INVENTORY
QUOTATNS.IXF	QUOTATIONS
SECURITY.IXF	SECURITY
SUPPLRS.IXF	SUPPLIERS

The following files contain IMS program specification blocks, database definitions, and data for the EX00AD program:

- **SUPPQDBD.DBD**—Database definition source file for the SUPPQDB database
- **SUPPINDX.DBD**—Database definition source file for the SUPPQDB database
- **INVDB.DBD**—Database definition source file for the INVDB database
- **INVINDX.DBD**—Database definition source file for the INVDB database
- **EX99PS.PSB**—Program specification block source file
- **SUPPQDB.DTA**—Data file for the SUPPQDB database
- **INVDB.DTA**—Data file for the INVDB database

The following files are data files for the EX00AV program.:

- INVNFILE.CSP
- QUOTFILE.CSP
- SUPFILEV.CSP
- EZERSMP.RAF (the resource association file)

These data files are in a format usable by the VisualAge Generator test facility. To use these files with EX00AV, you

must update the entries in the EZERSMP.RAF file with the correct physical path of the first 3 files, if it has changed from the default path.

To generate the sample applications, use the default generation options file, create a Resource Association part named EZERSMP, read the file EZERSMP.RSC into it and specify it as the resource association file.

However, if you intend to generate and run the sample DL/I program, EX00AD, on host environments other than IMS, you need to remove ELAWORK from PSB EX99PS before generation. Otherwise, message ELA00215P, PSB does not match VisualAge Generator Developer PSB definition is returned when the applications is run.

EZERADFJ.DAT / EZERADFS.DAT

VAGen ADF Work Database Sample / No configuration map

com.ibm.vgj.sample.ezerwadf / HptSampleEzerwadfParts

This sample contains examples of record definitions for the ADF Work Database.

ICPKGS.DAT (Smalltalk only)

VAGen Samples IC Packaging

HptICSample4GLApp

HptICSampleApp1

HptICSampleApp2

HptICSampleCommonPartsApp

This sample is used to demonstrate automated IC packaging. For more information on these samples, refer to the VisualAge Generator User's Guide.

MQJ.DAT / MQS.DAT

VAGen MQ Sample / No configuration map

com.ibm.vgj.sample.mq / HptSampleMQApp

This sample contains programs which demonstrate two ways to implement MQ programs: using the file level interface, and using direct MQ API calls. The programs which use the file level interface are described in the file MQ.TXT, which is located in the same directory as this DAT file. The programs which use direct MQ API calls are described in the *VisualAge Generator User's Guide*.

STAFFJ.DAT / STAFFS.DAT

VAGen Staff Sample / No configuration map

com.ibm.vgj.sample.staff / HptSampleStaffParts

This sample application is a simple employee database system with a graphical user interface that displays a list of employees. The list includes user IDs, last names, salaries, and commissions. The file STAFF.IXF contains a DB2 table and data for use with this application.

This application/package includes programs that are used by the stored procedure sample. See the VisualAge Generator Design Guide for more detailed information on this sample. The following files are included with this sample: STAFF.QMF and STFPROC.SQL.

WEBJ.DAT / WEBS.DAT

VAGen Web Transaction Sample / No configuration map

com.ibm.vgj.sample.map3270 / HptSample3270MapApp

com.ibm.vgj.sample.webmap / HptSampleWebApp

This first sample is a set of text programs created using VisualAge Generator templates. The programs list and update the EMPLOYEE and DEPARTMENT tables in the DB2 sample database.

The second sample is the same set of programs modified to use web user interface records (UI Records) instead of text maps. For additional information on this sample, see WEB.TXT.

Associated files

This list provides a brief overview of the other files associated with the VAGen samples that are located in the samples subdirectory.

CUSTOMER.IXF

This database table is associated with the VAGen tutorial in "Part 2. VisualAge Generator Tutorial" on page 51. You must install this database table in the SAMPLE database using CUSTOMER as the table name to run the SAMPLE program.

VBsamp.exe, PBsamp.exe

These samples show how to build Visual Basic and Power Builder clients for VisualAge Generator server programs using VisualAge Interspace. For more information on these samples, refer to the Visual Age Generator Client/Server Communications Guide section on calling server programs via VisualAge Interspace.

Index

Q

R

S

T

V

W

Readers' Comments — We'd Like to Hear from You

VisualAge Generator
Getting Started
Version 4.5

Publication No. GH23-0258-01

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

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. _____

Readers' Comments — We'd Like to Hear from You
GH23-0258-01

Cut or Fold
Along Line

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
Information Development
Department G7IA / Bldg 062
P.O. Box 12195
Research Triangle Park, NC
27709-2195

Fold and Tape **Please do not staple** Fold and Tape

GH23-0258-01

Cut or Fold
Along Line

Part Number: CT7P7NA

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

(1P) P/N: CT7P7NA

GH23-0258-01