

VisualAge COBOL



Getting Started on OS/2

Version 2.2

VisualAge COBOL



Getting Started on OS/2

Version 2.2

Note!

Before using this information and the product it supports, be sure to read the general information under "Notices" on page vii.

Third Edition (April 1998)

This edition applies to Version 2.2 of VisualAge COBOL (Program Number 5639-B92) and to all subsequent releases and modifications until otherwise indicated in new editions. This edition also obsoletes and replaces *VisualAge COBOL for OS/2: Introducing Redeveloper*, GC26-8749. Make sure you are using the correct edition for the level of the product.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

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

IBM Corporation, Department W92/H3
P.O. Box 49023
San Jose, California, 95161-9023
U.S.A.

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

© Copyright International Business Machines Corporation 1995, 1998. All rights reserved.

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

Contents

Notices	vii
Trademarks	viii
 About This Book	 ix
 Before You Begin	 1
Hardware and Software Requirements	1
Choosing Basic or Shared Installation	6
Which Procedure Should I Choose?	7
Installing on LAN-Connected Workstations	7
If You are a LAN User:	7
If You are a LAN Administrator:	8
Restricting Users from Changing the Compiler Options Default Tool	8
Installing VisualAge COBOL	9
 Introducing VisualAge COBOL	 11
Working in a Project Environment	11
Editing Source Code	12
Creating GUI Applications	12
Creating COBOL Logic	13
Debugging Workstation Applications	14
Accessing IMS Databases	15
Redeveloping Legacy Code	16
What is Application Understanding?	16
What is Program Conversion?	17
What is Program Structuring?	17
What is Program Understanding?	18
What is Year 2000 Impact Analysis?	18
Analyzing Program Performance	19
Developing CICS Host Applications from Your Workstation	19
 Working with Host Applications	 21
Using Remote E/C/D	21
Before You Begin Using Remote E/C/D	22
Configure Your Host and Workstation for Communications	22
Perform Remote E/C/D Setup	22
 Creating Your First VisualAge COBOL Application	 23
Creating a Non-GUI Project	24
Creating the Application	27
Building the Application	28
Running the Application	30
 Creating a Simple Visual Builder Application	 31
Creating the To-Do List Application	31

Creating a Visual Project	32
Starting Visual Builder	32
Placing Parts in the Application Window	34
Resizing and Aligning the Parts	36
Connecting the Parts	39
Generating the COBOL Code for your Application	43
Building the Application	43
Running the Application	43
Exiting the Composition Editor and Visual Builder	44
Expanding the Simple Visual Builder Application	45
Expanding the To-Do List Application	45
Opening the To-Do List Part	46
Enabling the Add Push Button	46
Adding and Aligning New Parts	48
Creating a Nonvisual Part	49
Defining Your System Interface	51
Generating Your Code	51
Updating Feature Source	52
Connecting the Parts	53
Generating the COBOL Code for Your Application	56
Building the Application	56
Running the Application	56
Exiting the Visual Builder	56
Redeveloping Legacy COBOL Applications	57
Initiating Redevelopment Actions	57
Understanding Your Applications	58
Scanning the JCL Libraries	59
Retrieving the JCL Scanned Output	60
Loading the Database with the JCL Scan Output	61
Viewing the Contents of Your Inventory Database	61
Updating Your Inventory Database	64
Application Understanding Hints and Tips	65
Reengineering Your Programs	65
Converting Your Programs	66
Viewing the Conversion Log	66
Program Conversion and Reserved Words	67
Structuring Unstructured Programs	67
Analyzing Your Program for Structuring	67
Preparing Your Program to Improve Structuring	68
Structuring Your Program	69
Modularizing Your Program	70
Testing Your Program	71
Understanding Your Programs	71
Selecting Programs for Program Understanding Analysis	72
Creating SYSADATA	72
Displaying a Flow Graph	73

Displaying a Smart Listing	73
Analyzing the Year-2000 Impact on Your Programs	73
Generating Year 2000 Impact Analysis Reports	75
More Information about Year 2000 Impact Analysis	75
Using VisualAge COBOL in the Analysis and Maintenance Process	75
Appendix A. Comparison of Workstation and Mainframe Concepts	77
Appendix B. Getting Support for Using VisualAge COBOL	79
Getting Product Support	79
Getting Consulting Services	80
Getting Education and Training	80
VisualAge COBOL Glossary	81
Index	87

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:

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 (1) the exchange of information between independently created programs and other programs (including this one) and (2) the mutual use of the information that has been exchanged, should contact:

IBM Corporation
555 Bailey Avenue, W92/H3
P.O. Box 49023
San Jose, CA 95161-9023

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

Trademarks

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

AIX	MQSeries
CICS	MVS
DB2	MVS/ESA
DB2/2	OS/2
DFSMS/MVS	OS/390
IBM	S/390
IMS	System Object Model
IMS/ESA	VisualAge
Language Environment	

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

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

About This Book

This manual helps you install and learn to use VisualAge COBOL.

- “Before You Begin” on page 1 lists the hardware and software requirements for installing and using VisualAge COBOL and summarizes the installation process.
- “Introducing VisualAge COBOL” on page 11 introduces VisualAge COBOL and its components.
- “Creating Your First VisualAge COBOL Application” on page 23 guides you through creating a simple non-GUI project.
- “Creating a Simple Visual Builder Application” on page 31 introduces you to the Visual Builder where, from a Project environment, you’ll use the Composition Editor to create a visual application, adding parts in a frame window, connecting the parts, building, and running the application.
- In “Expanding the Simple Visual Builder Application” on page 45 you’ll expand this visual application to include a nonvisual part. You’ll generate feature code from the nonvisual part, which you’ll update using the Code Assistant tool.
- “Redeveloping Legacy COBOL Applications” on page 57 introduces you the features in VisualAge COBOL that help you keep your legacy code abreast of today’s technology.
- Appendix A, “Comparison of Workstation and Mainframe Concepts” on page 77 helps you relate workstation and host terms and concepts.
- Appendix B, “Getting Support for Using VisualAge COBOL” on page 79 provides information on product services and support.

This book assumes familiarity with OS/2.

Hardware and Software Requirements

Before You Begin

Important! Before you install VisualAge COBOL:

- Make sure that your workstation meets the hardware and software requirements described in “Hardware and Software Requirements” below.
- Review the information in the **readme** file. It contains the very latest product information that might not appear in any other documentation.
- Determine if you have a previous version of VisualAge COBOL installed.

You cannot install VisualAge COBOL over an existing version or release of VisualAge COBOL. You must deinstall it. If you install VisualAge COBOL over a previous version or release, you can get unpredictable results.

Important!

If you have created projects on the desktop with a previous release of VisualAge COBOL, move those projects into a folder on the desktop **before** uninstalling your current release of VisualAge COBOL.

Projects left on the desktop will lose inheritance information, which may impact the ability to migrate them to VisualAge COBOL Version 2.2.

- Decide which installation procedure you should use. The different procedures are described in “Choosing Basic or Shared Installation” on page 6.

Note: The default root installation directory for VisualAge COBOL is **IBMCOBOL**. We refer to IBMCOBOL throughout this manual as the root (or target installation) directory. If you installed VisualAge COBOL in a directory other than IBMCOBOL, specify the directory you named when performing tasks that refer to IBMCOBOL.

Hardware and Software Requirements

Processor

486-based 66 MHz processor (minimum)
Pentium-based processor or higher (recommended)

Program Understanding or Visual Builder:
Pentium-based 100 MHz processor (minimum)

Operating System

IBM OS/2 Warp Version 4.0 (FixPak 5 is recommended)
IBM OS/2 Warp Server Version 4.0 (FixPak 5 is recommended)

Memory

24 megabytes (MB) RAM (minimum) 32 MB RAM (recommended)

Program Understanding or Visual Builder:
64 MB RAM or higher (recommended)

Hardware and Software Requirements

Hard Disk Space

- Basic Installation (all files on local workstation):
 - Full product: 300 MB
 - Swap space: 40 MB

The tools and information are broken down into separate *components*. You can reduce the space required by selectively installing components. The Installation Utility displays the amount of disk space required for the components you selected. Or, you can perform a shared install.

- Shared Installation:
 - 280 MB of shared files on the LAN server
 - 20 MB on the local workstation

For connection to a Local Area Network

A LAN adapter supported by the operating system is required, for example:

- IBM Token-Ring Network Adapter/A
- IBM Token-Ring Network 16/4 Adapter/A
- A suitable Ethernet network adapter
- Other suitable compatible adapters

For host communications

A communications adapter supported by the operating system is required

Other Software

This section lists the software required for specific applications. It also includes requirements for VisualAge COBOL components that require other software.

Hardware and Software Requirements

Table 1 (Page 1 of 4). Software Requirements

Application	Requirements
DB2	<ul style="list-style-type: none"> For stand-alone DB2 workstation applications: <p>One of the following:</p> <ul style="list-style-type: none"> DB2 Universal Developer's Edition V5.0 (comprised of the DB2 SDK plus the developer licenses and media for DB2 Universal Database Personal Edition, Workgroup Edition, and Enterprise Edition, and DB2 Connect Personal Edition and Enterprise Edition) DB2 Personal Developer's Edition V5.0 (comprised of the DB2 SDK plus the developer licenses and media for the DB2 Personal Edition plus DB2 Connect Personal Edition on OS/2 and Win95/NT) DB2 for OS/2 V2.1.2 Server plus the DB2 SDK for OS/2 V2.1 DB2 for OS/2 V2.1.2 Single-User plus FixPak 8120 plus the DB2 SDK for OS/2 V2.1 Database Server for OS/2 Warp Version 4.0 plus FixPak 8122 (comprised of DB2 for OS/2 V2.1 Server plus the DB2 SDK for OS/2 V2.1) For remote access or LAN-based DB2 applications, see the Configuration tab in the <i>Information Notebook</i> and click on Configuring for remote host development. <p>To determine the service level of your installed DB2 for OS/2 product, type SYSLEVEL at an OS/2 command prompt.</p> <p>For more information, use one of the following addresses:</p> <ul style="list-style-type: none"> World Wide Web: <ul style="list-style-type: none"> http://www.software.ibm.com/data/ http://www.software.ibm.com/data/db2/library/ CompuServe <ul style="list-style-type: none"> GO IBMDB2 Anonymous FTP Site <ul style="list-style-type: none"> ftp.software.ibm.com in the directory ps/products/db2
CICS Server	<ul style="list-style-type: none"> For developing and running production applications: <p>One of the following:</p> <ul style="list-style-type: none"> CICS Transaction Server for OS/2 Warp, Version 4.1 (which contains CICS for OS/2 Version 3.1) Transaction Server for OS/2 Warp, Version 4.0 (which contains CICS for OS/2 Version 3.0) For developing host applications: <ul style="list-style-type: none"> VisualAge CICS Enterprise Application Development (provided with VisualAge COBOL Enterprise)
CICS Client	<p>CICS Clients Version 2.0 or later</p> <p>Note: This is also required for Transaction Assistant (TA) to execute an application that contains code generated by TA.</p>
MQSeries	<p>One of the following:</p> <ul style="list-style-type: none"> MQSeries for OS/2 Warp Version 5.0 MQSeries for OS/2 Version 2.0.1 MQSeries Client for OS/2 (shipped with the above product)

Hardware and Software Requirements

Table 1 (Page 2 of 4). Software Requirements

Application	Requirements
Host Data Access (via APPC) using SMARTdata Utilities (SdU)	<ul style="list-style-type: none"> On the workstation: <ul style="list-style-type: none"> One of the following: <ul style="list-style-type: none"> IBM Communications Manager/2 Version 1.11 IBM Communications Server for OS/2 Warp Version 4.0 IBM eNetwork Communications Server for OS/2 Warp Version 5.0 On the host: <ul style="list-style-type: none"> DFSMS/MVS Version 1.2.0 (5695-DF1) is required on your host
Oracle**	Oracle7 or Oracle8
Sybase**	Sybase System 10
Application Understanding, Program Understanding, Data Assistant, or Year 2000 Analysis Tool	<p>One of the following:</p> <ul style="list-style-type: none"> DB2 for OS/2 Single-User Version 2.1 plus FixPak 8120 DB2 SDK for OS/2 Version 2.1 plus FixPak 8120 Database Server for OS/2 Warp Version 4 plus FixPak 8122 DB2 Universal Database Version 5 <p>Note: For Application Understanding to interact with your host from the workstation you must install the VisualAge COBOL Remote Edit/Compile/Debug component.</p>
Remote IMS DL/I through APPC	<p>You need the following:</p> <ul style="list-style-type: none"> On the OS/390 host: <ul style="list-style-type: none"> IMS/ESA Database Manager Version 5 Release 1 (5695-176) plus enabling PTF, or, IMS/ESA Database Manager Version 6 Release 1 (5655-158) One of the following: <ul style="list-style-type: none"> OS/390 Release 3 Language Environment feature (5645-001) IBM Language Environment for MVS & VM Release 5 (5688-198) On the MVS host: <ul style="list-style-type: none"> IMS/ESA Database Manager Version 5 Release 1 (5695-176) plus enabling PTF, or, IMS/ESA Database Manager Version 6 Release 1 (5655-158) IBM Language Environment for MVS & VM Release 5 (5688-198) On the OS/2 workstation: <ul style="list-style-type: none"> One of the following: <ul style="list-style-type: none"> IBM Communications Manager/2 Version 1.11 IBM Communications Server for OS/2 Warp Version 4.0 IBM eNetwork Communications Server for OS/2 Warp Version 5.0

Hardware and Software Requirements

Table 1 (Page 3 of 4). Software Requirements

Application	Requirements
Remote Edit/Compile/Debug function using TCP/IP	<ul style="list-style-type: none"> On the OS/390 Host: <ul style="list-style-type: none"> IBM COBOL for OS/390 & VM Version 2 Release 1 (5648-A25) Full Function Offering plus Debug Tool PTFs for APARs PQ6202 and PQ8277 and the PTF for APAR PQ13212 with OS/390 Release 3 Language Environment feature (5645-001) OR IBM COBOL for MVS & VM Version 1 Release 2 (5688-197) Full Function Offering plus Debug Tool PTFs for APARs PQ6202 and PQ8277 and the PTF for APAR PQ13211 with IBM Language Environment for MVS & VM Release 5 (5688-198) TCP/IP Version 3 Release 2 for MVS/ESA (5655-HAL) (TCP/IP Version 3 Release 3 is not recommended) DFSMS/MVS Version 1.2.0 (5695-DF1) with the Network File System Feature (minimum) or DFSMS/MVS Version 1.3.0 (5695-DF1) with the Network File System Feature and the PTF for APAR OW25973 (recommended) <ul style="list-style-type: none"> On the MVS Host: <ul style="list-style-type: none"> IBM COBOL for MVS & VM Version 1 Release 2 (5688-197) Full Function Offering plus Debug Tool PTFs for APARs PQ6202 and PQ8277 and the PTF for APAR PQ13211 IBM Language Environment for MVS & VM Release 5 (5688-198) TCP/IP Version 3 Release 2 for MVS/ESA (5655-HAL) (TCP/IP Version 3 Release 3 is not recommended) DFSMS/MVS Version 1.2.0 (5695-DF1) with the Network File Feature (minimum) or DFSMS/MVS Version 1.3.0 (5695-DF1) with the Network File System Feature and the PTF for APAR OW25973 (recommended) On the OS/2 Workstation: <ul style="list-style-type: none"> IBM TCP/IP for OS/2 Version 2.0 IBM TCP/IP for OS/2 Version 2.0 NFS kit with CSD UN57064 and the fix for APAR PQ00835

Table 1 (Page 4 of 4). Software Requirements

Application	Requirements
Remote Edit/Compile/Debug function using APPC	<ul style="list-style-type: none"> On the OS/390 Host: <ul style="list-style-type: none"> IBM COBOL for OS/390 & VM Version 2 Release 1 (5648-A25) Full Function Offering plus Debug Tool PTFs for APARs PQ6202 and PQ8277 and the PTF for APAR PQ13212 with OS/390 Release 3 Language Environment feature (5645-001) <p>OR</p> <ul style="list-style-type: none"> IBM COBOL for MVS & VM Version 1 Release 2 (5688-197) Full Function Offering plus Debug Tool PTFs for APARs PQ6202 and PQ8277 and the PTF for APAR PQ13211 with IBM Language Environment for MVS & VM Release 5 (5688-198) DFSMS/MVS Version 1.3.0 (5695-DF1) and the PTFs for APARs OW20884, OW27760, OW27934 On the MVS Host: <ul style="list-style-type: none"> IBM COBOL for MVS & VM Version 1 Release 2 (5688-197) Full Function Offering plus Debug Tool PTFs for APARs PQ6202 and PQ8277 and the PTF for APAR PQ13211 IBM Language Environment for MVS & VM Release 5 (5688-198) DFSMS/MVS Version 1.3.0 (5695-DF1) and the PTFs for APARs OW20884, OW27760, OW27934 On the OS/2 Workstation: <ul style="list-style-type: none"> IBM Communication Manager/2 Version 1.11
HTML-formatted Information	Netscape Navigator for OS/2 Version 2.02

Choosing Basic or Shared Installation

There are two main ways you can install VisualAge COBOL: basic or shared. The install command is different for each, but the steps you follow are the same.

- **Basic installation**

This procedure installs all files for the components you select to your hard drive. The drawback to a basic installation is that you need a large amount of disk space. The command for basic installation is **install**.

- **Shared installation**

For the components you select, this procedure installs only those files that must be local to your hard drive. The rest of the files remain on the CD-ROM or on a LAN server, and you access them from the CD-ROM or server. A shared installation for all components requires approximately 30 MB of disk space. The command for shared installation is **shrdinst**.

If You Run Shared Installation from a CD-ROM

There are some drawbacks to a shared installation from the CD-ROM:

Installing on LAN-Connected Workstations

- You cannot apply corrective service (CSDs) or fix packs to the CD-ROM. If you want to apply fixes to VisualAge COBOL components, you must use basic installation.
- VisualAge COBOL performance from the CD-ROM may be slower than from your hard drive. The speed of accessing and transferring information is usually considerably slower from the CD-ROM than from your hard drive.

Which Procedure Should I Choose?

Use basic installation if:

- You want to maintain files locally on your hard drive and you have enough space on your hard drive.
- You are concerned about performance for shared installation from the CD-ROM.

Use shared installation if:

- You want to minimize the disk space used on your hard drive.
- You want to maintain common files on a LAN server for the team to use.
- The LAN administrator has restricted access to the Compiler Options Default Tool.

Installing on LAN-Connected Workstations

If your workstations are connected to a LAN, you can put the VisualAge COBOL installation files on a LAN server and then install from the server to the individual client workstations. Installing from the server is faster than installing from the CD-ROM, and several client workstations can install at the same time or only necessary files, with the remainder left to reside on the server (a shared installation). Note that most of the problems associated with shared installation from the CD-ROM do not affect shared installation from a LAN. In addition, with a shared installation, you can ensure that all clients are using the same versions of the common files.

Important: When you install VisualAge COBOL across multiple workstations, make sure you observe the license agreement as described in the *License Information* booklet.

If You are a LAN User:

You need to know what kind of installation is available from the server (ask your LAN administrator). If both basic and shared are available, decide which you prefer. Then follow the same steps as you would for installing to a single workstation. You can run the install program either attended or unattended using a response file. For more information, see the INSTALL.TXT file on the VisualAge COBOL CD-ROM.

You **cannot** install VisualAge COBOL to a remote drive on the LAN because certain files are not accessible when rebooting your system.

Important Information for Shared Install! If you are adding components, (components that you did not initially install), make sure you have access to the LAN directory where the VisualAge COBOL installation files reside. If not, VisualAge COBOL will

Restricting Access to Default Tool

attempt to install on the drive where the local files from the shared install reside. VisualAge COBOL will then issue an error message stating the drive is not ready.

If You are a LAN Administrator:

You first need to decide whether to set up the server for basic installation, shared installation, or both. Follow the steps below to put the appropriate installation files on the server. Clients can choose basic or shared installation.

This procedure does not require changes to the server's CONFIG.SYS file.

1. Create a directory on the server that LAN users can access (for example, IBMCOBOL).
2. Change to the LAN directory that you have just created.
3. Use XCOPY to copy all the files from the CD-ROM to the server. The syntax for XCOPY is:

```
xcopy d:\* /s
```

where *d* is the CD-ROM drive.

4. Notify LAN users of where the installation files are located.

You can also install VisualAge COBOL directly on the LAN server following the instructions as you would for a single workstation.

Restricting Users from Changing the Compiler Options Default Tool

If you are allowing users to perform a shared installation, you must decide whether to allow them to change fixed compiler options from the Compiler Options Default Tool. To restrict access to this tool, decide which compiler options to set so that users cannot change them after installation.

To invoke the tool, from the task bar, select **VisualAge COBOL→Tools→Compiler Default Option Tool**.

Use the tool to set the default value of the compiler options when they are not specified in the compiler invocation or on CBL or PROCESS statements in the COBOL source file. Do *not* use the tool to set or change compiler options for a specific compilation.

You may:

- Accept the defaults provided by clicking on **OK**.
- Accept the changes you have made by clicking on **OK**.
- If you made changes, but decide to change back to the defaults provided, and stay within the tool, click on **Default**. This resets to the *existing* default options.
- If you made changes, but decide to change back to the defaults provided, and want to exit out of the tool, click on **Cancel**.

To restrict users from using the tool:

1. Remove and copy from the IBMCOBOL\BIN directory to a different directory the following files:

DIAMOND.EXE
IWZQDMND.EXE

By removing these files, the object for the Compiler Default Options Tool will not be created in the **Tools** folder. Users will not see the tool after installation.

2. Remove and copy from the IBMCOBOL directory to a different directory the following files:

IBMCOBOL.ICF
IBMCOBOL.PKG
INSTALL.CMD

3. Copy **SHRDINST.CMD** to **INSTALL.CMD**. Your users are now restricted to shared installations only. Users can now use **INSTALL** to perform a shared installation.

Installing VisualAge COBOL

To install VisualAge COBOL, switch to your CD-ROM drive and type **install** for a regular install, or **shrdinst** for a shared install.

If you get an error messages during the installation procedure, select the **Help** button.

For detailed installation information, see the **INSTALL.TXT** file on the VisualAge COBOL CD-ROM.

Introducing VisualAge COBOL

VisualAge COBOL is a COBOL development environment for creating applications on workstations running OS/2 or Windows NT. It offers the best of both traditional and cutting-edge COBOL programming.

VisualAge COBOL provides a set of visual tools to edit, compile, and debug your programs, integrated in a project environment. When you set up your application project, these tools are available from the pop-up menus of your files. The files can reside on either your workstation or the host.

The VisualAge COBOL remote functions lets you work with your host applications from your workstation. It provides a seamless workstation environment for the development of host programs. From the workstation, you can edit, compile, and debug COBOL applications on the host, even applications using DL/I calls. Files are directly loaded from the host to the workstation without the need for explicit downloading and workstation naming and mapping. Communication to the host is provided using TCP/IP or Advanced Program-to-Program Communication (APPC).

Not only does VisualAge COBOL offer the tools you need to efficiently and quickly develop applications of the future, it enables you to pull your legacy applications in to the future with you. You do not have to scrape your old applications and write new code. Using VisualAge COBOL, you can analyze your applications to develop migration plans, study your program code to find what ails it, convert older levels of code to the latest COBOL standards, structure unstructured code, and develop detailed report identifying potential year-2000 problems.

Working in a Project Environment

With the VisualAge COBOL project environment, your tools and files are integrated into a single development environment, where you can work with your files directly rather than accessing them through individual tools. For example, when you want to edit your source code, you select the icon representing the file and invoke the edit action from the file icon's pop-up menu. You can concentrate on the file itself since you can rely on the project to provide context-sensitive actions for your files.

With VisualAge COBOL, you organize your code by grouping related files into projects. A *COBOL project* is the complete set of data and actions you need to build a single *target*, such as a dynamic link library (DLL) or executable (EXE).

VisualAge COBOL provides pre-configured COBOL projects. When you create your own COBOL projects, you get a complete set of actions, types, and environment variable settings pre-configured for your particular project type. This means that you get an environment with the tools and actions already set up for you.

For more information, see "Creating Your First VisualAge COBOL Application" on page 23.

Introducing VisualAge COBOL

Editing Source Code

The COBOL Editor provides language-sensitive editing for your files. Different COBOL constructs, such as comments, are shown in different colors. In addition, context-sensitive help is available for most COBOL language elements. COBOL keywords are highlighted and by pressing F1, you get help for that particular language element.

Aside from standard editing functions, you can create GUI code, generate SQL statements, and generate calls to invoke CICS transactions.

Code Assistant

Generates code to access the data and attributes of your graphical user interface parts. Code Assistant is only available if you have installed the Visual Builder. Code Assistant works with files with CBV and CPV extensions. The Visual Builder must be running to access Code Assistant.

Data Assistant

Simplifies the process of constructing embedded SQL statements. SQL statements generated by Data Assistant can be compiled and executed in OS/2, Windows, and host COBOL programs. Data Assistant is only available for non-GUI applications.

Transaction Assistant

Generates a COBOL CALL to ECICALL and a parameter list, based on your input, for invoking CICS transactions. ECICALL uses the parameter list to make the actual ECI call. Transaction Assistant is only available for non-GUI applications.

The COBOL Editor displays certain tool bar icons and menu choices for the coding assistants only when you need them. The tool bar icons and menu choices for the Data and Transaction Assistants come up *only* if you are editing a COBOL file (CBL), a copy file (CPY), a DB2 file (SQB), or a CICS file (CCP).

Creating GUI Applications

The COBOL Visual Builder enables you to create COBOL GUI applications.

The Visual Builder is based on a “construction from parts” paradigm. It consists of the:

Composition Editor

Use the Composition Editor to create the views for your application, choose the parts that perform the logic you need, and make connections between the parts.

System Interface Editor

Use the System Interface Editor to specify the names of files and resources associated with the current part.

Part Interface Editor

Use the Part Interface Editor to define the features (attributes, actions, and events) for your parts, along with a list of preferred features for the pop-up connections menu. These features make up the

Introducing VisualAge COBOL

part's interface. You use them when you make connections between collaborating parts. You can also promote features of subparts from this editor.

For more information, see "Creating a Simple Visual Builder Application" on page 31.

Creating COBOL Logic

VisualAge COBOL supports development of new COBOL applications that are targeted for the workstation environment.

Compiler and Run-Time Environment

The VisualAge COBOL compiler and run-time environment supports the high subset of ANSI 85 COBOL functions, just like the other IBM COBOL products. Your applications can be compiled and run on supported platforms, whether they are created on a mainframe, an AIX workstation, or a personal computer with OS/2 or Windows NT.

Although the IBM COBOL language is practically the same across platforms, there are some minor differences between IBM COBOL for OS/390 & VM and VisualAge COBOL. These differences are documented in the *Summary of Differences: Host COBOL and Workstation COBOL* topics in the *VisualAge COBOL Programming Guide* and the *IBM COBOL Language Reference*. Porting considerations are described in "Porting Applications between Platforms" in the *VisualAge COBOL Programming Guide*.

COBOL Millennium Language Extensions (MLE)

MLE is IBM's patent-pending technology that provides support for automated date windowing to help you address the year 2000 challenge. MLE uses a windowing approach to identify and convert date data from a two-digit year to four-digit year. Provided through IBM extensions to the COBOL language, MLE gives you a mechanism to indicate to the compiler which dates should be windowed. It's a compiler-assisted solution for your windowed dates.

For more information, see *COBOL Millennium Language Extensions Guide*.

Object-Oriented Extensions

VisualAge COBOL's object-oriented language extensions are a syntax extension to COBOL that implement a complete object-oriented paradigm. These object-oriented extensions allow you to define object classes and subclass objects, to instantiate objects, and to have objects inherit characteristics from other objects.

VisualAge COBOL creates language-neutral objects that interoperate with objects created in other object-oriented languages enabled for IBM's System Object Model* (SOM*). This is provided through VisualAge COBOL's Direct-to-SOM capability.

Introducing VisualAge COBOL

DB2 Co-Processor

The DB2 co-processor eliminates the DB2 precompiling step, resulting in better optimization of EXEC SQL statements. The DB2 support is fully integrated with the compiler. Your source program containing embedded SQL statements is handled by the compiler without your having to use a separate precompiler. When the compiler encounters SQL statements and at significant points in the source program, the DB2 co-processor processes the SQL statements by taking appropriate actions and indicating to the compiler what native COBOL statements to generate at that point.

Distributed Data Access

VisualAge COBOL also provides a set of functions that enable your applications to handle data across distributed environments. The services include:

- Local VSAM record file system

- Access to remote VSAM record files using the file processing capabilities of COBOL

- Copy, sort, and merge functions for both record and byte files

- Access to local and remote DB2, CICS, Oracle, and Sybase database using native, SQL support

- Access to local and remote databases that support ODBC.

- Data managed by Btrieve** using the file processing capabilities of COBOL.

- Local sequential, relative, and indexed files using the STL (a VisualAge COBOL access method) file system that supports full ANSI 85 COBOL standard file I/O language.

These services complement their counterpart services on the mainframe, enabling you to create client/server and cooperative processing applications using the IBM COBOL language. Your applications can also call the utilities directly using the application programming interfaces (APIs) that come with the utilities.

Debugging Workstation Applications

The interactive debugger helps detect and diagnose errors in code developed with VisualAge COBOL. Using the interactive debugger, you can:

Step Through or Run a Program

You can step through your program one line at a time, or you can run the program until a breakpoint is encountered, the program is halted, or the program ends.

You can also select the way the interactive debugger steps through a program. If it is a call, the program's run can be halted when the call is complete, at the first statement in the called program, or at the return statement of the current program. The interactive debugger can also step over any program for which debugging is not available, for example, library and system routines.

Introducing VisualAge COBOL

Set Breakpoints

You can control how your program executes by setting breakpoints. A breakpoint stops the execution of your program at a specific location or when a specific event occurs.

For year 2000 projects, you can specify that execution halts on all date fields. Used in conjunction with MLE, "Creating COBOL Logic" on page 13, and Year 2000 Impact Analysis, "What is Year 2000 Impact Analysis?" on page 18, you have a powerful tool to meet the year 2000 challenge.

View the Program Source Code

You can view the source code of the program you are debugging. You can view it as a source, disassembly (assembler instructions), or mixed (a combination of source and disassembled code).

Monitor Variables

You can display and change the variables during debugging.

Monitor the Registers

You can view all the processor and coprocessor registers for a particular thread.

Monitor the Call Stack

You can display all of the active programs, the remaining stack size, the stack frame size, and the return address. When the state of the program changes, such as when you execute the program or you update displayed data, the debugger changes the information displayed to reflect the current state.

Monitor Storage

You can monitor variables in a storage window. For example, if you are monitoring a pointer, as the pointer changes, the storage window changes to show the new location referenced by the pointer.

Accessing IMS Databases

Remote DL/I provides access to IMS full function databases and GSAM databases from programs using DL/I calls running on a workstation. Remote DL/I provides the support to develop and test mainframe COBOL programs on a workstation that use DL/I calls in IMS batch applications that access IMS full function databases and GSAM databases.

With VisualAge COBOL's Remote DL/I support, you can develop, compile and test COBOL programs that run in an IMS batch environment that use CBLTDLI calls on the workstation. Additionally, you can develop, compile, and test CICS COBOL programs that use CBLTLI calls to access IMS full function databases.

Note:

- Remote DL/I does not provide access to IMS message queues or IMS fast path databases.
- Remote DL/I runs using only S/390 data types as input and output. It does not provide any data conversion function.

Introducing VisualAge COBOL

Remote DL/I uses APPC and an IMS batch environment to provide the remote DL/I call support. When Remote DL/I is first initialized on the workstation, you are prompted for your TSO userid and password, which are used when the remote job is started on MVS. APPC is used by Remote DL/I to start a job on MVS, which brings up an IMS batch environment. Once the IMS batch environment is available, remote DL/I calls can be processed. The DL/I calls on the workstation are sent to the IMS batch environment and executed. The results of the DL/I call are then sent back to the program running on the workstation.

Redeveloping Legacy Code

VisualAge COBOL can help you analyze, understand, and maintain legacy COBOL programs and applications. With VisualAge COBOL, you can reengineer existing programs to produce applications better suited for future tasks.

VisualAge COBOL helps you maintain COBOL program code and applications by:

- Helping you understand the resources, structure, and flow of an application.
- Helping you convert COBOL programs to later ANSI standards of COBOL.
- Helping you structure your unstructured COBOL programs.
- Helping you understand the structure, logic, and flow of a program.
- Helping you identify places in your code where two-digit dates could cause you trouble when processing dates later than 1999.

With VisualAge COBOL, COBOL program code and applications can be analyzed both before and after converting.

Support for legacy code includes:

- **Application Understanding**, which helps you understand your existing MVS and VSE inventory of applications.
- **Program Conversion**, which helps you convert older COBOL programs to higher levels of COBOL.
- **Program Structuring**, which helps you structure your unstructured COBOL programs.
- **Program Understanding**, which helps you understand the logic and flow of your COBOL programs.
- **Year 2000 Analysis**, which helps you locate occurrences of two digit year fields in your programs.

What is Application Understanding?

When maintaining applications, it is useful to understand how they are constructed, where they are used, and what files they process. Application Understanding helps you do this by extracting key information from your existing inventory of JCL and presenting this information to you in easy to understand graphic windows.

Introducing VisualAge COBOL

Key information about your applications is gathered by scanning your JCL. The information extracted by Application Understanding's host JCL scanner is downloaded and loaded into a DB2 database on your workstation. A scanner is provided for both MVS and VSE.

Through a workstation graphical interface, you can see the content of your applications and the resources that they use. For example, you can view all the jobs and jobsteps that process a particular file or execute a particular program, or you can see all the jobs that make up an application or all the job steps in a given job. You can follow the JCL connections in your applications. For example, if you are viewing a job, you can expand the view to see the job steps and then expand it again to see the DD statements of a job step. You can go from one view directly to another.

What is Program Conversion?

Program Conversion helps streamline the maintenance of COBOL programs by helping you convert your program code to higher levels of COBOL. The objective of Program Conversion is to extend the life of your COBOL programs by making them easier to convert. You can convert programs written to the ANSI COBOL Level 68 or 74 standard to programs that meet the ANSI COBOL Level 84 Standard.

Program Conversion in VisualAge COBOL is based on the technology that is currently available in the current IBM COBOL and CICS/VS Command Level Conversion AID.

Input Languages: Use VisualAge COBOL to automate the conversion of valid COBOL programs written in any of the following languages:

- DOS/VS COBOL
- OS/VS COBOL LANGLVL(1)
- OS/VS COBOL LANGLVL(2)
- VS COBOL II Release 1 and Release 2
- VS COBOL II Release 3 (CMPR2)

Output Languages: You can convert any of the input languages listed above to any of the following output languages:

- VS COBOL II Release 4
- COBOL for MVS & VM, VisualAge COBOL
- COBOL Set for AIX

Program Conversion eliminates user-defined name and reserved word conflicts, flags language elements it cannot convert, and produces a log of information that helps you assess the conversion and to track statistics about converting your COBOL inventory.

What is Program Structuring?

Use Program Structuring to analyze programs and optimize their structure. Program Structuring structures your programs and produces reports that help you understand the structured output program, how it relates to the input program, and help with possible modularization.

Introducing VisualAge COBOL

What is Program Understanding?

Program Understanding helps you understand complex COBOL software programs. You can analyze and extract information about an entire set of programs and examine this information on the workstation using a workstation graphical user interface to show both graphical and textual views of the physical design of the programs.

Graphically, you can explore various views of the physical design of programs. At the highest level, you can discover the relationships between the programs that compose the entire application. A mid-level view displays the calling structure between the routines within a program or executable, including routines external to a program. At the lowest level, you can examine the internal control flow of each program.

By default, multiple views are linked. A linked view depicts the association between the compiled source code and the graphical control flow representations of the program.

Program Understanding presents hypertext cross-reference information that details program usage, data structures, symbol usage, and COPY library usage. You can use the expanded source code browser view as a launching pad for navigation within and across programs.

You can request dataflow analysis on any user defined variable at any point in the program. You can also view all instructions that use the selected variable.

Program Understanding technology can:

- Provide complete user defined high-, medium-, and low-level information about large-scale software applications.
- Provide dataflow analysis.
- Provide program slicing to view business logic.
- Reduce the time application programmers need to become familiar with large-scale applications.
- Facilitate the migration of applications to the latest technology, such as client/server computing.
- Reduce errors that occur when maintenance tasks are performed manually on complex applications.
- Improve the productivity of software maintenance programmers.

What is Year 2000 Impact Analysis?

VisualAge COBOL Year 2000 Impact Analysis helps tackle the 2000 challenge by assisting you in locating occurrences of various kinds of year-related fields in your programs and examine them on the workstation.

Using the Cross Compilation Unit Analysis, you can propagate impact analysis information between programs.

Analyzing Program Performance

The Performance Analyzer helps you understand your program's flow and tune your program's performance.

It traces the execution of your application and allows you to analyze the call-return path of your COBOL programs as well as the paragraphs within your programs. The generated trace file contains trace analysis data that can be graphically displayed in diagrams. Using these diagrams, you can improve the performance of an application, examine occurrences that produce faults, and in general, understand what happens when your application runs.

The Performance Analyzer does not replace static analyzers or debug tools, but it can complement them by helping you understand aspects of the application that would otherwise be difficult or impossible to see.

For instance, you can:

Time and tune applications

The Performance Analyzer time stamps each trace event using a high resolution clock (about 838 nanoseconds per clock tick). As a result, the trace file contains a detailed record of when each traced function was called and when it returned.

The trace data also shows how long each function runs. This helps you find *hot spots*, the areas within an application where a disproportionate amount of time was spent.

Locate program hangs and deadlocks

The Performance Analyzer provides a complete history of events leading up to the point where a program stops. You can view the function call stack from anywhere in the application.

Trace multithreaded interactions

When multithreaded applications are traced, you can look at the sequencing of functions across threads in some of the diagrams. This highlights problems within critical areas of the application.

Trace the complete application

Not only does the analyzer trace procedures in the EXE file, but it traces the entry points to system calls and application DLLs.

Developing CICS Host Applications from Your Workstation

Included with VisualAge COBOL is the VisualAge CICS Enterprise Application Development offering, which enables complex transaction management applications to be developed and tested using the function available in VisualAge COBOL. VisualAge CICS Enterprise mirrors the enterprise environment for transaction systems on a typical desktop computer. It provides the capability to use the power of the workstation together with networks, which can be local area or wide area, stand-alone or host-attached.

Introducing VisualAge COBOL

With VisualAge CICS Enterprise, you can:

- Communicate between application programs and local and remote workstations, terminals, subsystems, printers, and other devices
- Add additional CICS resources while debugging and testing transactions.
- Use the standard CICS distributed features all in one desk-top computer namely, Distributed link, Transaction routing, Function shipping and Distributed Transaction Processing over MRO or ISC
- Enable a non-CICS application to call a VisualAge CICS program synchronously or asynchronously, using External Call Interface (ECI)
- Enable programs in one CICS system to have requests to address resources (such as files or queues) in any other connected CICS system executed on that system.
- Use the performance analyzing function to provide information that will assist in the resolution of application performance issues including a high level application trace and probe
- Use IBM's leading commercial messaging MQSeries (TM) software

Working with Host Applications

The VisualAge COBOL Remote Edit/Compile/Debug component (Remote E/C/D) provides a workstation environment for performing the edit, compile, and debug tasks associated with host COBOL application development. It also serves as the basis for the Application Understanding component to access the host.

Application parts, such as COBOL source code, COBOL copy books, and host JCL, are kept in PDS or PDSE data sets on the host. You access and work with these files through an MVS project. If you keep your source, copy files, or JCL in a library system on the host, you must copy them from the host library to a PDS to make them available for MVS project actions. Also, the names you see on the workstation for your host PDSs and PDS members depend on how you defined your MVS drives during Remote E/C/D setup.

Using Remote E/C/D

Using Remote E/C/D, you can:

Access host data sets in COBOL MVS projects

After completing the required configuration, you can connect to PDS or PDSE data sets and include members in your project. You can then access your host files as if they were workstation files.

Edit host files with the COBOL editor

You can edit remote files using the COBOL editor, which enables you to use the language-sensitive editing capability of the COBOL editor on host COBOL programs.

Compile on the workstation to check syntax

From the workstation, you can run a syntax-check compile on your source files to ensure they compile error free before submitting them for compile on the host.

Submit batch jobs to the host

You can submit jobs directly from your project without logging on to your host.

Monitor the status of batch jobs on the host

From your workstation, you can monitor the status of jobs submitted to the host, as well as look at job output.

Execute TSO commands on the host

You can open a TSO command prompt window from your project, allowing TSO commands to be submitted to the host.

Debug programs executing on the host

With the Debug Tool, you can debug applications running in the OS/390 or MVS environments. The supported environments are batch (JES), TSO, CICS, and IMS (BTS). The debugging sessions are cooperative; thus, the user interface for the session resides on the workstation. The Debug Tool runs on the host with your application. The Debug Tool and workstation interface communicate with each other through APPC or TCP/IP.

You can start or submit the following actions from your workstation, but the activity must be done on the host:

- Preprocessing, compiling and link-editing via batch job submission
- Running applications for debugging

You can find detailed information about using Remote E/C/D on the **MVS Projects** page of the **Information Notebook**.

Before You Begin Using Remote E/C/D

Before you can start using Remote E/C/D, the following must be completed:

1. Configure the Host and your workstation for TCP/IP or APPC communications
2. Perform Remote E/C/D setup

Your host system administrator will configure the host. You, or your workstation support personnel, will setup your workstation.

Before configuring your host system, ensure you have the appropriate software prerequisites installed on your host and workstation. For details, see “Other Software” on page 2.

Configure Your Host and Workstation for Communications

After installing VisualAge COBOL, you (or your host administrator) can navigate through a browser-driven scenario that builds the appropriate configuration steps, customized for your specific requirements and environment. From the task bar, select **VisualAge COBOL→Information Notebook**. Then, click the **Configuration** tab and double click on **Configuring for remote host development**.

Perform Remote E/C/D Setup

After completing the configuration necessary for your host, you then need to set up your workstation before you can use Remote E/C/D. With Remote E/C/D setup, you provide user-specific information required to complete the host and workstation communication. For example, you'll provide your TSO userid, define which host data sets you want to access and how you access them, and indicate whether you have optional software that enables some advanced features.

To complete the Remote E/C/D setup, you edit the MVSINFO.DAT file provided in the IBMCOBOL\MACROS directory. You can edit the file directly, or invoke the **MVS Setup** action from your MVS project, which opens the MVSINFO.DAT file.

Details on the information you need to collect before setting up Remote E/C/D, instruction on how to perform the setup, and steps to verify the communications are correct are included on the **Configuration** page of the **Information Notebook**.

Creating a VisualAge COBOL Application

Creating Your First VisualAge COBOL Application

This chapter guides you through creating your first VisualAge COBOL application. The steps you follow here teach you the basic principles that you use for further applications that you create.

Your first step in developing an application with VisualAge COBOL is to set up a project. The VisualAge COBOL development paradigm centers around the concept of a *COBOL project*, which is a container of your application files, such as COBOL source files, copy files, listings, object code, and executable files. Projects are set up to enable you to perform actions on those application files. The actions vary depending on the type of file. For example, edit is an action appropriate for a COBOL source file. However, the edit action would not be appropriate for an executable file.

With VisualAge COBOL, you can work with the following types of applications:

Non-GUI Applications

Enables you to create applications using basic COBOL language features.

Visual Builder Applications

Enables you to create applications with a graphic user interface (GUI). Using a "construction from parts" paradigm, you create applications by dropping parts onto a palette and then adding connections to define the actions for those parts. For more complex applications, you then add user code to complete the application.

Remote Host Applications

Using the Remote Edit/Compile/Debug feature of VisualAge COBOL, you can work with your host applications from the workstation, without having to download the applications or data to the workstation.

The Hello Application you create in this chapter is a non-GUI application, which contains a COBOL source file (a component) from which you build (compile and link) the running COBOL program (a target). When you finish, you have an application that displays a customized greeting.

Figure 1 on page 24 shows you what the application's interface looks like when you have finished.

Note: A complete Hello Application is provided in the samples folder. You can use the Hello Application sample to get an idea of how your Hello Application will run.

Creating a VisualAge COBOL Application



Figure 1. Hello Application. A COBOL application you can build using VisualAge COBOL.

The main steps you follow are:

- Creating the project
- Coding the application
- Building the application
- Running the application

Creating a Non-GUI Project

To create a new project, you use Project Smarts. Project Smarts contains notebook pages on which you define the type of project you want to create and other information needed to create the project. By default your project files are placed in a folder on your desktop, COBOL Projects. You open existing project files contained in this folder by double-clicking on the project's IWP file.

To create a non-GUI project do the following:

1. Select **VisualAge COBOL** → **Accessing COBOL Projects** from the Task Bar. The dialog box for creating new projects or opening existing project opens.
Note: The first time you create a new project, the COBOL Projects folder is created on your desktop.
2. Select **Create new project**. The Project Smarts window opens.
3. On the **Projects** page, you indicate the type of project you want to create. Select **COBOL Non-GUI Project**.
4. Click **Next**. The **Location** page opens.

Creating a VisualAge COBOL Application



Figure 2. Location page for creating new projects.

5. On the **Location** page:

- Enter a title for your project; in this case, enter the name Hello Application. The title shows in the title bar of the project.
- Enter the directory where you want to put your source files. In this case, enter E:\HELLOAPP, substituting the actual drive you'll be using in place of E. The HELLOAPP subdirectory is created after completing this page.
- Accept the default specification for Where to create the project file.

The project file (.IWP) contains general project information and settings. Associated with the .IWP file is the .IWO file, which contains the options settings for the actions you can invoke on files in your project. The .IWP and .IWO files must be in the same directory and they must have the same filename. They do not need to be the same directory as the project source files.

- In the **Project file name** field, enter the filename. For this project enter, HELLOAPP.

Click **Next**. The **Target** page opens.

6. On the **Target** page, indicate the name of the target file. The target file (usually an .EXE or .DLL) is generated when you build the application. For this application, use the default HELLOAPP. The default target file is assigned based on the first eight characters in the Project Title field on the Location page.
7. Click **Done**. A window opens confirming that the project was created successfully.
8. Click **OK**. The Hello Application project opens. Also, the project files (HELLOAPP.IWP and HELLOAPP.IWO) are added to the COBOL Projects folder on your desktop.

Creating a VisualAge COBOL Application

The project window includes:

1 System menu

Contains menu items for manipulating (minimizing, maximizing, or closing) the window that contains your project.

2 Project toolbar

Contains buttons for frequently used actions.

The left side of the project toolbar contains the Monitor button, which displays the project monitor window where the output of monitored actions is displayed (such as compile and link). It also contains a button that opens the How Do I help for COBOL projects.

The right-hand side of the project toolbar contains buttons for launching frequently-used actions like Build, Debug, and Run.

3 Menu bar

Contains menu items to launch project-scoped and file-scoped actions, bring up the project's settings notebook, set options, and link to online books and How Do Help.

Project lists all the project-scoped actions available to the project. The menu also contains actions to open or create another project, to close the project view, and to exit the project.

Selected contains the actions that apply to any selected file (or subproject). The menu changes depending on the type of file selected.

Both project-scoped and file-scoped actions are also accessible from the pop-up menus.

View contains controls for opening different views on the project, including the project's settings notebook. It also contains view options for the project toolbar and information area.

Options includes a list of the actions for which you can change option settings. When you select an action from this menu, the options dialog for the action displays.

Help contains items for referencing project online help and other VisualAge COBOL information..

4 Filter

Enables you to select which files display in the project container. The drop-down list box lists all the masks that are available to the project. To use multiple file masks, separate them with a semicolon.

5 Container

Displays the project files and any subprojects.

Creating the Application

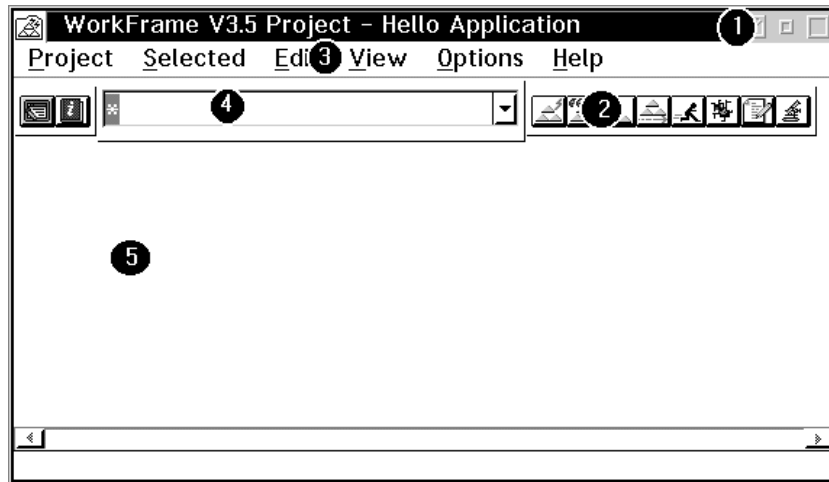


Figure 3. The project window

Now that you have created your project, you can create the files you need for the application as explained in “Creating the Application”

Creating the Application

Once you have created a new project, you have a set of actions available for the files you create for your program.

Note: This section is included to explain how to create COBOL source files and enable the language-sensitive editing feature. If you're familiar with these concepts, you can copy HELLOAPP.CBL from the SAMPLES\HELLOAPP subdirectory where you installed VisualAge COBOL to the directory where you have stored your source files for this tutorial. Then, skip this step and continue with “Building the Application” on page 28.

To create the COBOL source file for the Hello Application project:

1. From the project menu-bar, select **Project** → **Edit**. The live parsing editor opens with a default document titled *Editor - Untitled Document 1*.

To enable the COBOL language-sensitive editing features:

- From the Editor menu-bar, select **Options** → **Profiles** → **Change profile**. The Change profile window opens.
- In the Change profile window, click on the drop-down arrow to the right of the **Language Profile** drop-down combination box. Locate the item **cbl**; you might have to scroll to find it. Click on **cbl** to select it.
- Click on **OK**. The COBOL language-sensitive editing features are enabled.

Building the Application

2. Enter the following source code:

```
Identification division.
Program-ID. Helloapp.

Data division.
Working-storage section.

01 Program-work-fields.
   05 Input-name      Pic x(30).
   05 Output-name     Pic x(37).

01 Program-flags.
   05 Loop-flag       Pic 9(01).
   88 Loop-done       Value 1.

Procedure division.

    Initialize program-work-fields
        Program-flags.

    Perform until loop-done
        Display " "
        Display "Enter a name or Q to quit:"
        Accept input-name
        If function upper-case (input-name) = "Q"
            Set loop-done to true
        Else
            Move spaces to output-name
            Move "Hello, " to output-name (1:7)
            Move input-name to output-name (8:30)
            Display output-name
        End-if
    End-perform.


    Goback.
```

3. When you have finished entering the source code, save the file by selecting **File-Save** from the menu bar. Ensure that the **Directory** list box shows HELLOAPP as the selected directory. Enter HELLOAPP.CBL in the **File Name** field and click **OK**. Then close the COBOL Editor by selecting **File** → **Exit** from the editor menu bar.
4. Press **F5** to refresh the project window. The **HELLOAPP.CBL** file should appear.

You are now ready to build the application.

Building the Application

When you build your application, the target file that you specify is created. For the Hello Application, you have a single COBOL source object, which you build into a running COBOL program.

1. From the tool bar, click on , the Build Normal icon.

Building the Application

The target, an executable file titled HELLOAPP.EXE, is created from the COBOL source file. A monitor window opens titled *Editor - Project Monitor - Hello Application* and displays the progress of the build.



Figure 4. Hello Application - Project Monitor Window

If errors are detected during the build step, the monitor displays the return code and compiler error messages. Scroll up to the message lines that include the drive, path names, and source file name (HELLOAPP.CBL) as well as the error message text. Double-click on a message line.

The COBOL Editor displays, showing the line in the HELLOAPP.CBL source file where the error occurred. Correct the error and save the file. Rebuild the file and check the results in the monitor window.

For multiple errors messages, you can select **Actions** → **Select all messages**, and then **Actions** → **Process selected messages** to display the source files with all the messages.


Note: Some linker error messages also contain drive, path, and file names. You can only double-click on compiler error messages to fix errors in the source program.

2. Close the **Editor** window. New files appear in the **WorkFrame V3.5 Project - Hello Application** window,

You are now ready to run the Hello Application.

Running the Application

Running the Application

You can run your application from your Hello Application project. From the tool bar, click on , the Run icon.

A workstation window displays and runs your program, prompting you for a name.

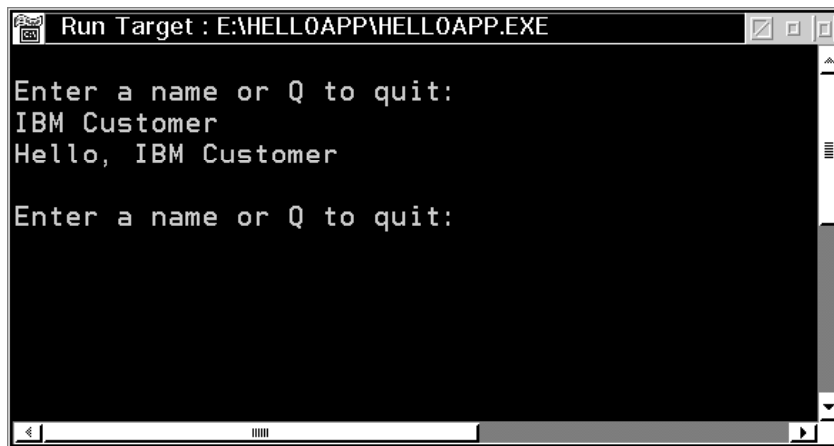


Figure 5. Hello Application. The Hello Application in action....

Creating a Visual Builder Application

Creating a Simple Visual Builder Application

This chapter takes you through an example of how you use Visual Builder to develop an application for creating and maintaining a simple to-do list. Figure 6 shows what the application looks like when it is finished.

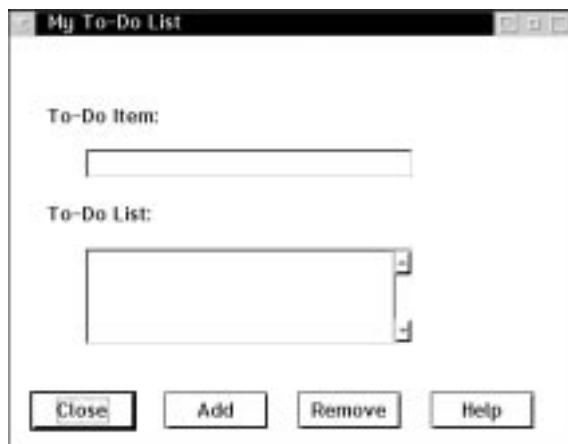


Figure 6. Finished To-Do List Application

A complete TODOLIST application is provided in the samples folder. The TODOLIST sample includes additional connections, which are not included in the To-Do List application that you'll create in this chapter. It also includes the date function, which is explained in "Expanding the Simple Visual Builder Application" on page 45. You can use the TODOLIST sample to get an idea of how your To-Do List application will run.

Creating the To-Do List Application

Creating the To-Do List application consists of the following steps:

1. Creating a visual project
2. Starting the Visual Builder
3. Creating a new visual part
4. Placing parts in the application window
5. Resizing and aligning the parts
6. Connecting the parts
7. Generating the COBOL code for your application
8. Building the application
9. Running the application
10. Exiting the Composition Editor and Visual Builder

Creating a Visual Builder Application

Creating a Visual Project

For this tutorial, we'll organize the files in a Visual Builder project. To create a project, follow the steps in "Creating Your First VisualAge COBOL Application" on page 23, except:

- On the **Project** page select **COBOL Visual Builder Project**.
- On the **Location** page specify:
 - TODOLIST as the Project title.
 - D:\VISUAL\TODOLIST as the Source file directory.
 - TODOLIST as the Project file name.
- On the **Target** page accept the default **TODOLIST** as the target file name.
- Click on **Done**. A message displays indicating your project was created successfully. Click **OK**. The WorkFrame V3.5 Project - TODOLIST window opens.

Starting Visual Builder

The TODOLIST project contains the following files:

READ.ME

A file explaining the basics for creating a visual project

TODOLIST.VCB

The TODOLIST visual part.

VBHELP.IWO/VBHELP.IWP

Subproject files, which contain a skeleton help source file (VBHELP.IPF). Project Smarts creates the HELPDIR subdirectory under VISUAL\TODOLIST for the help source file.

To start the Visual Builder, double-click on the TODOLIST.VCB icon in the TODOLIST project. The Visual Builder window and the Composition Editor window open.

Creating a Visual Builder Application

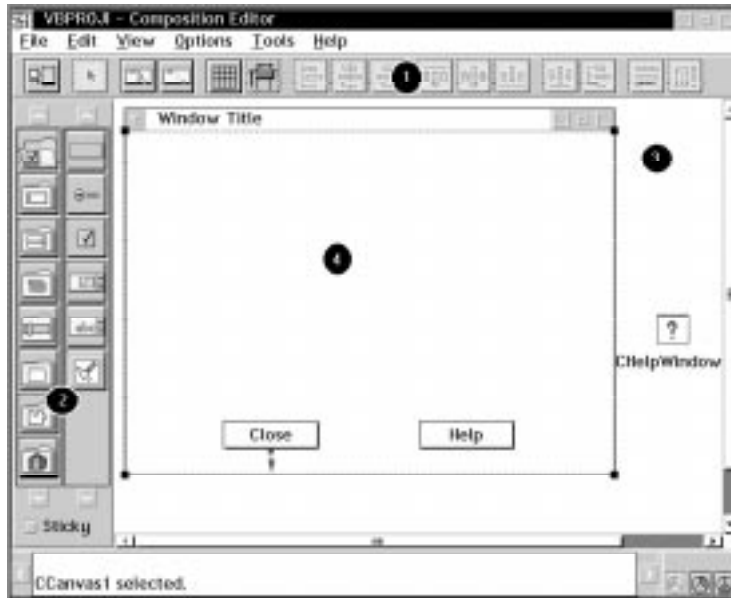


Figure 7. Visual Builder's Composition Editor

Areas to note in the Composition Editor are:

1 Tool Bar

Enables you to invoke frequently used actions by clicking on the icon.

2 Parts Palette

The left column of the part palette contains the categories of parts (such as buttons and list boxes). The right column contains the parts available for the category selected.

Each category contains a collection of similar or related parts. You drop parts on the free-form surface or on top of other parts, like a canvas (which is component of your visual part).

3 Free-Form Surface

The area on which you place parts, such as a canvas or nonvisual parts.

4 Visual Part

A frame window with a canvas and two push buttons. As you create your applications, you'll add more parts (such as push buttons and static text) to this canvas.

Close Push Button—When running the application, clicking the `Close` push button closes the window. It has no affect in the Composition Editor. The solid green arrow pointing to the To-Do List frame is the completed connection that closes the To-Do List application when a user presses the `Close` push button. You learn how to connect parts later in this tutorial.

Help Push Button—When running the application, pressing the `Help` push button causes a help window to display for the To-Do List application.

Creating a Visual Builder Application

The help file (VBHELP.HLP) is created from the supplied help source file (VBHELP.IPF) when you build the To-Do List application.

The settings of the Help push button determine which help panel displays based on a specific resource ID. For example, double-click on the Help push button to open the settings notebook. On the **Control** tab, **Help panel id** specifies 100 as the resource ID for the Help push button. If you edit VBHelp.IPF, you see that res=100 is included on the **Main Window Help** panel.

The TODOLIST part also contains a subpart titled CHelpWindow2, which is outside of the TODOLIST frame window on the *free-form surface*. This subpart represents the actual help window and it's where you define the help libraries. (If you double-click on CHelpWindow2, the setting notebook opens. The default help library is VBHELP.HLP).

Before you place any parts in the frame window, first edit its title.

Changing the title of the To-Do List frame window

The new visual part that you just created is a CFrameWindow part, titled *Window Title*. This will be the To-Do List application title bar. Change the title of this window by doing the following:

1. Position the cursor over the title bar.
2. Press the ALT key and click with mouse button 1.
3. Enter a new title (such as My To-Do List)
4. Click anywhere on the To-Do List canvas part.

You are now ready to place parts (such as push buttons and static text) in the application window.

Placing Parts in the Application Window


To select parts to place in the application window, you:

- Select the parts category from the left-hand column of the parts palette.
- Select the part from the right-hand column of the parts palette.
- Move the mouse pointer to the free-form surface and click on the spot where you want to place the part.


Note: You don't need to drag the part from the parts palette to the free-form surface. You just point and click.

Placing a static text part in the window

The To-Do List application needs two static text fields. Follow these steps to place the first static text part in the To-Do List application window:


1. Select  (the **Data entry** category) from the icons on the left-hand side of the parts palette.

Creating a Visual Builder Application

2. Select  (the **COBOL Text StaticText** icon) from the icons on the right-hand side of the parts palette. When you move the mouse pointer over the free-form surface, you see that it has changed to crosshairs. This means the mouse pointer is loaded with the COBOL Text StaticText part.
3. Place the crosshairs in the upper-left corner of the To-Do List application window's client area and click mouse button 1. A static text part is placed in the window.
4. Change the name of the static text part to: To-Do Item:
Use the same method for changing text that you learned previously when you changed the title of the To-Do List frame window.

Placing an entry field in the window

The To-Do List application needs an entry field so the user can type in new To-Do items.

1. Select  (the **COBOL Text Entry Field** icon) from the icons on the right-hand side of the parts palette. (The COBOL Text Entry Field icon is also contained on the Data entry category set of icons.)
2. Place the crosshairs beneath the first static text, indented a few spaces to the right, and click mouse button 1. The entry field is placed beneath the static text.



Placing another static text in the window

Follow these steps to position and modify the second static text:

1. Place the second static text in the To-Do List frame window under the entry field, but aligned with the first static text. Use the same method for placing a static text that you learned previously when you placed the first static text in the To-Do List frame window.
2. Change the name of the static text to: To-Do List: Use the same method for changing text that you learned when you changed the title of the To-Do List Application frame window.

Placing a list box in the window



Because the to-do list will consist of a list of text strings, you want to store that list in a COBOL ListBox. To place a list box in the To-Do List frame window:

1. Select  (the **Lists category**) from the icons on the left-hand side of the parts palette.
2. Select  (the **COBOL ListBox** icon) from the icons that Visual Builder displays on the right-hand side of the parts palette.
3. Place the crosshairs below the second static text, aligned with the entry field, and click mouse button 1. The list box is placed beneath the second static text.

Creating a Visual Builder Application

Placing the push buttons in the window

The To-Do List application needs two additional push buttons, one for adding items to the list and one for removing items from the list. To place two push buttons in the frame window:

1. Select  (the **Buttons category**) from the icons on the left-hand side of the parts palette.
2. Select  (the **COBOL PushButton** icon) from the icons on the right-hand side of the parts palette.
3. Place the crosshairs between the **Help** and **Close** push buttons and click mouse button 1. The first push button is placed in the window.
4. Change the name of the new push button to Remove. Also, to help identify subparts once you begin connecting parts, rename the subpart to removepb. Double-click on the Remove push button to display the Settings notebook. On the **General** page, change the Subpart name to removepb and change the Label to Remove. Click on **OK**.
5. Select the **COBOL PushButton** icon again.
6. Place the crosshairs to the left of the **Help** push button and click mouse button 1. The second push button is placed in the window.
7. Change the name of the push button to Add and the Subpart name to addpb.

Note: If you place the push buttons slightly out of alignment with the existing push buttons, that's OK. We'll align and size the push buttons in the next step.

Resizing and Aligning the Parts

Now that you have placed all of the parts you need in the To-Do List window, you can resize and align them. When you have finished, your frame window should look like Figure 6 on page 31.

Dragging and dropping parts in the frame window

Before you align the parts, you might want to drag and drop some of them to put them in closer proximity to each other. For example, you might want the static texts to be closer to the parts that they label. Follow these steps to drag and drop the parts in the application window:

Note: The following instructions are written for dragging and dropping multiple parts simultaneously. If you just want to drag and drop one part at a time, select the part and continue with step 3.

1. Select the first part you want to drag.
2. While holding down mouse button 1, move the mouse pointer to the second part you want to move. The selection handles on the first part become outlined, and black selection handles appear on the four corners of the second part. This means both parts are selected, but the second part is the *anchor* part. Therefore, any

Creating a Visual Builder Application

sizing actions performed using the tool bar cause the first part to match the anchor part for the sizing action selected.

3. Move the mouse pointer over one of the parts that you selected to drag.
4. Press and hold mouse button 2 and move the mouse cursor. Visual Builder displays an outline of the parts that you are dragging.
5. Move the outline to the place where you want to drop the parts and release the mouse button. The parts are moved to their new location.

Resizing the frame window (optional)


At this point, the parts in the frame window might be closer to the left window border than to the right window border (depending on where you've placed the parts). Follow these steps to resize the frame window:

1. Select the application window by clicking mouse button 1 on the title bar.
2. Move the mouse pointer over the selection handle on the lower-right corner of the frame window.
3. Press and hold mouse button 1.
4. Resize the frame window by dragging the mouse pointer towards the left until the right border of the frame window is approximately the same distance from the entry field and list box as the left border is.

To size the frame window in only one direction, either horizontally or vertically, hold down the Shift key while dragging the mouse pointer.

Matching the width of the list box to the width of the entry field


Follow these steps to match the width of the list box to the width of the entry field:

1. Using the multiple selection technique you learned in *Dragging and dropping parts in the frame window*, select the list box and then the entry field, making the entry field the anchor part. (The last part you select becomes the anchor part.)
2. Select  (the Match Width tool) from the icons on the tool bar, located beneath the menu bar. The width of the list box changes to match that of the entry field.

Sizing and aligning the push buttons




After changing the text for each push button, you'll notice the push buttons have different widths and heights, and aren't aligned.

Using the techniques you learned in the preceding steps, select all four push buttons, using the Remove push button as the anchor part.

- Match the width of the push buttons. Select  (the Match Width tool).



Creating a Visual Builder Application

While all four push buttons are still selected, let's match their height and align them in the frame window.

- Match the height of the push buttons. Select  (the Match Height tool).
- Match the top edges of the push buttons. Select  (the Align Top tool).
- Space the push buttons evenly across the frame window. Select  (the Distribute Horizontally tool).


Centering the entry field and list box within the frame window

The entry field and list box need to be centered within the application window. Follow these steps to center them:

1. Select the entry field.
2. Select  (the Distribute Horizontally tool) from the icons on the tool bar. Visual Builder centers the entry field between the left and right borders of the frame window.
3. Select the list box and then the entry field, making the entry field the anchor part. Use the multiple part selection technique you learned previously.
4. Select  (the Align Left tool) from the icons on the tool bar. The list box is aligned with the entry field.

Aligning the static text so their left edges are even

The two static text parts need to be aligned evenly. Follow these steps to align them:

1. Select both static text parts (making either one the anchor part). Use the multiple part selection technique you learned previously.
2. Select  (the Align Left tool) from the icons on the tool bar. The first static text part is aligned evenly with the second static text part.

Your To-Do List window should now look like the one shown below.

Creating a Visual Builder Application

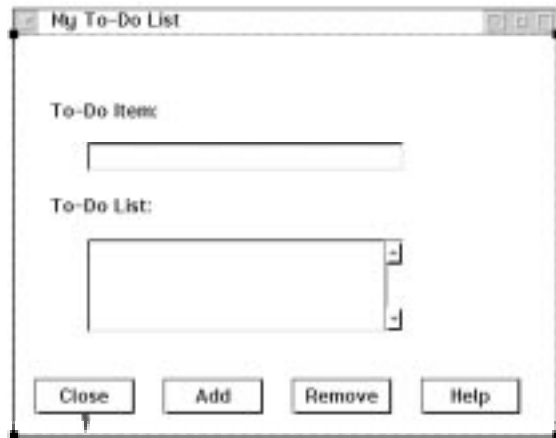


Figure 8. The TODOLIST with all its parts.

Connecting the Parts

The next step in developing your To-Do List application is to code the logic behind the parts. The COBOL program behind your To-Do List application follows the *event-driven programming* model.

An event-driven program runs segments of logic in response to events. It has entry and exit points that correspond to many events that can happen with respect to the program. When you run an event-driven program, all logic in your application waits for certain events to happen, such as when a user clicks the **Add** push button. Once a selected even occurs, only the logic for that event is performed, then the application waits for the next event. (With *procedural programming*, the program has one entry point of entry and exit. The program follows each step in the program logic sequentially until it reaches the end of the logic.)

The Visual Builder helps you code the event logic. Each part you place on your To-Do List canvas has a connections menu, from which you select the event to which your logic will respond.

When connecting parts, you follow these basic steps:

1. Select the part and right mouse click to display the connections menu
2. Select the event that triggers an action on that part
3. Select the part affected when that event occurs
4. Select the action to occur

For the To-Do List, you need to connect the push buttons to the list box and entry field. You'll add connections that will move the text that a user enters in the To-Do Item entry field to the end of the To-Do List list box when the Add push button is pressed. You'll also add connections to remove one item from the To-Do List list box when the Remove push button is pressed.

Creating a Visual Builder Application

Connecting the Add push button to the list box

The connection between the **Add** push button and the list box provides the information your application needs to add items to the list box.

1. With the mouse pointer over the **Add** push button, click mouse button 2. A pop-up menu displays.
2. Select **Connect**. A cascaded menu of the **Add** push button displays.
3. Select **press**. Selecting *press* means you want something to happen whenever a user presses this push button. The mouse pointer changes to look like a spider, indicating that it is ready for you to select another part.
4. Move the mouse pointer to the list box and click mouse button 1. A pop-up menu displays.
5. Select **addItemEnd**. Selecting *addItemEnd* means you want new items to be added to the end of the to-do list whenever a user presses the **Add** push button. The connection is shown in Figure 9.

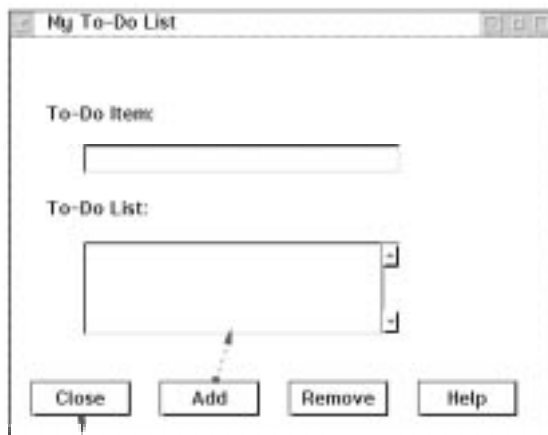


Figure 9. The connection between the Add push button and the list box

The line connecting the **Add** push button to the list box is dark green. It points from the push button to the list box, showing that the event that occurs when the push button is pressed causes the list box to perform an action.

The dashed line means the connection is incomplete. The connection is supposed to add something to the list box when the **Add** push button is clicked, but you have not yet supplied what needs to be added. You'll do that in the next step.

6. Move the mouse pointer to the dashed connection line between the **Add** push button and the list box. Click mouse button 2.
7. Select **Connect** → **Item**.
8. Move the mouse pointer over the entry field, and click mouse button 1.

Creating a Visual Builder Application

9. Select **contents**. Selecting *contents* means you want to move the text that a user enters in the entry field to the *Item* parameter of *addItemEnd*. This text string is added to the end of the to-do list whenever *addItemEnd* is called, which occurs whenever the **Add** push button is clicked. The completed connection is shown in Figure 10 on page 41.

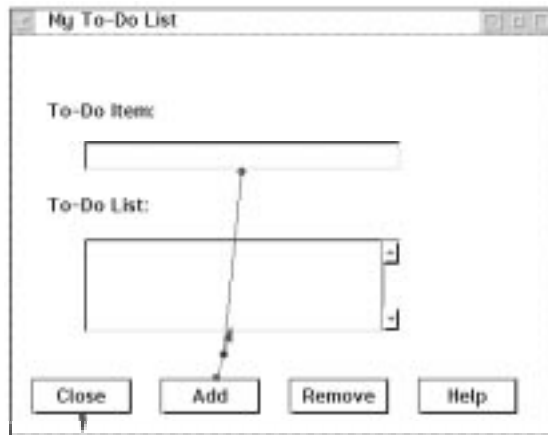


Figure 10. The completed connection for the Add push button

The completed connections indicate that the entry field contains the data to be added to the end of the list box when the user presses the Add push button.

The line connecting the entry field to the connection between the **Add** push button and the list box is violet. This is the parameter connection. The open circle end indicates that the *contents* attribute of the entry field is the source for the parameter to the *addItemEnd* action.

The solid circle touches the connection line between the Add push button and list box. The *Item* parameter of the *addItemEnd* action is the target of the connection. When the *Item* parameter needs a value, which occurs when a user presses on the **Add** button, the connection invokes the get member function of the entry field's *contents* attribute. The value of that attribute (the text in the entry field) is returned and moved to the *Item* parameter and the *addItemEnd* action puts the text string in the list box.

Notice that both of the connection lines are solid. This means the connection between the **Add** push button and the list box now has the information it needs to perform its function, so the connection is complete.

Connecting the Remove push button to the list box

The connection between the **Remove** push button and the list box programs your application to remove items from the list box.

1. With the mouse pointer over the **Remove** push button, click mouse button 2.
2. Select **Connect** → **Press**.

Creating a Visual Builder Application

3. Move the mouse pointer to the list box and click mouse button 1.

4. Select **removeOne**.

Selecting *removeOne* means you want your application to remove one item in the to-do list whenever a user presses the **Remove** push button. Once again, the connection is incomplete.

5. Move the mouse pointer to the connection between the **Remove** push button and the list box and click mouse button 2.

6. Select **Connect** → **itemIndex**.

7. Move the mouse pointer over the list box and click mouse button 1.

8. Select **firstSelected**.

Selecting *firstSelected* means you want to move the index of the first selected item in the list box to the *ItemIndex* parameter of the *removeOne* action. The *removeOne* action uses this index to determine which item to remove whenever the **Remove** push button is clicked.

Making this connection completes your application. It should now look like this:

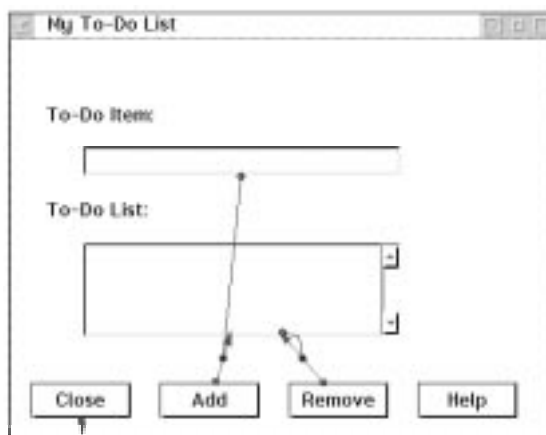


Figure 11. The completed To-Do List application


Now that you have made all of the connections, the next step is to generate your COBOL source code.

Creating a Visual Builder Application

Generating the COBOL Code for your Application

Before you can build your application, you must first generate source code and build files.

Generating the source code for your visual part

To generate the COBOL source code for your visual part, from the toolbar, select  (Generate Part Source).

Important: Visual Builder generates several files in the working directory (VISUAL\TODOLIST). Do not edit or move these files; they are used by Visual Builder and your project.


Generating the build files

To generate the build file, select **File** → **Save and generate** → **Build files**.

You have now generated the COBOL code for your application. The next step is to build the application.

Building the Application


Building your application consists of compiling and linking it. You'll build the application from your Visual Builder project that you created in the first step:

1. In your project, press F5 to refresh it.
2. From the toolbar, click on , the Build Normal icon.

Subprojects are built first, in this case VBHELP, which generates its make files in the HELPDIR subdirectory under VISUAL\TODOLIST. Then, the main project (TODOLIST) is built. The build action generates an executable, binary resource, several object files and some additional temporary files.

Now you can run your application.

Running the Application

To run your application from your Visual Builder project, from the toolbar, click on , the Run icon.

Once your application is running, experiment with it to make sure it works as designed. You'll notice that when you input text into the To-Do Item entry field and press Add, the text is added to the To-Do List entry box. However, the text remains in the entry field. You'll learn how to clear the entry field, as well as enable and disable the Add push button in "Expanding the Simple Visual Builder Application" on page 45.

Creating a Visual Builder Application

Exiting the Composition Editor and Visual Builder

You've created a simple Visual Builder application! If you want to expand on the To-Do List application, continue to the next tutorial. Do not close the Visual Builder or Composition Editor.

If you'd like to continue at another time, exit the Visual Builder and Composition Editor now. To exit, select **File** → **Exit**.

Note: You must exit Visual Builder before you can shut down the operating system. Otherwise, the operating system might not shut down completely, requiring you to turn the computer off.

Expanding the Simple Visual Builder Application

This chapter explains how to expand the To-Do List application, which you created in the previous section, to include user-supplied code contained in a nonvisual part. You will create a nonvisual part contained in the TODOLIST part. You will use the Part Interface Editor and System Interface Editor to generate feature code (skeleton COBOL code) in which you'll add code (using Code Assistant) that will generate the current date in the To-Do List Application.

You'll also add connections that will make the **Add** push button available (or unavailable), based on the content in the To-Do Item entry field. Figure 12 shows what the application looks like when it is finished.

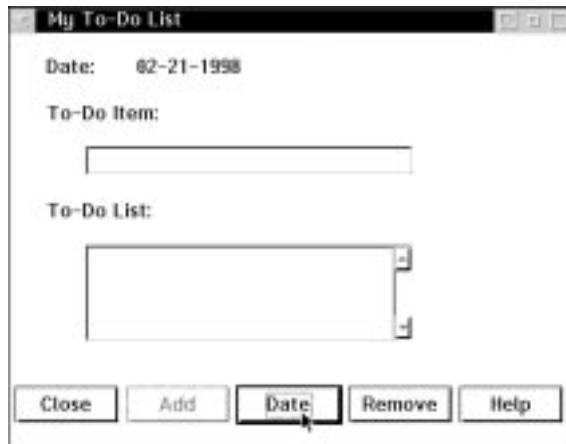


Figure 12. Expanding the To-Do List Application. Using the Date push button to display the date.

Expanding the To-Do List Application

Expanding the To-Do List application consists of the following steps:

1. Opening the To-Do List part
2. Enabling the Add push button
3. Adding and aligning new parts
4. Resizing and aligning the parts
5. Creating a nonvisual part
6. Defining your system interface
7. Generating your code
8. Updating feature source
9. Connecting the parts
10. Generating the COBOL code
11. Building the application
12. Running the application
13. Exiting the Composition Editor and Visual Builder

Expanding the TODOLIST

Opening the To-Do List Part

If you're continuing with the TODOLIST application from "Creating a Simple Visual Builder Application" on page 31, skip this step. If not, open the TODOLIST part:

1. Double-click on the COBOL Projects folder on your desktop. The TODOLIST - Icon View window opens. This is your TODOLIST project.
2. Double-click on the TODOLIST.VCB icon in the project view. The Visual Builder and the Composition Editor with your TODOLIST part open.

Now you're ready to continue with the To-Do List Application.

Enabling the Add Push Button

To fine tune the TODOLIST application, you'll probably want to make the Add push button unavailable, depending on whether text is entered in the To-Do Item entry field. Let's also clear the entry field of the To-Do Item after pressing the Add push button. You'll need to do the following:

1. Make the Add push button unavailable (since, initially, the text entry field is blank).
2. Make the ADD push button available once text is entered into the entry field.
3. Clear the entry field once the Add push button is clicked.
4. Make the Add push button unavailable once the Add push button is clicked.

Notice that steps 3 and 4 are driven by the same event (clicking the Add push button). When you have multiple actions occurring as a result of a single event, you must code the connections in the same sequence as the actions occur.

Making the Add push button unavailable

To disable the Add push button, you need to do the following:

1. Double-click on the Add push button.
2. Click on the **Control** notebook tab.
3. On the **Control** notebook page, click on the **enable** check box to remove the check mark. (The push button is available (checked) by default.)
4. Click on **OK**.

Making the Add push button available when text is entered in the entry field

To enable the Add push button once you enter an item in the To-Do Item entry field, you need to add a connection between the entry field and the Add push button. Do the following:

1. With the mouse pointer over the entry field, click mouse button 2.
2. Select **Connect** → **More....**
3. Under the event column, select **change**.
4. Select **OK**
5. Move the mouse pointer to the Add push button and click mouse button 1.

Expanding the TODOLIST

6. Select **enable**. Selecting the *enable* action means that when something is changed in the entry field (for example, inputting text), the *Add* push button is available.

Removing the entry field text

After you enter a To-Do Item in the entry field and press the Add push button, you should clear the text from the entry field. Do the following:

1. With the mouse pointer over the Add push button, click mouse button 2.
2. Select **Connect** → **press**.
3. Move the mouse pointer to the entry field and click mouse button 1.
4. Select **Contents**. A dashed line shows the connection between the Add push button and the entry field.
5. Double-click on the dashed connection line. The *Event-to-action connection - setting* dialog displays.
6. Click on the **Set parameters** button. The *Parameters Data Settings* notebook displays.
7. In the **Contents** field enter a blank space. (This will “empty” the entry field when the Add push button is pressed.)
8. Click on **OK** to close the Parameter Data Settings notebook.
9. Click on **OK** to close Event-to-action dialog.

Making the Add push button unavailable

Once again you need to disable the Add push button, after adding the To-Do Item to the To-Do List. Do the following:

1. With the mouse pointer over the **Add** push button, click mouse button 2.
2. Select **Connect** → **press**.
3. Move the mouse pointer to the Add push button and click mouse button 1.
4. Select **Disable**.

Now, the Add push button will not be available after clicking on it.

The connections are shown in Figure 13 on page 48.

Expanding the TODOLIST

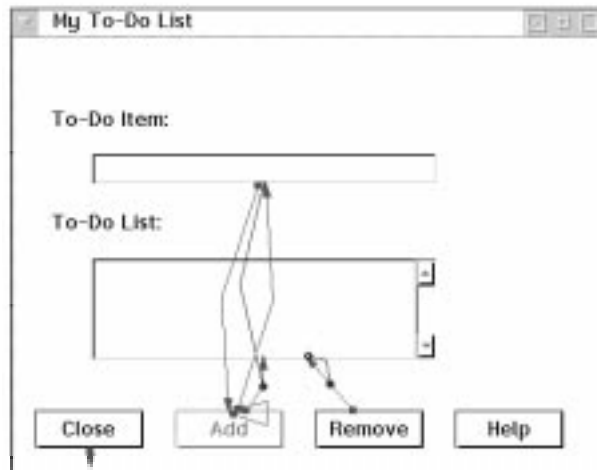


Figure 13. The connections so far...

Note: We changed the shape of the connections between the Add push button and entry field to make it easier for you to see. You can do this by selecting the connection and dragging the middle selection handles.

Adding and Aligning New Parts

To display the current date, you will need to add the following parts in the frame window:

- A push button, labeled *Date*
- Static Text for the *Date* label.
- Static Text where the application will display the date

In the To-Do List part, add the following in the frame window:

1. Add a static text field to the top of the window and change the text to **Date**.
2. Add another static text field to the right of the Date field. Double-click on the field to display the settings notebook. On the **General** page, remove *CStaticText4* from the *Text* field and change the *Limit* field value to *16*. (The application will display the date into this blank static text field.)

Click on **OK**. Selection handles show the location of the blank static text field. (You might want to move the blank static text field closer to the Date static text.)

3. Align the two text fields using the Align Top icon.
4. Align the Date static text field with the To-Do Item static text field.
5. Add a push button between the Add and Remove push buttons and change the text to **Date**. You'll probably need to move the existing push buttons to make room for the Date push button.

Expanding the TODOLIST

6. Select all five buttons, using Remove as the anchor part. Using the toolbar icons, match their height and width, align them, and then distribute them horizontally across the frame window.

When you have finished, your frame window should look like this:

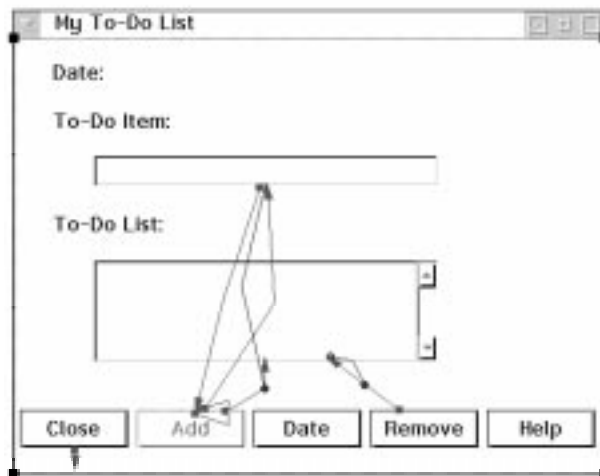


Figure 14. Adding the Date button and static text

You've added and aligned the new parts. Before you can connect them, you need to create a nonvisual part that will define an action to get the date.

Creating a Nonvisual Part

Now, you need to add data processing to your application. To perform data processing, you should use a nonvisual part, in which you'll add code to generate the system date. After creating a nonvisual part and updating code generated by the Visual Builder, we'll connect the nonvisual part to the main TODOLIST visual part.

To create a nonvisual part do the following:

1. From the main Visual Builder window (Visual Builder - TODOLIST), select **Part** → **New** from file menu.
2. In the **Part - New** window, enter **theDate** as the class name.
3. Click on the drop-down arrow next to **Part type** and select **Nonvisual part**.
4. Click on **Open**. The Part Interface Editor opens for the nonvisual part, *theDate*.

Expanding the TODOLIST

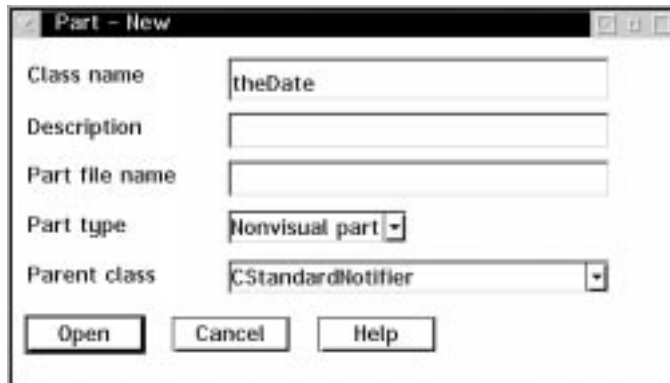


Figure 15. The Part-New Window. Creating a nonvisual part using the Part-New window

When working with a nonvisual part, the Part Interface Editor is the default editor.

Defining Features

To define a feature (an attribute, action, or event) for your part, you use the Visual Builder's Part Interface Editor. For additional details on the Part Interface Editor, see the *Visual Builder User's Guide*.

Attribute Page

On the Part Interface Editor **Attribute Page** we'll define a data item that will have a set and get method. Do the following:

1. Enter **Today** in the Attribute Name field.
2. Click on the arrow next to the Attribute Type field. Scroll down and click on **VarLengthString**, which is the data type that the static text field uses and that's where we'll connect the date data.
3. Click on **Defaults**. The Visual Builder displays the default set and get methods, which are the full method definition used in the generated feature code. The default Event identification is also displayed.
4. Click on **Add** to add Today as an attribute of the nonvisual part, theDate.

Visual Builder creates the **getToday** and **setToday** methods.

Action Page

On the **Action Page** we'll define a method that will determine the date from the system, format it, and set the value of the attribute, Today. Do the following:

1. Click on the **Action** notebook tab.
2. Enter **setDate** in the **Action name** field.
3. Click on **Defaults**.
4. Click on **Add**.

Expanding the TODOLIST

Preferred Page

The Preferred page allows you to specify which features are displayed on the cascaded menu when making connections.


1. Under **Actions**, select **setDate** and click on **Add**. The setDate action is added to the Preferred Features list.
2. Under **Attributes**, select **Today** and click on **Add**. The Today attribute is added to the Preferred Features list.

Now you need to define the files that will contain the code for this action (and other features) and will be included when you build the TODOLIST application.

Defining Your System Interface

Feature source is code that Visual Builder generates based on specifications in the Part Interface Editor and System Interface Editor. The first step in generating feature source is to define the file names for the feature code using the Visual Builder's System Interface Editor.

Defining users files

To define user files, click on  (the System Interface Editor icon) at the bottom right corner of the Part Interface Editor. The System Interface Editor displays.

Input the following under *User files included in generation*:

- **theDate.cpv** for the User declaration file (.cpv)
- **theDate.cbv** for the User code file (.cbv)

You'll notice the *Part file specification* contains the path location of the theDate part. And, the *COBOL code file (.cbl)* defaults to match the part name (theDate).

Generating Your Code

Now you need to generate the code for your nonvisual part. You need to generate both feature source, which you'll update with your own code, and part source.

Generating the feature source

To generate the feature source, from the System Interface Editor:

1. Select **File** → **Save and generate** → **Feature source**. The **theDate - Generate feature source code** window opens.
2. Select **Generate All**. An informational message opens saying the feature code generation is complete. Click on **OK**.

Any subsequent generations produce feature code that is appended to the selected features. Thus, when generating feature code for other methods you define, use *Generate selected* to avoid appending duplicate feature code to existing files.

Expanding the TODOLIST

3. Visual Builder generates the following files:

theDate.cbv	The COBOL feature code source file
theDate.cpv	The COBOL copy file

Generating the part source

To generate part source, select **File** → **Save and generate** → **Part source**.

Updating Feature Source

When you generated the feature source in the previous step, Visual Builder created a skeleton COBOL file for the theDate part based on the information you included in the Part Interface Editor and System Interface Editor. To generate the system date, you need to add some additional code to the LOCAL-STORAGE section and PROCEDURE DIVISION of the setDate method in the THEDATE.CBV file. You can use the Code Assistant tool to edit the file.

1. From the main Visual Builder window (Visual Builder - TODOLIST), select **theDate** in the **Nonvisual Parts** list.
2. Select **Part** → **Code Assistant**. The Code Assistant window opens.
3. Click on **setDate** in the upper part of the Code Assistant window. The method setDate appears in the code panel of the window.
4. Add the following code to the LOCAL-STORAGE SECTION of the setDate method:

```
01 tempdate    PIC X(8).
01 sep         PIC X  VALUE "-".
01 temp2       PIC X(10).
01 Tempday.
    03 Tempday-Length      PIC 9(9) COMP-5.
    03 Tempday-String.
        05 Tempday-Chars  PIC X
            OCCURS 1 TO 255 TIMES
            DEPENDING ON Tempday-Length.
```

Figure 16 on page 53 shows what the setDate method will look like before entering your code.

Expanding the TODOLIST



Figure 16. Using Code Assistant to update feature code

5. Scroll down a bit and add the following code to the PROCEDURE DIVISION of the setDate method:

Move function current-date(1:8) to tempdate.
String tempdate(5:2) sep tempdate(7:2) sep tempdate(1:4)
delimited by spaces into temp2.
Move 10 To Tempday-Length.
Move Temp2 To Tempday-String.
Invoke self "setToday" using Tempday.
6. Save the file and close the Code Assistant window. Now, you're ready to use the part.

Connecting the Parts

Back in the Composition Editor for the **TODOLIST** visual part, you'll need to add the nonvisual part to the TODOLIST free-form surface and connect the Date push button to the blank static text field.

Adding theDate nonvisual part

To add the nonvisual part to the TODOLIST:

1. From the TODOLIST Composition Editor, select **Options** → **Add part**.
2. In the **Add Part** window, click on the arrow beside the **Part Class** field. Select **theDate**.
3. Click on **Add**.

Expanding the TODOLIST

4. Move the mouse pointer anywhere on the free-form surface and click mouse button 1. The nonvisual object *theDate2* is added to the main TODOLIST visual part. Now, when the TODOLIST part is instantiated, it creates the instance *theDate2* of the *theDate* part.

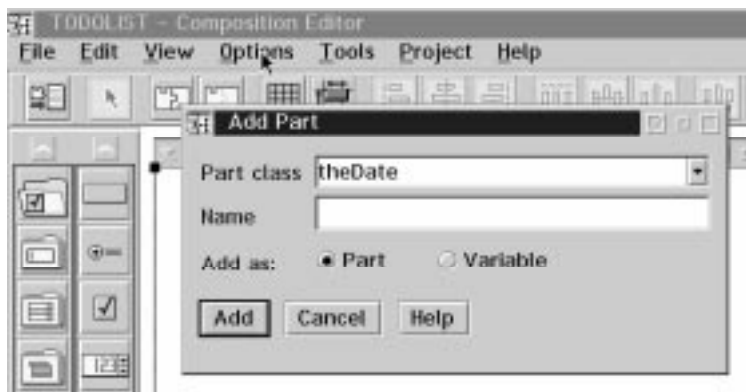


Figure 17. The Add Part Window. Adding the nonvisual part to the main visual part

Connecting the nonvisual part to the main part

You now need to connect the nonvisual part to the main TODOLIST part.

1. Click with mouse button 2 on the free-form surface.
2. Select **Connect**→**Ready**. Move the mouse pointer to the nonvisual object, **theDate2**.
3. Click mouse button 1. A pop-up menu shows the methods you added to the Preferred Features list in the Part Interface Editor.
4. Click on the **setDate** method. A solid line connects the free-form surface with the *theDate2* nonvisual object.

Connecting the ready event of the TODOLIST part with the *setDate* method means that when you run the application, as soon as the visual part (TODOLIST) is constructed and initialized, ready to run, it calls the *setDate* method in the nonvisual part to calculate the current date and store it in the attribute, *Today*.

Connecting the Date push button to the blank static text field

1. Connect the press event of the Date button to the label action of the blank static text field.
2. Then, connect the contents parameter of that connection to the Today attribute of the *theDate* part.

Selecting the *Today* attribute means that you want to pass the system date, as returned by the *getToday* method to the *contents* parameter of the *label* action. The date is displayed in the static text field whenever the *getToday* action is called, which occurs whenever the **Date** push button is clicked.

Expanding the TODOLIST

The completed connection is shown in Figure 18 on page 55. (Your connection lines might be slightly different.)

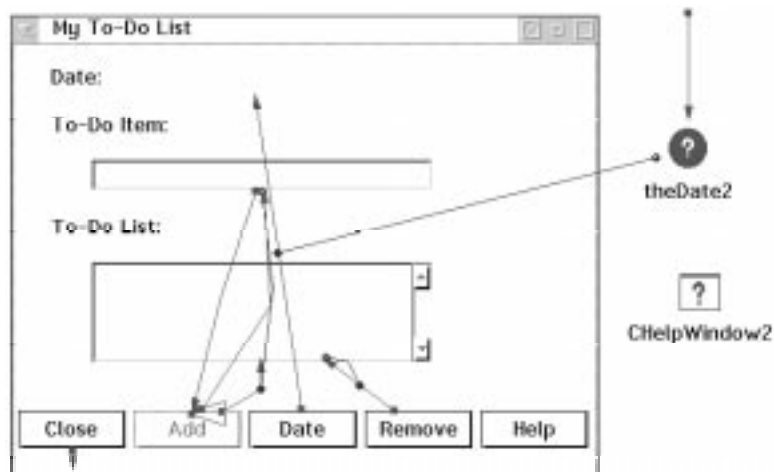


Figure 18. The completed connection for the TODOLIST part

The connection between the Date push button and blank static text field is green. The connection between the theDate2 object and this green line is violet. The hollow circle leads to the theDate object, showing that a feature of the theDate nonvisual part (the attribute Today) is the source of the connection. The solid circle leads to the connection line, indicating that the *contents* parameter is the target of the connection. (You can verify this by clicking on the connections. The features connected appear in the information area of the Composition Editor.)

When you press the Date push button, the application calls the *getToday* method in the nonvisual part and passes the string, returned as a parameter, to set the label of the static text field. When the contents parameter needs a value, it invokes the *getToday* method of *theDate2*, which returns *Today*. The value is passed as a parameter to the label action of the static text field. The label action then displays the current date in the static text field.

Now that you have made the connections, you need to generate the COBOL code.

Expanding the TODOLIST

Generating the COBOL Code for Your Application

As with the “Creating a Simple Visual Builder Application” on page 31, you need to get your application ready to build.

Generating the source code for your visual part

To generate the COBOL source code for your visual part, select **File** → **Save and generate** → **Part source**.

Generating the build files

To generate the make file, select **File** → **Save and generate** → **Build files**.


Building the Application

From the toolbar, click on , the Build Normal icon.

You have now built your application; the next step is to run your application.

Note: Typos or syntax errors in the feature source will cause compilation errors.

Running the Application

From the toolbar, click on , the Run icon.

Once your application is running, experiment with it to make sure it works as designed.

Exiting the Visual Builder

To exit the Visual Builder, close down the following windows (if they are still open):

- Visual Builder (Visual Builder - TODOLIST)
- TODOLIST project (TODOLIST - Icon view)

Redeveloping Legacy COBOL Applications

This chapter explains how to start redeveloping your legacy applications and programs. Redevelopment refers to the effort required to maintain legacy code or to upgrade old code to take advantage of the latest in programming technology. VisualAge COBOL redevelopment support helps you:

- Understand your applications: View the details of your application inventory.
- Convert your programs: Convert your old code to the latest levels of COBOL.
- Structure your programs: Structure unstructured programs.
- Understand your programs: View the flow and logic within a program or set of programs.
- Determine year-2000 impact: Find two-digit years in your program or set of programs.

The remainder of this chapter helps you to get started using the redevelopment support in VisualAge COBOL. For detailed information about this support, see the online help and online tutorials. The online tutorials are available through the Information Notebook in the VisualAge COBOL window and provide step-by-step lessons to help you learn how to use VisualAge COBOL to:

- Understand your applications
- Convert and structure your programs
- Understand your programs

Initiating Redevelopment Actions

You can request redevelopment actions from within the following types of projects:

- COBOL non-GUI project
- MVS project without W/S extensions
- MVS project with W/S extensions

You can also request some redevelopment actions independently of a project. We recommend using COBOL projects and selecting redevelopment actions from within the projects, but if you want to use redevelopment support without using a project, VisualAge COBOL provides stand-alone operation for some redevelopment support. To operate independently of a project:

1. From the VisualAge COBOL window, select the **Tools** icon to display the VisualAge COBOL Tools window.
2. Select the component that you want to run: Application Understanding or Program Conversion and Structuring.

Understanding Your Applications

Understanding Your Applications

Application Understanding helps you understand your inventory of applications. By scanning and analyzing your JCL, it helps you identify the jobs, job steps, procedures, and data that make up your applications. You can see the data sets and other resources used by your applications as well as their characteristics of all your application resources.

To use Application Understanding you must have selected it as an option during VisualAge COBOL installation. Additionally, to scan the JCL, you must have installed the Scan program on the host. The Scan program is distributed with VisualAge COBOL; its installation instructions are in the VisualAge COBOL installation instructions, INSTALL.TXT, on the VisualAge COBOL CD-ROM and are available from the Configuration page of the Information Notebook. Additionally, if you are planning to use VisualAge COBOL's Remote Edit/Compile/Debug during application understanding, you must have installed and configured the Remote support. More information about the use of Remote Edit/Compile/Debug is provided in "Scanning the JCL Libraries" on page 59.

Setup

Before using Application Understanding for the first time or if you have deleted and reinstalled Application Understanding, you must setup your local workstation database for use by Application Understanding. (This database is referred to as the inventory database because it can contain all of the information from your entire inventory of application JCL.)

Having installed IBM DB2 Universal Database (UDB), setup your inventory database as follows:

1. From the VisualAge COBOL window, select the **Tools** icon to display the VisualAge COBOL Tools window.
2. Select the **Application Understanding Migration/Setup** icon to perform the setup automatically.

As depicted in Figure 19 on page 59, using VisualAge COBOL to understand your applications includes:

1. "Scanning the JCL Libraries" on page 59.
2. "Retrieving the JCL Scanned Output" on page 60.
3. "Loading the Database with the JCL Scan Output" on page 61.
4. "Viewing the Contents of Your Inventory Database" on page 61.
5. "Updating Your Inventory Database" on page 64.

Understanding Your Applications

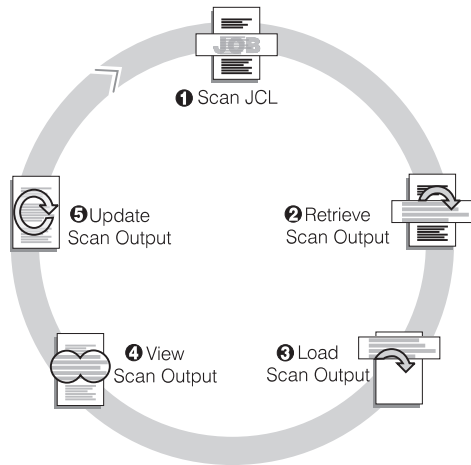


Figure 19. Analyzing your applications

Each of these steps are discussed below.

Scanning the JCL Libraries



To view the data about your applications, you must scan the JCL on the host, download (retrieve) the scan output to your workstation, and load this output into your inventory database. You can initiate all these tasks through VisualAge COBOL on the workstation, or you can do all of the tasks manually.

There are two ways to initiate the host JCL scanner:

- From a project on the workstation using remote processing.
- From the host, using sample JCL provided with VisualAge COBOL.

Preparing for Remote Processing

If you plan to scan your JCL on MVS from your workstation, Remote Edit/COMPILE/Debug needs to be installed and configured. Detailed instructions for installing Remote Edit/Compile/Debug and detailed configuration instructions are provided on the Configuration page of the Information Notebook in the VisualAge COBOL window.

If you are scanning JCL on MVS and have set up the host-workstation connection, you can initiate the scanner as described below. You do *not* have to be logged on to MVS.

1. From a project window, select **Project**→**Open Application Understanding** from the menu bar to display the Application Understanding window.
2. Select **Options**→**Connect to MVS** from the menu bar if you are not already communicating with the host.

Understanding Your Applications

VisualAge COBOL displays messages in a command window. Each time you see the message "Press any key to continue", do so.

Errors?

If you see error messages, the host-workstation setup may have a problem. Check to see whether you have set up the host-workstation connection as described in *Remote Host Development with IBM COBOL: requirements and configuration*.

3. Select **Options**→**JCL Scan** from the menu bar to display the Scan Request window.
4. In the **Application Understanding Scan Request** window, type the necessary information in the entry boxes. This information identifies where to locate the JCL members to scan, where to put the results of the scan, and where to put the log from the scan.
5. Select **Scan JCL**.

Note: The JCL scan request runs on MVS using the sample CLIST supplied with VisualAge COBOL. The TIME and REGION resource allocations must be large enough to accommodate the number of members to be scanned in a single execution of this job (depending on the number of PROCs per job and the size of the jobs, you may need to adjust these values). There is an upper limit of 3000 members (JCL scanned and PROC members called) that can be processed in a single run of the scan.

Retrieving the JCL Scanned Output



After the JCL library has been successfully scanned, you can download the host data sets containing the scan output to your workstation. Doing this allows you to later load the contents into your inventory database.

There are two ways to retrieve the output from the JCL scanner:

- Use remote processing from a project on the workstation.
- Use a suitable host-to-workstation download utility.

Preparing for Remote Processing

If you are downloading from MVS and have not set up the host-workstation connection, you need to set this up as described in *Remote Host Development with IBM COBOL: requirements and configuration*, which is available on the Configuration page of the Information Notebook in the VisualAge COBOL window. If you are downloading from VSE, you need to use a suitable host-workstation download utility.

If you are downloading from MVS and have set up the host-workstation connection, you can download the scanner output as described below:

Understanding Your Applications

1. If you are not already remotely accessing the host through VisualAge COBOL:
 - a. From a project window, select **Project→Open Application Understanding** from the menu bar to display the Application Understanding window.
 - b. Select **Options→Connect to MVS** from the menu bar and follow the on-screen messages to connect to MVS.
2. Select **Options→Retrieve** from the menu bar of the Application Understanding window to display the Retrieve Request window.
3. In the **Application Understanding Retrieve Request** window, enter the location of the scanned output on the host and the destination of that output on the workstation.
4. Select **Find** to select the file to be retrieved from the list of files created. The fields are automatically filled in.
5. Select **Retrieve** to begin downloading the data set.

Loading the Database with the JCL Scan Output



You populate your inventory database by loading it with the JCL scan output. This allows you to use Application Understanding's query and search functions to display and understand your application inventory.

Note: Before you can load the JCL scan output, you must have successfully scanned a JCL library and retrieved the output to your workstation.

To load the database with the JCL scan output:

1. From a project window, select the CDI file you want to load; then select **Selected→Load Database** from the menu bar to display the JCL Load Request window.
2. Enter the name and location of the retrieved file and select **Load** in the **Application Understanding JCL Load Request** window to load the JCL scan output file.

Viewing the Contents of Your Inventory Database



You can view your application inventories to see which JCL objects they use and the characteristics of each object. Application Understanding displays JCL objects such as jobs, DD statements, or programs, and their characteristics. Using these characteristics and displaying different Application Understanding views, you can determine whether an object is unique to an application or is shared.

When you open Application Understanding, it displays any or all of the following:

- **Inventory Groups** – select this icon to view and select the specific applications you want to study. The user creates this object by grouping the jobs in an application.

Understanding Your Applications

- **Job Inventory** – select this icon to view information about the jobs that make up the applications you have selected.
- **Procedure Inventory** – select this icon to view information about the procedures that make up the applications you have selected.
- **DD Statement Inventory** – select this icon to view information about the DD statements in the applications you have selected.
- **Dataset Inventory** – select this icon to view information about the data used in the applications you have selected.
- **Executable Inventory** – select this icon to view information about the executables used in the applications you have selected.
- **Program Unit Inventory** – select this icon to view information about the program units that make up the applications you have selected.
- **Source Module Inventory** – select this icon to view information about the source modules that make up the applications you have selected.
- **Library Inventory** – select this icon to view information about the libraries used in the applications you have selected.
- **Cycles** – select this icon to view how often an application is run, for example, daily, weekly, or monthly. The user specifies the Cycle value for each application.

Because your applications are usually extensive, with much information, Application Understanding lets you filter the information in the inventory database by setting a qualifier for an Inventory window. Then, when you open an Inventory window, Application Understanding displays only the information defined by your filter.

Recommendation

Use filters particularly when viewing DD statements and data sets. These categories can contain a very large number of objects, but the filters enable you display information about selected statements only.

Opening an Inventory Window

Application Understanding lets you select which JCL objects you want to view and then displays an Inventory window containing the information for those objects. For example, you can select to view the Job Inventory, and Application Understanding displays an Inventory window describing the characteristics from the Job statements that meet your filter criteria.

To display an Inventory window:

From the project window, select the CDI file you want to view; then select **Selected→Open** from the menu bar.

Application Understanding displays an Inventory window of the JCL objects from that file.

Understanding Your Applications

You can specify which of JCL objects you want to appear in the Inventory window by setting a qualifier.

Setting a Qualifier

Set qualifiers to selectively filter what information is to be displayed in an Inventory window. This allows you to view only the JCL objects and information that are of interest to you. You can set separate qualifiers for each inventory of objects. Once you set a qualifier, you can view it by selecting the **Details** view in the **Application Understanding** window.

To set a qualifier:

1. From the project window, select the CDI file for which you want to set; then select **Selected→Qualifier** from the menu bar.
2. Type an SQL Qualify statement into the Qualify dialog box

This tells the database what to display in the window.

Basic syntax:

attribute operator attribute value

for example:

JCJ_JCJ_LIB_NM LIKE 'MYLIB%'

Lets you open an inventory window containing the jobs located in the library. whose names start with MYLIB.

Note: The Qualify statement is not case sensitive, but the value may be. For more information about the SQL Qualifier statement, see the online help.

3. Select **OK**.

The qualify statement is saved and used in all sessions until you change it.

Opening a Structure View of JCL Objects

Open a Structure window to better understand structure of a JCL object. For example, to see what jobs are in an inventory group or what job steps are in a job. You may want to select a particular subset of an object and see if it is used anywhere else.

To open a Structure view of a JCL object:

1. From the project window, select the CDI file you want to view; then Select **Selected→Open** from the menu bar.
2. Select an Inventory object.
3. Select **Select→Open** from the menu bar to display the Inventory window.
4. In the Inventory window, select the JCL object whose structure you want to view.
5. Select **Selected→Open→ Structure** from the menu bar of the Inventory window. from the menu bar.

A structure window opens showing the relationship among the JCL objects in the selected inventory.

Understanding Your Applications

6. Select the + (plus) to expand a level or select on the - (minus) to contract a level.

Note: Select an object in the structure to see additional related JCL objects.

Updating Your Inventory Database



As you use your inventory database, you may want to make adjustments to the database by manually adding or deleting JCL objects.

Creating a New JCL Object

You may create a new instance of an object for any of the inventory objects.

To create a new JCL object:

1. From a project window, select the CDI file to which you want to add an object; then select **Selected→Open** from the menu bar.
2. From the **Application Understanding** window, open an Inventory window for the type of JCL object you want to create. For example, if you want to add a procedure, select **Procedure Inventory**.
3. In the Inventory window, select **Edit→Create** from the menu bar.

A definition notebook opens containing attributes that describe the JCL object.

4. Enter the attribute values as needed, then select **OK**.

Note: Attributes marked with an asterisk (*) are called Key attributes and cannot be modified once saved in the inventory database. If any one of these attributes contains an incorrect value, the object must be deleted and re-added to correct the value.

Note: Close then reopen the window to see the new object.

Deleting a JCL Object

You clean up your database by manually deleting unwanted JCL objects from the inventory database. When deleting an object, you are deleting all relationships associated with the deleted JCL object. The JCL object is also deleted from the inventory database.

To delete a JCL object:

1. From a project window, select the CDI file containing the object you want to delete; then select **Selected→Open** from the menu bar.
2. Select the object you want to delete.
3. Select **Edit→Delete** from the menu bar.
4. Select **Yes** to delete the JCL object.

Note: Close then reopen the window to see that the object was deleted.

Reengineering Your Programs

Application Understanding Hints and Tips

Each time you create or delete a JCL object manually in an Application Understanding inventory window, you must close the window and reopen it to refresh it. If you are unable to reopen the window, you may need to close Application Understanding and restart it.

If you receive SQL error message SQL1035N, the inventory database is currently in use. For example, when attempting to drop and recreate the inventory database, you may need to reboot your system.

Reengineering Your Programs

You can reengineer your legacy COBOL programs to take advantage of the latest in COBOL technology and to make them easier to understand, maintain, and update.

As depicted in Figure 20, using VisualAge COBOL to reengineer your programs includes:

1. “Converting Your Programs” on page 66
2. “Analyzing Your Program for Structuring” on page 67
3. “Preparing Your Program to Improve Structuring” on page 68
4. “Structuring Your Program” on page 69
5. “Modularizing Your Program” on page 70
6. “Testing Your Program” on page 71

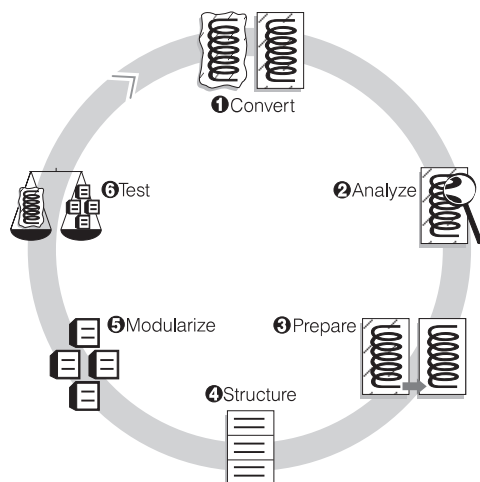
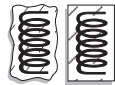


Figure 20. Steps for reengineering COBOL programs

Note that some of these steps are optional, depending on your reengineering scenario.

Viewing the Conversion Log

Converting Your Programs



To use Program Conversion, you must have selected the **Program Conversion and Structuring** option during the installation of VisualAge COBOL.

Note: If you want to generate a COBOL program that runs on VSE, you must first set the environment variable ECFVSE to '1'.

To convert a COBOL program from a lower level to a higher level of COBOL:

1. From a project window, select the program that you want to convert; then select **Selected→Convert Options** to display the conversion options.

Options include the file name of the program to convert, the COBOL language level used in that program, the name and location of the converted file, names and locations of any copy files, indication of whether the program contains CICS commands, and other conversion parameters and formatting options.

Suggestion

Use the same directory for your conversion output as you use for source for your project. This lets you continue to use project facilities to process these new files.

2. Specify the options that apply to your program, and then select **Convert**.

VisualAge COBOL converts your program, storing the converted file in the directory you specified in the convert options and creating a conversion log. These files are stored in your project if you specified the same directory for conversion output as you did for the source of your project.

The report generated as a result of conversion is automatically displayed.

3. To use the same options for subsequent conversions, select the file you want to convert from the project, and then select **Selected→Convert** from the menu bar.

Viewing the Conversion Log

You can open the Program Conversion Log to view information about all programs that have been converted up to the present date.

To open the Program Conversion Log, select **Project→Open Convert Log** from the task bar of the project.

The Program Conversion Log includes information on the following:

Status page

Shows the status of each converted program, including the date each was most recently converted.

Options page

Shows the options used for each converted program.

Analyzing Your Program for Structuring.

File page

Lists a cross reference of which data files are used by which COBOL programs, indicating those that do not need conversion and those that will need to be defined or checked for migration.

Copy Programs (CopyLIB) page

Provides a cross reference of which copy files are used by which COBOL programs; can be sorted by copy names or by program names. This page is only available if you convert a program containing COPY statements.

Calls page

Provides a cross reference of which call statements are used by which COBOL programs; can be sorted by call names or by program names. This page is only available if you convert a program that has CALL statements.

Program Conversion also produces a Convert Results file, which provides a detailed report of the converted COBOL program. This file has the same name as the converted program but an extension of CRS. To view the Results file, select it from the project window, then select **Selected**→**Edit** from the task bar.

Program Conversion and Reserved Words

Because older levels of COBOL did not have the following reserved words which are part of later COBOL standards, Program Conversion does not support them:

AUTOMATIC	METHOD-ID
CLASS-ID	OBJECT
COMP-5	OVERRIDE
COMPUTATIONAL-5	PREVIOUS
END-VOKE	RECURSIVE
INHERITS	REPOSITORY
VOKE	RETURNING
LOCAL-STORAGE	SELF
METAClass	SUPER
METHOD	

If you use these words in your source program, the output program will not execute successfully.

Structuring Unstructured Programs

Use Program Structuring to structure previously unstructured programs. Program Structuring helps you analyze your existing program to determine how to best structure it and then structures it based, in part, on formatting options you specify. (To use Program Structuring, you must have selected the Program Conversion and Structuring option during the installation of VisualAge COBOL.)

Analyzing Your Program for Structuring



To analyze a COBOL program for structuring:

1. From a project, select the COBOL program you want to analyze; then select **Selected→Analyze/Structure** from the menu bar.

VisualAge COBOL opens the Analyze/Structure options notebook.

2. Use the notebook to specify the options you want for structure analysis. You can specify characteristics of the input and the output files, location of any copy files, specification of CICS usage, and other structure and formatting requirements.

Note: Request modularization information if you think you will want to modularize your program. Refer to “Modularizing Your Program” on page 70 for more details on requesting information and modularizing your program.

3. Select **Analyze** to start the analysis.

VisualAge COBOL analyzes your program and stores the information:

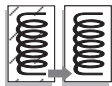
- As the Reengineering Report.
- As Expert Advice.
- In the Complexity log.

4. To use the same options for subsequent structure analysis request, select a COBOL file from the project, then select **Selected→Analyze** on the menu bar.

VisualAge COBOL displays a dialog box allowing you to use your previous options as your defaults or to specify other options. It then analyzes your program.

The reports generated as a result of the analysis are displayed as icon objects in the project. You can view the information in the reports by selecting the report icon in the project window.

Preparing Your Program to Improve Structuring



In about 25% of typical COBOL programs, some manual preparation of the input program before structuring helps you increase the benefits of structuring.

During Program Structuring analysis, VisualAge COBOL generates a Reengineering Report. This report contains a detailed analysis of the PERFORM logic in your input program, a set of program complexity metrics, advice to help modularize the output program, sections describing various properties of the output program, CICS HANDLE analysis for programs containing CICS HANDLE commands, and any messages that you should examine before structuring your program.

VisualAge COBOL also provides Expert Advice that explains how to prepare your program for structuring. This advice is provided for programs with complex performed procedures (Group I) or procedures that have multiple entries and/or exits (Group II).

You can use Expert Advice after you have analyzed or structured a program. To open Expert Advice:

1. Open the Restructuring report by selecting on its icon in the project window.

Structuring Your Program

2. Select **Open as Expert Advice** from **Expert Advice** on the menu bar, or select **Expert Advice** on the **Properties** page.

Expert Advice provide advice for the following Group I and Group II PERFORM statements:

<i>Characteristic</i>	<i>Advice</i>
Multiple exits	based on the type of multiple exits
Multiple entries	based on the type of multiple entries
Perform overlaps	shows range of PERFORM overlaps
End precedes begin	reported if the THRU label lexically precedes the initial label for a procedure
Same final label	range of code that has the same final label
Reinvocation	pointer to a line of code that is potentially recursive
Lower level	shows range of PERFORM procedure that has complex usage or multiple entries/exits

Use the Reengineering Report, along with the Expert Advice, to improve your program's design and decrease its complexity before structuring.

Structuring Your Program



To structure a COBOL program:

1. From a project, select the program you want to structure; then select **Selected→Analyze/Structure Options** to display the structure options.

These options include the input and output names and locations of the program being structured, identification of any copy files, indication of whether the program contains CICS commands, and other structuring parameters and formatting options.

2. Specify the options that apply to your program, and then select **Structure**.

VisualAge COBOL structures your program, storing the new file in the directory you specified in the structure options and updating the Complexity Metrics log.

3. To use the same options for subsequent structure requests, select the file you want to structure from the project, and then select **Selected→Structure** from the menu bar.

VisualAge COBOL displays a dialog box allowing you to use your default options or to specify different options before structuring your program.

Note: Structuring removes GOTO statements and may introduce new variables.

Note: Reserved words not supported during program structuring are the same ones listed in "Program Conversion and Reserved Words" on page 67.

The reports generated after structuring are displayed as objects in the project. You can view the information in the reports by selecting the report icons. These reports include:

- Updated Reengineering Report including Expert Advice.

Modularizing Your Program

- Updated Complexity Metrics Log summarizing complexity metrics for all analyzed and structured programs.
- Cross Reference Browser to browse both the input program and the structured output program simultaneously.
- Structure Chart displaying the flow of control through the newly structured program.

Modularizing Your Program



After you restructure your program, use the modularization advice in the Reengineering Report to find out which procedures represent the best candidates for modularization. Modularization can reduce the size of the program and its overall complexity by breaking your program into several smaller physical units.

To include modularization advice in your Reengineering Report, set the Analyze/Structure options in the Analyze/Structure Options notebook as follows:

1. Select the **Reporting** tab.
2. Make sure the **Perform modularization analysis** box is checked.
3. Click in the **Minimum** setting entry box under **Modularization statement** to set the minimum number of lines a PERFORM should contain to be considered for modularization in the report.
4. Click in the **Maximum** entry box to set the maximum number of lines a PERFORM should contain to be considered for modularization in the report.
5. After setting the options, select **Structure** in the Analyze/Structure options notebook to structure your program and generate the Reengineering Report.

You can then look at the Reengineering Report to see the modularization advice:

1. Select the report icon in the project.
VisualAge COBOL displays the report. This display contains a detailed analysis of your program and instructions on how to use this information.
2. Click on the **Mod Advice** tab to open the modularization advice page.
3. To see modularization statistics, select **Statistics table**.

You can use the modularization advice to manually modularize your program. Modularization advice is not provided for programs containing CICS HANDLE commands.

Understanding Your Programs

Testing Your Program



After reengineering your program, verify that it is equivalent in function to the input program.

You can use tools like IBM VisualAge COBOL, Test for OS/2 or IBM WITT Year2000 to regression test your programs.

Understanding Your Programs

Program Understanding helps you understand the control and data flow of your programs. It analyzes the SYSADATA files produced during compilation and graphically displays the information from this analysis.

To use Program Understanding, you must have selected the **Program Understanding** option during the installation of VisualAge COBOL.

Program Understanding does not support IMPLICIT transfers of control to COBOL declaratives. It does, however, support EXPLICIT transfers of control. Because Program Understanding is designed to process legacy code, it does not support analysis of programs containing Object Oriented COBOL class or method definitions.

As depicted in Figure 21, using VisualAge COBOL to understand your programs includes:

1. "Selecting Programs for Program Understanding Analysis" on page 72.
2. "Creating SYSADATA" on page 72.
3. "Displaying a Flow Graph" on page 73.
4. "Displaying a Smart Listing" on page 73.

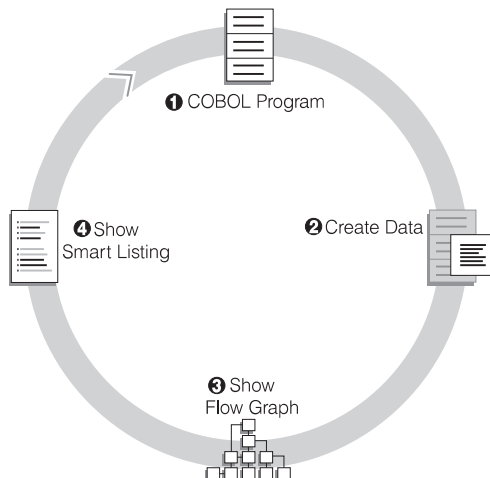


Figure 21. Steps for displaying graphical views using Program Understanding

Creating SYSADATA

Selecting Programs for Program Understanding Analysis



From the project window, select the programs you want to view with Program Understanding. If you have not generated SYSADATA for the programs, you must do so before requesting a Program Understanding show action. Program Understanding can process SYSADATA produced during the compilation of files with the following extensions:

- CBL - COBOL source code
- CVT - COBOL code produced by Program Conversion
- SCB - COBOL code produced by Program Structuring

When you request the first view of a SYSADATA file, Program Understanding develops additional analysis information and stores it in an EU file. Subsequently, you can select either the SYSADATA file or the EU file when requesting that Program Understanding show either the flow graph or the smart listing. Whenever you update your SYSADATA, you need to select that SYSADATA file for the next show action to generate a new EU file.

Note: Program Understanding generates the same EU file from the SYSADATA for both Show Flow Graph and Show Smart Listing; so you can use the same EU file for subsequent show actions.

Creating SYSADATA



You can create SYSADATA by compiling the programs using the ADATA option either on your workstation or on the MVS host. If you compile on the MVS host, provide access the binary SYSADATA files from your project using either Remote Edit/Compile/Debug in VisualAge COBOL or a download utility.

Note: A SYSADATA file in binary and must be accessed in binary. If you are using Remote Edit/Compile/Debug, assure that it was configured for binary data. Refer to the Configuration page of the Information Notebook for details on configuring Remote Edit/Compile/Debug. If you are using a download utility, be sure that it handles binary data correctly.

If your host compiler cannot produce SYSADATA, you can download your MVS host programs to your workstation first, then compile them on your workstation with the ADATA option.

Recommendation

Use the ANALYZE option during SYSADATA generation. This option allows better analysis of programs containing CICS or SQL statements. Note, that when ANALYZE is specified, no executable code is generated.

Analyzing the Year-2000 Impact on Your Programs

Displaying a Flow Graph



Use the **Show Flow Graph** action to display programs graphically so you can see control and data flow and cross-program impacts.

To view a flow graph:

1. From a project, select the programs you want to view. You can select the programs' SYSADATA or EU files.
2. Select **Selected**→**Show Flow Graph** from the menu bar.

VisualAge COBOL displays the programs graphically, illustrating control and data flow.

Displaying a Smart Listing



Use the **Show Smart Listing** action to display the source code in listing format and in a tree structure. It also displays formatted data declarations.

To view a smart listing:

1. From a project, select the program you want to view. You can select the programs' SYSADATA or EU files.
2. Select **Selected**→**Show Smart Listing** from the menu bar.

VisualAge COBOL displays the expanded source for the program. This display shows the expanded source as a listing and in a tree structure with navigational aids. It also shows details of any data declarations you select.

Analyzing the Year-2000 Impact on Your Programs

Use Year 2000 Impact Analysis to understand how your program handles dates and to determine the impact that the second millennium will have on your code. Impact Analysis generates formatted reports that assist you to locate two-digit year fields in your programs.

To perform year 2000 impact analysis, you must have selected the **Program Understanding** option during the installation of VisualAge COBOL.

Because Impact Analysis is designed to process legacy code, it does not support programs containing OO statements.

Impact Analysis can generate two types of reports:

- Formatted report for an individual program.
- Formatted reports for several programs, including cross compilation unit analysis that uses the information gathered from one program unit analysis in the analysis of other program units.

Analyzing the Year-2000 Impact on Your Programs

As depicted in Figure 22 on page 75, to use VisualAge COBOL to identify two-digit year fields in your programs:



1. Select the programs to be analyzed for possible occurrences of two-digit year fields, then compile the source with the ANALYZE and SYSADATA options to produce SYSADATA files, which are input to Impact Analysis. These are the same SYSADATA files used in Program Understanding. Refer to “Creating SYSADATA” on page 72 for details on creating and using SYSADATA.



2. Make a copy of the sample seed file provided with Impact Analysis on the VisualAge COBOL CD-ROM. Modify the seed file, making changes related to knowledge of the programs to be analyzed. For example, make required changes to reflect local naming and coding practices, file formats, and tables

What is a seed file?

It's a way to specify what is known about dates in a program. Impact Analysis follows the data flow from known dates and finds new dates. You can specify known dates, the seeds, in many ways. For example, by variable name pattern or fields in an input file.



3. Run Impact Analysis, using the modified seed file, to generate impact analysis reports. Refer to “Generating Year 2000 Impact Analysis Reports” on page 75 for details on running Impact Analysis.



4. Use the output from that run to further modify the seed file with the new knowledge gained.



5. Repeat the process of running the analysis and modifying the seed file to ensure greater accuracy of the results. Cross compilation unit analysis can assist in this process in order to achieve more accurate results.

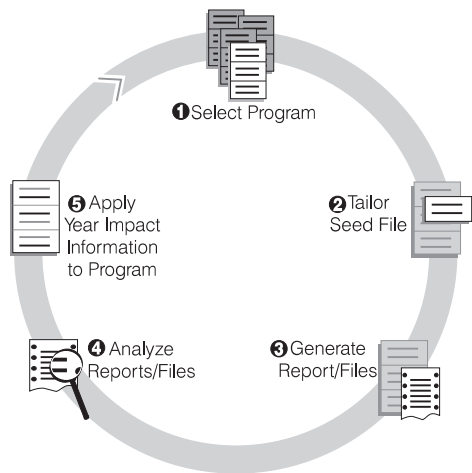


Figure 22. Steps for performing Year 2000 Impact Analysis

Generating Year 2000 Impact Analysis Reports



To create Impact Analysis reports:

1. From a project, select **Project→Open Year 2000 Analysis** from the menu bar to display the Year 2000 Analysis window.
2. Supply names and locations of seed files and SYSADATA files, and **Target Directory** for output reports as requested.
3. Select **Begin Analysis** to start the analysis and generate the report.
4. Repeat this process, refining the seed files to produce more accurate and detailed reports.

More Information about Year 2000 Impact Analysis

VisualAge COBOL provides a scenario that illustrates how to use Impact Analysis to discover two-digit years in a program. Use the Information Notebook to display this scenario.

Using VisualAge COBOL in the Analysis and Maintenance Process

The analysis and maintenance process generally begins with an evaluation of your applications to find the most suitable choices for redevelopment. You can use the Application Understanding to identify good redevelopment candidates and to predict the impact of changes you might want to make.

Your maintenance work might involve upgrading from older COBOL versions to the ANSI X3.23-1985 standard (the American National Standard for Programming Language COBOL, also known as ISO International Standard 1989-1985). Program Con-

version automates much of the upgrade process by converting most of the COBOL syntax for you and converting EXEC CICS commands. Program Structure helps you clean up code, identify dead code, and structure code, which reduces a program's size and makes the code easier to maintain.

You can use Program Understanding to analyze and understand your programs, and you can use Year 2000 Analysis to locate two-digit year fields that may cause a problem when the program starts processing dates later than 1999.

A Common Scenario for Modifying a Program to Meet New User Requirements:

To modify a program to meet new user requirements:

1. Identify impact of affected application program:
 - a. Use Application Understanding to identify programs and datasets affected by the new user-requirement for the current production environment.
 - b. Use Application Understanding to identify other applications that may use the same datasets to be changed.
 - c. If needed, convert the affected programs to a more current level of COBOL, using Program Conversion.
 - d. If needed, structure affected programs that are not structured using Program Conversion.
 - e. Using Program Understanding, learn about each COBOL program that makes up the affected application.
 - f. If you need to update your programs to meet the year-2000 challenge, use Year 2000 Impact Analysis to identify those areas of the code that need changing.
2. Make changes:
 - a. Edit, compile, debug the programs identified as impacted by the fix, using other VisualAge COBOL actions.
 - b. Use MLE support in the VisualAge COBOL compiler to implement changes for year 2000.
 - c. Test the programs identified as impacted by the fix, using other VisualAge COBOL actions.
 - d. Promote to production.

Comparison of Workstation and Mainframe Concepts

Appendix A. Comparison of Workstation and Mainframe Concepts

If workstation concepts are new to you, but you are familiar with the mainframe environment, this topic might help you.

The following table provides a comparison of workstation and mainframe concepts. In most cases, a term-to-term comparison is not possible.

Table 2 (Page 1 of 2). Comparison of workstation and mainframe concepts

Workstation Concept	Mainframe Concept
File: A unit of stored information for text, data, programs, etc.	member: In OS/390 and MVS, a data set is the major unit of data storage and retrieval. A member is a partition of a partitioned data set.
Batch file (.BAT) and Command file (.CMD): a file containing operating system commands organized for sequential processing. In workstation programs, command files are much like JCL. A command file can also function similarly to a CLIST (in OS/390 or MVS) or an EXEC (in VM).	JCL: Stands for job control language, which is used to identify a job to an operating system and to describe the job's requirements. A CLIST is a list of commands and statements designed to perform a specific function for the user, and an EXEC is a user-written command file that contains CMS commands and execution of control statements, such as branches.
Dynamic link library (DLL) file: This file contains executable code and data which is bound to a program at load or run time, rather than during linking. The code and data in a DLL can be shared by several applications at the same time.	DLL files are much like pre-loaded subprograms in the link pack area on MVS and OS/390 or in a shared segment on VM.
Environment variables: These are any number of variables that describe the way an operating system is going to run and the devices it is going to recognize. For example, in a WorkFrame project, an environment variable is an operating system variable like PATH and DPATH, and other environment variables that are defined using the OS/2 SET command such as TMP.	SYS1.PARMLIB and a STEPLIB or PARMLIB statement in JCL perform function similar to workstation environment variables, in that they provide global values or settings, and provide search information.
Executable (.EXE) file: This is a file that contains a program's executable code.	Load module, statically-linked program, statically-linked load module: All or part of a computer program in a form suitable for loading into main storage for execution. A load module is usually the output of a linkage editor.
Module definition (.DEF) file: Directives to the linkage editor on how to build the executable file.	Linkage editor input statements: Directives to the linkage editor, such as INCLUDE and NAME, on how to build the load module.

Comparison of Workstation and Mainframe Concepts

Table 2 (Page 2 of 2). Comparison of workstation and mainframe concepts

Workstation Concept	Mainframe Concept
Make: An action that invokes the make utility, a tool that automates the process of updating project files. This includes compiling and linking programs.	Any automated way of controlling compiling and link-editing.
workstation desktop: A graphical way of accessing your tools and files. It fills the entire screen and holds the objects with which you can interact to perform operations on the operating system.	No exact equivalent. The closest equivalent on the mainframe are menu-driven tools such as ISPF or Office Vision.

Appendix B. Getting Support for Using VisualAge COBOL

You or your company may need more assistance in using VisualAge COBOL. IBM provides support, consulting services, and education for using VisualAge COBOL and the IBM COBOL family of products.

Getting Product Support

There are several ways for you to get product support for VisualAge COBOL: voice support, CompuServe, mail, fixes, World Wide Web, and FAX.

- **Voice support:** Voice support provides assistance for a variety of incidents, such as installation problems and defect reporting. There may be a fee associated with voice support. You may pay for the call on a per incident basis; elect to use bulletin board services (BBS), CompuServe, or the Internet; or purchase an annual contract for voice support. However, if IBM determines that the defect is in the IBM code, you are not charged for the incident.

To report a problem, call 1-800-237-5511 or 1-800-992-4777. These phone numbers are available Monday through Friday, 8:00 a.m. to 5:00 p.m., your time zone.

- **CompuServe:** If you have access to CompuServe, you can enter your comments about COBOL. At the ! command prompt enter, GO IBMLANG. Place your messages or comments regarding COBOL in "Section 11, COBOL Language." Note that if you want a guaranteed response to a problem, call 1-800-237-5511.

If you want to email a *Defect Report Form* through CompuServe, you can find the form in the **Library** area. To submit, send it to the Personal Systems Support Family at 76711.610@CompuServe.com.

For CompuServe membership information, call 1-800-848-8199 and request Representative 239.

- **Fixes:** You can access fixes from the following sources.
 - Call 1-800-237-5511 to request packaged fixes in the form of a CD-ROM.
 - Download the fixes from the FTP site at:
<ftp://ftp.software.ibm.com/ps/products/cobol/fixes>
 - Access fixes from your respective bulletin board services (BBS).
 - Access the World Wide Web and go to the **IBM COBOL Family** page:
 1. Enter the Uniform Resource Locator (URL):
<http://www.software.ibm.com/ad/cobol/cobol.htm>
 2. Click on the **Support** button.

VisualAge COBOL Support

- **Mail:** Mail your comments to:
IBM Corporation
Personal Systems Support Family
Internal Zip 2901
11400 Burnett Road
Austin, Texas 78758
- **FAX:** You can also fax the *Defect Report Form* to IBM at 1-800-426-8602. To receive a copy of this form, call 1-800-992-4777, Monday through Friday, 8:00 a.m. to 5:00 p.m., your time zone.

For support in other countries, contact your local IBM-authorized sales representative.

Getting Consulting Services

IBM provides service offerings for VisualAge COBOL as well as the rest of the IBM COBOL family of products. For more information about consulting services in the United States, call 1-800-IBM-3333, ext. STAR703. To arrange for an IBM representative to discuss your specific COBOL services requirements, call 1-800-IBM-4YOU. For consulting services in other countries, contact your local IBM-authorized sales representative.

If you have access to the World Wide Web, you can access the **IBM COBOL Family** page as follows:

1. Enter the Uniform Resource Locator (URL):
<http://www.software.ibm.com/ad/cobol/cobol.htm>
2. Click on the **Support** button.

Getting Education and Training

IBM provides education and training for VisualAge COBOL as well as the rest of the IBM COBOL family of products. You can request information or enroll in courses in one of the following ways:

- For more information about the course offerings in the United States and Canada, call 1-800-IBM-8322. For education and training in other countries, call 001-520-574-4500. These phone numbers are available Monday through Friday, 8:00 a.m. to 8:00 p.m., Eastern Standard Time (EST).
- If you have access to the World Wide Web, you can access the **IBM COBOL Family** page as follows:
 1. Enter the Uniform Resource Locator (URL):
<http://www.software.ibm.com/ad/cobol/cobol.htm>
 2. Click on the **Education** button.

VisualAge COBOL Glossary

A

action. In a project, a tool or function that can be used to manipulate a project's files, or build a project's target.

advanced program-to-program communication.. Communications protocol between the workstation and the host. SdU for remote edit and compile, including the debug tool, uses the APPC communications protocol.

analyze. In Program Structuring, a function that helps you determine which programs require structuring. VisualAge COBOL analyzes an input program's structural properties. It produces a Reengineering Report and appends to the Complexity Metrics Log.

application. (1) The use to which an information processing system is put; for example, a payroll application, an airline reservation application, a network application. (2) A collection of software components used to perform specific types of user-oriented work on a computer. (3) In the Visual Builder, a GUI project whose target is built as an EXE instead of a DLL.

attribute. (1) In Application Understanding, a characteristic of a component. (2) In DB2/2, a characteristic of data contained in a column or row, such as length, data type, or data.

B

block. In Program Understanding, the largest detectable sequence of statements of either data declarations or procedural statements.

build. An action that invokes the Build tool to create the project's target. The Build tool manages the project's makefile and builds dependencies between projects in a project hierarchy.

build actions. A series of actions that are invoked to build a project's target. These actions are set in the Build options window, or in MakeMake.

C

CICS HANDLE commands. A Customer Information Control System (CICS) command that lets application programs intercept and process exception conditions including user-specified conditions and abnormal termination.

class. The entity that defines common behavior and implementation for zero, one, or more objects. The objects that share the same implementation are considered to be objects of the same class.

complex performed procedures. A procedure has complex usage if the combination of its invocations (through PERFORM, GO TO, or fall-through), together with its internal control flow require Program Structuring to integrate the procedure into the places from which it is invoked in order to structure it. A procedure that has complex usage is also called a Group I procedure.

Complexity Metrics Log. In Program Structuring, a file that contains a summary of the complexity metrics of all programs analyzed or structured. This report provides a useful management tool to help evaluate the maintainability of entire libraries (or systems) of COBOL programs. During each Analysis or Structure run, the complexity metrics are appended to this summary.

component. (1) a functional grouping of related files. (2) In the Visual Builder, a GUI project whose target is built as a DLL instead of an EXE. (3) In Application Understanding, one of the types of information about JCL that you can use to add, modify, or query.

convert. A function in Program Conversion that converts down-level COBOL source code to the more current level of COBOL.

Conversion Log. A Program Conversion file that contains a history of all workstation programs that have been converted.

copy files. A file or library member containing a sequence of code that is included in the source program at compile time using the COPY statement. The file can be created by the user, supplied by COBOL, or supplied by another product.

cycle. The frequency in which a job runs.

Glossary

D

data set. In MVS, a named collection of related data records that is stored and retrieved by an assigned name.

desktop. A metaphor for a computer's working environment—the screen layout, the menu bar, and the program icons associated with the machine's operating environment.

DLL. See *dynamic link library*.

dynamic link library (DLL). A file containing executable code and data bound to a program at load time or run time, rather than during linking. The code and data in a dynamic link library can be shared by several applications simultaneously.

E

event. A representation of a change that occurs to a part. For example, a push button generates an event, as in signalling that it has been pressed.

EXE. See *executable file*.

executable file. A file that contains a program's executable code.

F

file-scoped action. An action that applies only to a file or a group of files, rather than to the project as a whole.

filter. A mask, usually a regular expression, a logical-OR, a logical-AND, or logical-NOT.

G

graphical user interface. A type of computer interface consisting of a visual metaphor of a real-world scene, often of a desk top. Within that scene are icons, representing actual objects, that the user can access and manipulate with a pointing device.

GUI. Acronym for *graphical user interface*.

I

information area. An area of a window in which information about the object or choice that the cursor is on is displayed. The information area can also contain a message about the normal completion of a process. The information area is usually located at the bottom of the window.

information line. The information area of a window.

Input COBOL Report. In Program Structuring, a listing of the input COBOL program with line numbers are attached to each statement. You can use the line numbers as you investigate the control flow properties of the input program documented in the Reengineering Report.

Inventory window. A type of window in Application Understanding. The inventory window contains the set of all components in the database represented by the selected component icon.

inventory database. The database on the workstation used by Application Understanding to store the JCL scan output.

L

link. To interconnect items of data or portions of one or more computer programs, for example, linking object programs by a linkage editor to produce an executable file. Also called object module.

M

make. An action in which a project's target is built from a makefile by a make utility.

makefile. A text file containing a list of your application's files. The make utility uses makefile to maintain application files and dependencies.

MakeMake. The makefile generation utility used in VisualAge COBOL projects.

make utility. A tool that automates the process of updating project files. The make utility compares the modification dates for one set of files (the targets) with those of another set of files (the dependent files, such as source files). If any dependent files have changed more

Glossary

recently than the target files, the make utility runs a series of commands to bring the targets up-to-date.

message line. A type of status area for the COBOL Editor.

modularize. In Program Structuring, to break up a large program into a main program and one or more separately compilable sub-programs.

Monitor. A window that displays output from monitored actions. The Monitor window is attached to the project view.

multiple entries. A statement that can generate more than one entry within the same procedure. Such statements are:

- GO TO statements outside the procedure whose targets lie strictly within the procedure.
- ENTRY statements within the procedure.

multiple exits. A statement that can generate more than one exit within the same procedure: Such statements are:

- GO TO statements within the procedure whose targets lie outside of the procedure
- GOBACK statements within the procedure
- STOP RUN statements within the procedure
- EXIT PROGRAM statements within the procedure
- CALL statements whose call-literal is named in the list of non-return calls.

N

node. In Program Understanding, a graphical representation of one or more blocks. See *block*.

O

object. An entity that has state (its data values) and operations (its methods). An object is a way to encapsulate state and behavior.

object code. Output from a compiler or assembler that is itself executable machine code or is suitable for processing to produce executable machine code.

object program. A set or group of executable machine language instructions or other material designed to interact with data to provide problem solutions. In this

context, an object program is generally the machine language result of the operation of a COBOL compiler on a source program.

P

parent project. A project that contains other projects.

part. In the Visual Builder, the graphic representations of GUI controls that you use to create a GUI, for example, a push button part.

procedure. In MVS, a named collection of job steps that can be included in a job.

program unit. In MVS, the name of a compilable block of code that gains control when the compiled program is run.

project. The complete set of data and actions required to build a target, such as a dynamic link library (DLL) or other executable (EXE).

project hierarchy. A project tree that represents dependencies between projects. The project paradigm requires that one project should be created for every target. Dependencies between projects and their targets should be expressed in a project hierarchy. That is, if a project's build depends on the target of another project, the dependent project should contain the project it depends on. The dependent project is then said to *nest* the other project. This enables the Build tool to perform Builds in a depth-first search manner from anywhere in the project hierarchy.

project-scoped action. An action that applies to a project as a whole, or to a project's specially designated files. Specially designated project files are the project's makefile and target. An example of a project-scoped action is Debug, which is invoked on the project's target.

Q

qualifier. In DB2/2, a short identifier used to filter data from the database. All objects such as tables or queries in DB2/2 are prefixed with a unique qualifier that allows them to be grouped, named, and displayed under a particular category or userid, for example (USERID.QUERYNAME). The object can be viewed only if the correct qualifier is specified.

Glossary

R

Reengineering Report. In Program Structuring, the results of an analysis of the PERFORM logic and control flow, and input program complexity metrics. This report can help you evaluate the complexity and maintainability of the input program. From this, you can determine whether the program requires structuring.

routine. The largest collection of nodes or blocks identified by a callable entry node.

S

scan output. The results of a request for Application Understanding to analyze MVS JCL libraries for information.

scanner. The portion of Application Understanding residing on the MVS host that extracts data from JCL libraries in order to provide information about production applications and the components that comprise them.

Set window. A window in Application Understanding that contains the set of all components in the database represented by the selected component icon.

SOM. See *System Object Model*.

source directory. A directory where a project's files are physically stored. A project may have many source directories.

source type. A list of the kinds of files to which an action applies. The Build tool and MakeMake utility use the source and target types of build actions to determine the order in which the actions should be run to produce the project's target.

status area. The area of a window that displays information indicating the state of the current view of an object. The status area is usually located just below the title bar and menu bar.

sub-program. In COBOL, synonym for called program. A called program is a program that is the object of a CALL statement combined at object time with the calling program to produce a run unit.

subproject. A project contained by another project. Nesting expresses a dependency of the main project on

the sub-project's target. This dependency is managed by the Build utility.

System Object Model (SOM). IBM's object-oriented programming technology for building, packaging, and manipulating class libraries. SOM conforms to the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA) standards.

T

target. In a project, the file that is produced as a result of a project build. For example, an EXE or a DLL.

target directory. Directory in which a target will be built. Usually defaults to the source directory. This is the first source directory listed on the **Locations** page of the project's settings notebook.

target type. A list of files that only apply to actions that participate in a project build, such as Compile and Link. The Build tool and MakeMake utility use the source and target types of build actions to determine the order in which the actions should be run to produce the project's target.

template. An object that you can use as a model to create other objects. When you drag and drop a template, you create a copy of the original object. The new object has the same settings and contents as the original template object.

thread. In an operating system, the smallest unit of operation to be performed within a process.

token highlighting. In the COBOL Editor, a feature that enables you to view the token types of the programming language in different colors and fonts. It makes the structure of the program more visible.

U

user interface. A hardware, software, or both that allows a user to interact with and perform operations on a system, program, or device.

V

view. In an operating system, the appearance of the contents of an open object.

Glossary

W

window. An area of the screen with visible boundaries within which information is displayed. A window can be smaller than or the same size as the screen. Windows can appear to overlap on the screen.

working directory. The directory where files that are

copied or dragged into the project are stored. Actions are also executed in this directory, so this directory is where many output files are placed. Compare with *source directory*.

Glossary

Index

A

- actions
 - for redeveloping legacy code 57
- addItemEnd event 40
- administrator tasks 8
- Analyze/Structure options notebook 68
- analyzing your program 67
- Application Understanding 64
 - creating a new object 64
 - deleting an object 64
 - inventory database 64
 - Inventory window 62
 - JCL object 61
 - qualifiers 63
 - structure view 63
- Application Understanding inventory database
 - viewing contents 62

B

- build files 43
- buttons 33, 36

C

- canvas, in visual builder 33
- CCCA technology 17
- CICS transactions 12
- co-processor, DB2 14
- COBOL/SF technology 17
- Code Assistant
 - description 12
 - updating feature source 52
- coding applications, visual builder 39
- compiler options, making non-overidable 8
- compiler, overview 13
- compiling an application 43
- complex PERFORMs 68
- Complexity log 68
- composition editor 33
 - changing window title 34
 - connecting user code 53
- configuring for Remote E/C/D 22
- connection lines 40
- consulting services 80

- contents parameter 40
- conversion reports 66
- convert log 66
- Convert Results file 67
- converting source code 17
- converting your program 66
- creating a new object in Application Understanding 64

D

- Data Assistant 12
- DB2 co-processor 14
- Debug Tool, remote debugging 21
- debugger
 - description 14
 - for remote E/C/D 21
- deleting an Application Understanding object 64
- disk space requirements 1
- DL/I, accessing remotely 15

E

- editing
 - creating a new file 27
 - description 12
- education 80
- entry field, defining 35
- EU files 72
- event-driving programming 39
- Expert Advice 68

F

- feature source 51
- firstSelected parameter 42
- fixes, access 79
- Flow Graph 73
- free-form surface, in visual builder 33

G

- glossary 81
- GUI applications 12

H

- hard disk requirements 1
- hardware requirements 1
- help files in visual builder projects 32
- host, accessing 21

I

- IMS, accessing remotely 15
- installing
 - command 6
 - shared install 6
 - using a LAN
 - description 7
 - preparing the server 8
- interactive debugger 14
- inventory database 58
- inventory database, modifying 64
- Inventory window 62

J

- JCL object 61
- JCL, retrieving scanned output 60
- JCL, scanning 59

L

- LAN installation 7
- linking an application 43
- list box, defining 35
- loading inventory database
 - with JCL Scan output 61

M

- mainframe concepts, compared with workstation 77
- maintaining legacy code 75
- memory requirements 1
- MLE 13
- modifying Application Understanding inventory database 64
- MVS, accessing 21

N

- nonvisual parts, creating 49

O

- object, JCL 61
- Open a structure view in Application Understanding 63
- opening an Inventory window 62
- operating system requirements 1
- OS/390, accessing 21

P

- Part Interface Editor
 - input into feature source 51
 - when creating a nonvisual part 49
- parts
 - aligning 36
 - changing settings 36
 - completing connections 40
 - connecting parts 39
 - creating in visual builder 34
 - in visual builder project 32
 - nonvisual, creating 49
- parts palette 33
- performance, improving 19
- PERFORMs, complex 68
- populating inventory database 59
- preinstallation tasks 1
- press event 40
- Program Conversion and Structuring
 - structuring unstructured programs 67
- Program Conversion, accessing 66
- Program Structuring
 - advice 68
 - preparing for 68
 - Reengineering Report 68
- Program Understanding
 - producing SYSADATA 72
 - selecting executables 72
 - show flow graph 73
 - show smart listing 73
 - SYSADATA 71
 - using 71
- project
 - building a target 28
 - correcting syntax errors 29
 - creating 23
 - running a target 30
 - visual, creating 32
- Project Smarts 24
- Projects file, creating 24
- push buttons 33

pushbuttons 36

Q

qualifiers, setting in Application Understanding 63

R

RAM, minimum requirements 1
redeveloping legacy code
 invoking actions 57
 overview 16
 Year 2000 Impact Analysis 73
 Year 2000 Impact Analysis reports 75
redeveloping legacy programs
 reengineering programs 65
 structuring programs 67
reengineering programs 65
Reengineering Report 68
remote DL/I 15
remote E/C/D 21
requirements 1
retrieving JCL scanned output 60
running an application 43

S

sample application
 building 23
 running 30
 source code 27
scanning JCL libraries 59
server, setting up as installation base 8
service 79
set qualifier in Application Understanding 63
setting notebook, visual parts 36
shared installation
 description 6
 from CD-ROM 6
Show Flow Graph 73
Show Smart Listing 73
Smart Listing 73
SMARTdata UTILITIES 14
software requirements 1
source code
 generating for visual builder 43
SQL statements 12
starting redevelopment components 57
static text, defining 34

Structure view in Application Understanding 63
structuring unstructured programs 67
structuring your program 17, 70
syntax errors, correcting 29
SYSADATA 72
SYSADATA used with Program Understanding 71
System Interface Editor 51

T

testing your program 71
title bar, changing text in 34
Transaction Assistant 12

U

understanding applications 58
user code
 adding to feature source 52
 connecting 53

V

VCB file, opening 46
Visual Builder
 aligning parts 36
 building applications 43
 changing settings 36
 completing connections 40
 connecting parts 39
 defining a list box 35
 feature source 51
 generating source code 43
 overview 12
 running applications 43
 working with parts 34
visual builder projects
 composition editor 33
 creating 32
 parts 33

W

workstation concepts, compared with mainframe 77

X

XCOPY command 8

Y

Year 2000

- Impact Analysis overview 18

- MLE 13

- using Impact Analysis 73

Year 2000 Impact Analysis

- generating reports 75

- overview 18

- using 73

We'd Like to Hear from You

VisualAge COBOL
Getting Started on OS/2
Version 2.2
Publication No. GC26-9051-02

Please use one of the following ways to send us your comments about this book:

- Mail—Use the Readers' Comments form on the next page. If you are sending the form from a country other than the United States, give it to your local IBM branch office or IBM representative for mailing.
- Fax—Use the Readers' Comments form on the next page and fax it to this U.S. number: 800-426-7773.
- Electronic mail—Use one of the following network IDs:
 - IBMMail: USIB2VVG at IBMMAIL
 - IBMLink: COBPUBS at STLV27
 - Internet: COBPUBS@VNET.IBM.COM

Be sure to include the following with your comments:

- Title and publication number of this book
- Your name, address, and telephone number if you would like a reply

Your comments should pertain only to the information in this book and the way the information is presented. To request additional publications, or to comment on other IBM information or the function of IBM products, please give your comments to your IBM representative or to your IBM authorized remarketer.

IBM may use or distribute your comments without obligation.

Readers' Comments

VisualAge COBOL
Getting Started on OS/2
Version 2.2

Publication No. GC26-9051-02

How satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Technically accurate	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Complete	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to find	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to understand	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Well organized	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Applicable to your tasks	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Grammatically correct and consistent	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Graphically well designed	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Overall satisfaction	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

May we contact you to discuss your comments? Yes No

Would you like to receive our response by E-Mail?

Your E-mail address

Name

Address

Company or Organization

Phone No.

Readers' Comments
GC26-9051-02



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
Department W92/H3
PO Box 49023
San Jose, CA 95161-9945



Fold and Tape

Please do not staple

Fold and Tape

GC26-9051-02

Cut or Fold
Along Line



Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

IBM VisualAge COBOL

GC26-9051	Getting Started on OS/2
GC26-8944	Getting Started on Windows
SC26-9046	Language Reference
SC26-9050	Programming Guide
SC26-9053	Visual Builder User's Guide

GC26-9051-02



Spine information:



VisualAge COBOL

Getting Started

Version 2.2